Bottom-up Integration

Bottom-up integration testing, as its name implies, 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 sub-function.

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 structu

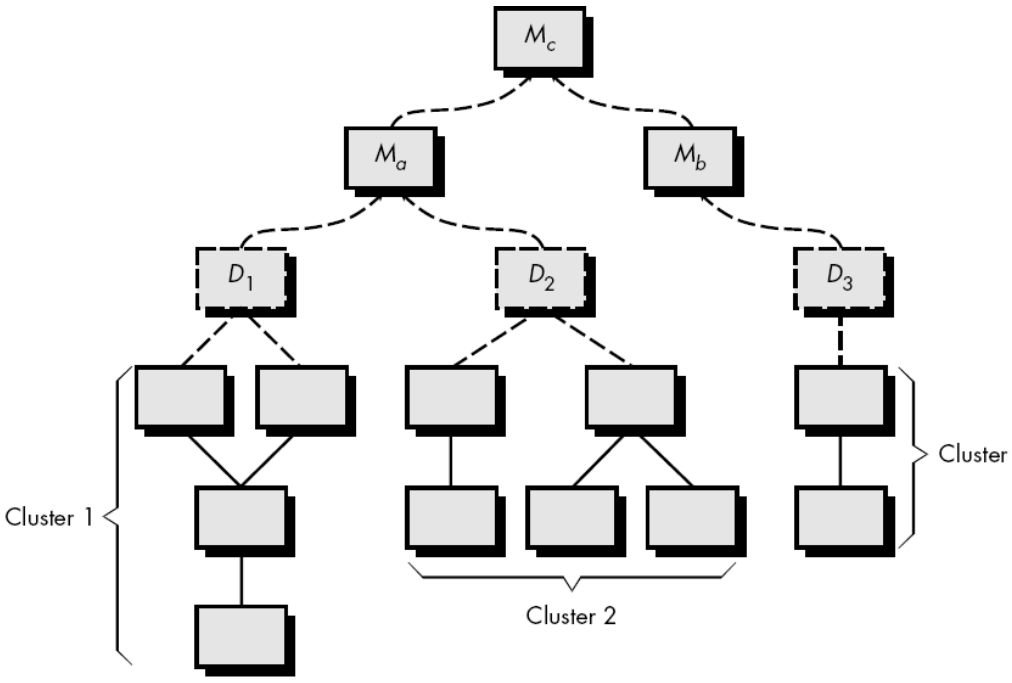


Fig. Bottom-up Integration

Integration follows the pattern illustrated in Figure above. Components are combined to form clusters 1, 2, and 3. Each of the clusters is tested using a driver (shown as a dashed block). Components in clusters 1 and 2 are subordinate to Ma. Drivers D1 and D2 are removed and the clusters are interfaced directly to Ma. Similarly, driver D3 for cluster 3 is removed prior to integration with module Mb. Both Ma and Mb will ultimately be integrated with component Mc , and so forth.

As integration moves upward, the need for separate test drivers lessens. In fact, if the top two levels of program structure are integrated top down, the number of drivers can be reduced substantially and integration of clusters is greatly simplified.

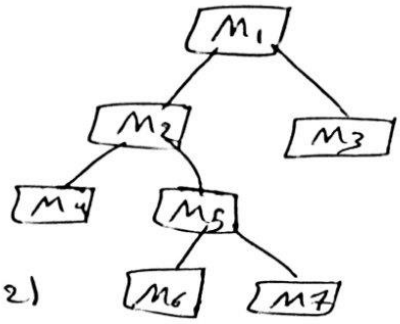
Comments on Integration Testing

There has been much discussion of the relative advantages and disadvantages of top-down versus bottom-up integration testing. In general, the advantages of one strategy tend to result in disadvantages for the other strategy. The major disadvantage of the top-down approach is the need for stubs and the attendant testing difficulties that can be associated with them.

The major disadvantage of bottom-up integration is that "the program as an entity does not exist until the last module is added". This drawback is tempered by easier test case design and a lack of stubs.

Selection of an integration strategy depends upon software characteristics and, sometimes, project schedule. In general, a combined approach (sometimes called ***sandwich testing***) that uses top-down tests for upper levels of the program structure, coupled with bottom-up tests for subordinate levels may be the best compromise.

Bottom-up



S1: M6, d(M5)
S2: M7, d(M5)
S3: M6, M7, d(M5)
S4: M4, d(M2)
S5: M6, M7, M5, d(M2)
S6: M4, M6, M7, M5, d(M2)
S7: M4, M6, M7, M5, M2, d(M1)
S8: M3, d(M1)
S9: M6, M7, M4, M5, M2, M3, M1

No of drivers = nodes - leaves = 3

Top-down

S1: M1, stub(M2), stub(M3)
S2: M1, M2, stub(M3)
S3: M1, stub(M2), M3
S4: M1, M2, stub(M4), stub(M5), M3
S5: M1, M2, M4, stub(M5), M3
S6: M1, M2, M4, M5, stub(M6), stub(M7), M3
S7: M4, M2, M4, M5, M6, stub(7), M3
S8: M4, M2, M4, M5, stub(M6), M7, M3
S9: M1, M2, M4, M5, M6, M7, M3

No of stubs = nodes - 1 = 7 - 1 = 6

No of sessions = (nodes - leaves) + edges = 3 + 6 = 9

VALIDATION TESTING

At the culmination of integration testing, software is completely assembled as a package, interfacing errors have been uncovered and corrected, and a final series of software tests—*validation testing*—may begin. Validation can be defined in many ways, but a simple definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer.

Validation Test Criteria

Software validation is achieved through a series of black-box tests that demonstrate conformity with requirements. Both the plan and procedure are designed to ensure that all functional requirements are satisfied, all behavioral characteristics are achieved, all performance requirements are attained, documentation is correct, and human engineered and other requirements are met (e.g., transportability, compatibility, error recovery, maintainability).

After each validation test case has been conducted, one of two possible conditions exist:

(1) The function or performance characteristics conform to specification and are accepted or

(2) a deviation from specification is uncovered and a deficiency list is created. Deviation or error discovered at this stage in a project can rarely be corrected prior to scheduled delivery. It is often necessary to negotiate with the customer to establish a method for resolving deficiencies.

Alpha and Beta Testing

It is virtually impossible for a software developer to foresee how the customer will really use a program. Instructions for use may be misinterpreted; strange

combinations of data may be regularly used; output that seemed clear to the tester may be unintelligible to a user in the field.

When custom software is built for one customer, a series of *acceptance tests* are conducted to enable the customer to validate all requirements.

Most software product builders use a process called **alpha and beta testing** to uncover errors that only the end-user seems able to find.

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.

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.