UNIT–IV

SOFTWARE TESTING

Student-Friendly • Exam-Oriented • Relatable Examples • Quick Revision

🏫 RUNNING CASE STUDY — COLLEGE CANTEEN ORDERING SYSTEM

We will use one familiar system throughout the unit: Student logs in → selects food → adds items to cart → pays → receives confirmation → canteen prepares the order. Testing examples will use this system wherever possible.

UNIT-IV Syllabus Coverage

Area	Topics
Testing Basics	Testing Process, Test Case Design
Functional Testing	Boundary Value Analysis, Equivalence Class Testing, Decision Table Testing, Cause-Effect Graphing
Structural Testing	Path Testing, Data Flow Testing, Mutation Testing
Testing Levels	Unit Testing, Integration Testing, System Testing
Supporting Activities	Debugging, Alpha & Beta Testing, Regression Testing

⭐ EXAM TIP

A strong answer usually follows: Definition → Purpose → Steps/Types → Relatable Example → Advantages/Limitations → Short conclusion.

1. Software Testing

Software Testing is the process of evaluating software to find defects and to check whether the software behaves as expected according to its requirements.

💡 RELATABLE EXAMPLE

For the canteen system, if a student enters the correct login details, the system should allow login. If the student enters an invalid password, the system should reject it and show a useful message. Testing checks both expected and unexpected situations.

🧠 MEMORY AID

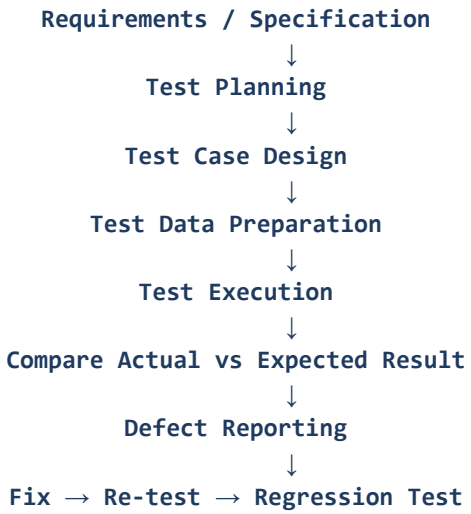
Testing = Find defects + Verify expected behavior + Build confidence in software quality.

1.1 Why is Testing Needed?

- To discover defects before software reaches users.
- To check whether requirements are satisfied.
- To verify expected inputs produce expected outputs.
- To check important error and boundary conditions.
- To improve reliability, usability and confidence in the system.

1.2 Testing Process

A testing process is a planned sequence of activities used to prepare, execute and evaluate tests.



1.3 Important Testing Terms

Term	Meaning	Canteen Example
Test Case	A specific set of inputs, conditions and expected result used to test one situation	Valid login test
Test Data	Input values used during testing	Roll No. + password
Expected Result	What should happen if the software works correctly	Dashboard opens
Actual Result	What actually happens during execution	Dashboard does not open
Defect/Bug	A problem where actual behavior differs from expected behavior	Valid user is rejected
Pass	Actual result matches expected result	Order confirmation appears
Fail	Actual result differs from expected result	Payment succeeds but order is not created

2. Test Case Design

Test Case Design is the process of creating test cases that systematically check whether software functions correctly.

2.1 Typical Test Case Format

Field	Example
Test Case ID	TC-LOGIN-01
Objective	Check login with valid credentials
Precondition	Student account exists
Input	Valid roll number and password
Steps	Open login → enter details → click Login
Expected Result	Student dashboard is displayed
Actual Result	Filled during execution
Status	Pass / Fail

2.2 Good Test Case Characteristics

- Clear and easy to understand.
- Traceable to a requirement.
- Has a definite expected result.
- Can be repeated consistently.
- Includes valid, invalid and boundary situations where appropriate.

⭐ **EXAM TIP**

Do not write only normal cases. Good test design deliberately checks valid, invalid, boundary and special conditions.

