MENOUFIA UNIVERSITY
جامعة المنوفية
منارة المعرفة فى قلب الدلتا

Software Engineering

Dr. Mohammed Badawy

mbmbadawy@yahoo.com
mohamed.badawi@el-eng.menofia.edu.eg
Room: 417,   Ext. 7332

13 May 2016

Chapter 7

Software Testing

Contents

1. Testing Principles
2. Testing Objectives
3. Test Oracles
4. Levels of Testing
5. White-Box Testing/ Structural Testing
6. Functional/ Black-Box Testing
7. Test Plan
8. Test-Case Design

Introduction

- Testing is a set of activities that can be planned in advance and conducted systematically.
- A template for software testing—a set of steps into which we can place specific test-case design techniques and testing methods—should be defined for the software process.
- A number of software-testing strategies have been proposed in the literature. All provide the software developer with a template for testing and all have the following generic characteristics:

Introduction

1. To perform effective testing, a software team should conduct effective formal technical reviews. By doing this, many errors will be eliminated before testing commences.
2. Testing begins at the component level and works “outward” toward the integration of the entire computer-based system.
3. Testing is conducted by the developers of the software and (for large projects) an independent test group.
4. Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

Introduction

- A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source-code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.
- A strategy must provide guidance for the practitioner and a set of milestones for the manager.

Introduction

- **Software testing** has been defined as:
- The process of analyzing a software item to detect the differences between existing and required conditions (i.e., bugs) and to evaluate the features of the software items.
- The process of analyzing a program with the intent of finding errors.

OR

- Testing is the process of executing a program with the intent of finding errors.

1. Testing Principles

- There are many principles that guide software testing.
- The following are the main principles for testing:
  - All tests should be traceable to customer requirement: This is in order to uncover any defects that might cause the program or system to fail to meet the client’s requirements.
  - Tests should be planned long before testing begins: Soon after the requirements model is completed, test planning can begin. Detailed test cases can begin as soon as the design model is designed.

1. Testing Principles

- The Pareto principle applies to software testing: Stated simply, the Pareto principle implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components. The problem, of course, is to isolate these suspect components and to thoroughly test them.
- Testing should begin “in the small” and progress toward testing “in the large”: The first tests planned and executed generally focus on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.

1. Testing Principles

- Exhaustive testing is not possible: The number of path permutations for even a moderately-sized program is exceptionally large. For this reason, it is impossible to execute every combination of paths during testing. It is possible, however, to adequately cover program logic and to ensure that all conditions in the component-level design have been exercised.
- To be most effective, testing should be conducted by an independent third party: The software engineer who has created the system is not the best person to conduct all tests for the software.

2. Testing Objectives

- The testing objective is to test the code, whereby there is a high probability of discovering all errors.
- This objective also demonstrates that the software functions are working according to software requirements specification (SRS).
- It should be noted, however, that testing will detect errors in the written code, but it will not show an error if the code does not address a specific requirement stipulated in the SRS but not coded in the program.

