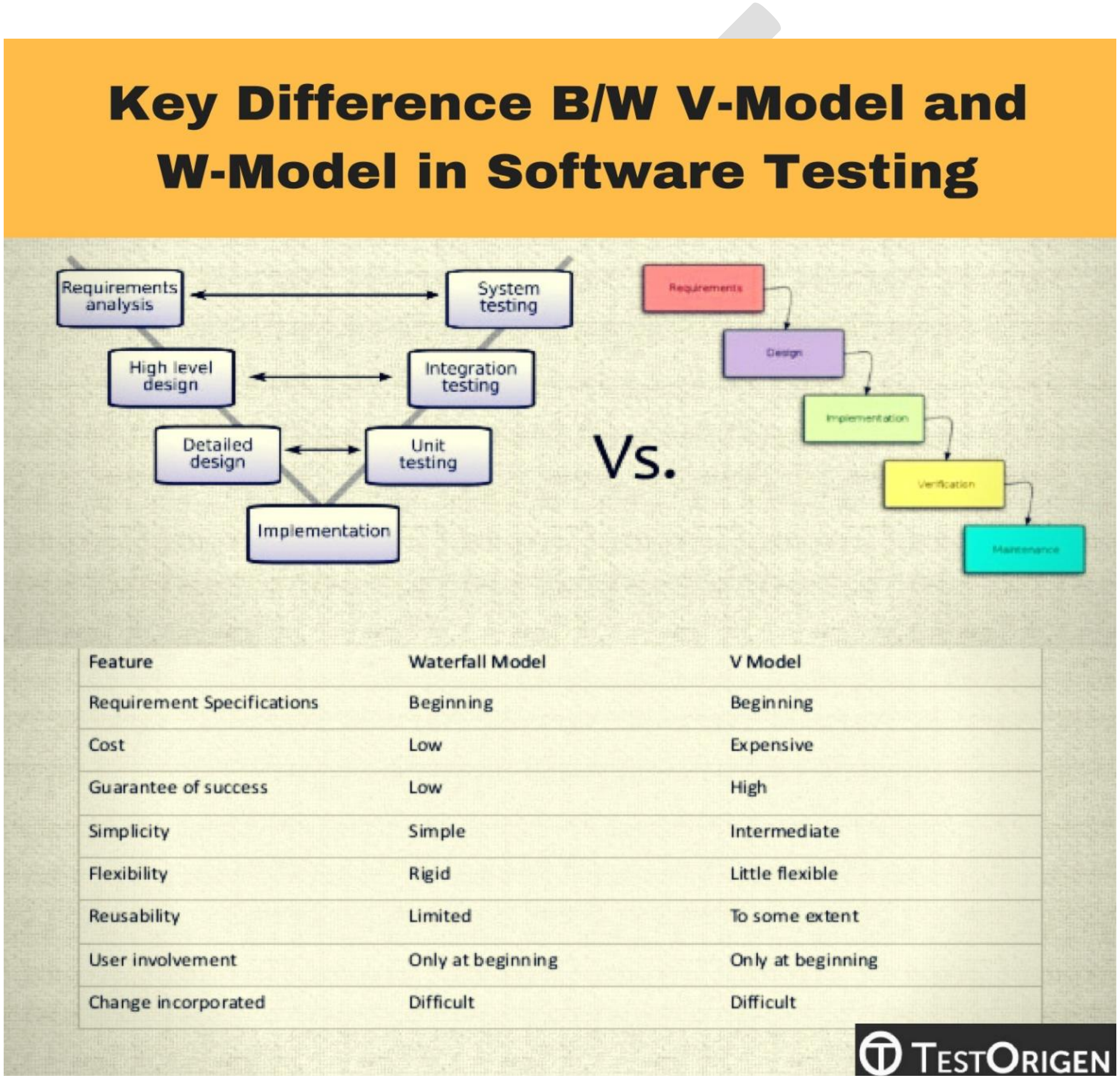
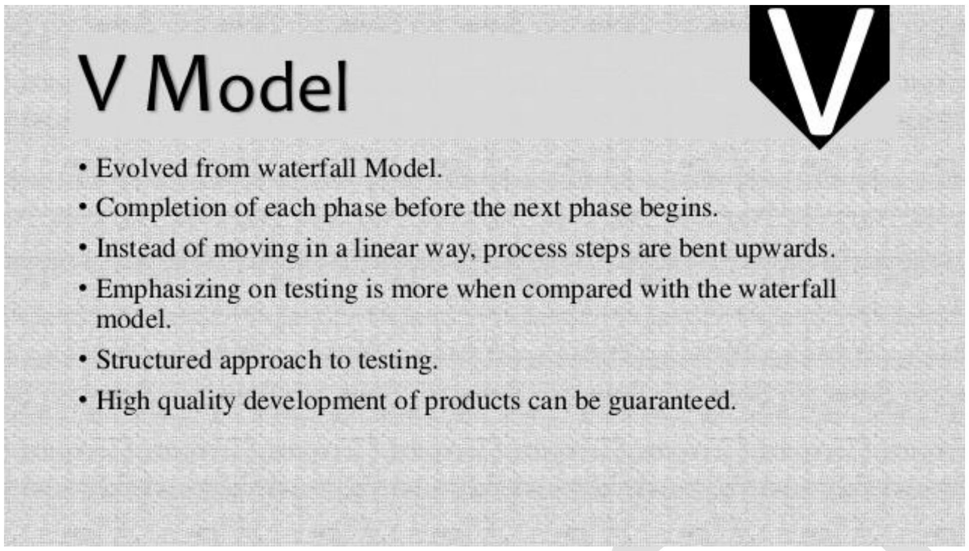
1. What is the difference between V-model and W-model?

