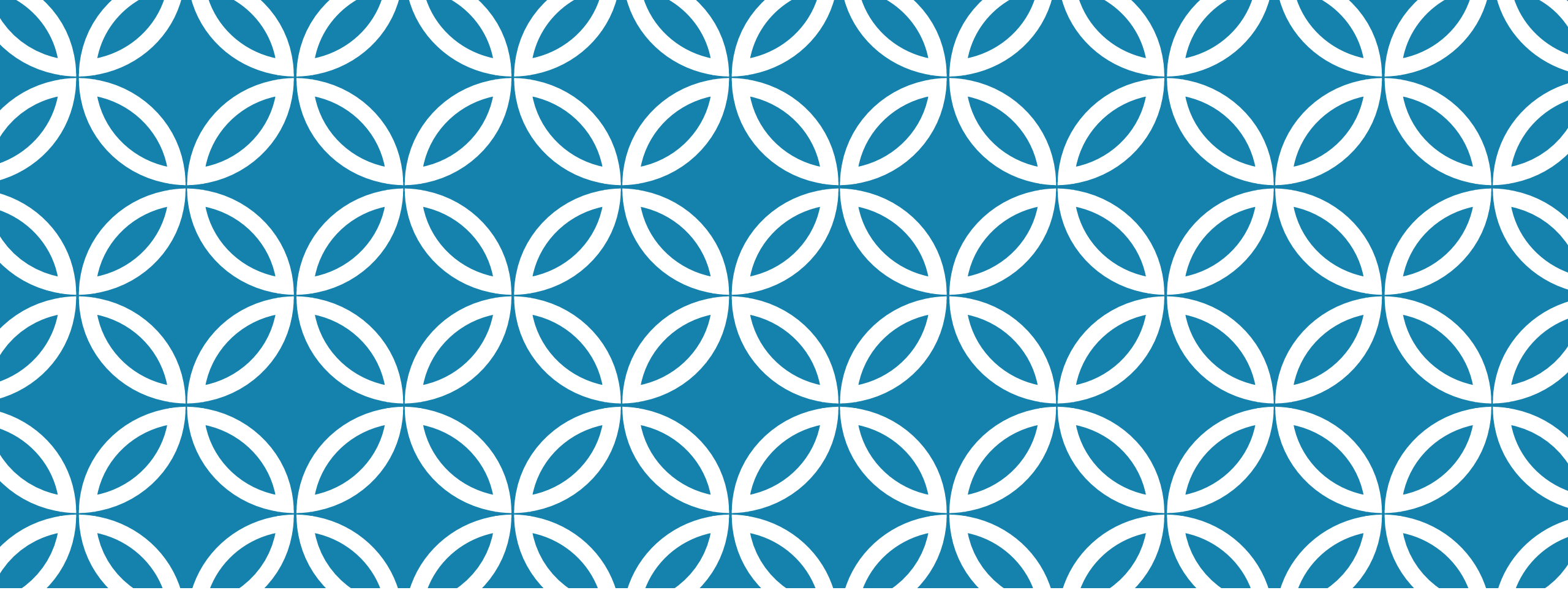


CHAPTER – 7
SYSTEM INTEGRATION TESTING

OUTLINE OF THE CHAPTER

- The Concept of Integration Testing
- Different Types of Interfaces
- Different Types of Interface Errors
- Granularity of System Integration Testing
- System Integration Techniques: Incremental, Top-down, Bottom-up, and Sandwich and Big-bang
- Test Plan for System Integration
- Off-the-shelf Component Testing

THE CONCEPT OF INTEGRATION TESTING

- A software module is a self-contained element of a system
- Modules are individually tested commonly known as *unit testing*
- Next major task is to put the modules, i.e., pieces together to construct the complete system
- Construction of a working system from the pieces is not a straightforward task because of numerous *interface errors*
- The objective of *system integration testing* (SIT) is to build a "working" version of the system
  - Putting modules together in an incremental manner
  - Ensuring that the additional modules work as expected without disturbing the functionalities of the modules already put together
- *Integration testing* is said to be complete when
  - The system is fully integrated together
  - All the test cases have been executed
  - All the severe and moderated defects found have been fixed

THE CONCEPT OF INTEGRATION TESTING

- Interface errors cannot be detected by performing unit testing on modules since unit testing causes computation to happen within a module, whereas interactions are required to happen between modules for interface errors to be detected.

