Unit Testing

•A unit is the smallest testable part of an application like functions, classes, procedures, interfaces.

•Unit testing is a method by which individual units of source code are tested to determine if they are fit for use.

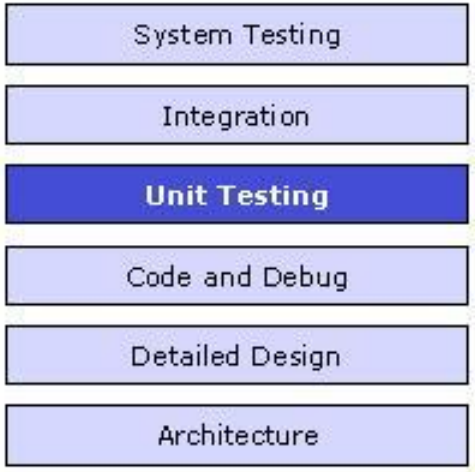
•**Unit tests are basically written and executed by software developers** to make sure that code meets its design and requirements and behaves as expected.

•The goal of unit testing is to segregate each part of the program and test that the individual parts are working correctly.

•This means that for any function or procedure when a set of inputs are given then it should return the proper values. It should handle the failures gracefully during the course of execution when any invalid input is given.

•A unit test provides a written contract that the piece of code must assure. Hence it has several benefits.

•Unit testing is basically done before integration as shown in the image.



Method Used for unit testing: White Box Testing method is used for executing the unit test.

When Unit testing should be done?
Unit testing should be done before Integration testing.

By whom unit testing should be done?
Unit testing should be done by the **developers.**

Advantages of Unit testing:

1. Issues are found at early stage. Since unit testing are carried out by developers where they test their individual code before the integration. Hence the issues can be found very early and can be resolved then and there without impacting the other piece of codes.

2. Unit testing helps in maintaining and changing the code. This is possible by making the codes less interdependent so that unit testing can be executed. Hence chances of impact of changes to any other code gets reduced.

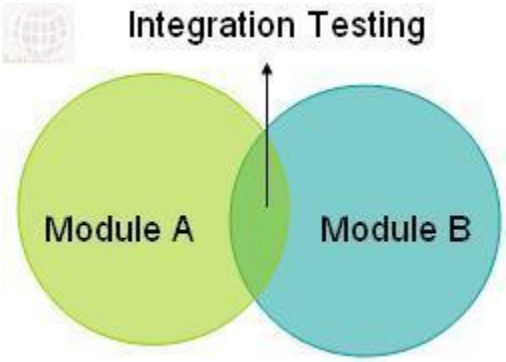
3. Since the bugs are found early in unit testing hence it also helps in reducing the cost of bug fixes. Just imagine the cost of bug found during the later stages of development like during system testing or during acceptance testing.

4. Unit testing helps in simplifying the debugging process. If suppose a test fails then only latest changes made in code needs to be debugged.

Integration Testing

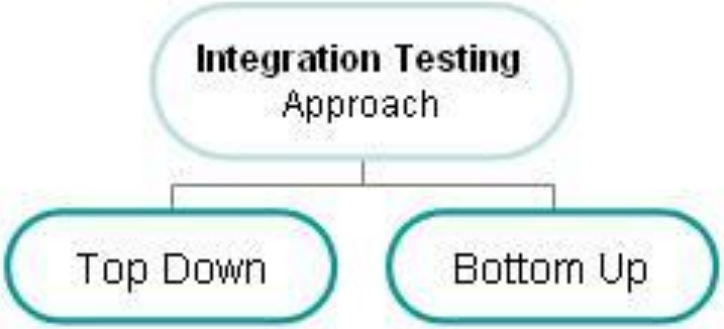
•Integration testing tests integration or interfaces between components, interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems.

•Also after integrating two different components together we do the integration testing. As displayed in the image below when two different modules ‘Module A’ and ‘Module B’ are integrated then the integration testing is done.



Integration testing is done by a specific integration tester or test team.