A. **V model and W model** are two of the most important **models** that are followed **in the** process of software testing. ... On the other hand, **W model** is a sequential approach to test a product and can be done only once the development of the product is complete **with** no modifications required to be done.

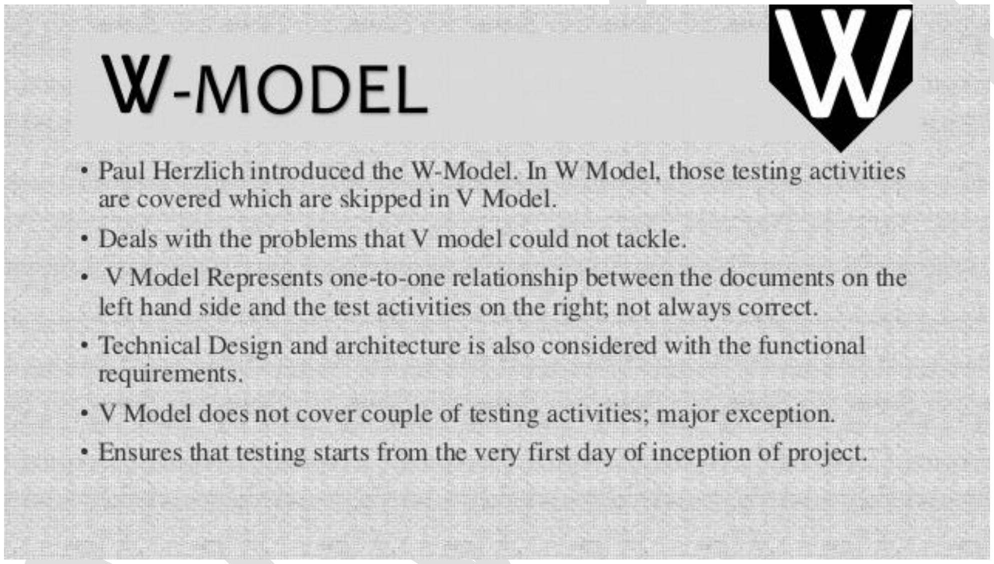


Feature	Waterfall Model	V Model
Requirement Specifications	Beginning	Beginning
Cost	Low	Expensive
Guarantee of success	Low	High
Simplicity	Simple	Intermediate
Flexibility	Rigid	Little flexible
Reusability	Limited	To some extent
User involvement	Only at beginning	Only at beginning
Change incorporated	Difficult	Difficult



V Model

- Evolved from waterfall Model.
- Completion of each phase before the next phase begins.
- Instead of moving in a linear way, process steps are bent upwards.
- Emphasizing on testing is more when compared with the waterfall model.
- Structured approach to testing.
- High quality development of products can be guaranteed.



W-MODEL

- Paul Herzlich introduced the W-Model. In W Model, those testing activities are covered which are skipped in V Model.
- Deals with the problems that V model could not tackle.
- V Model Represents one-to-one relationship between the documents on the left hand side and the test activities on the right; not always correct.
- Technical Design and architecture is also considered with the functional requirements.
- V Model does not cover couple of testing activities; major exception.
- Ensures that testing starts from the very first day of inception of project.

