GYAAN SANDHI
YOUR BRIDGE TO BRILLIANCE

SPPU

UNIT 3

Test case design techniques

white box testing

Introduction

- White Box Testing examines the internal code, logic, and structure of the software.
- Tester must have complete knowledge of the source code.
- Also known as Clear Box, Glass Box, or Structural Testing.
- Uses control structures of the program to design test cases.

What White Box Testing Checks

- Program flow control
- Loops, conditions, and branches
- All possible execution paths
- Internal logic errors
- Security vulnerabilities such as weak key handling
- Helps detect errors early in development

white box testing

Levels of White Box Testing

- Unit Testing – testing individual functions
- Integration Testing – testing combined modules
- System Testing – testing full internal structure
- Suitable for both small and large routines

Advantages of White Box Testing

- Can be performed by the development team
- No need for a completed GUI
- Highly systematic and repeatable
- Covers every possible path
- Detects issues early in the SDLC

white box testing

Disadvantages of White Box Testing

- Requires significant developer time
- Code changes can invalidate test cases
- Hard to maintain a separate testing environment
- Requires high technical skill
- Not suitable for large and complex codebases (costly effort)

Black box testing

Introduction

- Black box testing is a testing method in which there is no need of knowledge of code structure,logic or functions.
- Black box testing only focuses on input and output
- Tester need not to know about how output is generated
- It only focuses on providing input and checking if output generated is valid or not.
- It is also known as functional testing.

How black box testing works ?

Steps included :

1.Define the initial state of component.

Black box testing

2.Decide the tests inputs
3. Write the expected output
4. observe the output
5. match the expected as well as actual output.
If it matches test is passed and if not bug is detected.

Here , no internal knowledge about code is required to the tester.
Test cases are generated on the basis of specifications/requirements .
System is said to be as black box as there is no knowledge about inside

Example : Search Engine :
When you type on search bar and click on search button you do not know the process behind backend , how data is being fetched or how the result is being calculated. only input and output is checked .

Input : Search item & output : list of results

Black box testing

Black box testing mainly focuses on :-

1. Functional Behavior
2. if requirements is met or not
3. Input-output correctness
4. behaviour of software on basis of user perspective.

It means behaviour-oriented testing

Grey box testing

Introduction

- Grey Box Testing is a hybrid technique combining Black Box and White Box testing.
- Tester has partial knowledge of the internal code structure.
- Focuses on both functional behavior and internal architecture.
- Suitable for improving overall product quality.

Characteristics of Grey Box Testing

- Tester knows high-level design, not full source code.
- Test cases are created using algorithms for:
  - Internal states
  - Program behavior
  - Application architecture
- Tester only executes and interprets test results.
- Covers both functional and non-functional testing.

Grey box testing

Grey Box Testing Techniques

1. Matrix Testing

- Examines all variables and their relationships within the application.

2. Regression Testing

- Ensures new changes or bug fixes do not introduce errors in existing components.

3. Pattern Testing

- Analyzes past defects to identify recurring patterns leading to errors.

4. Orthogonal Array Testing

- Used when the application has complex or large input sets.
- Optimizes test cases to balance coverage and testing effort.

Grey box testing

Advantages of Grey Box Testing

- No programming expertise required for testers.
- Combines strengths of both Black Box and White Box testing.
- Reduces conflicts between developers and testers.
- More cost-effective than full integration testing.
- Provides better test coverage than pure Black Box testing.

Disadvantages of Grey Box Testing

- Limited access to internal structure leads to restricted code path traversal.
- Cannot be used for algorithm-level testing.
- Designing effective test cases can be difficult.
- Coverage is not as comprehensive as White Box testing.

Grey box testing

Advantages of Grey Box Testing

- No programming expertise required for testers.
- Combines strengths of both Black Box and White Box testing.
- Reduces conflicts between developers and testers.
- More cost-effective than full integration testing.
- Provides better test coverage than pure Black Box testing.

Disadvantages of Grey Box Testing

- Limited access to internal structure leads to restricted code path traversal.
- Cannot be used for algorithm-level testing.
- Designing effective test cases can be difficult.
- Coverage is not as comprehensive as White Box testing.

Static Test Case Design Techniques

Introduction

- A test design technique helps to select the best test cases from all possible tests.
- Static testing does not execute the code.
- It checks documents, design, and source code manually to detect early defects.
- Evaluates the software product and related artifacts without running the program.