The major advantages of conducting SIT are as follows:

- Defects are detected early
- It is easier to fix defects detected earlier
- We get earlier feedback on the health and acceptability of the individual modules and on the overall system
- Scheduling of defect fixes is flexible, and it can overlap with the development

DIFFERENT TYPES OF INTERFACES

- Modularization is an important principle in software design, and modules are interfaced with other modules to realize the system's functional requirements.
- An interface between two modules allows one module to access the service provided by the other.
- It implements a mechanism for passing control and data between modules.
- Three common paradigms for interfacing modules:
  - Procedure call interface
  - Shared memory interface
  - Message passing interface
- The problem arises when we "put modules together" because of interface errors
- **Interface errors**
  - Interface errors are those that are associated with structures existing outside the local environment of a module, but which the module uses

DIFFERENT TYPES OF INTERFACE ERRORS

1. Construction
2. Inadequate functionality
3. Location of functionality
4. Changes in functionality
5. Added functionality
6. Misuse of interface
7. Misunderstanding of interface
8. Data structure alteration
9. Inadequate error processing
10. Additions to error processing
11. Inadequate post-processing
12. Inadequate interface support
13. Initialization/value errors
14. Validation of data constraints
15. Timing/performance problems
16. Coordination changes
17. Hardware/software interfaces

GRANULARITY OF SYSTEM INTEGRATION TESTING

- System Integration testing is performed at different levels of granularity
- Intra-system testing
  - This form of testing constitutes low-level integration testing with the objective of combining the modules together to build a cohesive system
- Inter-system testing
  - It is a high-level testing phase that requires interfacing independently tested systems
  - For example, Integrating a call control system and a billing system in a telephone network is another example of intersystem testing
- Pairwise testing
  - In pairwise integration, only two interconnected systems in an overall system are tested at a time
  - The purpose of pairwise testing is to ensure that two systems under consideration can function together, assuming that the other systems within the overall environment behave as expected

SYSTEM INTEGRATION TECHNIQUES

- One of the objectives of integration testing is to combine the software modules into a working system so that system-level tests can be performed on the complete system.
- Integration testing need not wait until all the modules of a system are coded and unit tested. Instead, it can begin as soon as the relevant modules are available.
- Common approaches to perform system integration testing
  - Incremental
  - Top-down
  - Bottom-up
  - Sandwich
  - Big-bang

INCREMENTAL

- Integration testing is conducted in an incremental manner as a series of test cycles
- In each test cycle, a few more modules are integrated with an existing and tested build to generated larger builds
- The complete system is built, cycle by cycle until the whole system is operational for system-level testing.
- The number of SIT cycles and the total integration time are determined by the following parameters:
  - Number of modules in the system
  - Relative complexity of the module (cyclomatic complexity)
  - Relative complexity of the interfaces between the modules
  - Number of modules needed to be clustered together in each test cycle
  - Whether the modules to be integrated have boon adequately tested before
  - Turnaround time for each test-debug-fix cycle

INCREMENTAL

- A software image is a compiled software binary
- A build is an interim software image for internal testing within an organization
- Constructing a build is a process by which individual modules are integrated to form an interim software image.
- The final build is a candidate for system testing
- Constructing a software image involves the following activities
  - Gathering the latest unit tested, authorized versions of modules
  - Compiling the source code of those modules
  - Checking in the compiled code to the repository
  - Linking the compiled modules into subassemblies
  - Verifying that the subassemblies are correct
  - Exercising version control

INCREMENTAL

- Creating a daily build is very popular among many organization
- It facilitates to a faster delivery of the system
- It puts emphasis on small incremental testing
- It steadily increases number of test cases
- The system is tested using automated, re-usable test cases
- An effort is made to fix the defects that were found within 24 hours
- Prior version of the build are retained for references and rollback
- A typical practice is to retain the past 7-10 builds

INCREMENTAL

A release note containing the following information accompanies a build.