What is the difference between Iterative and Incremental model

The Incremental approach is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the product is finished. It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements
The Iterative Design is a design methodology based on a cyclic process of prototyping, testing, analyzing, and refining a product or process. Based on the results of testing the most recent iteration of a design, changes and refinements are made. This process is intended to ultimately improve the quality and functionality of a design. In iterative design, interaction with the designed system is used as a form of research for informing and evolving a project, as successive versions, or iterations of a design are implemented.

The **Incremental Approach** uses a set number of steps and development goes from start to finish in a linear path of progression.
Incremental development is done in steps from design, implementation, testing/verification, maintenance. These can be broken down further into sub-steps but most incremental models follow that same pattern. The **Waterfall Model** is a traditional incremental development approach.
The **Iterative Approach** has no set number of steps, rather development is done in cycles. Iterative development is less concerned with tracking the progress of individual features. Instead, focus is put on creating a working prototype first and adding features in development cycles where the Increment Development steps are done for every cycle. **Agile Modeling** is a typical iterative approach.

What is component testing ?

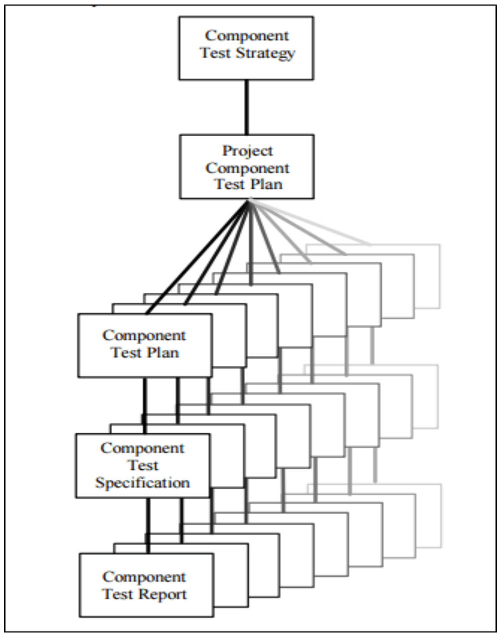
A. **Component testing** is defined as a software **testing** type, in which the **testing** is performed on each individual **component** separately without integrating with other **components**. ... **Component Testing** is also referred to as Unit **Testing**, Program **Testing** or Module **Testing**.

Component testing is defined as a software testing type, in which the testing is performed on each individual component separately without integrating with other components. It's also referred to as Module Testing when it is viewed from an architecture perspective. Component Testing is also referred to as Unit Testing, Program Testing or Module Testing.

Generally, any software as a whole is made of several components. Component Level Testing deals with testing these components individually.

It's one of most frequent black box testing types which is performed by QA Team.

As per the below diagram, there will be a test strategy and test plan for component testing. Where each and every part of the software or application is considered individually. For each of this component a Test Scenario will be defined, which will be further brought down into a High Level Test Cases -> Low Level detailed Test Cases with Prerequisites.



The usage of the term "**Component Testing**" varies from domain to domain and organization to organization.

The most common reason for different perception of Component testing are

1. Type of Development Life Cycle Model Chosen
2. Complexity of the software or application under test
3. Testing with or without isolation from rest of other component in software or application.

As we know Software Test Life Cycle Architecture has lots many test-artifacts (Documents made, used during testing activities). Among many tests – artifacts, it's the Test Policy & Test Strategy which defines the types of testing, depth of testing to be performed in a given project.

Component testing is performed by testers. 'Unit Testing' is performed by the developers where they do the testing of the individual functionality or procedure. After Unit Testing is performed, the next testing is component testing. Component testing is done by the testers.

When to perform Component testing

Component testing is performed soon after the Unit Testing is done by the developers and the build is released for the testing team. This build is referred as UT build (Unit Testing Build). Major functionality of all the components are tested in this phase,

Entry criteria for component testing

- Minimum number of the component to be included in the UT should be developed & unit tested.

Exit criteria for component testing

- The functionality of all the component should be working fine.
- There should not presence of any Critical or High or Medium severity & priority defects Defect log.

Component Testing Techniques

Based on depth of testing levels, Component testing can be categorized as

1. **CTIS - Component Testing In Small**
2. **CTIL - Component Testing In Large**

CTIS – Component Testing in Small

Component testing may be done with or without isolation of rest of other components in the software or application under test. If it's performed with the isolation of other component, then it's referred as Component Testing in Small.