What is Static Testing?

- Detects defects without executing software.
- Uses white box methods such as reviews, inspections, and walkthroughs.
- Ensures correctness of code logic, algorithms, and documentation.
- Helps identify errors before compilation, making further testing efficient.

Static Test Case Design Techniques

Static Testing Methods

1. Code Inspections

- Manual review of software code to find defects.

2. Code Analyzers

- Automated tools that detect:
  - Type errors
  - Memory leaks
  - Syntax issues
  - Style violations

3. Compiling

- Errors found during compilation are part of static defect detection.

4. Modeling Techniques

- Requirements are modeled to verify correctness and completeness.

Static Test Case Design Techniques

Features of Static Testing

- Conducted early in the development life cycle.
- Prevents major defects from appearing in later stages.
- Improves quality by validating logic, structure, and documentation.
- Reduces overall testing and debugging effort.

Advantages of Static Testing

- Early feedback on quality issues.
- Low cost of fixing defects when detected early.
- Improved development productivity.
- Encourages collaboration and knowledge exchange within the team.
- Increases awareness of quality standards.
- Highest chance of detecting defects before execution.

Static Test Case Design Techniques

Disadvantages of Static Testing

- Time-consuming due to manual reviews.
- Cannot test data dependencies since execution is not involved.
- Requires high-level skills and expertise to review code and documents.

Informal review , walkthroughs & inspections.

1.Informal Reviews

- Conducted during early stages of the document life cycle.
- Usually performed by 2-person teams (author + colleague).
- Reviewers are peers of the author.
- Aim: Find defects and discuss them in a short meeting.
- Goal: Help the author & improve document quality.
- Not documented formally; no strict process.
- Can review handwritten notes, code listings, documents, etc.
- Very flexible and quick.

2. Walkthroughs

- Semi-formal type of review.
- Conducted by a group of people, usually without much preparation.

Informal review , walkthroughs & inspections.

- Example: Requirements traceability review (checking requirement → design → code → test).
- All issues found are documented on CAR (Corrective Action Request) forms.
- A designer/programmer leads the walkthrough.
- Participants ask questions and point out defects or standard violations.
- Best for source code, small modules, documents.
- The effectiveness depends on the producer (review leader).

3. Inspections

- Most formal and disciplined type of review.
- Used to detect & correct defects early and prevent defect leakage.
- More formal than walkthroughs; includes metrics collection.

Informal review , walkthroughs & inspections.

- Helps in process control: planning, measurement, and control.
- Used to collect quantitative quality data.

Provides:

- Feedback to developers
- Feed-forward to future development
- Input to next process steps
- Inspections are done at various stages (Fagan’s recommendation):
- Detailed design
- Cleanly compiled code
- Completed unit tests

Team structure includes:

- Moderator
- Author

Informal review , walkthroughs & inspections.

- At least one peer inspector
- Walkthroughs vs Inspections:
- Walkthrough → led by author, less formal
- Inspection → led by moderator, more formal
- Inspection Steps:
- Planning
- Overview
- Preparation
- Meeting
- Rework
- Follow-up

Statement Coverage Testing

What is Statement Coverage?

- A white-box testing technique.
- Ensures every statement in the source code executes at least once.
- Used to measure how many statements were executed vs total statements.
- Helps verify that all parts of the code are tested at the statement level.

Purpose

- To check whether all statements are executed.
- Helps identify unused or skipped parts of code.
- Good for basic verification of code behavior.
- Not useful for testing control flow paths or conditions.

Statement Coverage Testing

Limitations

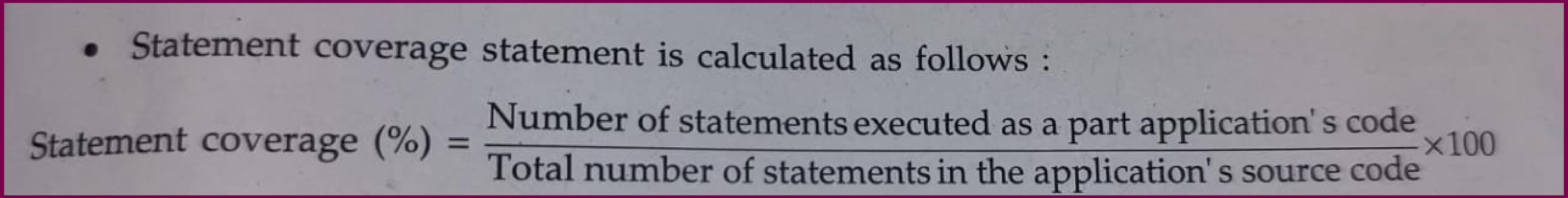
- Does not guarantee testing of all decision-making paths.
- Example:
- A statement may be reachable from multiple conditions, but only one path might be tested → other conditions remain untested.

