Levels of Testing
There are three levels of testing:

- Unit testing
- Integration testing
- System testing
Unit Testing (1)

- The process taking of a module and run it in isolation from the rest of the software product by using prepared test cases and comparing actual result with expected output.
- The purpose of this test is to find (and remove) as many errors in the software as practical.
Unit Testing (2)

The reasons for unit testing:

1. The size of single module is small enough that we can locate an error fairly easily
2. The module is small enough that we can attempt to test it in some demonstrably exhaustive fashion
3. Confusing interaction of multiple errors in widely different parts of the software are eliminated
The problems associated with unit testing:

1. How do we run a module without anything to call it, to be called by it?
2. Possibly, to output intermediate values obtained during execution?

The approach is to construct an appropriate drive routine to call it and simple stubs to be called by it, and to insert output statements in it.
The overhead cost due to construct drivers and stubs, is called scaffolding.

Scaffolding does not appear in the delivered product.
The target of integration testing is the interface: whether parameters match on both sides as to type, permissible ranges, meaning and utilization.

Decomposition Based Integration: is based on the functional decomposition of the System. Four Choices are available:
- Top-down Approach
- Bottom-up Approach
- Mixed Approach
- Big bang Approach
Integration process is performed in a series of following 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, subordinate stubs are replace 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 replace with the real component.
Integration Testing (3)

Top Down Integration Testing - Example

- Top module is tested with Stubs
- Stubs are replaced one at a time, "depth first"
- As new modules are integrated, some subset of tests is re-run
<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages if major flaws (thiều sót) occur toward the top of the program</td>
<td>Stub modules must be produced.</td>
</tr>
<tr>
<td>Once the I/O functions are added, representation of a test cases is easier</td>
<td>Test conditions may be impossible, or very difficult, to create</td>
</tr>
<tr>
<td>Early skeletal program allows demonstrations and boosts morale.</td>
<td>Observation of test output is more difficult</td>
</tr>
<tr>
<td></td>
<td>Allows to think that design and testing can be overlapped.</td>
</tr>
<tr>
<td></td>
<td>defer completion of the testing of certain modules</td>
</tr>
</tbody>
</table>
Bottom-up integration testing begins construction and testing with components at the lowest levels in the program structure. Need for stub is eliminated as the required component subordinate is always available.

Integration Strategy

- Low level components are combined into cluster/builds that perform a specific software sub function
- A driver/control program is written to coordinate test case input and output
- Cluster is tested
- Drivers are removed and cluster are combined moving upward in programme structure
## Integration Testing (6)

### Bottom-up Integration Testing

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages of the major flaws occur toward the bottom of the program</td>
<td>Driver modules must be produced</td>
</tr>
<tr>
<td>Test conditions are easier to create</td>
<td>The program as an entity does not exist until the last modules is added</td>
</tr>
<tr>
<td>Observation of test results is easier</td>
<td></td>
</tr>
</tbody>
</table>
Sandwich Testing combines the ideas of bottom-up and top-down testing by defining a certain target layer in the hierarchy of the modules.

The modules below this layer are tested following bottom-up approach, whereas those above the target layer are subject to top-down testing.
Integration Testing (8)

Big bang Approach

- Simplest approach to Integration Testing, where all modules are simply put together and tested.
- This technique is practicable only for very small systems

Problem

Once an error is found, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated.
System Testing (1)

- Objective: to ensure that the system does what the customer wants it to do.
System Testing (2)

Flowchart:

1. Component code
   - Unit test
     - Tested components
   - Integration test
     - Design Specification
     - Integrated modules
System Testing (3)

System functional requirements

Integrated modules → Function test → Functioning systems

Other software requirements

Verified validated software

Verified validated software

Acceptance test

Customer requirements spec.

Accepted system

Installation test

User environment

System In Use!
A function test checks that the integrated system performs its function as specified in the requirement.

Guidelines:
- use a test team independent of the designers and programmers
- know the expected actions and output
- test both valid and invalid input
- never modify the system just to make testing easier
- have stopping criteria
Termination Problem

How decide when to stop testing

- The main problem for managers!
  - Termination takes place when
    - resources (time & budget) are over
    - some coverage is reached