Integration testing follows two approach known as ‘Top Down’ approach and ‘Bottom Up’ approach as shown in the image below:

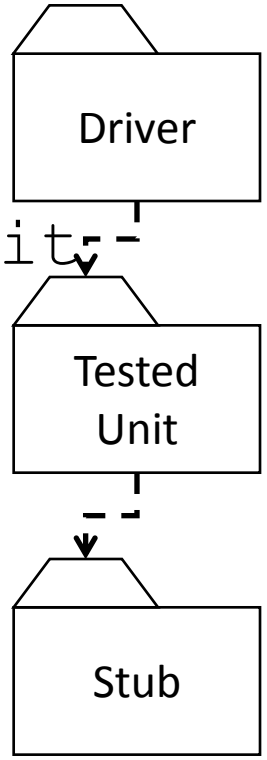


Why do we do integration testing?

- Unit tests only test the unit in isolation
- Many failures result from faults in the interaction of subsystems
- Often many Off-the-shelf components are used that cannot be unit tested
- Without integration testing the system test will be very time consuming
- Failures that are not discovered in integration testing will be discovered after the system is deployed and can be very expensive.

Stubs and drivers

- Driver:
  - A component, that calls the `TestedUnit`
  - Controls the test cases
- Stub:
  - A component, the `TestedUnit` depends on
  - Partial implementation
  - Returns fake values.



1. Big Bang integration testing: (Non Incremental Integration Testing)

•In Big Bang integration testing all components or modules are integrated simultaneously, after which everything is tested as a whole.

•In this approach individual modules are not integrated until and unless all the modules are ready.

•In Big Bang integration testing all the modules are integrated without performing any integration testing and then it's executed to know whether all the integrated modules are working fine or not.

•This approach is generally executed by those developers who follows the 'Run it and see' approach.

•Because of integrating everything at one time if any failures occurs then it become very difficult for the programmers to know the root cause of that failure.

•In case any bug arises then the developers has to detach the integrated modules in order to find the actual cause of the bug.

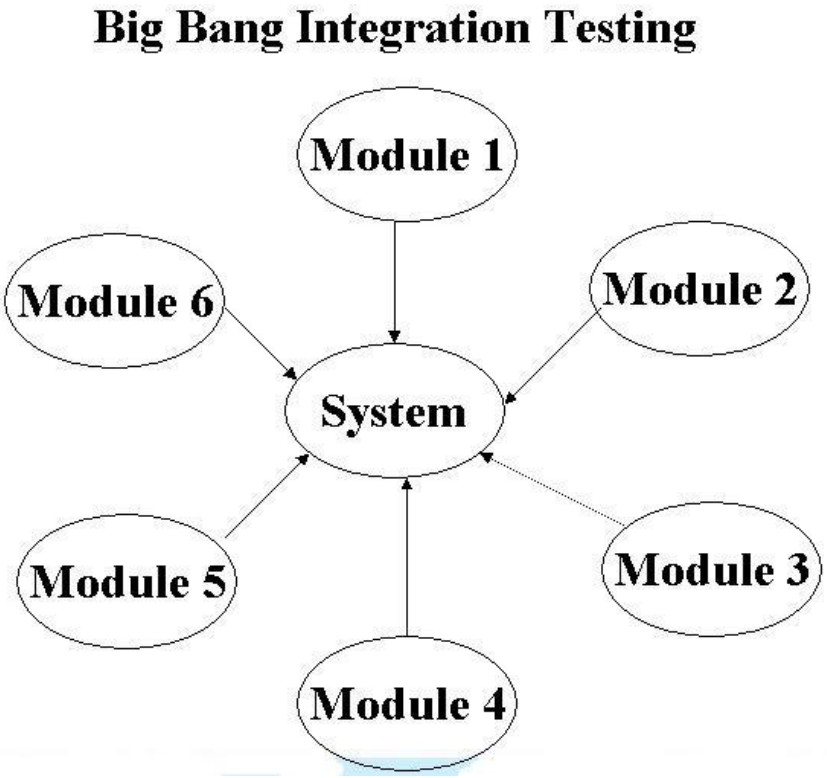
Advantage: Big Bang testing has the advantage that everything is finished before integration testing starts.

Disadvantage:

The major disadvantage is that in general it is **time consuming** and

difficult to trace the cause of failures because of this late integration.

The chances of having critical failures are more because of integrating all the components together at same time.