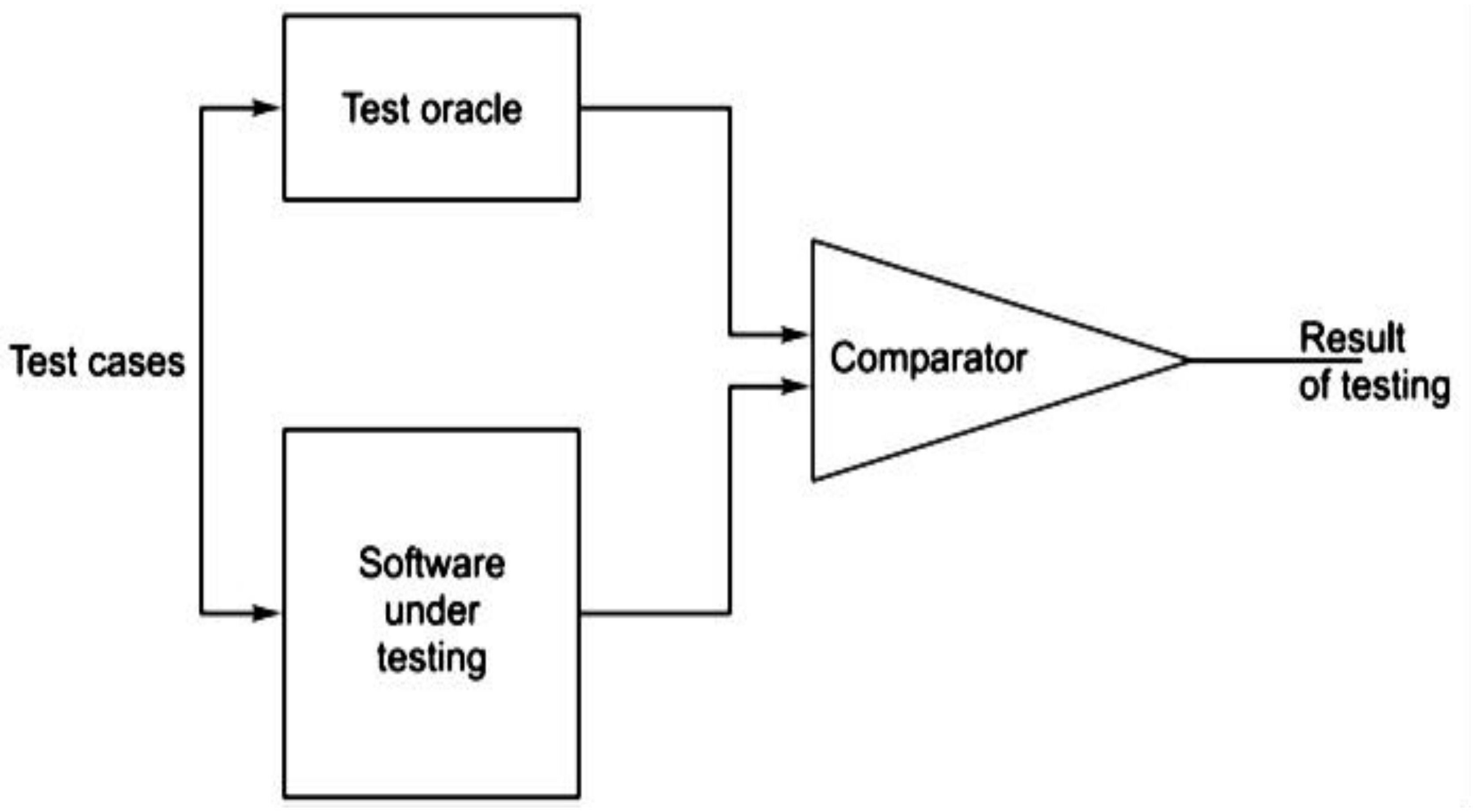
2. Testing Objectives

- Testing objectives are:
  - Testing is a process of executing a program with the intent of finding an error.
  - A good test case is one that has a high probability of finding an as-yet-undiscovered error.
  - A successful test is one that uncovers an as-yet-undiscovered error.

3. Test Oracles

- To test any program, we need to have a description of its expected behaviour and a method of determining whether the observed behaviour conforms to the expected behaviour.
- For this we need a test oracle.
- A test oracle is a mechanism, different from the program itself, which can be used to check the correctness of the output of the program for the test cases.
- Conceptually, we can consider testing a process in which the test cases are given to the test oracle and the program under testing.
- The output of the two is then compared to determine if the program behaved correctly for the test cases.

