UNIT IV –LIFE CYCLE BASED TESTING

Life Cycle Based Traditional Waterfall Testing, Testing in Iterative Life Cycles, Agile Testing, Agile Model–Driven Development - Model-Based testing: Testing Based on Models - Integration Testing: Decomposition-Based Integration, Call Graph–Based Integration.

Integration Testing :

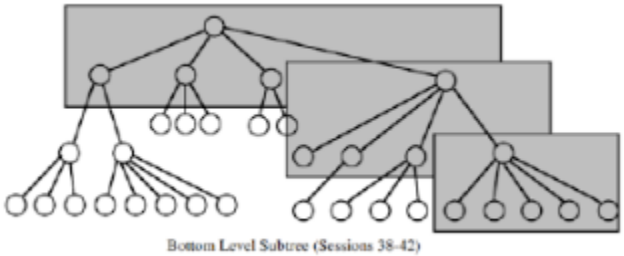
The goal of decomposition-based integration is to test the interfaces among separately tested units. In this view of integration, all the units are compiled together and tested at once.

Four choices are available:

1) top down integration
2) bottom up integration
3) sandwich integration
4) big bang integration

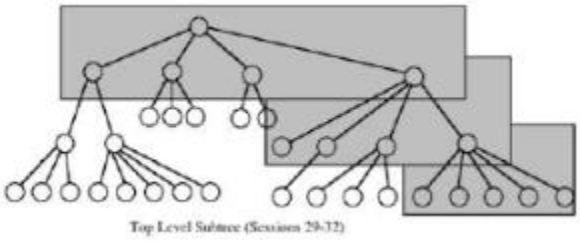
Top down

In this Top Down approach the higher level modules are tested first after higher level modules the lower level modules are tested.Then these modules undergo for integration accordingly.Here the higher level modules refers to main module and lower level modules refers to submodules.



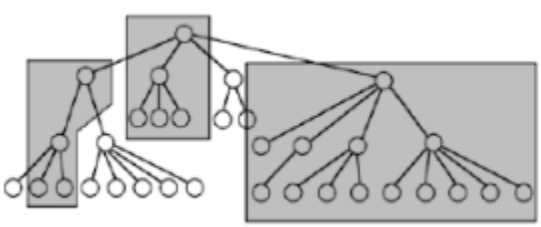
Bottom up

In this Bottom Up approach the lower level modules are tested first after lower level modules the higher level modules are tested.Then these modules undergo integration accordingly.Here the lower level modules refers to submodules and higher level modules refers to main modules.



Sandwich

Sandwich Testing is the combination of the bottom-up approach and top-down approach, so it uses the advantage of both the bottom-up approach and top-down approach.Initially, it uses the stubs and drivers where stubs simulate the behavior of a missing component. It is also known as Hybrid Integration Testing.



Call Graph-Based Integration Testing

- In integration testing, we need to test the interactions between different program units.
- A call graph can be used to represent these interactions.
- The basic idea is to use a call graph instead of a decomposition tree.
- A call graph shows which program unit calls which other program unit.

Call Graph

A **call graph** is a directed, labeled graph that represents the calling relationships among program units.

Components of a Call Graph

a) Vertices / Nodes

- Each **vertex (node)** represents a program unit.
- A program unit may be:
  - Function
  - Method
  - Procedure
  - Module

b) Directed Edges

- A directed edge represents a **calling relationship**.
- The arrow goes from the **calling unit** to the **called unit**.

c) Labeled Graph

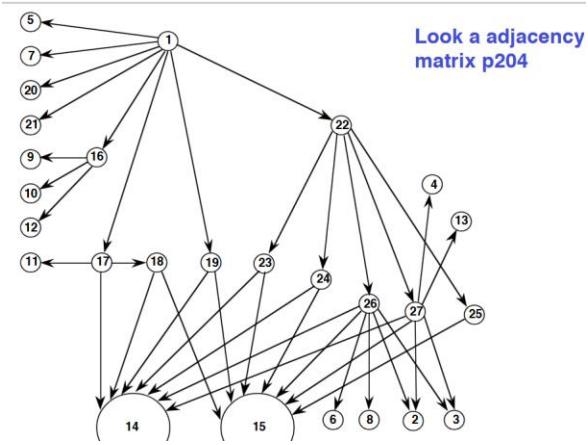
- A call graph is a **directed, labeled graph**.
- Labels can be used to describe the relationship between program units.

d)Adjacency Matrix

- An **adjacency matrix** is another way of representing a call graph.

- It uses a table to show whether one program unit calls another.
- Usually:
    - **1** → a call exists
    - **0** → no call exists

SATM call graph example

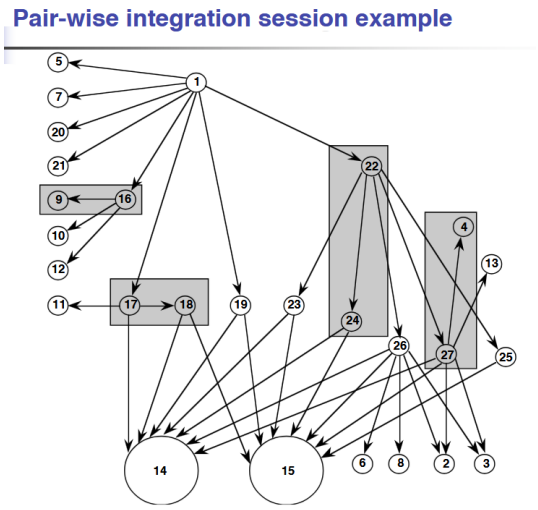


Two types of call graph based integration testing

- ❖ Pair-wise Integration Testing
- ❖ Neighborhood Integration Testing

Pair-wise Integration Testing :

- ❖ Pair-Wise Integration Testing is an integration testing technique in which two program units are tested together at a time. The main idea is to eliminate the need for developing stubs and drivers by using the actual code of the program units.
- ❖ This approach helps avoid the testing process becoming a Big-Bang integration strategy, where all units are integrated and tested at once. In pair-wise integration testing, each testing session is restricted to a pair of connected units in the call graph.
- ❖ For example, if unit A calls unit B, then A and B are tested together as one pair. The testing continues by selecting another connected pair from the call graph. Thus, each edge in the call graph results in one integration test session. This approach helps test the interactions between program units in a controlled and systematic manner.



Neighborhood Integration Testing

The neighborhood of a node in a graph is the set of nodes that are one edge away from the given node. In a directed graph, the neighborhood of a node includes both its immediate predecessor nodes and its immediate successor nodes. A predecessor is a node that directly calls the given node, while a successor is a node that is directly called by the given node.

Neighborhood Integration Testing is an integration testing approach in which a program unit is tested together with the units in its neighborhood. Instead of testing only one pair of units at a time, several directly connected units can be included in the same testing session. This reduces the number of test sessions compared with pair-wise integration testing. However, because several units are tested together, fault isolation becomes more difficult. If a test fails, it may be harder to identify exactly which interaction or program unit caused the fault.