When is it used?

- During the build and development phases.
- Applied as part of code validation in SDLC.
- Ensures correctness and helps detect missing or unreachable statements.

Statement Coverage Testing

Formula for Statement Coverage



- Statement coverage statement is calculated as follows :

$$\text{Statement coverage (\%)} = \frac{\text{Number of statements executed as a part application's code}}{\text{Total number of statements in the application's source code}} \times 100$$

Example Code

```
1. int x = 10;
2. if (x > 5)
3.     x = x + 1;
4. else
5.     x = x - 1;
6. print(x);
```

Statement Coverage Testing

Test Case

- Input: x = 10

Path executed:

- Line 1 → Line 2 → Line 3 → Line 6

Coverage Calculation

- Total statements: 6
- Executed statements: 4 (Lines 1, 2, 3, 6)

$$\text{Statement Coverage} = \frac{4}{6} \times 100 = 66.6\%$$

Branch Coverage Testing

Definition

- Branch coverage ensures that each branch (true/false path) of every decision point (if, else, switch, loops) is executed at least once.
- Also known as Decision Coverage Testing.
- Helps cover all possible execution paths that can occur after a decision.

Purpose

- To verify that every branch/edge in the control flow has been tested.
- Ensures that no branch leads to unexpected behavior.
- Solves the limitation of statement coverage, which may skip untested paths.

Branch Coverage Testing

Why is Branch Coverage Important?

- Shows how thoroughly the application is tested.
- Low branch coverage means many decision paths are untested → possible hidden bugs.
- Detects defects that occur only in edge cases or rare conditions.

How Branch Coverage Works

- Each condition must be tested with:
  - True branch
  - False branch
- Ensures complete decision-level testing.

Branch Coverage Testing

Code:

- if (x > 10)
-     y = 1;
- print(y);
- If tested only with x = 20, only the true branch executes.
- The false branch (x <= 10) is not executed → branch coverage incomplete.

Advantages (Pros)

1. Ensures all branches are reached at least once.
2. Guarantees no branch causes abnormal behavior.
3. Removes the issues found in statement coverage (missing conditions).

Branch Coverage Testing

Formula for Branch Coverage

$$\text{Branch Coverage (\%)} = \frac{\text{Number of executed branches}}{\text{Total number of branches}} \times 100$$

Path Coverage Testing

Definition

- Path coverage testing is a white-box testing technique.
- Ensures testing of all possible paths from program entry to exit.
- A path is a unique sequence of decisions (true/false branches) executed in the program.

Purpose

- To test every independent path in the program at least once.
- Helps find errors caused by:
  - Different sequence of execution
  - Complex decision combinations
- Works with loops, branches, and conditionals.

Path Coverage Testing

Features

- Test cases are prepared based on logical complexity of the procedural design.
- Every statement in the program is executed at least once.
- Flow graphs, cyclomatic complexity, and graph metrics are used to identify basis paths.
- Both true and false branches of all conditions are executed.

Limitations

- Testing all possible paths can be impractical due to:
  - Loops with many iterations
  - Large number of decision combinations
  - Different orders of execution causing errors
- Some bugs may arise because programmers forgot certain code, making some paths non-existent.

Path Coverage Testing

How it Works

- Identify independent paths in the flow graph.
- Execute each path at least once.
- Use cyclomatic complexity to reduce redundant test cases.

Example

Code:

```
1. if (x > 0)
2.    y = 1;
3. else
4.    y = 2;
5. if (z > 0)
6.    y = y + 1;
```

Path Coverage Testing

- Independent paths:
  a. x>0 true, z>0 true → Path 1
  b. x>0 true, z>0 false → Path 2
  c. x<=0 false, z>0 true → Path 3
  d. x<=0 false, z>0 false → Path 4\

- Each path ensures different branches and conditions are tested.

