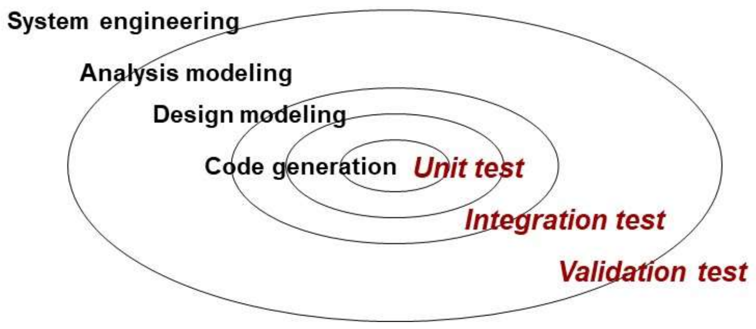
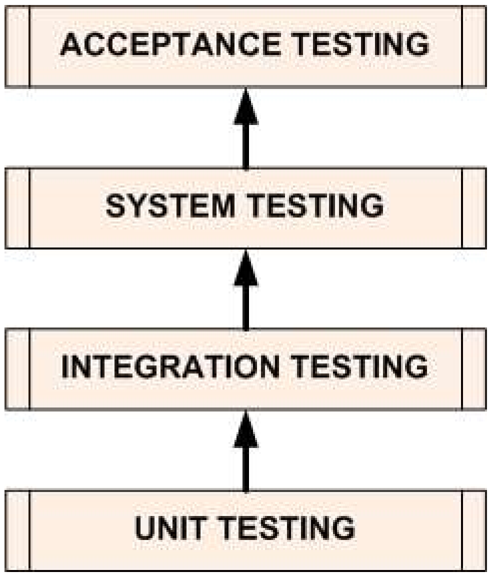
LEVELS OF TESTING

Testing Strategy

- We begin by 'testing-in-the-small' and move toward 'testing-in-the-large'
- For conventional software
  - The module (component) is our initial focus
  - Integration of modules follows
- For OO software
  - our focus when "testing in the small" changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration



SOFTWARE TESTING LEVELS are the different stages of the software development lifecycle where testing is conducted. There are four levels of software testing: Unit >> Integration >> System >> Acceptance.



Level	Summary
Unit Testing	A level of the software testing process where individual units of a software are tested. The purpose is to validate that each unit of the software performs as designed.
Integration Testing	A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.
System Testing	A level of the software testing process where a complete, integrated system is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.
Acceptance Testing	A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

1. UNIT TESTING

- A Unit is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately.
- Unit testing is performed by the respective developers on the individual units of source code assigned areas.
- The developers use test data that is different from the test data of the quality assurance team.
- The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.
- Initially, tests focus on each component individually, ensuring that it functions properly as a unit. Hence, the name unit testing. Unit testing makes heavy use of white-box testing techniques, exercising specific paths in a module's control structure to ensure complete coverage and maximum error detection.

Unit Test Consideration

- The module interface is tested to ensure that information properly flows into and out of the program unit under test.
- The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
- Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.

- All independent paths (basis paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once.
- And finally, all error handling paths are tested.