- What has changed since the last build?
- What outstanding defects have been fixed?
- What are the outstanding defects in the build?
- What new modules, or features, have been added?
- What existing modules, or features, have been enhanced, modified, or deleted?
- Are there any areas where unknown changes may have occurred?

A test strategy is created for each new build and the following issues are addressed while planning a test strategy

- What test cases need to be selected from the SIT test plan?
- What previously failed test cases should now be re-executed in order to test the fixes in the new build?
- How to determine the scope of a partial regression tests?
- What are the estimated time, resource demand, and cost to test this build?

TOP-DOWN

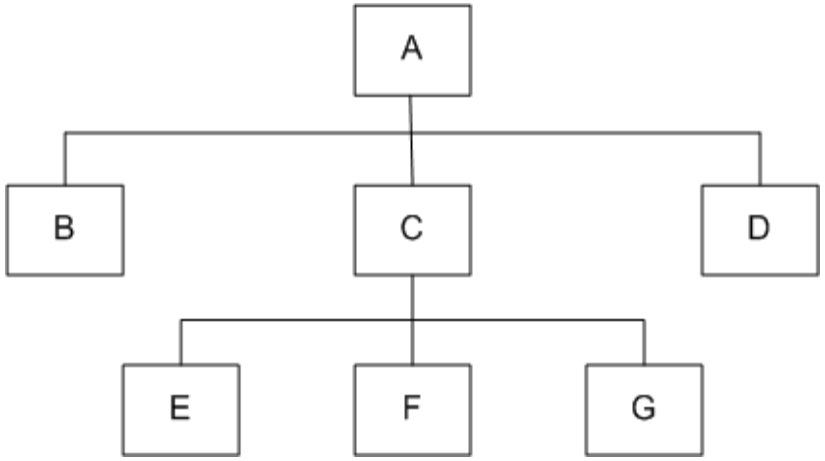


Figure 7.1: A module hierarchy with three levels and seven modules

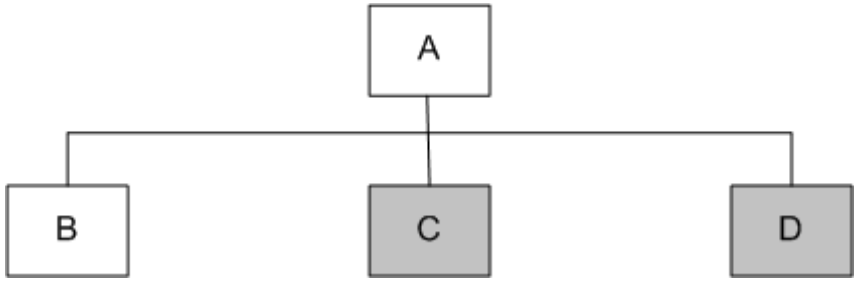


Figure 7.2: Top-down integration of modules A and B

Module A has been decomposed into modules B, C, and D

Modules B, D, E, F, and G are terminal modules

First integrate modules A and B using stubs C` and D` (represented by grey boxes)

Next stub D` has been replaced with its actual instance D

Two kinds of tests are performed:

- Test the interface between A and D
- Regression tests to look for interface defects between A and B in the presence of module D

