Concept of Integration Testing

Contains:

- Integration Testing
- Interface Errors
- System Integration Techniques

Introduction to Integration Testing

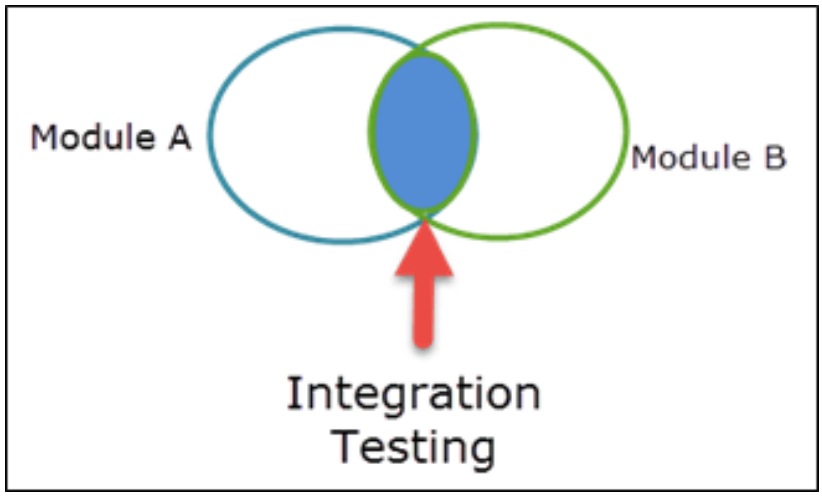
Definition: Integration testing is the phase where individual modules are combined to form a system and tested together.

Objective: Ensure that modules work together seamlessly and identify interface errors.

Key Elements:

Modules are self-contained software components.

System = Collection of modules working together.



Importance of Integration Testing

Reasons:

Developer Discrepancies: Different developers often create modules leading to interface errors.

Limitations of Unit Testing: Unit tests focus on individual modules, not interactions.

Complexity of Modules: Some modules are error-prone due to complexity.

Phases in Building a System

Unit Testing: Focuses on individual module functionality.

Integration Testing: Combines and tests modules together.

System Testing: Comprehensive testing in the actual environment.

Objective of Integration Testing

Goal: Assemble a stable system in a controlled environment.

Testing Phases:

Incrementally integrate modules.

Address interface errors and ensure all modules work correctly.

Ensure system withstands full-blown system testing.

Types of Interfaces in Systems

Common Paradigms:

Procedure Call Interface: One module calls another procedure.

Shared Memory Interface: Two modules share a memory block.

Message Passing Interface: Modules communicate via messages in client-server systems.

Interface Errors

Common Causes:

Construction Errors: Misuse of interface specifications (e.g., incorrect header files).

Inadequate Functionality: One module assumes functionality that another doesn't provide.

Location of Functionality: Misunderstanding where certain functionalities reside.

Changes in Functionality: Modifying one module affects others without proper adjustments.

Types of Interface Errors

Additional Errors:

Misuse of Interface: Wrong parameters or usage in procedure-call interfaces.

Data Structure Alteration: Incorrect data structure sizes leading to failures.

Timing Issues: Synchronization errors in multi-threaded environments.

Advantages of System Integration Testing

Early Detection: Identifies defects early in the process.

Easier Fixes: Earlier issues are easier to fix.

Feedback: Provides feedback on the health of individual modules and the overall system.

Granularity in System Integration Testing

Levels of Granularity:

Intrasystem Testing: Low-level integration of individual modules.

Intersystem Testing: High-level integration of independent systems.

Pairwise Testing: Testing interaction between two interconnected systems.

System Integration Testing Techniques

Approaches:

Incremental Testing: Add modules incrementally and test them.

Top-Down Testing: Start from the top module and integrate downwards.

Bottom-Up Testing: Start from lower modules and build upwards.

Sandwich Testing: Combine top-down and bottom-up approaches.

Big Bang Testing: All modules integrated and tested at once.

