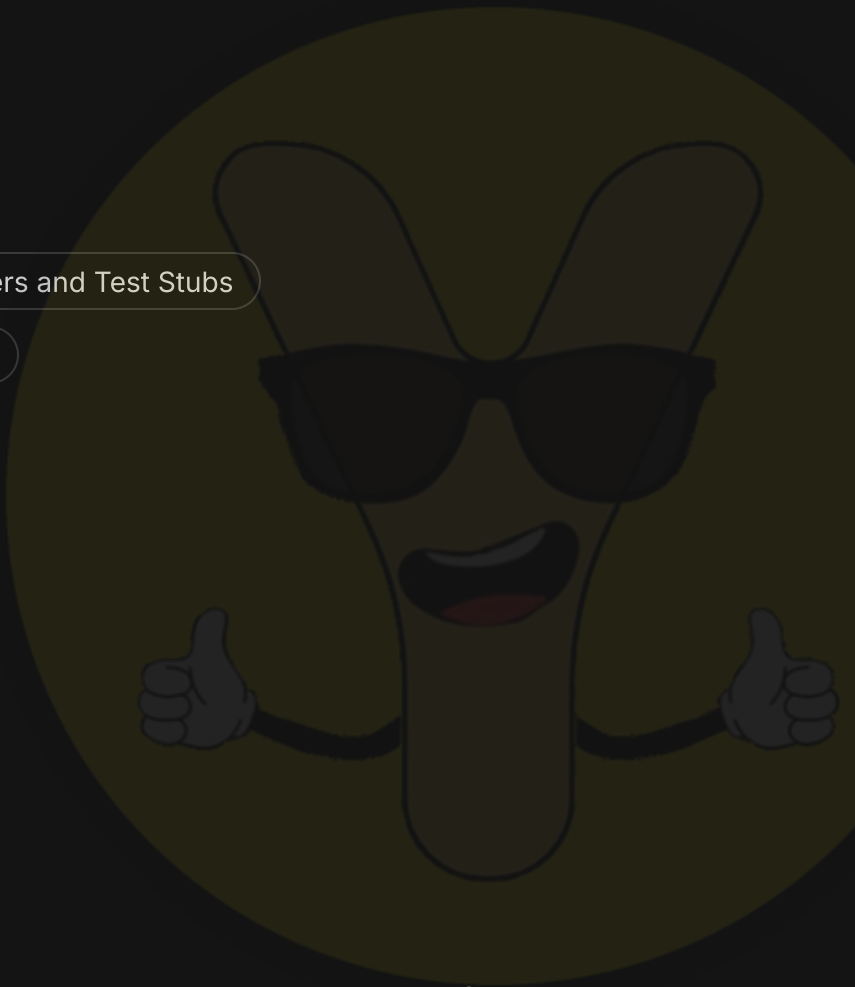
AKTU · 3RD YEAR · SEMESTER 6 · BCS601

Software Engineering

Unit 4 — Software Testing

- Testing Objectives
- Unit Testing
- Integration Testing
- Acceptance Testing
- Regression Testing
- Testing for Functionality and Testing for Performance
- Top-Down and Bottom-Up Testing Strategies: Test Drivers and Test Stubs
- Structural Testing (White Box Testing)
- Functional Testing (Black Box Testing)
- Test Data Suite Preparation
- Alpha and Beta Testing of Products