Conditional Coverage Testing

Definition

- Conditional coverage testing is a white-box testing technique.
- Focuses on testing individual conditions in a decision statement.
- Ensures that each condition evaluates to both true and false at least once.

Purpose

- To verify that every Boolean condition in a program is tested.
- Ensures no condition is left untested.
- Helps detect errors that may occur due to specific condition outcomes.

Conditional Coverage Testing

Key Points

- Decision outcome itself is not the main focus; focus is on individual conditions.
- All possible combinations of conditions are not required (unlike multiple condition coverage).
- Each condition should be:
  - True at least once
  - False at least once

Example

Code:

- if (x > 0 && y < 5)
-     z = 1;

Conditional Coverage Testing

Conditions:

1. x > 0 → must be true and false
2. y < 5 → must be true and false

- Test Cases for Conditional Coverage:
- Case 1: x = 1, y = 2 → both true
- Case 2: x = -1, y = 2 → first false, second true
- Case 3: x = 1, y = 6 → first true, second false
- (Combinations are not the focus, just each condition’s outcome)

Loop Coverage Testing

Definition

- Loop coverage is a white-box testing technique.
- Focuses on testing loops (while, do-while, for, etc.) in a program.
- Ensures loops work correctly for different iteration scenarios.

Purpose

- Test loops with minimal, typical, and maximal iterations.
- Detect errors caused by:
  - Fewer than minimum iterations
  - More than maximum iterations
- Helps ensure program stability under different loop conditions.

Loop Coverage Testing

Importance of Loop Testing

1. Detects loop repetition issues.
2. Verifies performance and functionality of loops.
3. Identifies uninitialized variables inside loops.
4. Detects loop start and boundary issues.
5. Helps resolve routine/logic errors in loops.
6. Reveals performance bottlenecks caused by loops.

Types of Loops Tested

1. Simple Loops:
    - Loops with no nested loops inside.
2. Nested Loops:
    - Loops inside another loop.
3. Concatenated Loops:
    - Loops executed one after another sequentially.

Loop Coverage Testing

Testing Approach

- Test loops at different iteration counts:
    - Zero iterations (if allowed)
    - Minimum iterations
    - Typical iterations
    - Maximum iterations
    - Beyond maximum (to check robustness)

Boundary Value Analysis

Definition

- Boundary Value Analysis is a black-box testing technique.
- Focuses on testing the boundaries of input, output, or internal values.
- Programmers often make errors at the boundary of equivalence classes.

Purpose

- To detect defects at edge cases of input and output values.
- More effective than testing random values inside equivalence classes.
- Ensures system behaves correctly for minimum, maximum, and near-boundary values.

Boundary Value Analysis

How It Works

- Test values directly on, above, and below the edges of equivalence classes.
- Helps catch errors that occur at limits of the input domain.
- Often used as an extension of Equivalence Partitioning.

BVA Strategy

1. Choose one arbitrary value in each equivalence class.
2. Choose values exactly on the lower and upper boundaries.
3. Choose values just below and just above each boundary (if applicable).

Boundary Value Analysis

Example

- Input range: 1 to 10
- Equivalence Classes:
  - Valid: 1–10
  - Invalid: <1, >10
- Test Cases:
  - On boundaries: 1, 10
  - Just below/above: 0, 11
  - Arbitrary inside: 5
- This ensures all boundary-related defects are tested.

Equivalence Class Partitioning

Meaning

- Equivalence Class Partitioning is a test design technique where input values of a system are divided into groups based on similar behavior.
- Instead of testing each possible input, one value is chosen from every group (class) to represent the entire group.

Why We Use It

- Reduces the total number of test cases.
- Saves time and effort while still maintaining good test coverage.
- Ensures that all meaningful input ranges are tested.

Equivalence Class Partitioning

How It Works

1. Identify the input condition(s).
2. Divide inputs into valid and invalid classes.
3. Select one test value from each class.