If any bug is found then it is very difficult to detach all the modules in order to find out the root cause of it. There is high probability of occurrence of the critical bugs in the production environment

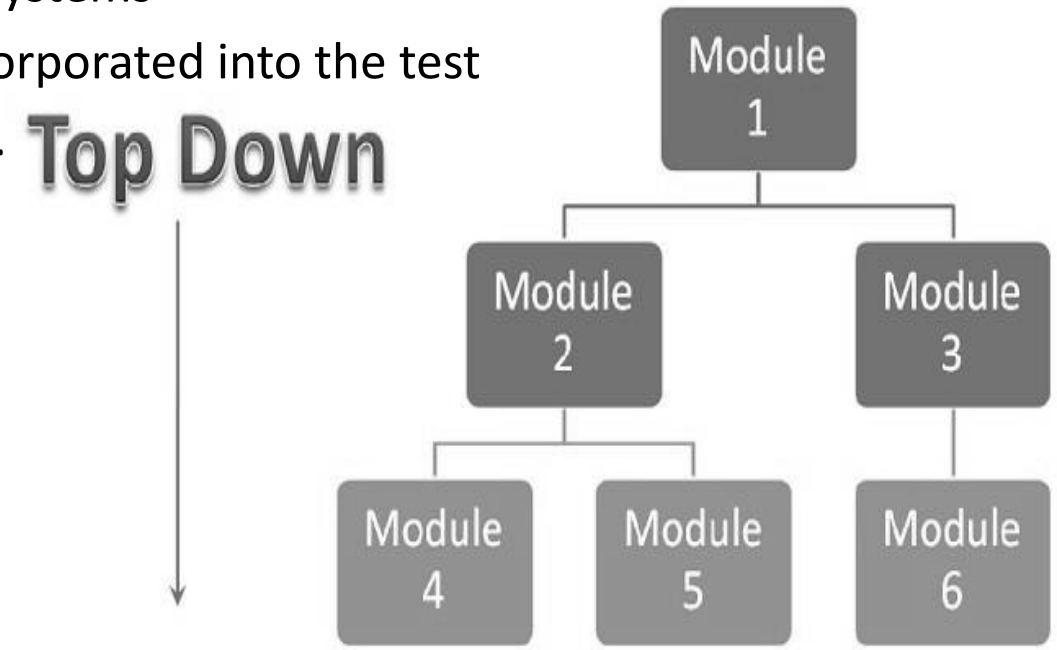
Incremental Integration Testing – Top Down and Bottom Up Integration Testing

2. **Top-down integration testing:**

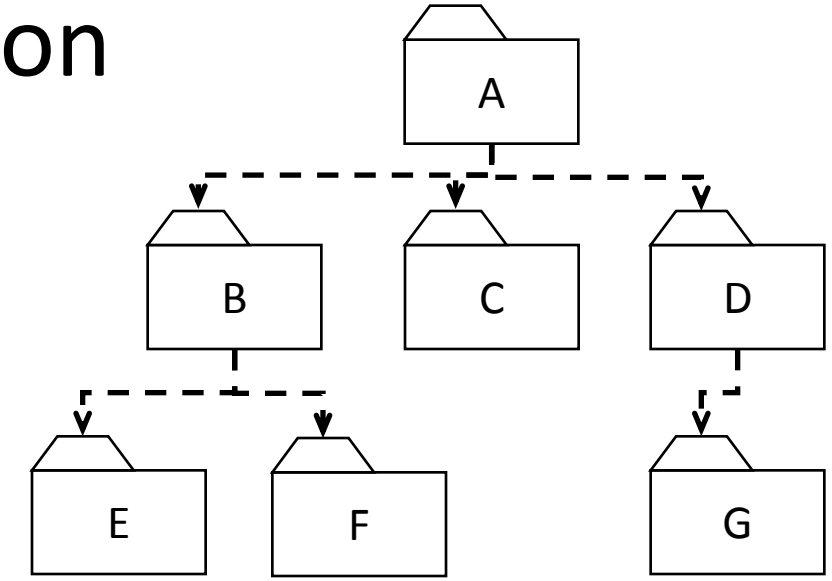
Testing takes place from top to bottom, following the control flow or architectural structure (e.g. starting from the GUI or main menu).

