UNIT-II

Integration Testing:

Integration is defined as the set of interactions among components. Testing the interaction between the modules and interaction with other systems externally is called integration testing.

OR

Integration testing is the process of testing the interface between two software units or module. It's focus on determining the correctness of the interface. The purpose of the integration testing is to expose faults in the interaction between integrated units.

The final round of integration involving all components is called Final Integration Testing (FIT), or system integration.

Integration is both a phase and a type of testing. Integration testing is aimed at testing the interactions among the modules, this testing just like white box, black box testing.

Integration Testing are of two types:

Integration testing means testing of interfaces. There are two types of interfaces.

1. **Internal Interface.**
2. **External Interface.**

Internal Interfaces are those that provide communication across two modules within a project or product, internal to the product, and not exposed to the customer or external developers.

External Interfaces were provided through API's and Software Development List (SDKs).

Integration testing approaches –

There are four types of integration testing approaches. Those approaches are the following:

1. Big-Bang Integration Testing–

- It is the simplest integration testing approach, where all the modules are combining and verifying the functionality after the completion of individual module testing.

- In simple words, all the modules of the system are simply put together and tested.
- This approach is practicable only for very small systems. If once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated.
- So, debugging errors reported during big bang integration testing are very expensive to fix.

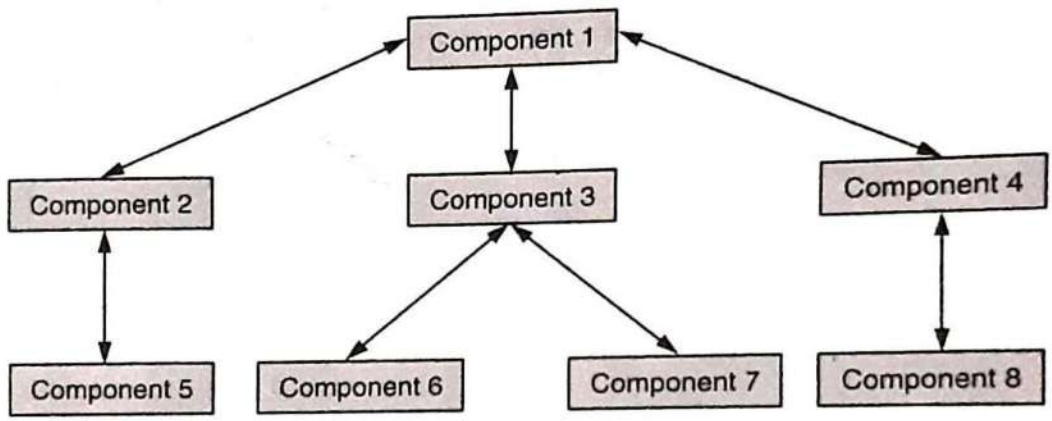
Advantages:

- It is convenient for small systems.

Disadvantages:

- There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
- High risk critical modules are not isolated and tested on priority since all modules are tested at once.

2. Top-Down Integration:



- Integration testing involves testing the topmost component interface with other components in same order as you navigate from top to bottom, till you cover all the components.

- To understand this methodology, Let us assume a new product or software development where components become available one after another in the order

of component numbers specified. The integration starts with testing the interface between Component 1 and Component 2 to complete the integration testing all interfaces mentioned in the above figure covering all the arrows, have to be tested together.

- Top-down integration testing is an integration testing technique used in order to simulate the behavior of the lower-level modules that are not yet integrated. Stubs are the modules that act as temporary replacement for a called module and give the same output as that of the actual product.
- The replacement for the 'called' modules is known as 'Stubs' and is also used when the software needs to interact with an external system.

