6.6 Levels of Testing

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

Unit Testing

- Individual components are tested.
- It is a path test.
- To focus on a relatively small segment of code and aim to exercise a high percentage of the internal path.

Disadvantage :

- The tester may be biased by previous experience. And the test value may not cover all possible values.

Objectives :

- To test the function of a program or unit of code such as a program or module.
- To test internal logic.
- To verify internal logic and design.
- To test path and condition coverage.
- To test exception conditions and error handling.

When :

- After modules are coded.

Input :

- Internal application design.
- Master test plan.
- Unit test plan.

Output :

- Unit Test Report.

Who :

- Developer.

Methods

- While Box Testing Techniques.
- Test Coverage Techniques.

Tools :

- Debug.
- Re-structure.

- Code-Analyzers.
- Path/Statement Coverage Tools.

Education :

- Testing methodology
- Effective use of tools.

Integration Testing

- Testing of combined parts of an application to determine their functional correctness.
- Parts can be
  - Code modules.
  - Individual applications.
  - Client/server applications on a network.
- Testing interfaces between components
- First step after unit-testing.
- Defects may exist in one module but manifest in another.
- Black-box tests.

Top-Down Integration Test

- To control program is tested first. Modules are integrated one at a time. Emphasize on interface testing.

Advantage :

- No test drivers needed interface errors are discovered early modular features aid debugging.

Disadvantage :

- Test stubs are needed errors in critical modules at low levels are found late.

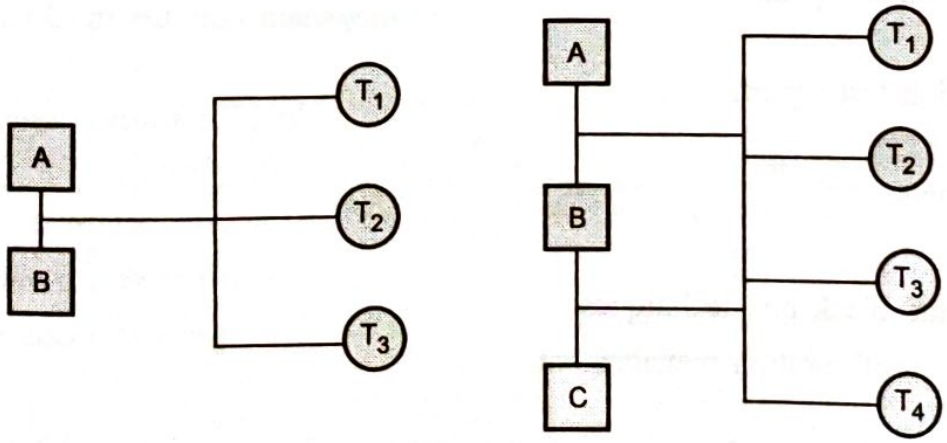

Fig. 6.6.1

Bottom-up Integration Test

- Allow early testing aimed at proving feasibility emphasize on module functionality and performance.

Advantages :

- No test stubs are needed errors in critical modules are found early.

Disadvantages :

- Test drivers are needed interface errors are discovered late.

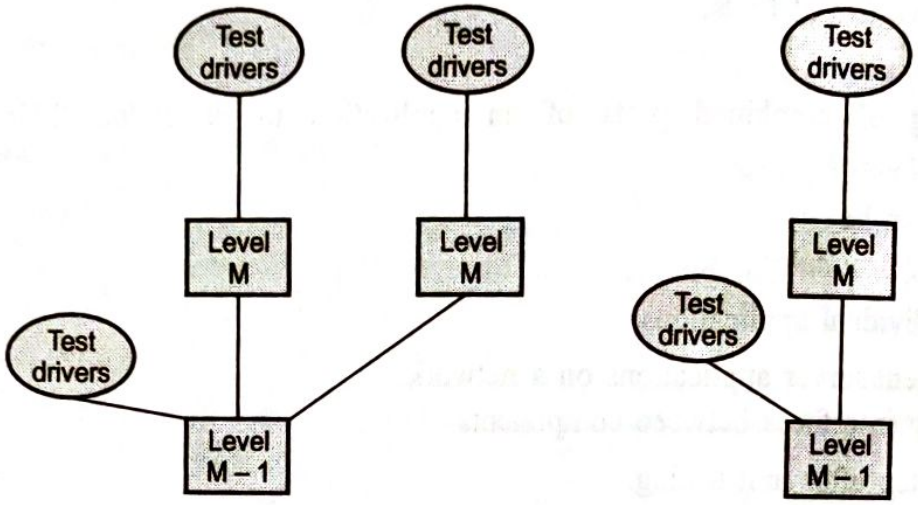

Fig. 6.6.2

Objectives :

- To technically verify proper interfacing between modules and within sub-systems.

When :

- After modules are unit tested.

Input :

- Internal and external application design.
- Master test plan.
- Integration test plan.

Output :

- Integration test report.

Who :

- Developers.

Methods :

1. White and black box techniques.
2. Problem/configuration management.

Tools :

- Debug.
- Re-structure.
- Code analyzers.

Education :

- Testing methodology.
- Effective use of tools.

System Testing

- To test all system after integration testing.

Objectives :

1. To verify that the system components perform control functions.
2. To perform inter-system test.
3. To demonstrate that the system performs both functionally and operationally as specified.
4. To perform appropriate types of tests relating to transaction flow, installation, reliability, regression etc.

