Software Testing

Notes By Mannan Ul Haq

Software Faults and Failures

Fault (Bug):

- A fault, also known as a bug, is a programming or design error in the software. This means the delivered system doesn't work as it was supposed to according to the specifications.
- **Examples:**
  - **Coding Error:** A mistake in the code, like a typo or incorrect logic.
  - **Protocol Error:** Miscommunication between different parts of the software or between the software and external systems.

Failure:

- A failure happens when the software doesn't deliver the expected service to the user. This usually means there is a mismatch between what the user expects and what the software does.
- **Examples:**
  - **Mistake in Requirements:** The requirements may not correctly reflect what the user needs.
  - **Confusing User Interface:** The software might be difficult to use, leading to user errors.

Why Does Software Fail?

Software can fail for several reasons, including:

1. **Wrong Requirement:**
   - The software does not meet what the customer actually wants.
   - **Example:** A client wants a banking app that shows transaction history, but the app only shows the last 5 transactions instead of all.
2. **Missing Requirement:**
   - The software lacks a necessary feature or functionality.
   - **Example:** An e-commerce website fails to include a checkout option.
3. **Requirement Impossible to Implement:**
   - The requested feature or functionality cannot be built due to technical constraints.
   - **Example:** A client requests real-time language translation for handwritten notes, but the current technology cannot support it accurately.
4. **Faulty Design:**

- There are errors in the system's design, which means it won't function correctly even if coded perfectly.
- **Example:** A poorly designed database that slows down when accessed by multiple users simultaneously.

5. **Faulty Code:**
   - Errors in the code that prevent the software from functioning correctly.
   - **Example:** A loop in the code that never ends, causing the program to freeze.
6. **Improperly Implemented Design:**
   - The design may be correct, but it is not implemented correctly in the code.
   - **Example:** The design specifies a user-friendly navigation menu, but the implemented menu is complex and hard to use.

Objective of Testing

- **Main Goal:** Find faults (bugs) in the software to ensure it works correctly.
- **Successful Test:** A test is successful if it finds a fault.

Fault Identification

- **Definition:** Figuring out which fault caused a failure.
- **Example:** If the software crashes during a file upload, determine if it’s due to a coding error or another issue.

Fault Correction

- **Definition:** Fault correction is the process of making changes to the system so that the faults are removed.
- **Example:** After identifying incorrect file handling as the cause of a crash, modify the code to handle files correctly.

Elements of a Test Case

1. **Purpose:** The reason for the test.
2. **Input:** The data used in the test.
3. **Expected Output:** The result that should be produced if the software works correctly.
4. **Actual Output:** The result that is actually produced during the test.
5. **Result:** Indicates whether the test passed or failed based on comparing expected and actual outputs.

Sample Test Case Format

Test Case ID	Purpose	Input Data	Expected Output	Actual Output	Result

TC001	Verify login function	Username: user, Password: pass	Login successful	Login successful	Pass
TC002	Check error message for wrong password	Username: user, Password: wrongpass	Error message displayed	Error message displayed	Pass

Types of Faults

Algorithmic Fault (Logic)

- **Definition:** Errors in the logic or algorithm used in the software.
- **Example:** Incorrect implementation of a formula.

Computation and Precision Fault

- **Definition:** Errors in calculations or precision of values.
- **Example:** A formula that produces incorrect results.

Documentation Fault

- **Definition:** Discrepancies between documentation and the actual program behavior.
- **Example:** User manual says the software supports feature X, but it does not.

Capacity or Boundary Faults

- **Definition:** Performance issues when certain limits are reached.
- **Example:** System slows down when processing more than 1000 transactions.

Timing or Coordination Faults

- **Definition:** Errors related to the timing or synchronization of tasks.
- **Example:** Two processes trying to access the same resource simultaneously, causing a conflict.

Performance Faults

- **Definition:** The system does not perform at the required speed.
- **Example:** Software takes too long to load.

Different Levels of Failure Severity

1. Catastrophic

Causes death or total system loss.

2. Critical

Causes severe injury or major system damage.

3. Marginal

Causes minor injury or minor system damage.

4. Minor

Causes no injury or system damage.

Black Box Testing

A testing method that focuses on the functionality of the software without looking at the internal code structure. The tester checks if the software performs as expected by providing inputs and examining outputs.

Techniques

1. **Equivalence Class Partitioning**

   Dividing input data into equivalent partitions (classes) where all the inputs in a class are expected to produce similar results.

   Example:

   Imagine a software application that requires users to input their age, which must be between 18 and 65. The goal is to verify that the software correctly handles valid and invalid age inputs.

   Steps:

   1. Identify equivalence classes:
      - **Valid class:** Ages from 18 to 65.
      - **Invalid classes:** Ages below 18 and above 65.
   2. Select representative values from each class:
      - **Valid class:** Pick a value within the range (e.g., 25).
      - **Invalid class 1 (below 18):** Pick a value below the range (e.g., 15).