3. Functional Testing

Functional Testing checks whether software functions behave according to the specified requirements. It focuses on inputs, outputs and externally visible behavior rather than the internal code structure.

💡 RELATABLE EXAMPLE

For the canteen system, functional testing asks: “If the student orders 2 burgers and pays successfully, does the system create the correct order and show the correct confirmation?”

🧠 MEMORY AID

Functional Testing = WHAT the software does.

3.1 Boundary Value Analysis (BVA)

Boundary Value Analysis is a test-case design technique that focuses on values at the edges of an input range, because defects often occur at boundaries.

💡 RELATABLE EXAMPLE

Suppose the canteen allows a student to order between 1 and 10 items in one order. Important boundary values include 1 and 10, plus nearby values such as 0, 2, 9 and 11.

Category	Example value	Expected idea
Below minimum	0	Invalid
Minimum	1	Valid
Just above minimum	2	Valid
Just below maximum	9	Valid
Maximum	10	Valid
Above maximum	11	Invalid

⭐ EXAM TIP

For a range Min–Max, think about Min−1, Min, Min+1, Max−1, Max, Max+1. The exact set depends on the test-design convention being used.

3.2 Equivalence Class Testing (Equivalence Partitioning)

Equivalence Class Testing divides the input domain into groups (classes) in which values are expected to behave similarly. Instead of testing every value, representative values from each class are tested.

💡 RELATABLE EXAMPLE

If an age field accepts 18–60 years: Class 1 = below 18 (invalid), Class 2 = 18–60 (valid), Class 3 = above 60 (invalid). We can select representative values such as 17, 30 and 61.

Input Class	Example	Result
Invalid: below range	17	Reject
Valid: within range	30	Accept
Invalid: above range	61	Reject

🧠 MEMORY AID

Equivalence Class = Group similar inputs → choose a representative from each group.

3.3 Decision Table Testing

Decision Table Testing is useful when the output depends on a combination of conditions. Conditions are listed with their possible values, and each combination is represented by a rule.

💡 RELATABLE EXAMPLE

Canteen discount rule: A student gets a discount if the student is a registered member AND the order total is at least ₹500.

Rule	Member?	Total ≥ ₹500?	Discount?	Action
R1	Yes	Yes	Yes	Apply discount
R2	Yes	No	No	Normal price
R3	No	Yes	No	Normal price
R4	No	No	No	Normal price

🧠 MEMORY AID

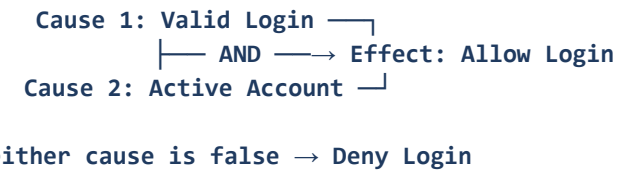
Decision Table = Conditions + combinations → expected action.

3.4 Cause-Effect Graphing

Cause-Effect Graphing is a technique that represents logical relationships between input conditions (causes) and resulting outputs/actions (effects). The graph is then used to derive test cases.

💡 RELATABLE EXAMPLE

Canteen example: Cause 1 = valid login; Cause 2 = account active. Effect = allow login. If either condition is false, the system should deny login.



Common logical relationships represented include AND, OR and NOT. The resulting logical combinations help identify test cases.

⭐ EXAM TIP

Decision Table and Cause-Effect Graphing are especially useful when several conditions interact to determine an outcome.

4. Structural Testing

Structural Testing, also called white-box testing, designs tests using the internal structure, logic, control flow or code of the program.

💡 RELATABLE EXAMPLE

For a canteen payment function, structural testing can ensure that every important decision path—payment success, payment failure, retry and cancellation—is exercised.

🧠 MEMORY AID

Structural Testing = HOW the program works internally.