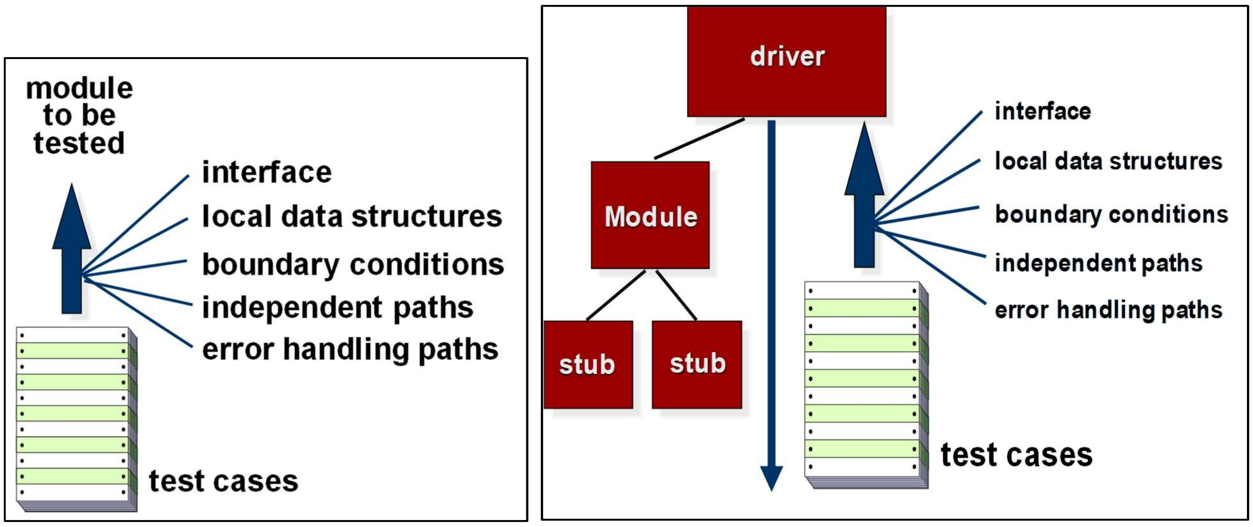


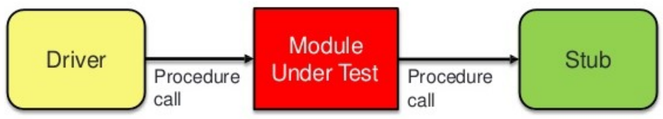
Fig: Unit Test Environment

- Each unit/ component is not a stand-alone program, hence driver and/or stub software must be developed for each unit test.
- The unit test environment is illustrated in Figure. In most applications a driver is nothing more than a "main program" that accepts test case data, passes such data to the component (to be tested), and prints relevant results.
- Stubs serve to replace modules that are subordinate (called by) the component to be tested. A stub or "dummy subprogram" uses the subordinate module's interface, may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing.
- Drivers and stubs represent overhead. That is, both are software that must be written (formal design is not commonly applied) but that is not delivered with the final software product.
- If drivers and stubs are kept simple, actual overhead is relatively low. Unfortunately, many components cannot be adequately unit tested with "simple" overhead software. In such cases, complete testing can be postponed until the integration test step (where drivers or stubs are also used).

2. INTEGRATION TESTING

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design. Incremental Approach is carried out by using dummy programs called Stubs and Drivers. Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.

Stubs and Drivers



Non- Incremental Integration/ big bang" approach.

- Here all component are integrated together at once and then tested.
- The entire program is tested as a whole. And chaos usually results!
- A set of errors is encountered. Correction is difficult because isolation of causes is complicated by the vast expanse of the entire program. Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop.

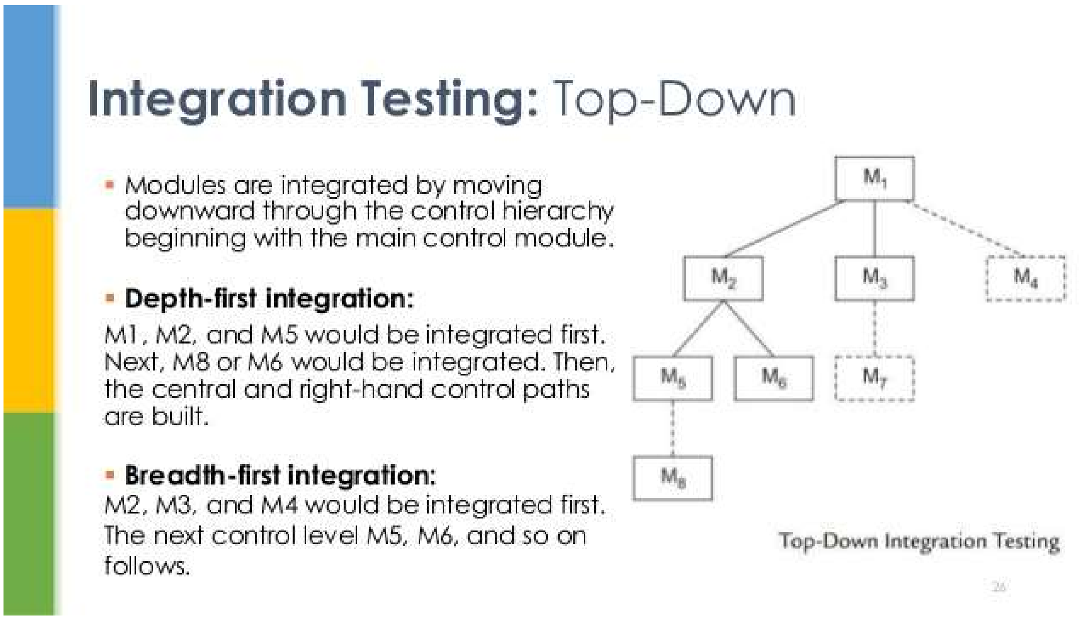
Incremental Approach: which is further divided into the following

- Top Down Approach
- Bottom Up Approach
- Sandwich Approach - Combination of Top Down and Bottom Up

The program is constructed and tested in small increments, where errors are easier to isolate and correct; interfaces are more likely to be tested completely; and a systematic test approach may be applied

i) Top-down Integration

Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program). Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner.