Incremental Integration Testing

Definition: Modules are integrated and tested incrementally through multiple cycles.

Approach: Start with core modules and add layers incrementally.

Benefits:

Ensures a self-contained, stable system at each step.

Easier to fix defects as they arise.

Key Points

Summary: Integration testing is vital for ensuring that individual modules work together as a cohesive system.

Key Takeaways:

Interface errors are common and require focused testing.

Incremental and structured testing approaches help detect issues early and ensure system stability.

System integration testing is crucial for the success of large software systems.

❖ Granularity of System Integration Testing

- System integration testing is performed at different levels of granularity. Integration testing includes both white- and black-box testing approaches.

Concept of intrasystem testing, intersystem testing, and pairwise testing

Intrasystem Testing

- **Definition**: Low-level integration testing to combine modules into a cohesive system.
- **Example**: Client-server-based system where both client and server are constructed individually before being tested together.
- **Process**: Incremental module construction.
- **Source of Test Cases**: Low-level design document detailing module specifications.

Intersystem Testing

- **Definition**: High-level testing phase where independently tested systems are connected for end-to-end testing.
- **Goal**: Ensure system interaction works together without full comprehensive testing.
- **Example**: Client-server system integration or a call control system with a billing system in a network.
- **Test Cases**: Derived from the high-level design document detailing system architecture.

Pairwise Testing

- **Definition**: Intermediate level of integration testing where only two interconnected systems are tested.
- **Purpose**: Ensure two systems work together while assuming other systems function as expected.
- **Considerations**: Requires a simple and stable network infrastructure.
- **Challenges**: Unintended side effects, such as triggering a high volume of traps during communication testing between network elements.

- **System Integration Techniques**
  Overview of Incremental, Top-Down, Bottom-Up, Sandwich, and Big Bang Approaches

Objectives of Integration Testing

- Combine software modules into a working system
- Perform system-level tests on the complete system
- Integration testing can start once modules are ready
- Focus on the integration of modules, not just waiting for completion

Integration Approaches Overview

Common approaches to system integration:

- **Incremental**
- **Top-Down**
- **Bottom-Up**
- **Sandwich**
- **Big Bang**

Incremental Integration

- Incremental approach uses multiple test cycles to integrate modules
- Modules are added in layers starting with core modules
- Each cycle involves testing and fixing errors

- Constructing a build involves:
  - Gathering, compiling, and linking modules
  - Version control and automation of build process

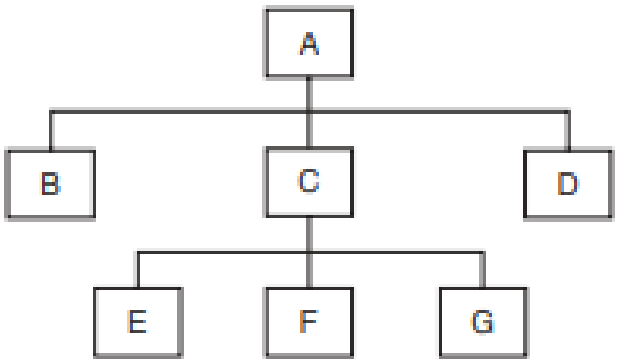
Incremental Integration - Build Process

- **Daily Build**:
  - Facilitates faster delivery through small incremental testing
  - Regression testing from build to build
  - Use of automated, reusable test cases
- **Check-in Request Form**:
  - Tracks new or fixed modules being integrated
  - Ensures unit tests are completed before integration

Top-Down Integration

- Works well with hierarchical systems
- Starts with top-level modules, integrating downward
- **Testing steps:**

    - Start with the top-level module and stubs for lower-level modules
    - Replace stubs with actual modules in successive steps
    - Perform interface tests and regression tests
- **Advantages**:
    - Early observation of system functions
    - Easy to isolate interface errors
    - Reusable test cases
- **Limitations**:
    - Difficult to test system functions early with stubs
    - Designing tests for distant stubs is challenging



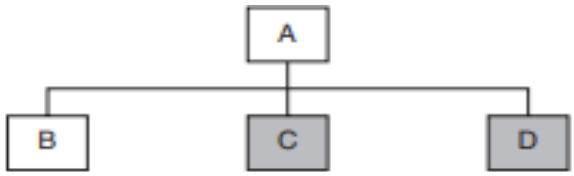


Figure 7.2 Top-down integration of modules A and B.