Example 1: Consider a website which has 5 different web pages then testing each webpage separately & with the isolation of other components is referred as Component testing in Small.

Home, Testing, SAP, Web, Must Learn!, Big Data, Live Projects, Blog and etc.

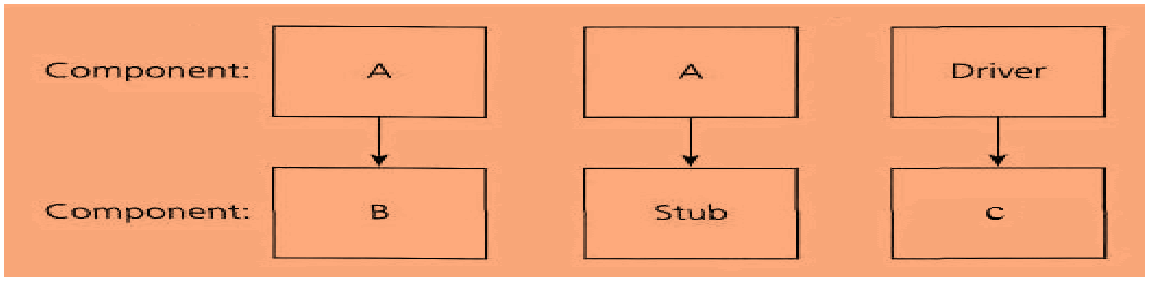
Similarly, any software is made of many components and also, every component will have its own subcomponents. Testing each modules mentioned in example 2 separately without considering integration with other components is referred as **Component Testing in Small.**

CTIS – Component Testing in Small

Component testing done without isolation of other components in the software or application under test is referred as Component Testing Large.

Let's take an example to understand it in a better way. Suppose there is an application consisting of three components say **Component A**, **Component B,** and **Component C**.

The developer has developed the component B and wants it tested. But in order to **completely** test the component B, few of its functionalities are dependent on component A and few on component C.



Functionality Flow: **A** -> B -> **C** which means there is a dependency to B from both A & C, as per the diagram stub is the **called function,** and the driver is the **calling function**.

But the component A and component C has not been developed yet. In that case, to test the component B completely, we can replace the component A and component C by stub and drivers as required. So basically, component A & C are replaced by stub & driver's which acts as a dummy object till they are actually developed.

- **Stub:** A stub is called from the software component to be tested as shown in the diagram below 'Stub' is called by Component A.
- **Driver:** A driver calls the component to be tested as shown in the diagram below 'Component B' is called by Driver.

So here login page is one component, and the home page is another. Now testing the functionality of individual pages separately is called **component testing**.

Component testing scenario's on web page1 –

- Enter invalid user id and verify if any user-friendly warning pop up is shown to the end user.
- Enter invalid user id and password and click on 'reset' and verify if the data entered in the text fields user-id and password are cleared out.
- Enter the valid user name and password and click on 'Login' button.

Component testing scenario's on web page2 –

- Verify if the "Welcome to manager page of aditya" message is being displayed on the home page.
- Verify if all the link on the left side of the web page are clickable.
- Verify if the manager id is being displayed in the center of the home page.
- Verify the presence of the 3 different images on the home page as per the diagram.

Unit Testing Vs Component Testing

Unit Testing	Component Testing
• Testing individual programs, modules to demonstrate that program executes as per the specification is called **Unit Testing**	• Testing each object or parts of the software separately with or without isolation of other objects is called **Component Testing**
• Its validated against design documents	• Its validated against test requirements, use cases

• Unit testing is done by Developers	• Component testing is done by Testers
• Unit testing is done first	• Component testing is done after unit testing is complete from the developers end.

What is Integration testing?

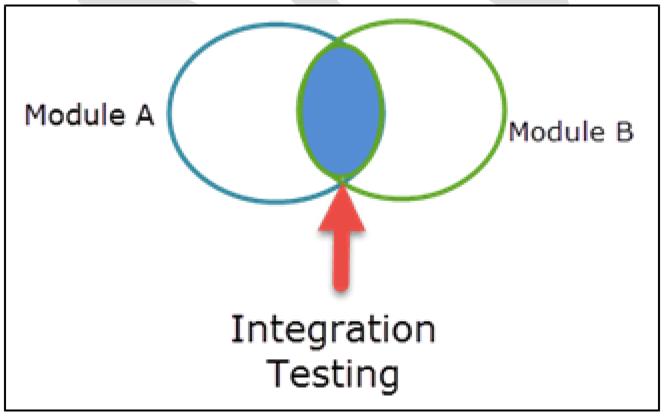
System **Integration Testing** is defined as a type of **software testing** carried out in an **integrated** hardware and **software** environment to verify the behavior of the complete system. ... For **Example**, **software** and/or hardware components are combined and **tested** progressively until the entire system has been **integrated**.