Integration Testing: Top-Down

- Modules are integrated by moving downward through the control hierarchy beginning with the main control module.
- **Depth-first integration:** M1, M2, and M5 would be integrated first. Next, M8 or M6 would be integrated. Then, the central and right-hand control paths are built.
- **Breadth-first integration:** M2, M3, and M4 would be integrated first. The next control level M5, M6, and so on follows.

The integration process is performed in a series of five steps:

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.

2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.

3. Tests are conducted as each component is integrated.

4. On completion of each set of tests, another stub is replaced with the real component. 5. Regression testing may be conducted to ensure that new errors have not been introduced.

The process continues from step 2 until the entire program structure is built.

Advantages:

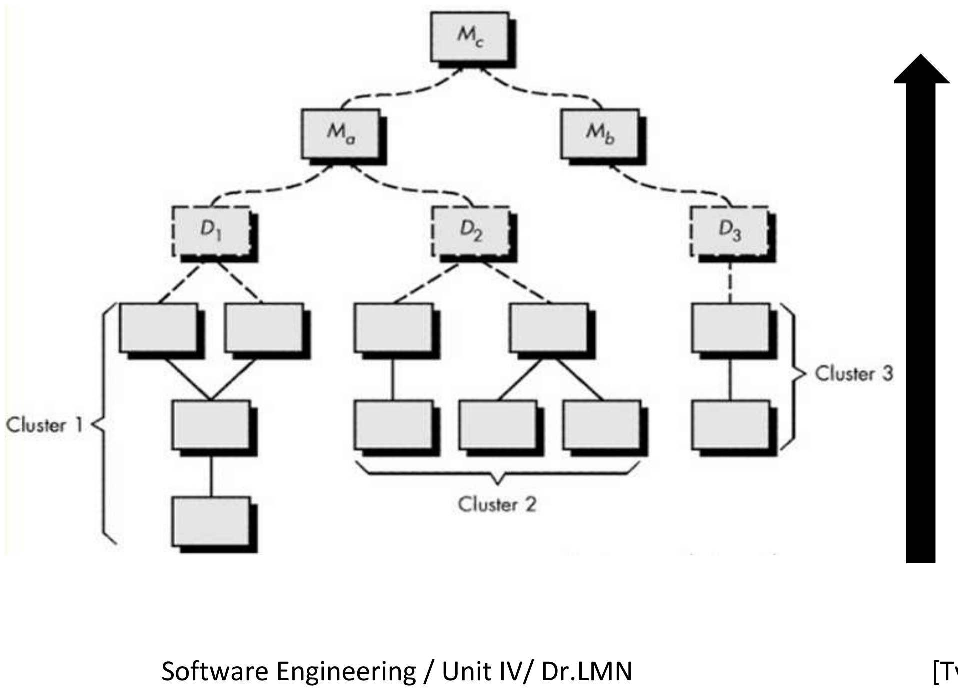
- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at a lower level are tested inadequately.

ii) Bottom Up Integration

- Bottom-up Integration begins construction and testing with atomic modules (i.e., components at the lowest levels in the program structure).
- Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated.

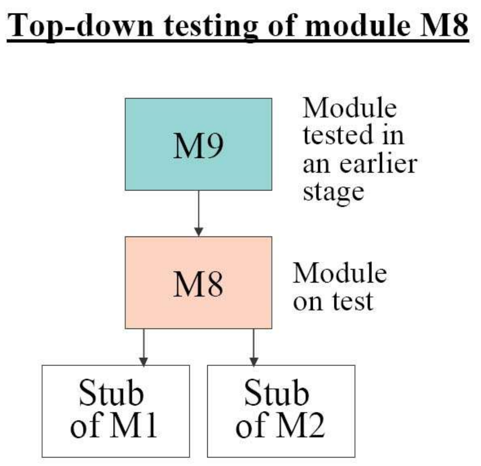


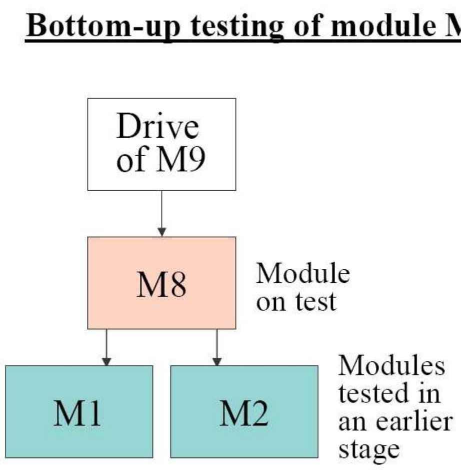
A bottom-up integration strategy may be implemented with the following steps:

1. Low-level components are combined into clusters (sometimes called builds) that perform a specific software subfunction.