4.1 Functional vs Structural Testing

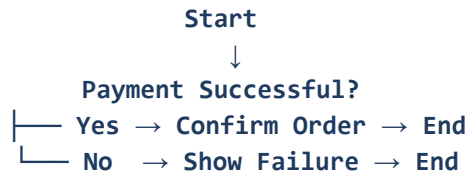
Point	Functional Testing	Structural Testing
Focus	External behavior / requirements	Internal code structure / logic
Common view	Black-box approach	White-box approach
Typical question	Does the function give the correct result?	Have important code paths/conditions been exercised?
Example	Correct bill displayed	Every important branch in bill calculation is executed

4.2 Path Testing

Path Testing is a structural testing technique that uses the control flow of a program to identify and execute independent paths through the code.

💡 RELATABLE EXAMPLE

Consider: if payment is successful, confirm the order; otherwise display payment failure. Path testing ensures both branches are tested.



A common measure used with path testing is cyclomatic complexity, which indicates the number of linearly independent paths in a control-flow graph.

$$V(G) = E - N + 2P$$

E = edges, N = nodes, P = connected components (for a single connected component, P = 1)

⭐ EXAM TIP

For a single connected flow graph, the commonly used form is V(G) = E − N + 2. Cyclomatic complexity gives a basis for the number of independent paths to consider.

4.3 Data Flow Testing

Data Flow Testing focuses on the life of variables: where a variable is defined/assigned, where it is used, and where it may be incorrectly redefined or used.

💡 RELATABLE EXAMPLE

In the canteen system, the variable totalAmount may be defined after calculating the cart total and then used when generating the payment request. Testing checks whether totalAmount is properly defined before every use.

Term	Meaning	Simple example
Definition (def)	A value is assigned to a variable	total = 500
Use (use)	The variable's value is read	display(total)
Definition-use path	A path from a definition to a use without an invalid redefinition	total=500 → display(total)

⭐ EXAM TIP

Data Flow Testing asks: Where is the data defined, where is it used, and is the flow between them correct?

4.4 Mutation Testing

Mutation Testing evaluates the effectiveness of test cases by making small deliberate changes (mutations) to the program and checking whether the existing tests detect them.

💡 RELATABLE EXAMPLE

Original condition: if (total >= 500). Mutant: if (total > 500). A good test suite should detect the mutant by including a test where total = ₹500.

🧠 MEMORY AID

Mutation Testing = Change the code slightly → run tests → good tests should “kill” the mutant.

If a test detects the changed program behavior, the mutant is said to be killed. If the tests do not detect it, the mutant survives and the test suite may need improvement.

⭐ EXAM TIP

The purpose is not to improve the program by mutation; it is to measure how strong the test suite is.

5. Unit, Integration & System Testing

Testing is performed at different levels so that defects can be found at the appropriate scope.

Level	What is tested?	Canteen Example	Main Goal
Unit Testing	Smallest testable component/module	Test calculateTotal() alone	Check individual component
Integration Testing	Interaction between modules/components	Order module ↔ Payment module	Check interfaces and data exchange
System Testing	Complete integrated system	Login → Menu → Order → Payment → Confirmation	Check the complete system against requirements

🧠 **MEMORY AID**

Unit = ONE part; Integration = parts WORK TOGETHER; System = WHOLE system.

Unit Testing → Integration Testing → System Testing

5.1 Quick Comparison

Question	Unit	Integration	System
Scope	Small	Several connected modules	Entire system
Main concern	Module correctness	Module interaction	End-to-end behavior
Example	Calculate bill	Order + payment exchange	Complete online ordering flow

6. Debugging

Debugging is the process of locating, analyzing and correcting the cause of a detected software defect.

Testing vs Debugging

Testing reveals that something is wrong; debugging investigates why it is wrong and fixes the cause.

6.1 Basic Debugging Process

Detect Failure → Reproduce Problem → Locate Cause → Fix Code → Re-test → Regression Test

💡 **RELATABLE EXAMPLE**

Suppose the canteen shows ₹600 when the student ordered ₹500 worth of food. Testing reports the failure. Debugging traces the calculation, finds that tax was added twice, fixes the code, then re-tests the bill.