3. Test Oracles

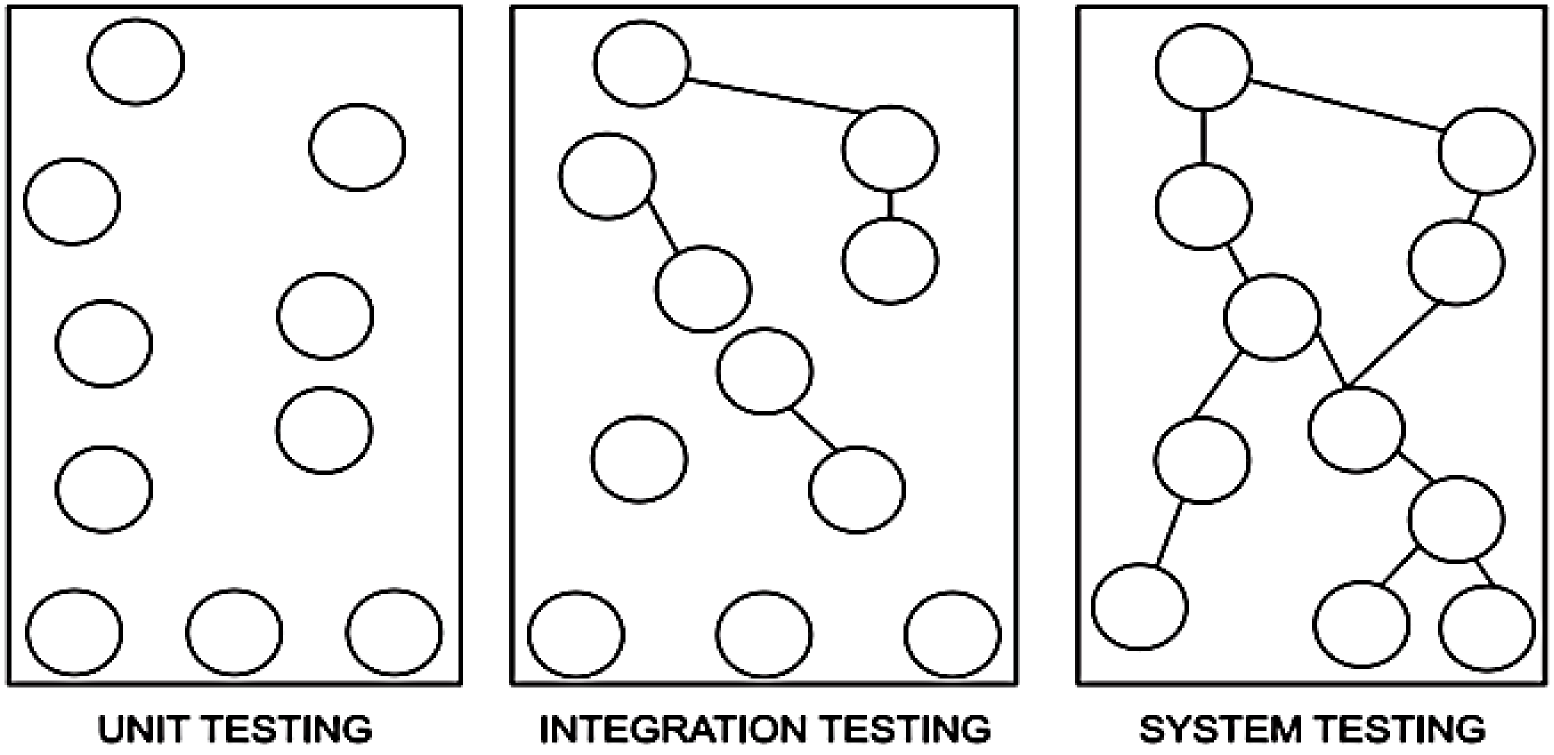


3. Test Oracles

- The human oracles generally use the specifications of the program to decide what the “correct” behaviour of the program should be.
- To help the oracle to determine the correct behaviour, it is important that the behaviour of the system be unambiguously specified and the specification itself should be error-free.

4. Levels of Testing

- There are three levels of testing: unit testing, integration testing, and system testing.