When :

- After integration testing.

Input :

1. Detailed requirements and external application design.
2. Master test plan.
3. System test plan.

Output :

- System test report.

Who :

- Development team and users.

Method

- Problem/Configuration management.

Tools :

- Recommended set off tools.

Education :

- Testing methodology.
- Effective use of tools.

System Integration Testing

Objectives :

- To test the co-existence of products and applications that are required to perform together in the production like operational environment (hardware, software, network).
- To ensure that the system functions together with all the components of its environment as a total system.

- To ensure that the system release can be deployed in the current environment.

When :

1. After system testing.
2. Often performed outside of project life-cycle.

Input :

- Test strategy.
- Master test plan.
- System integration test plan

Output :

- System integration test report.

Who :

- System testers.

Methods :

1. White and black box techniques.
2. Problem/Configuration management.

Tools :

- Recommended set of tools.

Education :

- Testing methodology.
- Effective use of tools.

User Acceptance Testing

- Last milestones in testing phase.
- Ultimate customer test and sign-off.
- Sometimes synonymous with beta tests.
- Customer is satisfied software meets their requirements.
- Based on "Acceptance Criteria".
  - Conditions the software must meet for customer to accept the system.
  - Ideally defined before contract is signed.
  - Use quantifiable, measurable conditions.

Objectives :

- To verify that the system meets the user requirements.

When :

- After system testing.

Input :

- Business needs and detailed requirements.

- Master test plan.
- User acceptance test plan.

Output :

- User acceptance test report.

Who :

- Users / End users.

Methods :

- Black box techniques.
- Problem/configuration management.

Tools :

- Compare, key store capture and playback regression testing.

Education

- Testing methodology.
- Effective use of tools.
- Product knowledge.
- Business release strategy.

6.7 Test Strategies

- We begin by "testing-in-the-small" and more toward 'testing-in-the-large'.
- For **Conventional Software**
  - The module (component) is our initial focus.
  - Integration of modules follows.
- For **Object Oriented Software**
  - Our focus when "testing-in-the-small" changes from an individual module (the conventional view) to an object oriented class that encompasses attributes and operations and implies communication and collaboration.

Strategic Issues

- Specify product requirements in a quantifiable manner long before testing commences.
- State testing objectives explicitly.
- Understand the users of the software and develop a profile for each user category.
- Develop a testing plan that emphasizes "rapid cycle testing".
- Build "robust" software that is designed to test itself.
- Use effective technical reviews as a filter prior to testing.
- Conduct technical reviews to assess the test strategy and test cases themselves.
- Develop a continuous improvement approach for the testing process.

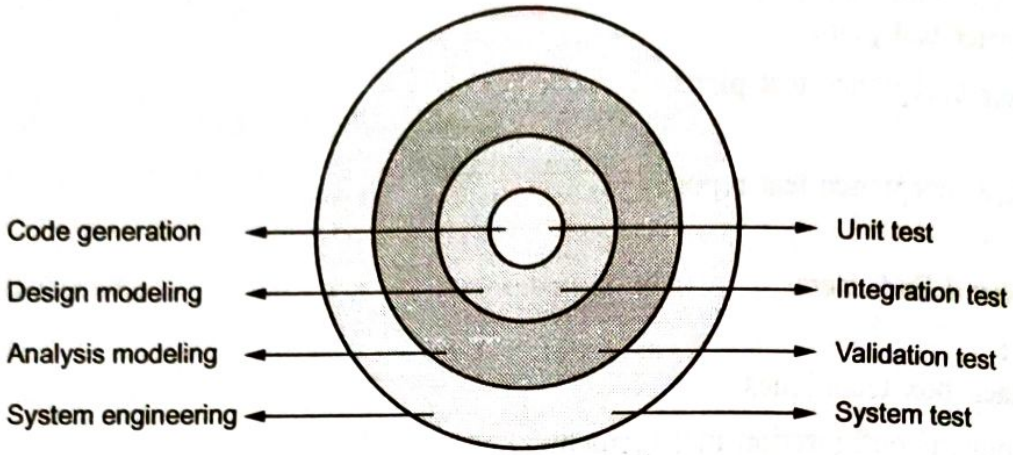

Fig. 6.7.1

Testing Strategy Applied to Conventional Software

(1) Unit Testing

- Exercise specific paths in a components control structure to ensure complete coverage and maximum error detection.
- Components are then assembled and integrated.

Integrating Testing

- Focuses on inputs and outputs and how well the components fit together and work together.

Validation Testing

- Provides final assurance that the software meets all functional, behavioural and performance requirement.

System Testing

- Verifies that all system elements (software, hardware, people, databases) mesh properly and that overall system function and performance is achieved.

Testing Strategy Applied to Object Oriented Software

- Must broaden testing to include detections of errors in analysis and design models.
- Unit testing loses some of its meaning and integration testing changes significantly.
- Use the same philosophy but different approach as in conventional software testing.
- Test "in the small" and then work out to testing "in the large".
- Testing in the small involves class attributes and operations; the main focus is on communication and collaboration within the class.
- Testing in the large involves a series of regression test to uncover errors due to communication and collaboration among classes.
- Finally, the system as a whole is tested to detect errors in fulfilling requirements.