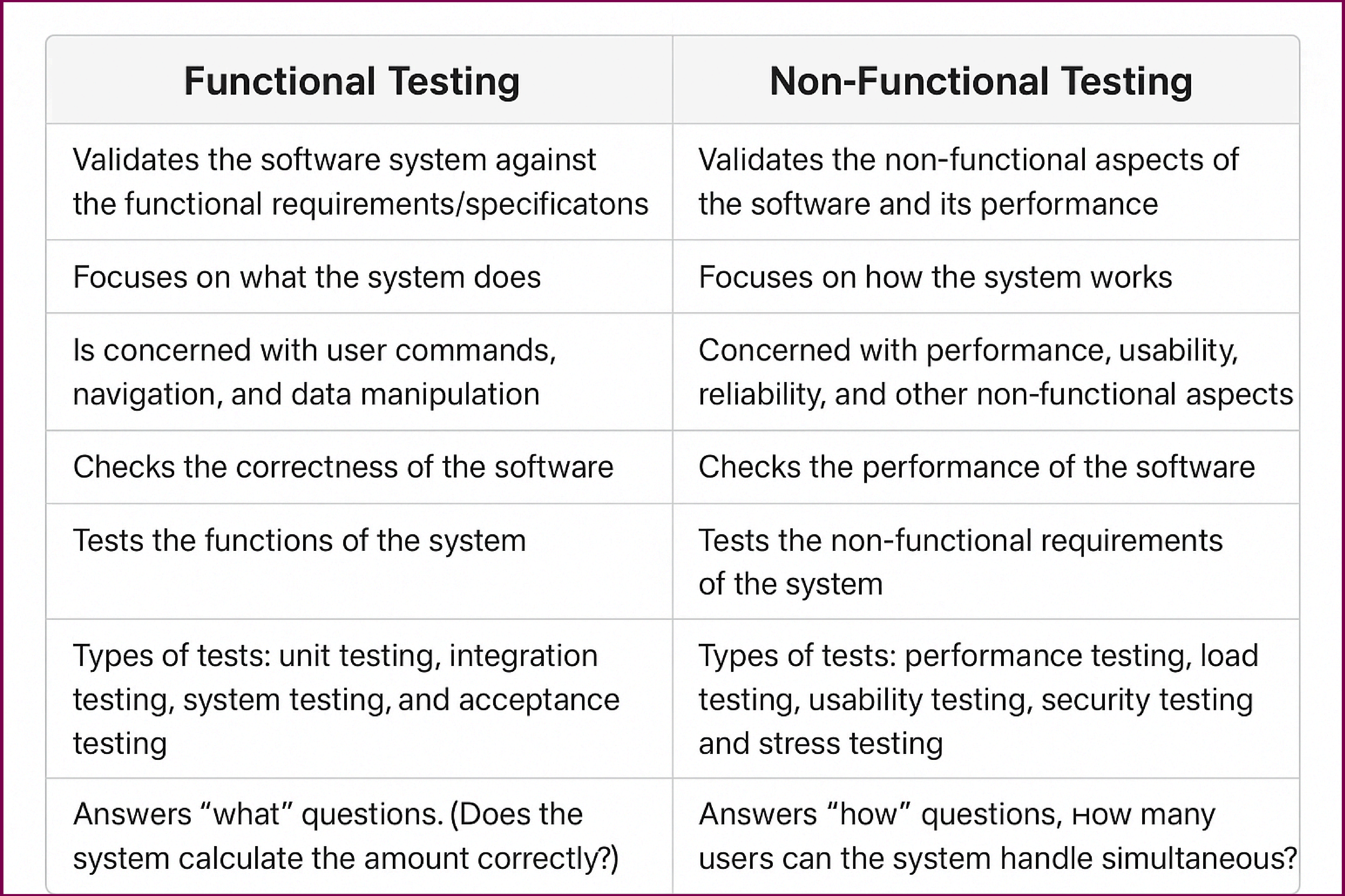
Example

- A text field "AGE" accepts values only between 18 to 60.
- Classes formed:
  - Invalid Class 1: ≤17
  - Valid Class: 18–60
  - Invalid Class 2: ≥61
- Only 3 test cases are required to cover all possibilities.