2. **Boundary Value Analysis**

   Testing the boundaries of input ranges. This technique involves creating test cases for values at the edges of equivalence classes.

   To perform boundary value testing, you need to set the boundary values or input variables at:

   - Minimum value.
   - Just above the minimum.
   - Nominal Value.
   - Just below Max value.
   - Max value.

   Example:

Example: Consider a system that accepts ages from 18 to 56.

Boundary Value Analysis(Age accepts 18 to 56)		
Invalid (min-1)	Valid (min, min + 1, nominal, max – 1, max)	Invalid (max + 1)
17	18, 19, 37, 55, 56	57

White Box Testing

Also known as clear box or glass box testing, this method involves testing the internal structure, design, and implementation of the software. The tester needs to have knowledge of the code and the system's internal workings.

Technique

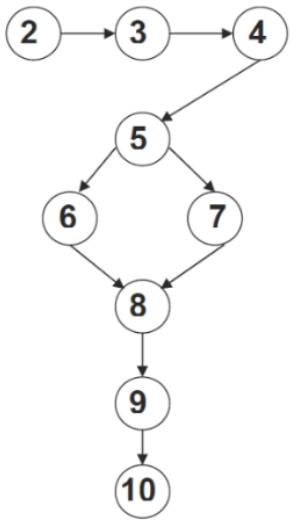
Control Flow Testing

Control flow testing involves testing the execution paths and flow of the program. This helps ensure that all paths and branches of the program are tested.

Steps:

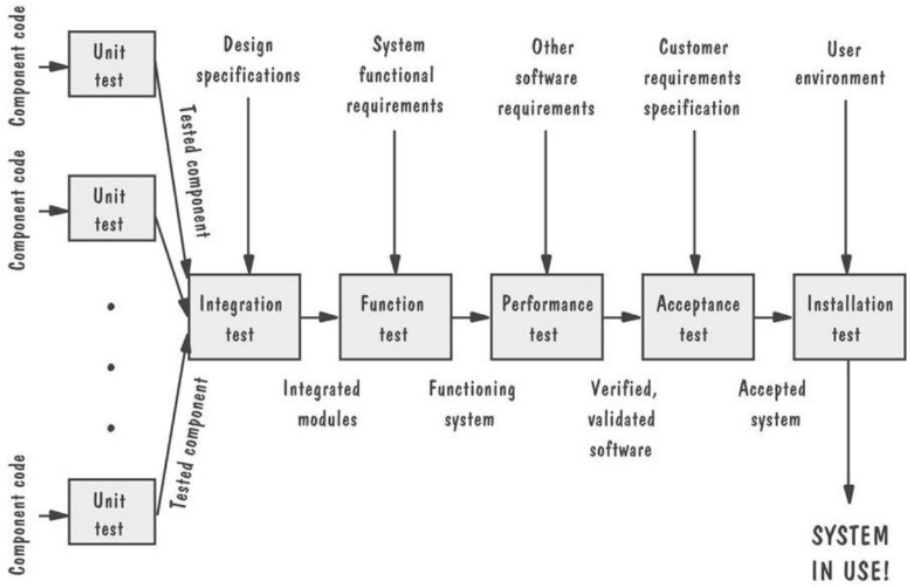
1. Create a flowchart of the code.
2. Determine the cyclomatic complexity (number of independent paths through the code).

1. *Program 'Simple Subtraction'*
2. *Input (x, y)*
3. *Output (x)*
4. *Output (y)*
5. *If x > y then DO*
6. *x – y = z*
7. *Else y – x = z*
8. *EndIf*
9. *Output (z)*
10. *Output "End Program"*



Cyclomatic Complexity = E – N + 2 = 9 – 9 + 2 = 2

Levels of Testing

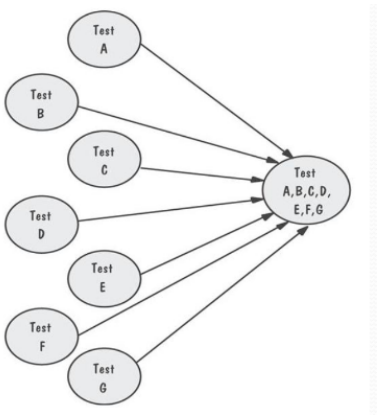


1. Unit Testing

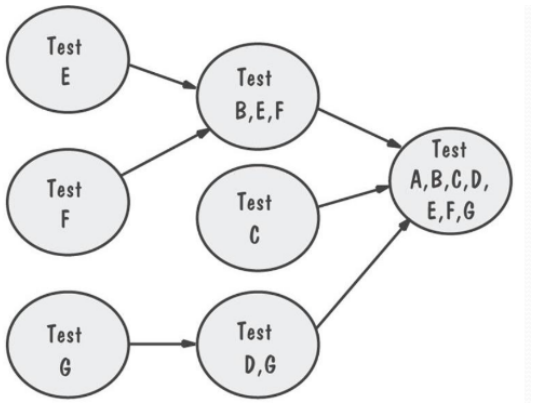
- **Purpose:** Test individual units or components for correct functionality and execution.
- **Steps:**
  - **Determine Test Objectives:** Define what you aim to achieve with the tests.
  - **Select Test Cases:** Choose specific scenarios to test the unit.
  - **Execute Test Cases:** Run the tests and verify the results.