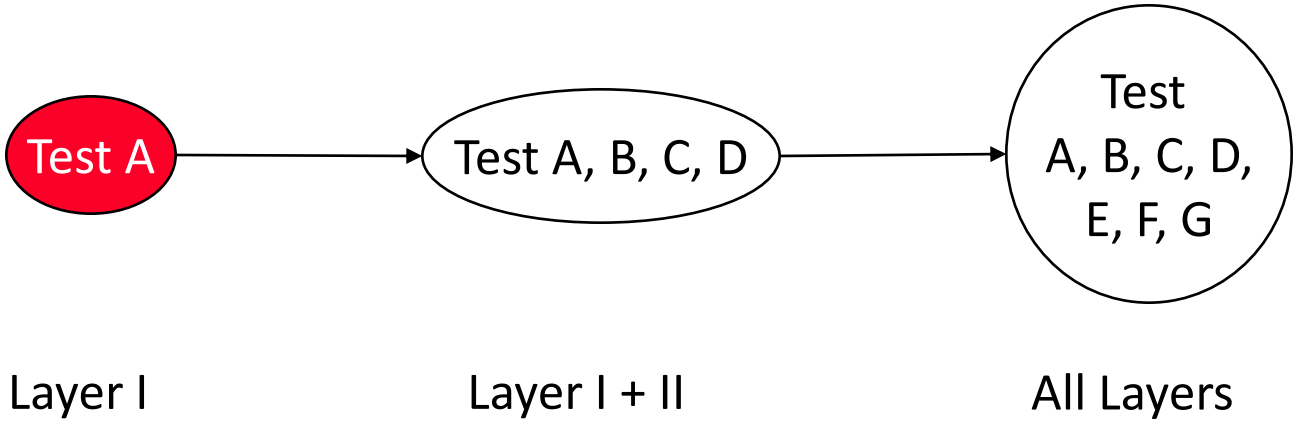
Components or systems are substituted by stubs. Below is the diagram of 'Top down Approach':

- Test the top layer or the controlling subsystem first
- Then combine all the subsystems that are called by the tested subsystems and test the resulting collection of subsystems
- Do this until all subsystems are incorporated into the test
- Stubs are needed to do the testing.



Top-down Integration





Advantages of Top-Down approach:

The tested product is very consistent because the integration testing is basically performed in an environment that almost similar to that of reality

Stubs can be written with lesser time because when compared to the drivers then Stubs are simpler to author.

Test cases can be defined in terms of the functionality of the system (functional requirements)

No drivers needed

Disadvantages of Top-Down approach:

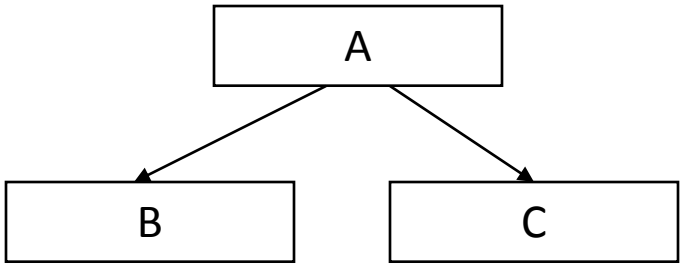
Basic functionality is tested at the end of cycle

Writing stubs is difficult: Stubs must allow all possible conditions to be tested.

Large number of stubs may be required, especially if the lowest level of the system contains many methods.

Some interfaces are not tested separately.

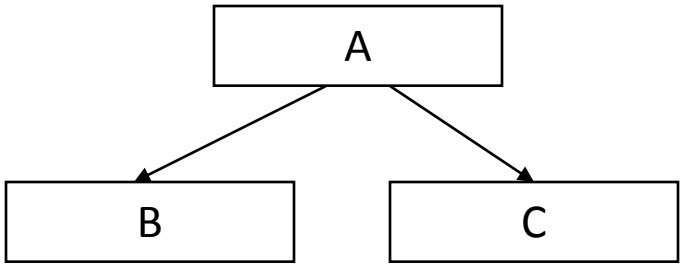
Bottom-Up Testing Example



1) Test B, C individually (using drivers)
2) Test A such that it calls B
   If an error occurs we know that the problem is in A or in the interface between A and B
3) Test A such that it calls C
   If an error occurs we know that the problem is in A or in the interface between A and C

(-) Top level components are the most important yet tested last.