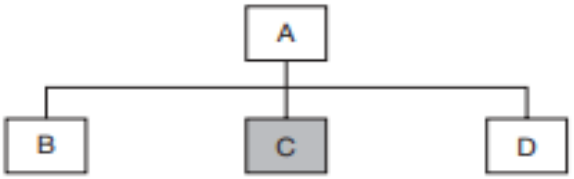


Figure 7.3 Top-down integration of modules A, B, and D.

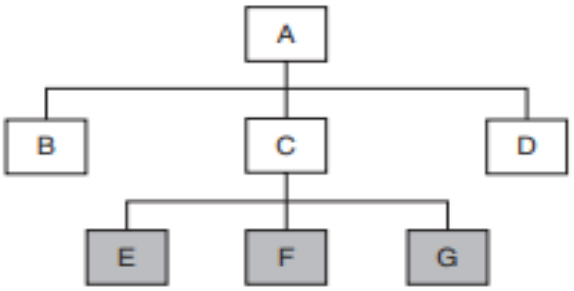


Figure 7.4 Top-down integration of modules A, B, D, and C.

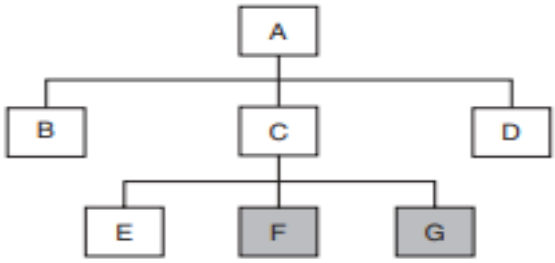


Figure 7.5 Top-down integration of modules A, B, C, D, and E.

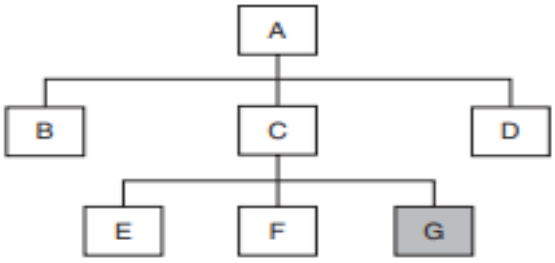


Figure 7.6 Top-down integration of modules A, B, C, D, E, and F.

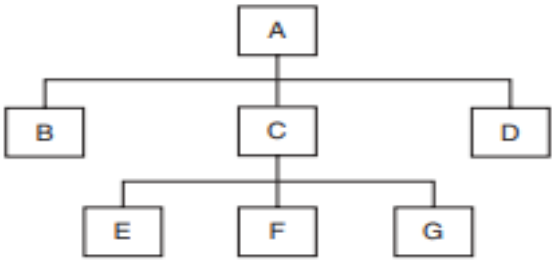


Figure 7.7 Top-down integration of modules A, B, C, D, E, F and G.