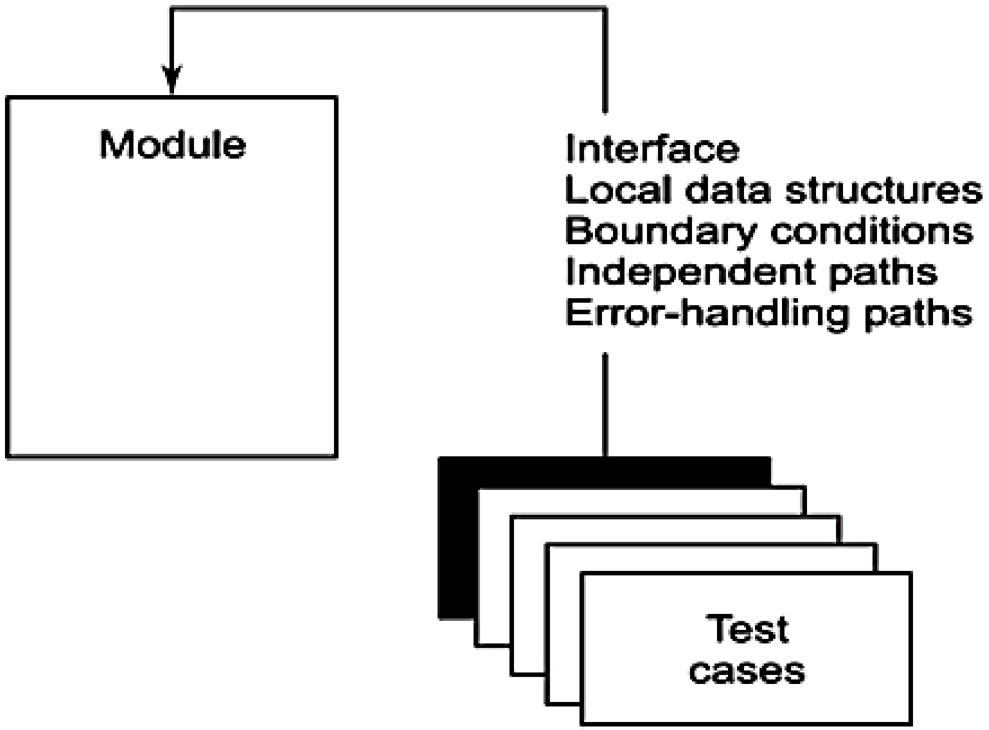


4.1 Unit Testing

- In unit testing individual components are tested to ensure that they operate correctly.
- On the smallest unit of software design, each component is tested independently without other system components.
- The reasons to do unit testing rather than testing the entire product:
  - The size of a single module is small enough that we can locate an error fairly easily.
  - The module is small enough that we can attempt to test it in some demonstrably exhaustive fashion.
  - Confusing interactions of multiple errors in widely different parts of the software are eliminated.

4.1 Unit Testing

- The tests that occur as part of unit tests are illustrated schematically as shown



4.1 Unit Testing

- The module interface is tested to ensure that information properly flows into and out of the program unit under testing.
- 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 the boundaries established to limit or restrict processing.
- All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once.

4.1 Unit Testing

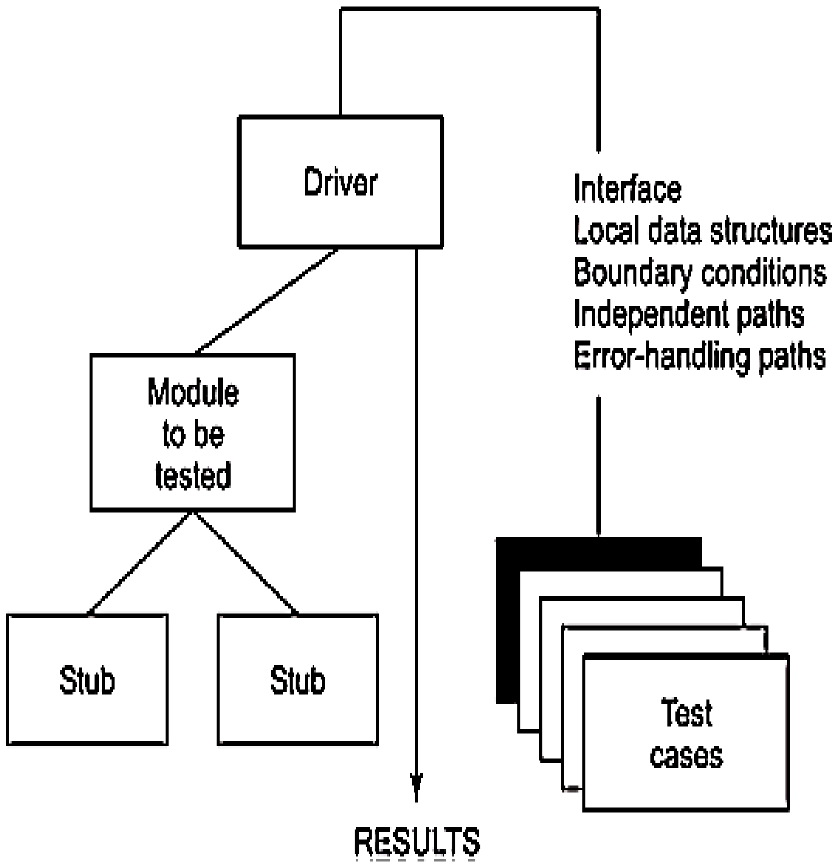
- Common errors in computation are:
  - Incorrect arithmetic precedence
  - Mixed code operations
  - Incorrect initialization
  - Precision inaccuracy
  - Incorrect symbolic representation of an expression

4.1 Unit Testing

- Test cases in unit testing should uncover errors, such as:
  - Comparison of different data types
  - Incorrect logical operators or precedence
  - Expectation of equality when precision error makes equality unlikely
  - Incorrect comparison of variables
  - Improper loop termination
  - Failure to exit when divergent iteration is encountered
  - Improperly modified loop variables

4.1 Unit Testing

- The unit-test environment is illustrated
- In most applications a driver is nothing more than a “main program” that accepts test-case data, passes such data to the component, and prints relevant results.
- Stubs serve to replace modules that are subordinate (called by) to the component to be tested.