2. Integration Testing

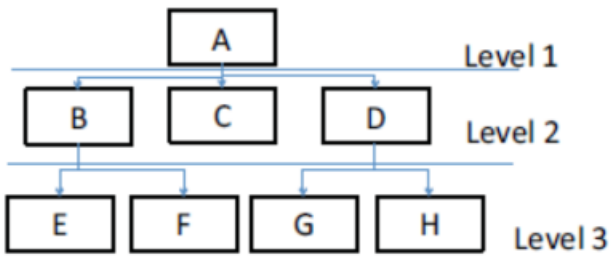
- **Purpose:** Test the interaction between integrated units or components.
- **Approaches:**
  - **Big-Bang:** All components are integrated at once.



- **Bottom-Up:** Testing starts from the lower-level components up to the main system.
  - Uses **drivers** to call child functions.

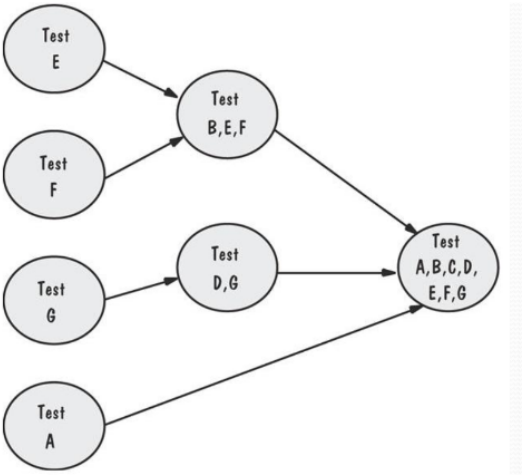


- **Top-Down:** Testing starts from the top-level modules down to the lower-level components.
  - Uses **stubs** to simulate missing components.



- **Sandwich Testing:** Combines bottom-up and top-down approaches.

- Three layers are used, applying bottom-up where writing drivers is not costly and top-down where stubs are easier to write.



- **Big-Bang:** Integrate the entire login system with the database at once and test.
- **Bottom-Up:** Test lower-level functions like database connection and then move upwards.
- **Top-Down:** Test the login interface first with stubs for the database interactions.
- **Sandwich:** Combine testing of UI and database interactions in layers.

3. System Testing

- **Purpose:** Test the complete system as a whole to ensure it meets requirements.
- **Types of Testing:**
  - **Functional Testing:** Tests the GUI and overall functionality.
  - **Scalability:** Ensures the system can handle increased loads.
  - **Performance Testing:** Verifies the speed and responsiveness.
  - **Sanity Testing:** Quick checks to see if the system works at a basic level.
  - **Usability Testing:** Checks if the system is user-friendly.
  - **Smoke Testing:** Preliminary tests to check basic functionalities.
  - **Load Testing:** Tests system performance under heavy loads.
  - **Regression Testing:** Ensures new code changes do not affect existing functionalities.
  - **Volume Testing:** Tests system performance with large volumes of data.
  - **Compatibility Testing:** Ensures the system works across different environments.
  - **Stress Testing:** Tests system behavior under extreme conditions.
  - **Installation Testing:** Verifies the system installs and operates correctly in the target environment.

- **Security Testing:** Ensures the system is secure from threats.
- **Ad hoc Testing:** Informal testing without planning and documentation.

4. Acceptance Testing

- **Purpose:** Enable customers and users to determine if the system meets their needs and expectations.
- **Conducted By:** Customers or users.
- **Results:** Identifies requirements that are not satisfied, must be deleted, revised, or added.

5. Installation Testing

- **Purpose:** Verify that the system has been installed properly and works in the target environment.
- **Steps:**
  - **Before Testing:** Configure the system, attach proper devices, and establish communication with other systems.
  - **During Testing:** Verify the system's correct installation and functionality.

Software Testing

Automated Testing Tools: Selenium, QTP (Micro Focus UFT), SilkTest, WinRunner, LoadRunner, JMeter

Testing Management Tools: TestManager, TestDirector (Micro Focus ALM)

Bug Tracking/Configuration Management Tools: Bugzilla, Jitterbug, SilkRadar