Top-Down Testing Example

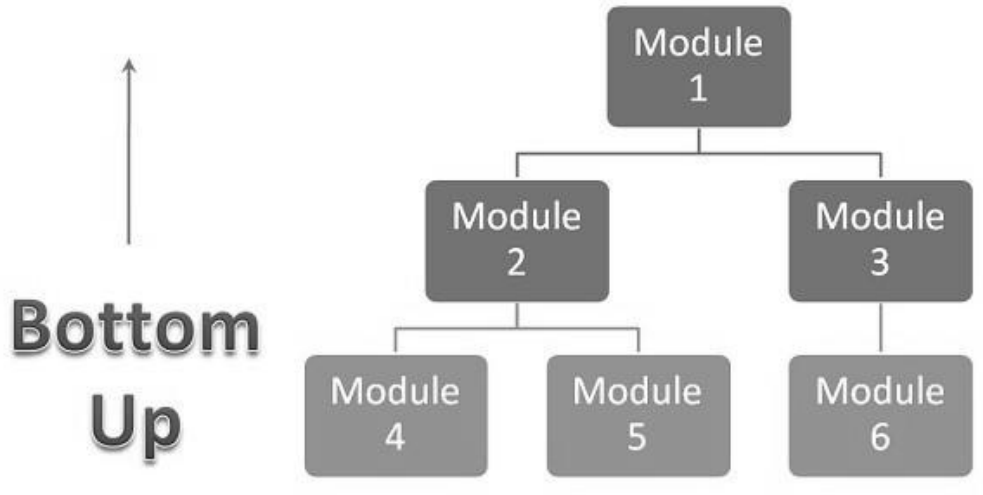


1) Test A individually (use stubs for B and C)
2) Test A such that it calls B (stub for C)
   If an error occurs we know that the problem is in B or in the interface between A and B
3) Test A such that it calls C (stub for B)
   If an error occurs we know that the problem is in C or in the interface between A and C

* Stubs are used to simulate the activity of components that are not currently tested; (-) may require many stubs

3. **Bottom-up integration testing:**

Testing takes place from the bottom of the control flow upwards.
Components or systems are substituted by drivers.
Below is the image of ‘Bottom up approach’:



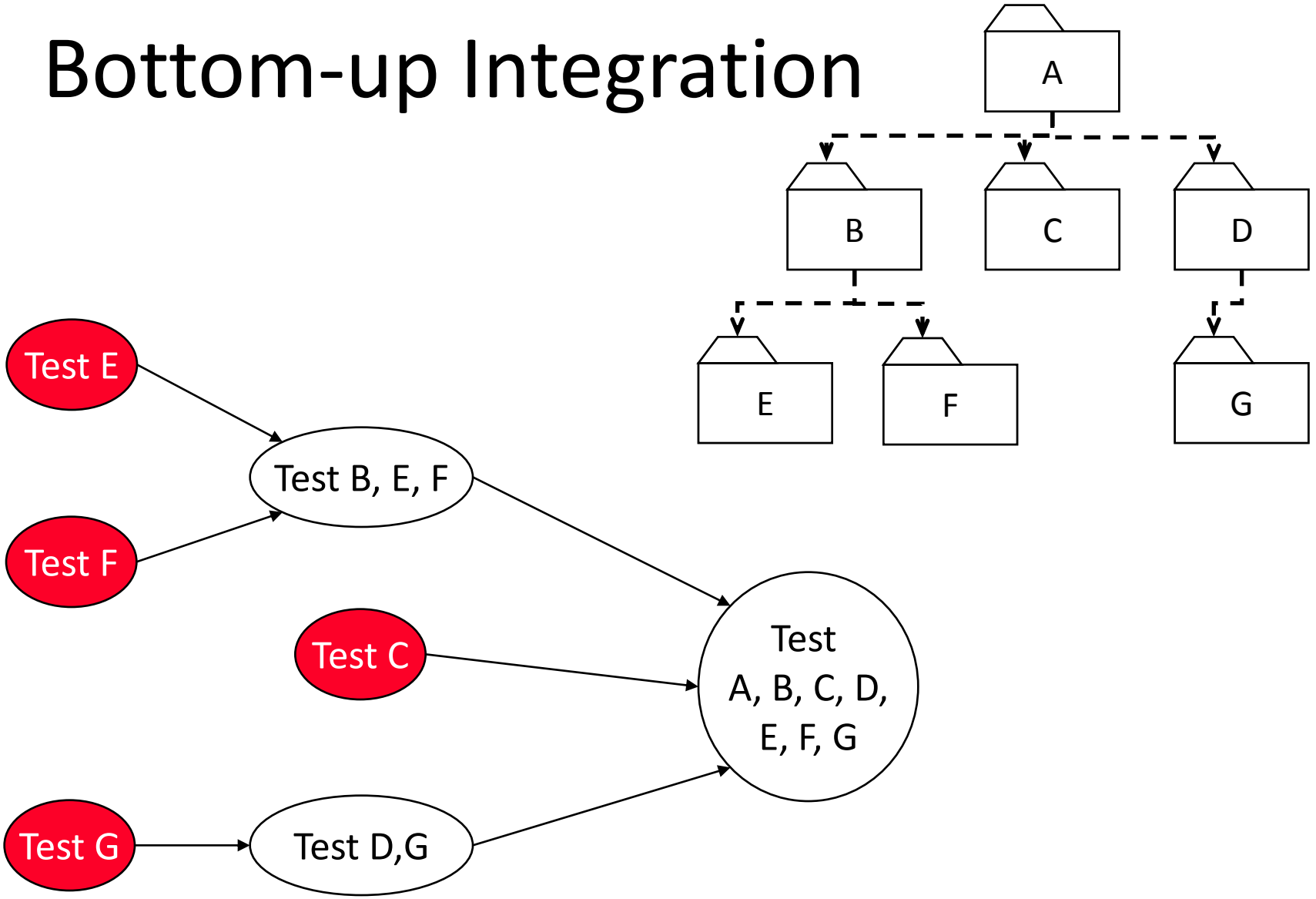
The subsystems in the lowest layer of the call hierarchy are tested individually