2. A driver (a control program for testing) is written to coordinate test case input and output.
3. The cluster is tested.

4. Drivers are removed and clusters are combined moving upward in the program structure.


Top-down testing of module M8
M9
Module tested in an earlier stage
M8
Module on test
Stub of M1
Stub of M2


Bottom-up testing of module M8
Drive of M9
M8
Module on test
M1
M2
Modules tested in an earlier stage

Advantages:

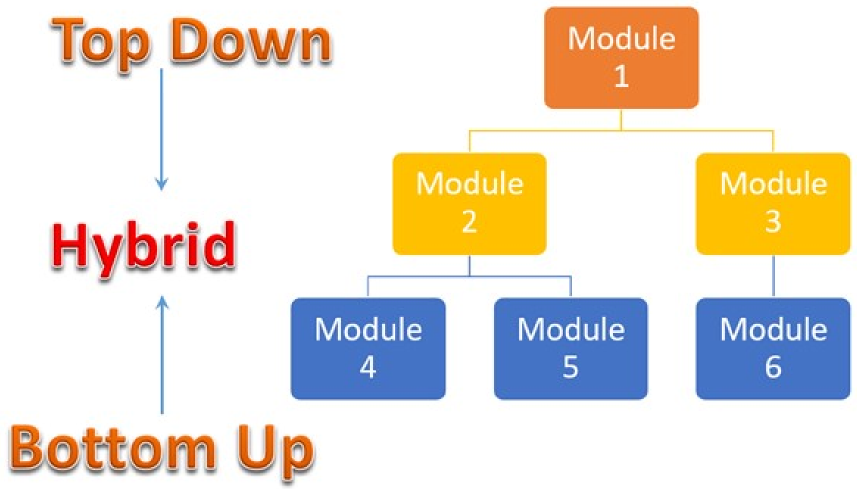
- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- An early prototype is not possible

<u>(iii) Sandwich Testing</u>

In the sandwich/hybrid strategy is a combination of Top Down and Bottom up approaches. Here, top modules are tested with lower modules at the same time lower modules are integrated with top modules and tested. This strategy makes use of stubs as well as drivers.



Regression testing

- *Regression testing* is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects
- Whenever software is corrected, some aspect of the software configuration (the program, its documentation, or the data that support it) is changed.
- Regression testing helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behavior or additional errors.
- Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.
- The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Smoke Testing

- ✓ Smoke Testing Smoke testing is an integration testing approach that is commonly used when "shrinkwrapped" software products are being developed. Software components that have been translated into code are integrated into a "build."
  - A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.
  - A series of tests is designed to expose errors that will keep the build from properly performing its function.
    - The intent should be to uncover "show stopper" errors that have the highest likelihood of throwing the software project behind schedule.
  - The build is integrated with other builds and the entire product (in its current form) is smoke tested daily.
    - The integration approach may be top down or bottom up.

3. SYSTEM TESTING

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

System testing is important because of the following reasons −

- System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment that is very close to the production environment where the application will be deployed.
- System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

4. ACCEPTANCE TESTING

Configuration Review

- ✓ An important element of the validation process is a configuration review.
- ✓ The intent of the review is to ensure that all elements of the software configuration have been properly developed, are catalogued, and have the necessary detail to support phase of the software life cycle.
- ✓ The configuration review, sometimes called an audit

When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements. Conducted by the end user rather than software engineers, an acceptance test can range from an informal "test drive" to a planned and systematically executed series of tests

Alpha Testing

- ✓ The alpha test is conducted at the developer's site by a customer.
- ✓ The software is used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems.
- ✓ Alpha tests are conducted in a controlled environment.

Beta Testing

- ✓ The beta test is conducted at one or more customer sites by the end-user of the software.
- ✓ Unlike alpha testing, the developer is generally not present. Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer.
- ✓ The customer records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals.
- ✓ As a result of problems reported during beta tests, software engineers make modifications and then prepare for release of the software product to the entire customer base.