4.2 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.
- In this testing many unit-tested modules are combined into subsystems, which are then tested.
- The goal here is to see if the modules can be integrated properly.
- The primary objective of integration testing is to test the module interfaces in order to ensure that there are no errors in the parameter passing, when one module invokes another module.

4.2 Integration Testing

- During integration testing, different modules of a system are integrated in a planned manner using an integration plan.
- The integration plan specifies the steps and the order in which modules are combined to realize the full system.
- After each integration step, the partially integrated system is tested.

4.2 Integration Testing

- The various approaches used for integration testing are:
  1. Incremental Approach
  2. Top-down Integration
  3. Bottom-up Integration
  4. Regression Testing
  5. Smoke Testing
  6. Sandwich Integration

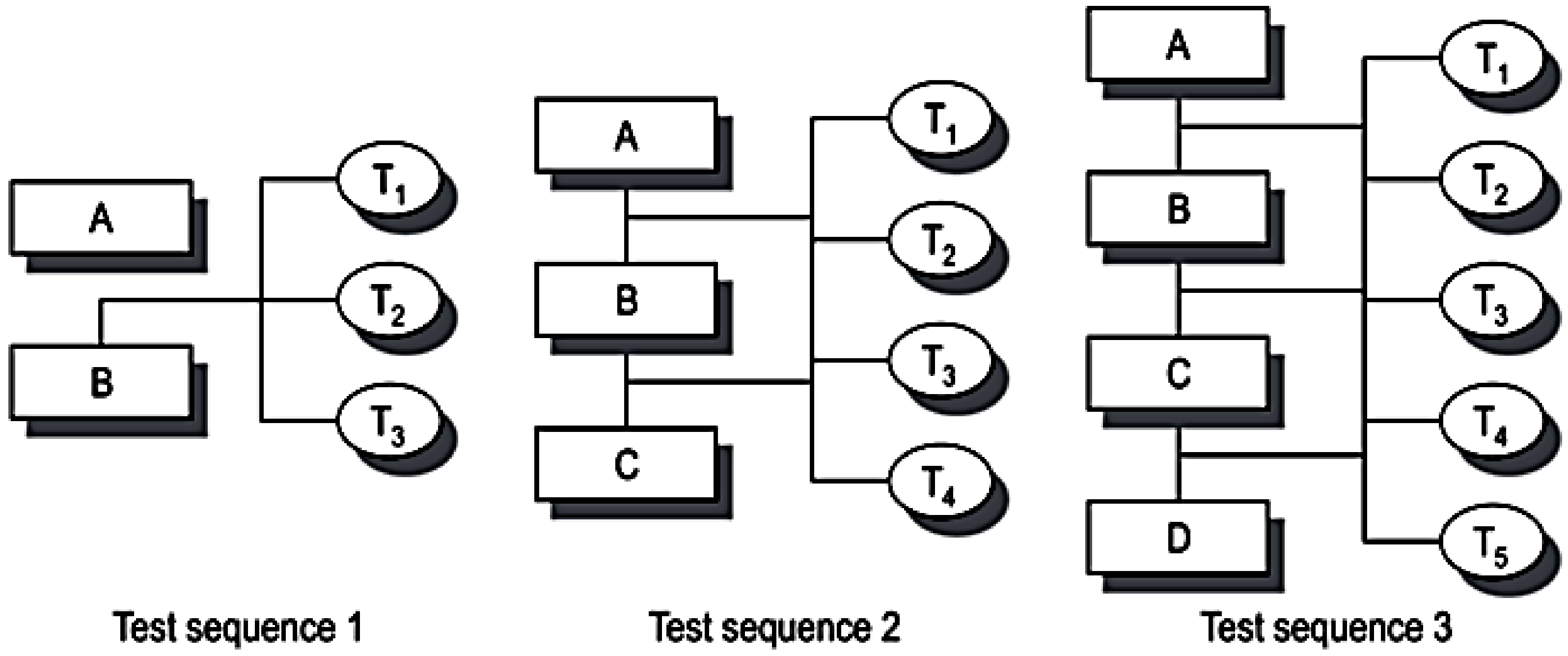
4.2 Integration Testing

1. **Incremental Approach**: The incremental approach means to first combine only two components together and test them.