DECODED BY

Chirag Jain

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

theycotes.com

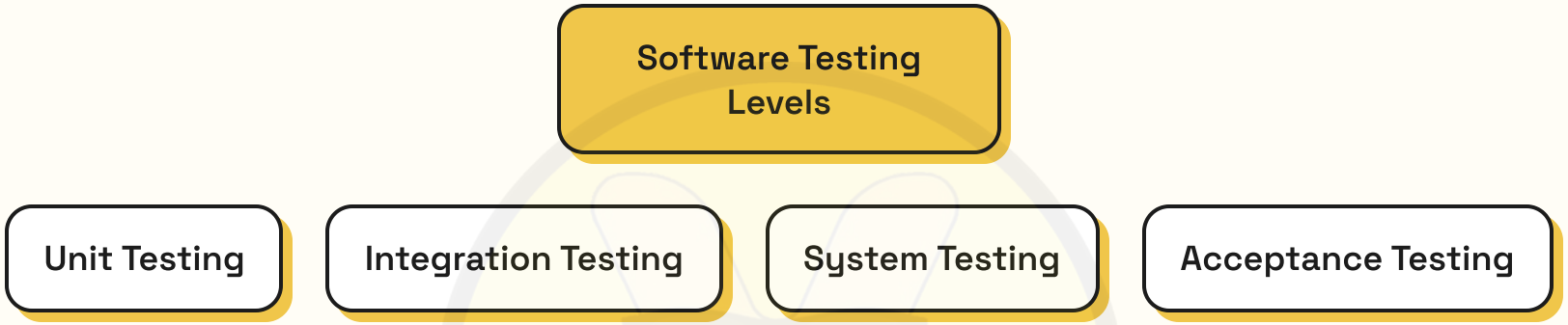
Software Testing Objectives

SOFTWARE TESTING

Software testing is the process of executing a program with the explicit intent of finding defects, verifying requirements fulfillment, and evaluating software quality.

- Primary objective is defect detection before product delivery to end-users.
- Ensures software compliance with specified functional and non-functional requirements.
- Validates system reliability, performance, security, and maintainability.
- Reduces overall development cost by identifying and fixing defects early in the SDLC.

Hierarchy of Software Testing Levels



Sequential levels of software testing in the classic V-Model.

Unit Testing & Test Environment

UNIT TESTING

Testing individual modules or software components in isolation to verify their internal logic, algorithms, and boundary conditions.

- Focuses on functional correctness of small testable parts (functions, classes, methods).
- Test Driver: A dummy main module that calls the unit under test and passes test inputs.
- Test Stub: A dummy subprogram that simulates low-level modules called by the unit under test.
- Drivers simulate parent callers while Stubs simulate child dependencies during isolated testing.

Integration Testing Strategies



Architectural approaches for combining individual modules into a coherent system.

Top-Down vs Bottom-Up Integration Testing

Parameter	Top-Down Integration	Bottom-Up Integration
Integration Order	Main control modules to lower sub-modules	Lowest worker modules to higher control modules
Dummy Code Needed	Requires Test Stubs heavily	Requires Test Drivers heavily
Major Advantage	Early prototype & structural flow verification	Early validation of low-level utility modules
Major Disadvantage	Stubs can be complex to write	System-level behavior verified late in cycle
Defect Isolation	Easy to isolate control logic defects	Easy to isolate data processing defects
Fault Detection	High-level architectural bugs found early	Data handling & utility bugs found early

Integration & Regression Testing

Combining modules and validating overall system integrity after code modifications.

- Integration Testing
  - Verifies interfaces, parameter passing, and data flow between integrated components.
  - Detects interface mismatches, uninitialized data passing, and side effects.
- Regression Testing
  - Re-testing previously passed software after bug fixes, enhancements, or refactoring.
  - Ensures new changes have not introduced unexpected side effects into existing functionality.
  - Requires automated test suites for rapid continuous execution during maintenance phases.

Acceptance Testing: Alpha and Beta Testing

Validation performed to determine whether software satisfies client business requirements.

- Acceptance Testing
  - Final dynamic testing phase conducted prior to official product deployment.
  - Validates system readiness from an end-user business workflow perspective.
- Alpha Testing
  - Conducted at developer site by internal software testers or select end-users.
  - Executed in a controlled environment; bugs are logged and fixed immediately.
- Beta Testing
  - Conducted at client or end-user sites in real-world deployment environments.
  - Performed by actual users who provide feedback on bugs, reliability, and usability.

Functional Testing vs Performance Testing

Evaluating operational functionality versus system response characteristics.

- Functional Testing
  - Validates what the software system does based on requirements specifications.
  - Focuses on input verification, output validation, business rules, and UI features.
  - Includes Unit, Integration, System, and User Acceptance Testing levels.
- Performance Testing
  - Validates how fast and stable the software performs under specific workloads.
  - Load Testing: Validates behavior under normal and peak anticipated user loads.
  - Stress Testing: Evaluates system robustness beyond operational limits to find breaking points.
  - Volume & Endurance Testing: Checks behavior under massive data volumes over long durations.