Top-Down Integration Example

- Example with module hierarchy: A → B, C, D → E, F, G
  - Initial integration with stubs
  - Progressive replacement of stubs with actual modules
  - Testing interfaces between modules

- Visual diagrams of integration steps (with stubs and actual modules)

Bottom-Up Integration

- Starts with integration of lowest-level modules
- **Test Driver**: Required for integrating lower-level modules
- Modules E, F, G integrated first, then gradually higher-level modules are added
- **Advantages**:
  - Effective if low-level modules are often invoked
  - Avoids the need for stubs for lower-level modules
- **Disadvantages**:
  - System-level functions not observable until top-level modules are integrated
  - Major design flaws may go undetected early

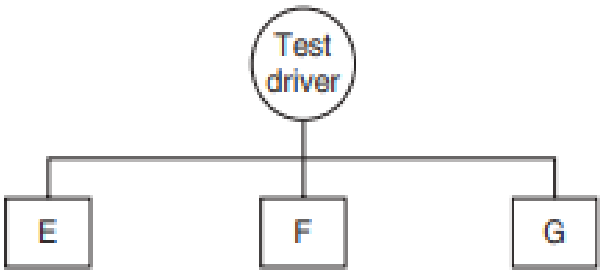


Figure 7.8 Bottom-up integration of modules E, F, and G.

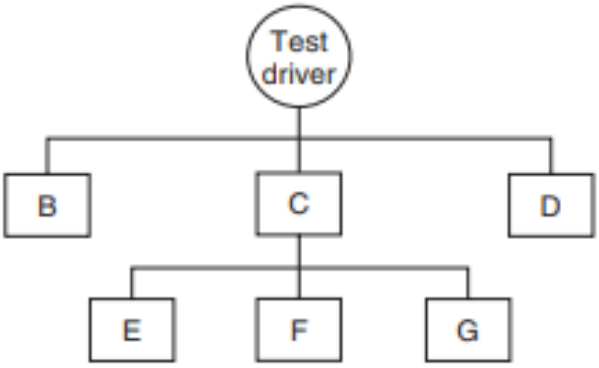


Figure 7.9 Bottom-up integration of modules B, C, and D with E, F, and G.

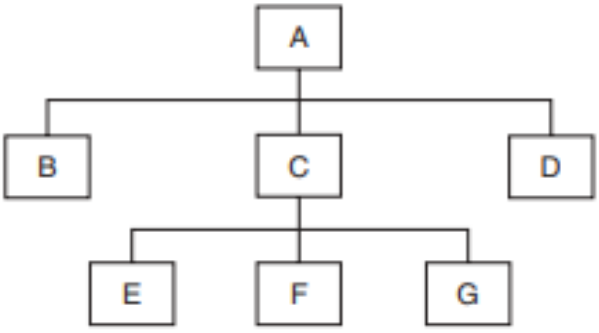


Figure 7.10 Bottom-up integration of module A with all others.

Bottom-Up Integration Example

- Example with module hierarchy: E, F, G → B, C → A
  - Initial integration with test drivers for lower-level modules
  - Gradual replacement of test drivers with actual modules
  - System integration completed once top-level module is integrated
- Visual diagrams of integration process

Top-Down vs. Bottom-Up Comparison

- **Major Design Decisions**:
  - Top-Down: Detects flaws in design decisions early
  - Bottom-Up: Identifies flaws toward the end of the process
- **System-Level Functions**:
  - Top-Down: Observed early
  - Bottom-Up: Observed only at the end
- **Test Case Design**:

    - Top-Down: More difficult with distant stubs
    - Bottom-Up: Easier with simplified test drivers

Sandwich Approach

- Mixes **Top-Down** and **Bottom-Up** approaches
    - Bottom layer: Integrated using Bottom-Up
    - Top layer: Integrated using Top-Down
    - Middle layer: Integrated using Big-Bang (if applicable)
- Benefits: No need for stubs in lower levels
- Aims to combine the best of both approaches

Big Bang Approach

- All modules are individually tested first
- Then, all modules are integrated together at once
- **Disadvantages**:
    - Difficult to pinpoint interface errors in a large system
    - Not cost-effective for large systems with many modules
- Typically used for smaller systems

Key Points

- **Integration Techniques**:
  - Different approaches suit different system types and project needs
  - Incremental testing ensures early error detection
  - Top-Down and Bottom-Up approaches are effective for hierarchical systems
  - Sandwich and Big-Bang are less common but have their specific uses