- Remove the errors if they are there, otherwise combine another component to it and then test again, and so on until the whole system is developed.
- According to next Figure, in test sequence 1 tests T1, T2, and T3 are first run on a system composed of module A and module B.
- If these are corrected or error-free then module C is integrated, i.e., test sequence 2 and then tests T1, T2, and T3 are repeated, if a problem arises in these tests, then they interact with the new module.

4.2 Integration Testing

- The source of the problem is localized, thus it simplifies defect location and repair. Finally, module D is integrated, i.e., test sequence 3 is then tested using existing (T1 to T5) and new tests (T6).

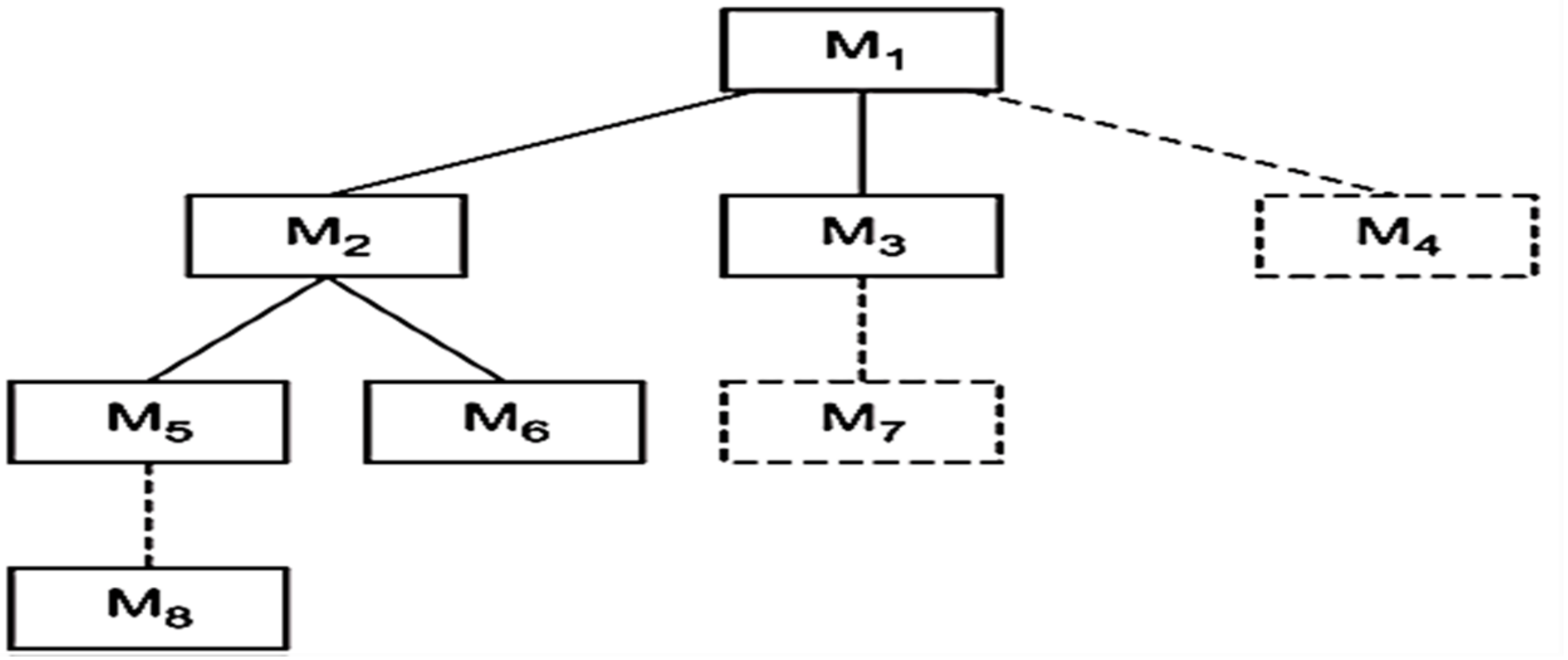


4.2 Integration Testing

2. **Top-Down Integration Testing**: Top-down integration testing is an incremental approach to construction of program structures. Modules are integrated by moving downward through the control hierarchy beginning with the main-control module.