Then the next subsystems are tested that call the previously tested subsystems

This is repeated until all subsystems are included

Drivers are needed.

Bottom-up Integration



Advantage of Bottom-Up approach:

In this approach development and testing can be done together so that the product or application will be efficient and as per the customer specifications.

No stubs needed

Useful for integration testing of the following systems

- Object-oriented systems
- Real-time systems
- Systems with strict performance requirements.

Disadvantages of Bottom-Up approach:

We can catch the Key interface defects at the end of cycle

It is required to create the test drivers for modules at all levels except the top control

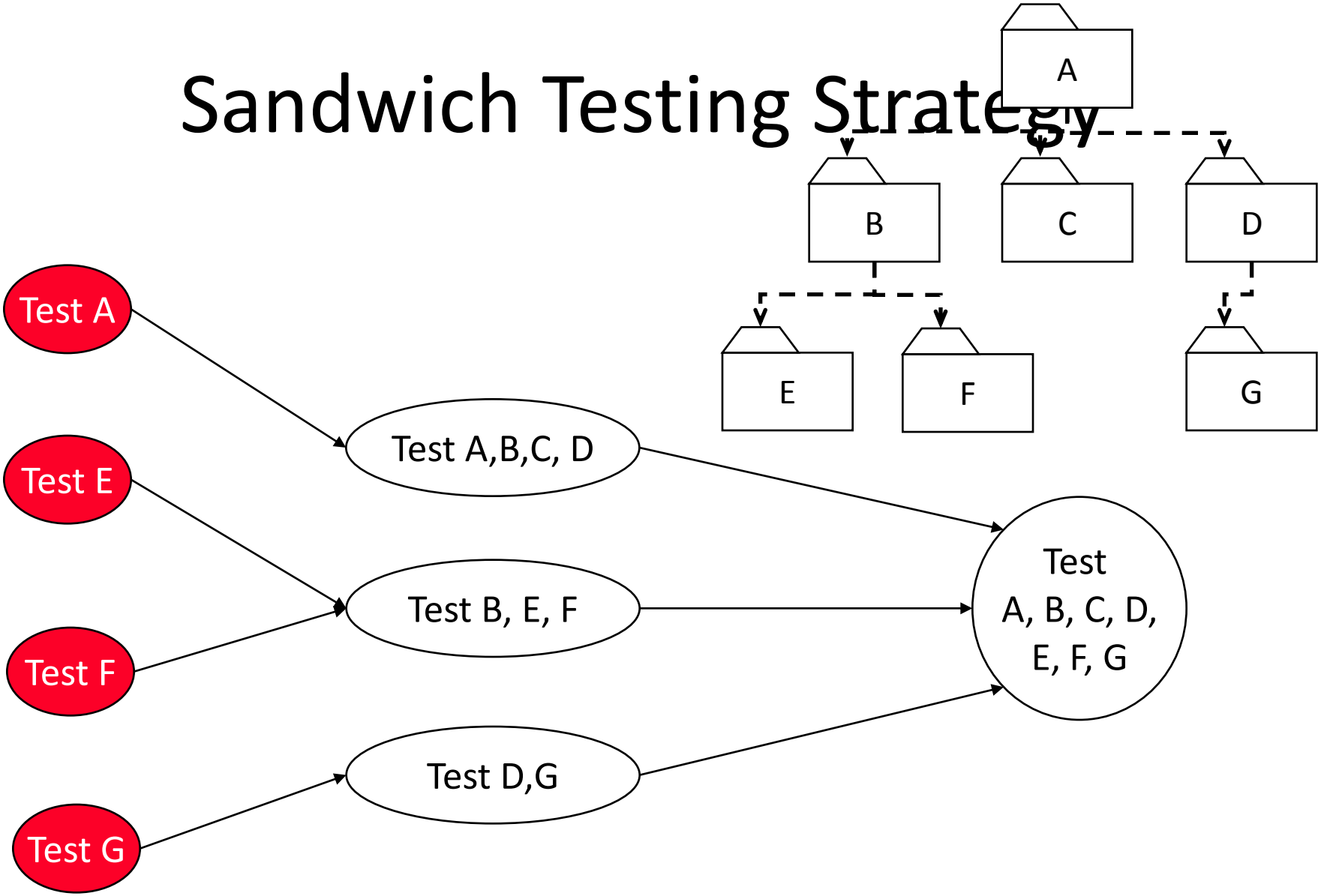
Tests the most important subsystem (user interface) last