System Integration Testing is defined as a type of software testing carried out in an integrated hardware and software environment to verify the behavior of the complete system. It is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirement.

System Integration Testing (SIT) is performed to verify the interactions between the modules of a software system. It deals with the verification of the high and low-level software requirements specified in the Software Requirements Specification/Data and the Software Design Document.

It also verifies a software system's coexistence with others and tests the interface between modules of the software application. In this type of testing, modules are first tested individually and then combined to make a system.

For Example, software and/or hardware components are combined and tested progressively until the entire system has been integrated.



In Software Engineering, System Integration Testing is done because,

- It helps to detect Defect early
- Earlier feedback on the acceptability of the individual module will be available
- Scheduling of Defect fixes is flexible, and it can be overlapped with development
- Correct data flow
- Correct control flow
- Correct timing
- Correct memory usage
- Correct with software requirements

Usually while performing Integration Testing, ETVX (Entry Criteria, Task, Validation, and Exit Criteria) strategy is used.

Entry Criteria:

- Completion of Unit Testing

Inputs:

- Software Requirements Data
- Software Design Document
- Software Verification Plan
- Software Integration Documents

Activities:

- Based on the High and Low-level requirements create test cases and procedures
- Combine low-level modules builds that implement a common functionality
- Develop a test harness
- Test the build
- Once the test is passed, the build is combined with other builds and tested until the system is integrated as a whole.
- Re-execute all the tests on the target processor-based platform, and obtain the results

Exit Criteria:

- Successful completion of the integration of the Software module on the target Hardware
- Correct performance of the software according to the requirements specified

Outputs

- Integration test reports
- Software Test Cases and Procedures [SVCP].

Incremental Approach

Incremental testing is a way of integration testing. In this type of testing method, you first test each module of the software individually and then continue testing by appending other modules to it then another and so on.

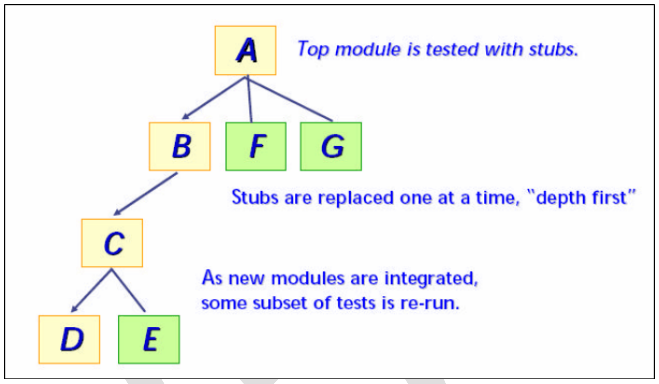
Incremental integration is the contrast to the big bang approach. The program is constructed and tested in small segments, where errors are easier to isolate and correct. Interfaces are more likely to be tested completely, and a systematic test approach may be applied.

There are two types of Incremental testing

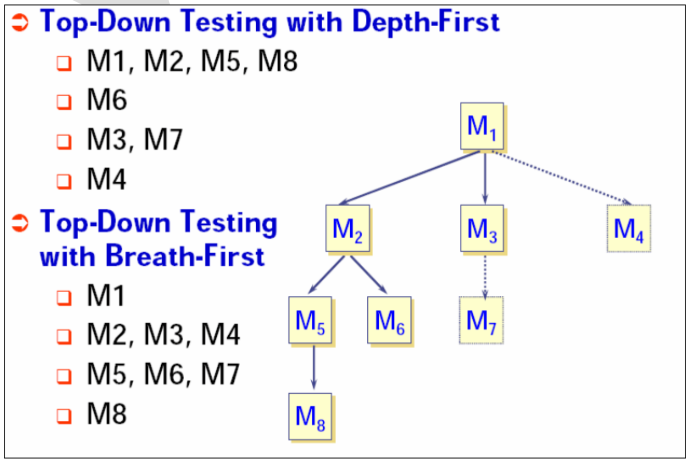
- Top down approach
- Bottom Up approach

Top-Down Approach

In this type of approach, individual start by testing only the user interface, with the underlying functionality simulated by stubs, then you move downwards integrating lower and lower layers as shown in the image below.