(i) **Depth-first integration**: This would integrate all components on a major control path of the structure. For example, 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.

4.2 Integration Testing



(ii) **Breadth-first integration**: This incorporates all components directly subordinate at each level, moving across the structure horizontally. From Figure 5.6, components M2, M3, and M4 would be integrated first. The next control level M5, M6, and so on follows.

4.2 Integration Testing

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

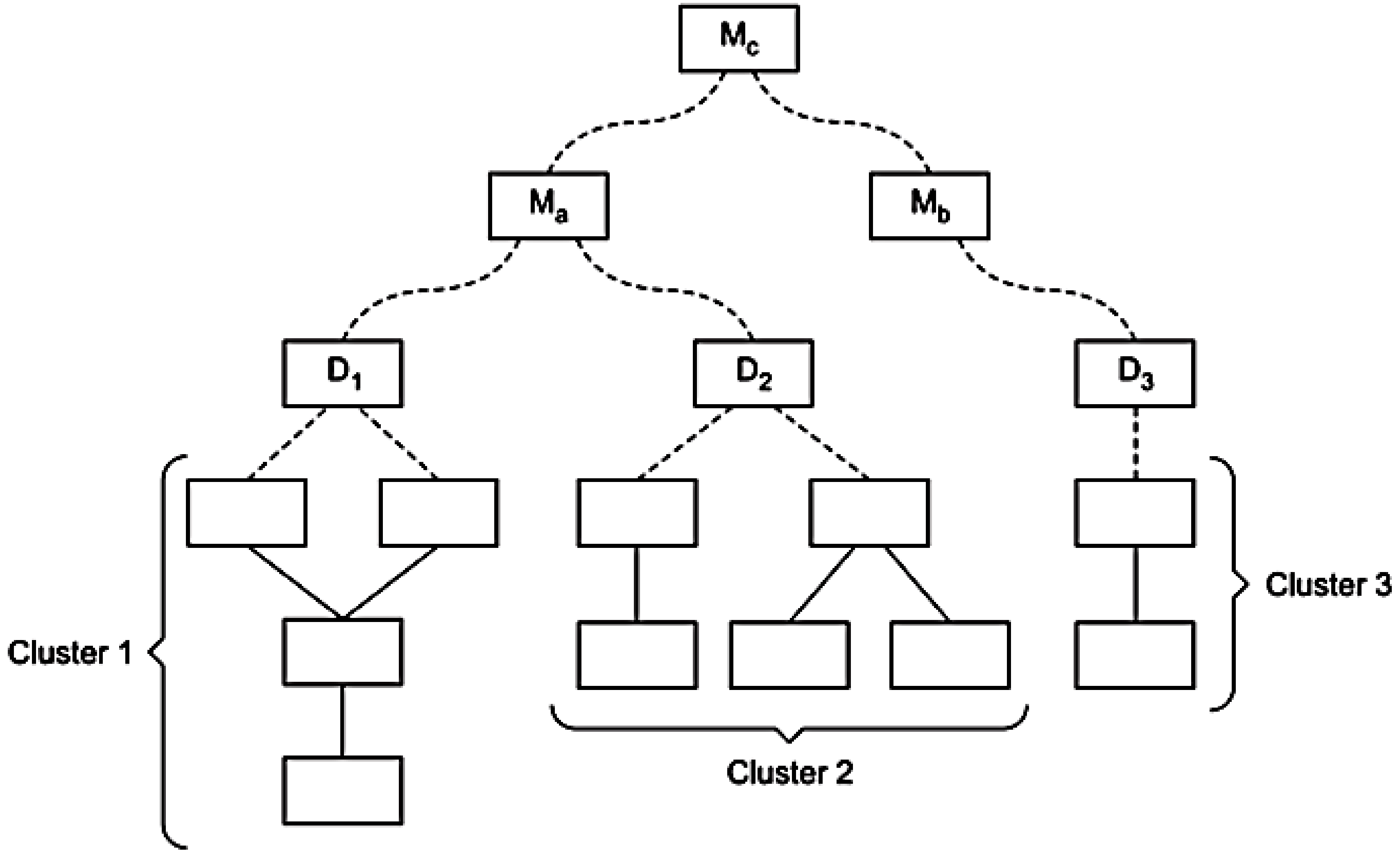
- The main-control module is used as a test driver and stubs are substituted for all components directly subordinate to the main-control module.
- Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.
- Tests are conducted as each component is integrated.
- On completion of each set of tests, another stub is replaced with the real component.
- Regression testing may be conducted to ensure that new errors have not been introduced.