Drivers needed

Sandwich Testing Strategy

- Combines top-down strategy with bottom-up strategy
- The system is viewed as having three layers
  - A target layer in the middle
  - A layer above the target
  - A layer below the target
- Testing converges at the target layer.

Sandwich Testing Strategy



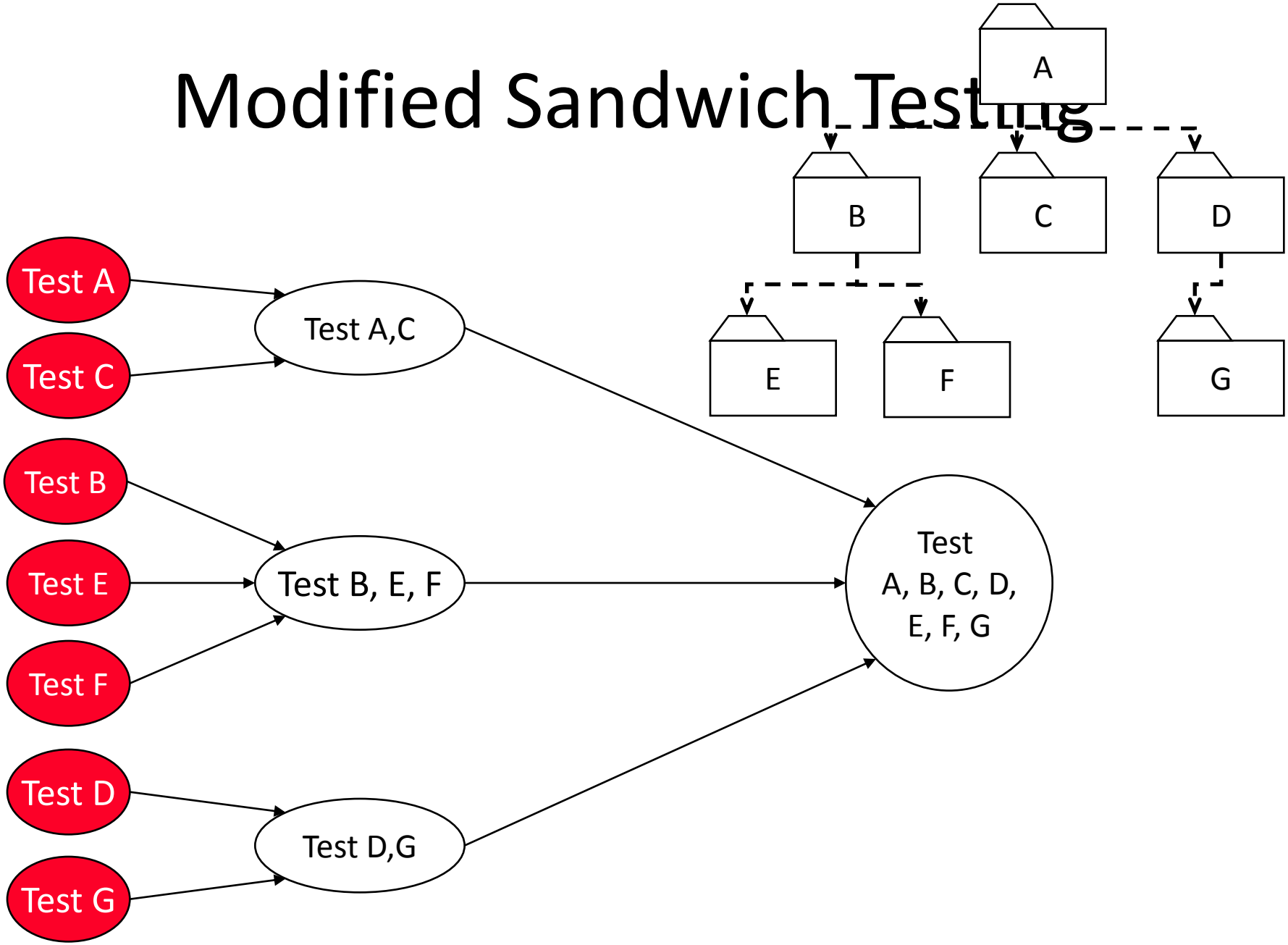
Pros and Cons of Sandwich Testing

- Top and Bottom Layer Tests can be done in parallel
- Problem: Does not test the individual subsystems and their interfaces thoroughly before integration
- Solution: Modified sandwich testing strategy

Modified Sandwich Testing Strategy

- Test in parallel:
  - Middle layer with drivers and stubs
  - Top layer with stubs
  - Bottom layer with drivers
- Test in parallel:
  - Top layer accessing middle layer (top layer replaces drivers)
  - Bottom accessed by middle layer (bottom layer replaces stubs).

Modified Sandwich Testing



Integration Strategies Comparison

TABLE 8.7 Comparison of Integration Strategies (Myers 1979)

	Bottom-up	Top-down	Modified top-down	Big-bang	Sandwich	Modified sandwich
Integration	Early	Early	Early	Late	Early	Early
Time to basic working program	Late	Early	Early	Late	Early	Early
Component drivers needed	Yes	No	Yes	Yes	Yes	Yes
Stubs needed	No	Yes	Yes	Yes	Yes	Yes
Work parallelism at beginning	Medium	Low	Medium	High	Medium	High
Ability to test particular paths	Easy	Hard	Easy	Easy	Medium	Easy
Ability to plan and control sequence	Easy	Hard	Hard	Easy	Hard	Hard

Functionality testing is performed to verify that a software application performs and functions correctly according to design specifications.

During functionality testing we check the core application functions, text input, menu functions and installation and setup on localized machines, etc.

The following is needed to be checked during the functionality testing:

- Installation and setup on localized machines running localized operating systems and local code pages.
- Text input, including the use of extended characters or non-Latin scripts.
- Core application functions.
- String handling, text, and data, especially when interfacing with non-Unicode applications or modules.
- Regional settings defaults.
- Text handling (such as copying, pasting, and editing) of extended characters, special fonts, and non-Latin scripts.
- Accurate hot-key shortcuts without any duplication.

In **non-functional testing** the quality characteristics of the component or system is tested.

Non-functional refers to aspects of the software that may not be related to a specific function or user action such as scalability or security.

Eg. How many people can log in at once? Non-functional testing is also performed at all levels like functional testing.

Non-functional testing includes:

Functionality testing	Reliability testing
Usability testing	Efficiency testing
Maintainability testing	Portability testing
Baseline testing	Compliance testing
Documentation testing	Endurance testing
Load testing	Performance testing
Compatibility testing	Security testing
Scalability testing	Volume testing
Stress testing	Recovery testing

Internationalization testing and Localization testing