6.2 Common Debugging Approaches

- Brute Force: collect logs, traces, print/debug output and inspect program behavior.
- Backtracking: start from the failure and trace backward through relevant code.
- Cause Elimination: list possible causes and eliminate them using evidence/tests.

⭐ **EXAM TIP**

Debugging is corrective activity: find the cause of the defect and remove it, then verify the fix.

7. Alpha & Beta Testing

Alpha and Beta testing are forms of acceptance-oriented testing performed before a software product is released widely.

Point	Alpha Testing	Beta Testing	

Where	Usually at the developer/organization environment	Usually in the real user/customer environment	
Who	Internal testers/development organization	Selected external/real users	
Purpose	Find problems before external release	Collect real-user feedback and discover remaining issues	
Canteen Example	College IT team tests the app internally	A selected group of students uses a pre-release version	

🧠 **MEMORY AID**

Alpha = internal/pre-release; Beta = selected real users before wider release.

⭐ **EXAM TIP**

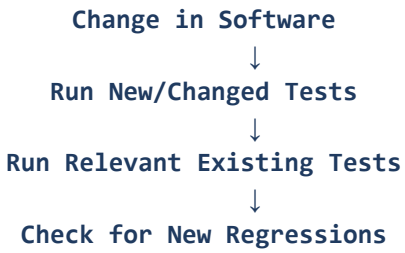
Do not confuse Alpha/Beta testing with Unit/Integration/System testing: Alpha and Beta describe pre-release user/organizational testing contexts.

8. Regression Testing

Regression Testing is re-running relevant tests after a change to ensure that existing, previously working functionality has not been broken by the change.

💡 **RELATABLE EXAMPLE**

The canteen team changes the payment module to add UPI support. Regression testing checks not only UPI, but also existing ordering, cart, bill, login and other affected features to ensure the change did not break them.



8.1 When is Regression Testing Needed?

- After fixing a defect.
- After adding a new feature.
- After modifying existing code.
- After changing interfaces or dependent components.
- After significant configuration/environment changes.

⭐ **EXAM TIP**

Regression = “Did my change break something that was already working?”

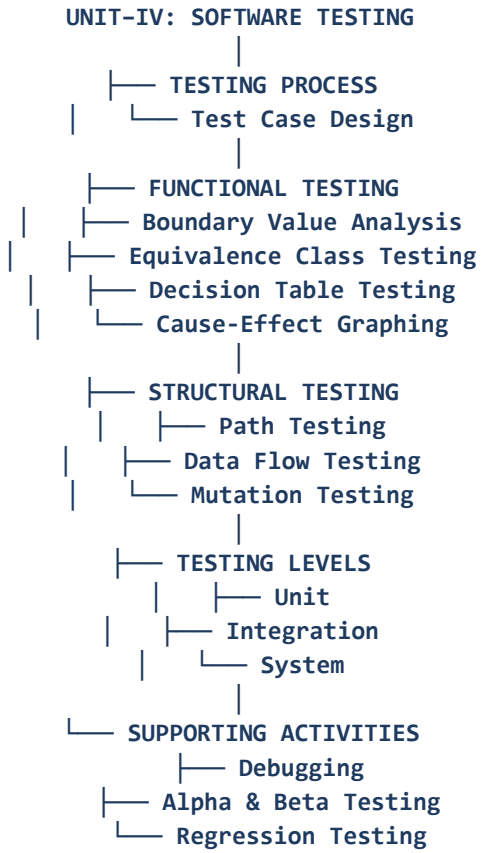
9. Functional Testing Techniques — Quick Comparison

Technique	Best Used When	Main Idea	Canteen Example
Boundary Value Analysis	Inputs have meaningful limits	Test edges and nearby values	Order quantity 1–10
Equivalence Class Testing	Input values can be grouped	Test representative value from each class	Age 18–60
Decision Table Testing	Several conditions combine	Test condition combinations and actions	Member + order amount → discount
Cause-Effect Graphing	Logical relationships are important	Map causes to effects and derive cases	Valid login AND active account → allow login