Benefit

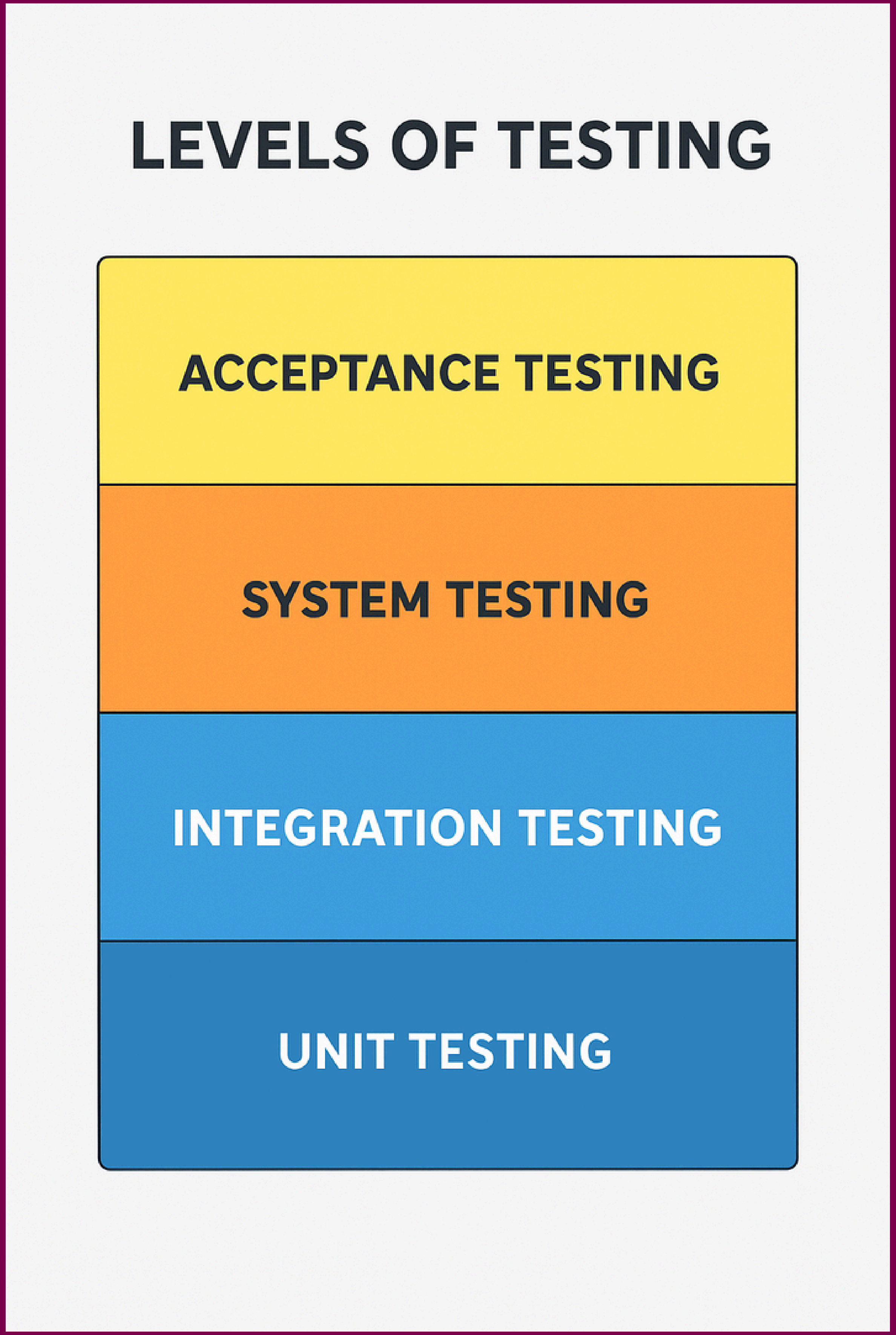
- Maximum

Functional & non – functional testing



Functional Testing	Non-Functional Testing
Validates the software system against the functional requirements/specificatons	Validates the non-functional aspects of the software and its performance
Focuses on what the system does	Focuses on how the system works
Is concerned with user commands, navigation, and data manipulation	Concerned with performance, usability, reliability, and other non-functional aspects
Checks the correctness of the software	Checks the performance of the software
Tests the functions of the system	Tests the non-functional requirements of the system
Types of tests: unit testing, integration testing, system testing, and acceptance testing	Types of tests: performance testing, load testing, usability testing, security testing and stress testing
Answers “what” questions. (Does the system calculate the amount correctly?)	Answers “how” questions, How many users can the system handle simultaneous?

Levels of testing



SHARE

subscribe

Thank You