Course 2/ Lecture 8

SOFTWARE TESTING TECHNIQUES

What is it?

Designing effective test cases is important, so is the strategy you use to execute them.

-Should you develop a formal plan for your tests?

- Should you test the entire program as a whole or run tests only on a small part of it?

-Should you rerun tests you've already conducted as you add new components to a large system?

-When should you involve the customer?

These and many other questions are answered when you develop a software testing strategy.

Who does it? A strategy for software testing is developed by the project manager, software engineers, and testing specialists.

Verification and Validation

Software testing is one element of a broader topic that is often referred to as verification and validation (V&V). Verification refers to the set of activities that ensure that software correctly implements a specific function. Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements. Boehm [BOE81] states this another way:

Verification: "Are we building the product right?"

Validation: "Are we building the right product?"

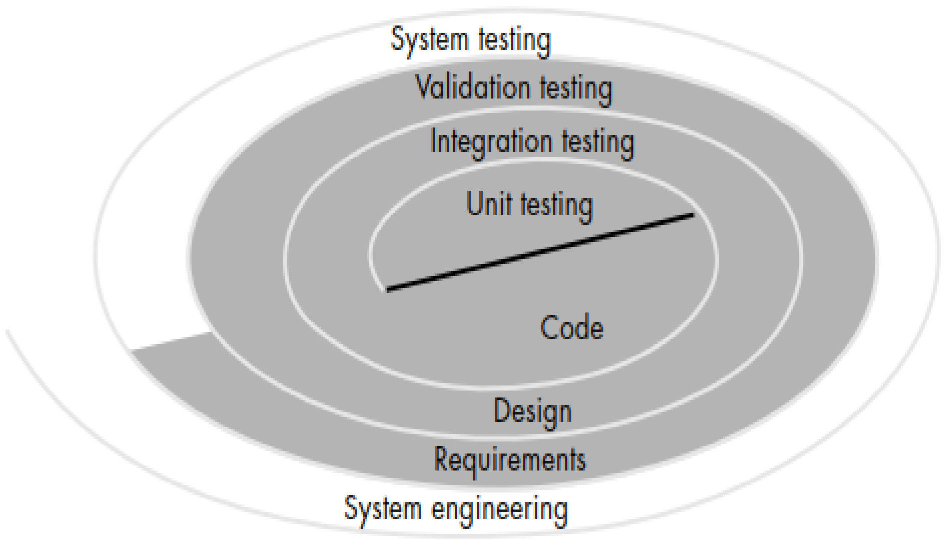
The definition of V&V encompasses many of the activities that we have referred to as software quality assurance (SQA).

Verification and validation encompasses a wide array of SQA activities that include:

performance monitoring, documentation review, database review, algorithm analysis, development testing and installation testing

A Software Testing Strategy

The software engineering process may be viewed as the spiral illustrated in Figure 1. Initially, system engineering defines the role of software and leads to software requirements analysis, where the information domain, function, behavior, performance, constraints, and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software, we spiral inward along streamlines that decrease the level of abstraction on each turn.



Testing strategy

A strategy for software testing may also be viewed in the context of the spiral (Figure .1). Unit testing begins at the vortex of the spiral and concentrates on each unit (i.e., component) of the software as implemented in source code. Testing progresses by moving outward along the spiral to integration testing, where the focus is on design and the construction of the software architecture. Taking another turn outward on the spiral, we encounter validation testing, where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, we arrive at system testing, where the software and other system elements are tested as a whole. To test computer software, we spiral out along streamlines that broaden the scope of testing with each turn.

1- UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented, and the step can be conducted in parallel for multiple components.