4.2 Integration Testing

3. **Bottom-Up Integration Testing**: as its name implies, begins construction and testing with the components at the lowest level in the program structure. A bottom-up integration strategy may be implemented with the following steps:

- Low-level components are combined into clusters that perform specific software sub-functions.
- A driver (a control program for testing) is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure.

4.2 Integration Testing



4.3 System Testing

- Subsystems are integrated to make up the entire system.
- The testing process is concerned with finding errors that result from unanticipated interactions between subsystems and system components.
- It is also concerned with validating that the system meets its functional and non-functional requirements.
- There are essentially three main kinds of system testing:
  - Alpha testing
  - Beta testing
  - Acceptance testing

4.3 System Testing

Alpha Testing:

- Alpha testing refers to the system testing carried out by the test team within the development organization.
- The alpha test is conducted by the customer under the project team's guidance.
- Alpha tests are conducted in a controlled environment. It is a simulation of real-life usage.
- Once the alpha test is complete, the software product is ready for transition to the customer site for implementation and development.

4.3 System Testing

Beta Testing:

- Beta testing is the system testing performed by a selected group of friendly customers.
- If the system is complex, the software is not taken for implementation directly. It is installed and all users are asked to use the software in testing mode; this is not live usage. This is called the beta test.
- Beta tests are conducted at the customer site in an environment where the software is exposed to a number of users.
- In this test, end users record their observations, mistakes, errors, and so on and report them periodically.

4.3 System Testing

Acceptance Testing:

- Is the system testing performed by the customer to determine whether to accept or reject the delivery of the system.
- When customer software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements.
- In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time.

5 White-Box Testing/ Structural Testing

- In this approach, test groups must have complete knowledge of the internal structure of the software.
- Structural testing is an approach to testing where the tests are derived from knowledge of the software’s structure and implementation.
- Structural testing is usually applied to relatively small program units, such as subroutines, or the operations associated with an object.
- As the name implies, the tester can analyze the code and use knowledge about the structure of a component to derive test data.

5 White-Box Testing/ Structural Testing

- The analysis of the code can be used to find out how many test cases are needed to guarantee that all of the statements in the program are executed at least once during the testing process.
- White-box testing is also known by other names, such as glass-box testing, structural testing, clear-box testing, open-box testing, logic-driven testing, and path-oriented testing.
- In white-box testing, test cases are selected on the basis of examination of the code.

5 White-Box Testing/ Structural Testing

- Using white-box testing methods the software engineer can test cases that:
  - Guarantee that all independent paths within a module have been exercised at least once.
  - Exercise all logical decision on their true and false sides.
  - Exercise all loops at their boundaries.
  - ……….

6 Functional/ Black-Box Testing

- In functional testing the structure of the program is not considered.
- Test cases are decided on the basis of the requirements or specifications of the program or module.
- Functional testing refers to testing that involves only observation of the output for certain input values.
- Black-box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.
- Other names for black-box testing (BBT) include specifications testing, behavioural testing, data-driven testing, functional testing, and input/output-driven testing.

6 Functional/ Black-Box Testing

- Black-box testing is not an alternative to white-box techniques; rather, it is a complementary approach that is likely to uncover a different class of errors than white-box methods.
- Black-box testing identifies the following kinds of errors:
  - Incorrect or missing functions.
  - Interface missing or erroneous.
  - Errors in data model.
  - Errors in access to external data source.

6 Functional/ Black-Box Testing

Categories of Black-box Testing

- **Positive Functional Testing**: This testing entails exercising the application's functions with valid input and verifying that the outputs are correct.
  - a positive test for the printing function might be to print a document containing both text and graphics to a printer that is online, filled with paper and for which the correct drivers are installed.
- **Negative Functional Testing**: This testing involves exercising application functionality using a combination of invalid inputs, unexpected operating conditions, and other out-of-bounds scenarios.
  - a negative test for the printing function might be to disconnect the printer from the computer while a document is printing.

6 Functional/ Black-Box Testing

Advantages of Black-box Testing

- The test is unbiased because the designer and the tester are independent of each other.
- The tester does not need knowledge of any specific programming languages.
- The test is done from the point-of-view of the user, not the designer.
- Test cases can be designed as soon as the specifications are complete.