White Box Testing vs Black Box Testing

Structural Testing (White Box Techniques)

White box testing derives test cases directly from the control logic and internal structure.

- Statement Coverage
  - Ensures every executable line of code is executed at least once during testing.
- Branch / Decision Coverage
  - Ensures every decision outcome (True and False branches) is executed at least once.
- Basis Path Testing
  - Guarantees execution of every independent control flow path through the program logic.
- Cyclomatic Complexity
  - Quantitative metric measuring the logical complexity of a control flow graph (CFG).
  - Defines the upper bound for the number of test cases needed for basis path coverage.

Cyclomatic Complexity Calculation Worked Example

Functional Testing (Black Box Techniques)

Black box techniques design test cases based on specification requirements without reading code.

- Equivalence Class Partitioning (ECP)
  - Divides input domain into valid and invalid data classes.
  - Assumes one test case from each class represents all items in that class.
- Boundary Value Analysis (BVA)
  - Tests values at boundary limits of input partitions where defect probability is highest.
  - For range $[a, b]$, test values are: $a - 1, a, a + 1, b - 1, b, b + 1$.
- Decision Table Testing
  - Tabular representation mapping complex combinations of conditions to actions.
- State Transition Testing
  - Validates system behavior when transitioning across valid and invalid states.

Worked Example: Equivalence Partitioning & BVA

Test Data Suite Preparation

Systematic design, creation, and management of test cases and data sets.

- Test Case Structure
  - Contains Test Case ID, Description, Pre-conditions, Test Input Data, Expected Result, Actual Result, Pass/Fail status.
- Test Data Generation Strategy
  - Includes manual creation, production data anonymization, and synthetic data generation tools.
- Coverage Criteria
  - Ensures complete coverage of functional paths, error handling scenarios, and edge conditions.
- Test Suite Maintenance
  - Regularly updating test cases when system requirements change to prevent test erosion.

Classification of Software Testing Methods

Walkthrough vs Code Inspection vs FTR

Parameter	Walkthrough	Code Inspection	Formal Technical Review (FTR)
Formality	Informal to semi-formal	Highly formal and structured	Formal technical evaluation
Led By	Author of the document	Trained Moderator (not author)	Review Leader / Chair
Primary Goal	Educate, collect feedback, find bugs	Detect defects, ensure standards compliance	Validate architectural quality
Documentation	Optional informal notes	Formal defect log & inspection report	Formal review summary & action items
Roles Assigned	Author, Presenter, Audience	Moderator, Reader, Recorder, Inspector	Chair, Reviewer, Author, Scribe
Process Rigor	Low rigor, open discussions	Strict checklist-driven process	Structured meeting with defined agenda

Static Testing Strategies & Inspection Steps

Evaluating software artifacts without executing source code.

- Static Testing Overview
  - Examines requirements, design diagrams, source code, and user manuals without code execution.
  - Identifies root-cause defects early, significantly reducing rework costs.
- Fagan Inspection Process
  - 1. Planning: Prepare inspection package and assign roles (Moderator, Reader, Recorder).
  - 2. Overview: Brief team on domain context and background.
- Inspection Execution & Rework
  - 3. Preparation & Meeting: Inspectors use checklists to find bugs; Recorder logs defects.
  - 4. Rework & Follow-up: Author corrects defect log; Moderator verifies all fixes.

Compliance with Design and Coding Standards

Enforcing architectural consistency, code readability, and maintainability.

- Design Standard Compliance
  - Verifies adherence to architectural patterns (MVC, microservices) and interface contracts.
- Coding Standard Compliance
  - Enforces uniform naming conventions, indentation, code commenting, and exception handling rules.
- Automated Static Analysis
  - Tools (e.g., SonarQube) analyze source code automatically for security vulnerabilities and code smells.
- Key Engineering Benefits
  - Enhances software readability, maintainability, porting ease, and reduces technical debt.

AKTU Exam Corner: Likely Questions

Unit 4 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 5 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