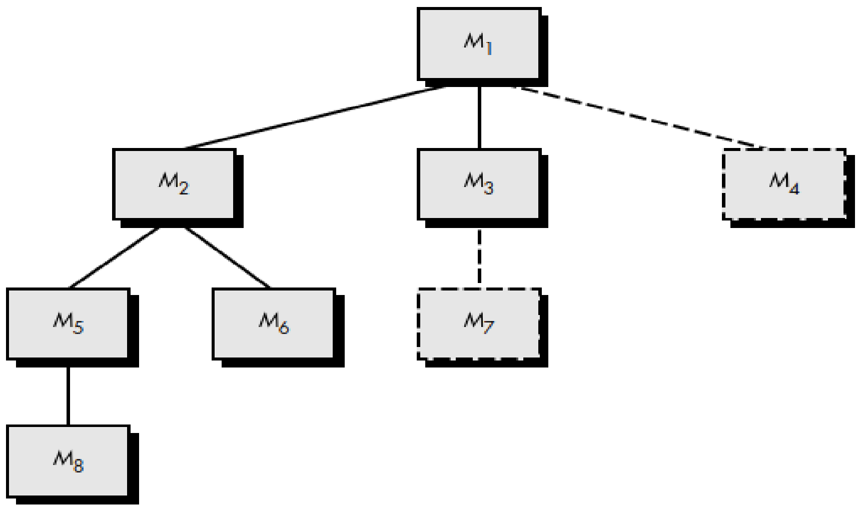
2- INTEGRATION TESTING

A neophyte in the software world might ask a seemingly legitimate question once all modules have been unit tested: "If they all work individually, why do you doubt that they'll work when we put them together?" The problem, of course, is "putting them together"—interfacing. Data can be lost across an interface; one module can have an inadvertent, adverse affect on another; subfunctions, when combined, may not produce the desired major function; individually acceptable imprecision may be magnified to unacceptable levels; global data structures can present problems. Sadly, the list goes on and on. 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.

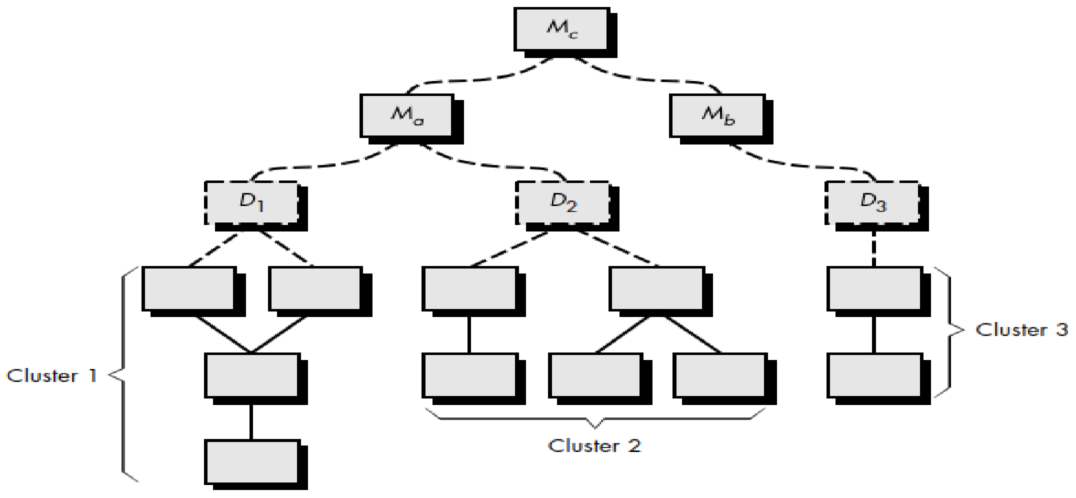
1.1 Top-down Integration

Top-down integration testing is an incremental approach to construction of program structure. 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.



2 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).



Testing Objectives

In software testing, a number of rules that can serve well as testing objectives:

1. Testing is a process of executing a program with the intent of finding an error.

2. A good test case is one that has a high probability of finding an as-yet-undiscovered error.

3. A successful test is one that uncovers an as-yet-undiscovered error.