A
Top module is tested with stubs.
B
F
G
Stubs are replaced one at a time, "depth first"
C
As new modules are integrated,
some subset of tests is re-run.
D
E

- Starting with the main control module, the modules are integrated by moving downward through the control hierarchy
- Sub-modules to the main control module are incorporated into the structure either in a breadth-first manner or depth-first manner.
- Depth-first integration integrates all modules on a major control path of the structure as displayed in the following diagram:



Top-Down Testing with Depth-First
- M1, M2, M5, M8
- M6
- M3, M7
- M4

Top-Down Testing with Breath-First
- M1
- M2, M3, M4
- M5, M6, M7
- M8

The module integration process is done in the following manner:

1. The main control module is used as a test driver, and the stubs are substituted for all modules directly subordinate to the main control module.
2. The subordinate stubs are replaced one at a time with actual modules depending on the approach selected (breadth first or depth first).
3. Tests are executed as each module is integrated.
4. On completion of each set of tests, another stub is replaced with a real module on completion of each set of tests
5. To make sure that new errors have not been introduced Regression Testing may be performed.

The process continues from step2 until the entire program structure is built. The top-down strategy sounds relatively uncomplicated, but in practice, logistical problems arise.

The most common of these problems occur when processing at low levels in the hierarchy is required to adequately test upper levels.

Stubs replace low-level modules at the beginning of top-down testing and, therefore no significant data can flow upward in the program structure.

Challenges Tester might face:

- Delay many tests until stubs are replaced with actual modules.
- Develop stubs that perform limited functions that simulate the actual module.
- Integrate the software from the bottom of the hierarchy upward.

The second approach is workable but can lead to significant overhead, as stubs become increasingly complex.

Bottom-up Approach

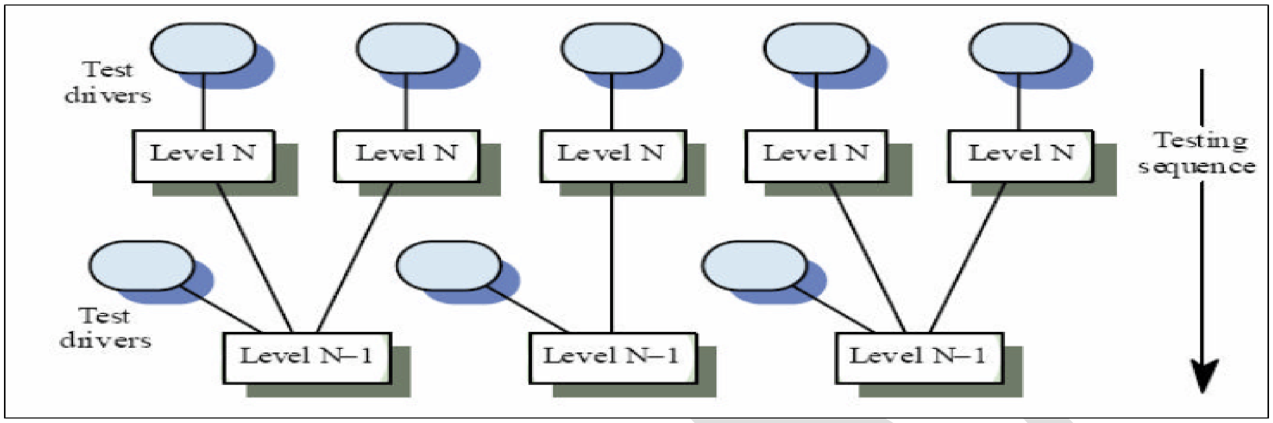
Bottom-up integration begins construction and testing with modules at the lowest level in the program structure. In this process, the modules are integrated from the bottom to the top.

In this approach processing required for the modules subordinate to a given level is always available and the need for the stubs is eliminated.

This integration test process is performed in a series of four steps

1. Low-level modules are combined into clusters that perform a specific software sub-function.
2. A driver is written to coordinate test case input and output.
3. The cluster or build is tested.
4. Drivers are removed, and clusters are combined moving upward in the program structure.

As integration moves upward, the need for separate test drivers lessons. 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. Integration follows the pattern illustrated below. As integration moves upward, the need for separate test drivers lessons.


Test drivers
Level N
Level N
Level N
Level N
Level N
Testing sequence
Test drivers
Level N–1
Level N–1
Level N–1