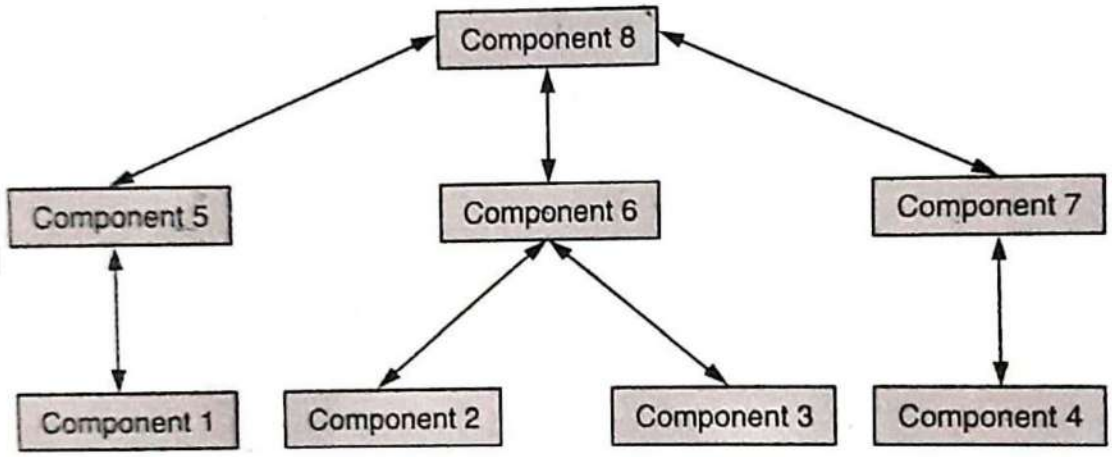
Advantages:

- Separately debugged module.
- Few or no drivers needed.
- It is more stable and accurate at the aggregate level.

Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.

3. Bottom-Up Integration:



- Bottom-up integration is just the opposite of top-down integration, where the components for a new product development become available in reverse order, starting from the bottom.

- Double arrows indicate both the logical flow of component and integration approach. Logic flow is from top to bottom and integration path is from bottom to top.
- The main difference between top-down approach and bottom-up approach is The arrows from top to bottom (that is downward pointing arrows) indicates interaction on control flow. The arrows from bottom to top (that is upward pointing arrows).
- It may be easy to say that top-down integration approach is best suited for the waterfall and V model and the bottom-up approach iterative(spiral) and agile methodology.

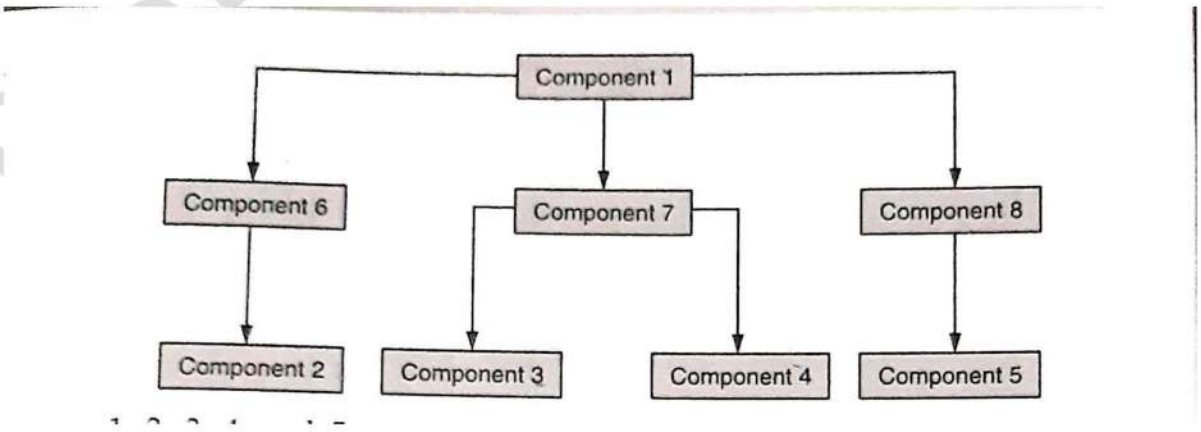
Advantages:

- In bottom-up testing, no stubs are required.
- A principal advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.

Disadvantages:

- Driver modules must be produced.
- In this testing, the complexity that occurs when the system is made up of a large number of small subsystems.

4. Bi –Directional Integration



- Bi directional integration is a combination of the top down and bottom-up integration approaches used together to derive integration steps.
- Let us assume the software components become available in the order mentioned by the component numbers. The individual components 1, 2, 3, 4, and 5 are tested separately and bidirectional integration is performed initially with the use of stubs and drivers.

Stubs: Stubs are used in top-down testing approach.

Drivers: Drivers are used in bottom-up approach.

- A driver is a function which redirects the requests to some other component and stubs simulates the behavior of the missing component. After the functionality of these integrated components are tested, the drivers and stubs are discarded.
- Once components 6, 7, 8 are become available the integration methodology then focuses only on those components, as these are the components which need focus and are new. This approach is also called as **"Sandwich integration".**

Advantages:

1. Sandwich approach is useful for very large projects having several subprojects.
2. Both Top-down and Bottom-up approach starts at a time as per development schedule.
3. Units are tested and brought together to make a system Integration is done downwards.