10. Structural Testing Techniques — Quick Comparison

Technique	Focus	Typical Question	Example
Path Testing	Control-flow paths	Have important independent paths been executed?	Success and failure payment paths
Data Flow Testing	Variable definitions/uses	Is each important data flow correct?	totalAmount defined before use
Mutation Testing	Test-suite strength	Would tests detect a small code change?	>= changed to >

11. Unit-IV at a Glance



12. Most Important One-Line Definitions

Topic	Exam-ready definition
Software Testing	Process of evaluating software to find defects and verify expected behavior against requirements.
Test Case	A set of inputs, conditions and expected results used to test a specific situation.

Functional Testing	Testing that checks whether software functions behave according to requirements.
BVA	Tests values at and around the boundaries of an input range.
Equivalence Class Testing	Divides input data into classes expected to behave similarly and tests representatives.
Decision Table Testing	Tests combinations of conditions and their corresponding actions.
Cause-Effect Graphing	Represents logical relationships between causes and effects to derive test cases.
Structural Testing	Testing based on the internal structure, logic or control flow of the program.
Path Testing	Tests independent execution paths through a program's control flow.
Data Flow Testing	Tests how variables are defined and used along program paths.
Mutation Testing	Measures test-suite effectiveness by checking whether tests detect small deliberate code changes.
Unit Testing	Tests an individual module/component in isolation.
Integration Testing	Tests interactions and interfaces between integrated modules/components.
System Testing	Tests the complete integrated system against its requirements.
Debugging	Locating, analyzing and correcting the cause of a software defect.
Alpha Testing	Pre-release testing generally performed within the developer/organization environment.
Beta Testing	Pre-release testing performed by selected real/external users in a realistic environment.
Regression Testing	Re-running relevant tests after changes to ensure existing functionality still works.

13. Last-Minute Memory Sheet

FUNCTIONAL TESTING

BVA = Boundaries • Equivalence = Classes • Decision Table = Conditions + combinations • Cause-Effect = Causes → Effects

STRUCTURAL TESTING

Path = Control flow • Data Flow = Definitions + Uses • Mutation = Change code slightly → tests should detect it

TESTING LEVELS

Unit = One module • Integration = Modules together • System = Whole system

ALPHA vs BETA

Alpha = Internal/pre-release • Beta = Selected real/external users

REGRESSION

After a change → check that old working features still work.

TESTING vs DEBUGGING

Testing finds/reveals a problem; Debugging finds and fixes its cause.

14. Practice Questions

1. Define Software Testing. Explain the testing process with a suitable diagram.
2. What is a test case? Design a test case for student login in a canteen ordering system.
3. Explain Boundary Value Analysis with a suitable example.
4. Explain Equivalence Class Testing and distinguish it from Boundary Value Analysis.
5. Explain Decision Table Testing with an example.
6. What is Cause-Effect Graphing? Explain how it helps in test-case design.
7. Differentiate Functional Testing and Structural Testing.
8. Explain Path Testing. What is cyclomatic complexity?
9. Explain Data Flow Testing with definition and use example.
10. What is Mutation Testing? Explain the idea of killing and surviving mutants.
11. Differentiate Unit, Integration and System Testing.
12. What is Debugging? Explain the basic debugging process.
13. Differentiate Alpha Testing and Beta Testing.
14. What is Regression Testing? When is it performed?
15. Compare BVA, Equivalence Class, Decision Table and Cause-Effect Graphing.
16. Write short notes on Path Testing, Data Flow Testing and Mutation Testing.

⭐ EXAM TIP

For a 5–10 mark answer: Definition → Explain the technique/process → Give a small example → Add a diagram/table when useful → Mention the purpose/benefit → Finish with one-line conclusion.