Disadvantages:

1. It requires very high cost for testing because one part has Top-down approach while another part has bottom-up approach.
2. It cannot be used for smaller system with huge interdependence between different modules. It makes sense when the individual subsystem is as good as complete system.

5. Defect Bash Testing:

- Defect bash testing is an ad-hoc testing where people performing different roles in an organization test the product together at the same time.
- This is very popular among application development companies, where the product can be used by people who perform different roles.
- The testing by all the participants during defect bashing is not based on written test cases. What is to be tested is left to an individual's decision and creativity. This

is usually done when the software is close to being ready to release.

Ad hoc Testing is an informal or unstructured software testing type that aims to break the testing process in order to find possible defects or errors at an early possible stage.

Defect bash involves several steps:

1. Choosing the frequency and duration of defect bash.

Defect bash is an activity involving a large amount of effort and an activity involving huge planning. Frequent defect bashes will incur low return on investment, and too few defect bashes may not meet the objective of finding all defects. Duration also an important factor.

2. Selecting the Right Product Build

Since the defect bash involves a large number of people, effort and planning, a good quality build is needed for defect bash.

3. Communicating the Objective of Defect Bash

Even though defect bash is an ad-hoc activity, its purpose and objective have to be very clear. Since defect bash involves people performing different roles, the contribution they make has to be focused towards meeting the purpose and objective of defect bash. The main objective should be to find large number of uncovered defects or finding out system requirements or finding the random defects.

4. Setting Up and Monitoring the Lab

Since defect bashes are planned, short term and resource intensive activities, it makes sense to setup and monitor laboratory for this purpose. Finding out the right product configuration, resources are activities that have to be planned carefully before a bash actually starts. Since the effort involved more.

5. Taking actions and fixing issues

It is the last step, is to take the necessary corrective action after the defect bash. Getting large number of defects from users is the purpose and also the normal end result from the defect bash. Many defects could be duplicate defects. In this step the developer will fix the issues.

6. Optimizing the effort involved in defect bash

Since a defect bash involves a large number of people, spending much effort is normal for conducting defect bashes. There are several ways to optimize the effort involved in defect bash if a record of objectives and outcome is kept.

System and Acceptance Testing:

- The testing conducted on the complete integrated products and solution to evaluate system compliance with specified requirements on functional and non- functional aspects is called system testing.
- The purpose of acceptance test is to evaluate the system's compliance with the business requirements and calculate whether it is acceptable at the user end.

Functional vs. Non-Functional Testing

Functional Testing	Non-Functional Testing
Execution is performed before non-functional testing.	Execution is performed after the functional testing.
Focus area is based on customer requirements.	Focus area is based on customer expectation.
It is easy to define functional requirements.	It is difficult to define the requirements for non-functional testing.
Helps to validate the behavior of the application.	Helps to validate the performance of the application.
Carried out to validate software actions.	It is done to validate the performance of the software.
It describes what the product does.	It describes how the product works.
Easy to use black box testing	Easy to execute white box testing.
Test cases are repeated many times	Repeated only in case of failures.
Failures normally due to code	Failures normally due to architecture, design, and code
Testing focus on defect detection	Testing focus on qualification of product.

It tests product behavior	It tests product behavior and experience
It involves product feature and functionality.	It involves quality factors.

Functional System Testing:

- Functional testing is performed at various testing phases, there are two obvious problems. One is duplication and other one is gray area. Duplication refers to the same tests being performed multiple times. Gray area refers to certain tests being missed out in all the phases. A small percentage of duplication across phases is unavoidable as different teams are involved.
- Gray areas in testing happen due to lack of product knowledge, lack of knowledge of customer usage, and lack of coordination across test teams.
- There are multiple ways system functional testing is performed. There are also many ways product level test cases are derived for functional testing. Some of the common techniques are:
  - Design/ architecture verification
  - Business vertical testing
  - Deployment testing
  - Beta testing
  - Certification, standards, and testing for compliance.

Non-Functional Testing

- Non-functional testing is similar to that of functional testing but differs from the aspects of complexity, knowledge requirement, effort needed, and number of times the test cases are repeated.
- Since repeating non-functional test cases involves more time, effort, and resources, the process for non-functional testing has to be more robust stronger than functional testing to minimize the need for repetition.