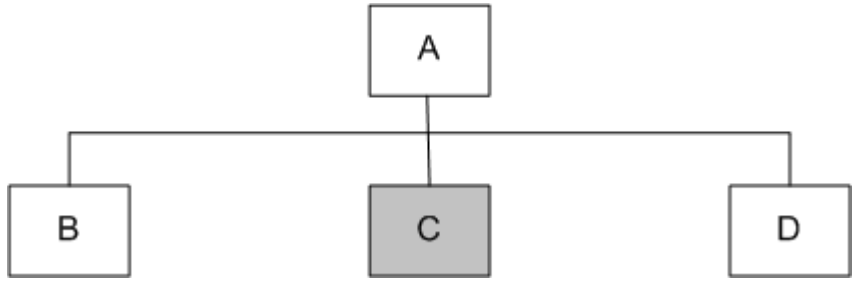


Figure 7.3: Top-down integration of modules A, B and D

TOP-DOWN

Stub C` has been replaced with the actual module C, and new stubs E`, F`, and G`

Perform tests as follows:

- first, test the interface between A and C;
- second, test the combined modules A, B, and D in the presence of C

The rest of the process depicted in the right hand side figures.

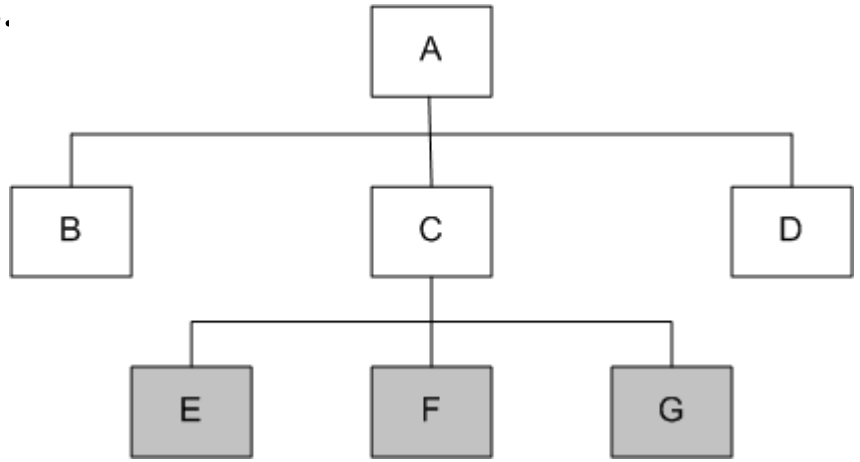


Figure 7.4: Top-down integration of modules A, B, D and C

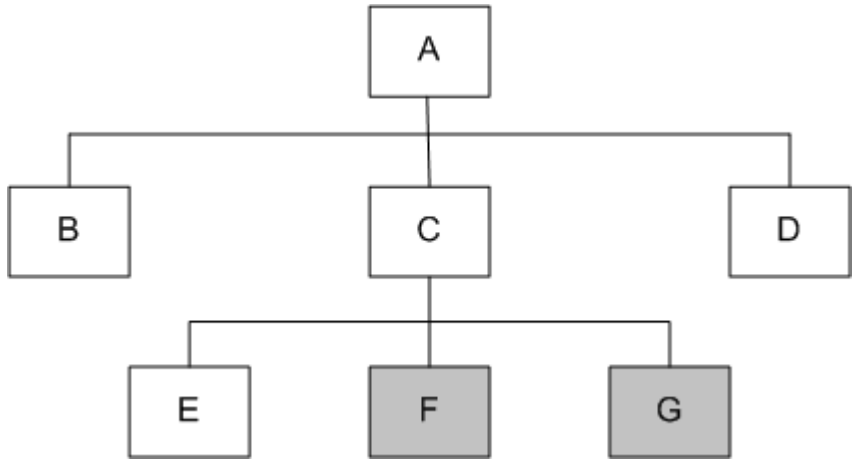


Figure 7.5: Top-down integration of modules A, B, C, D and E

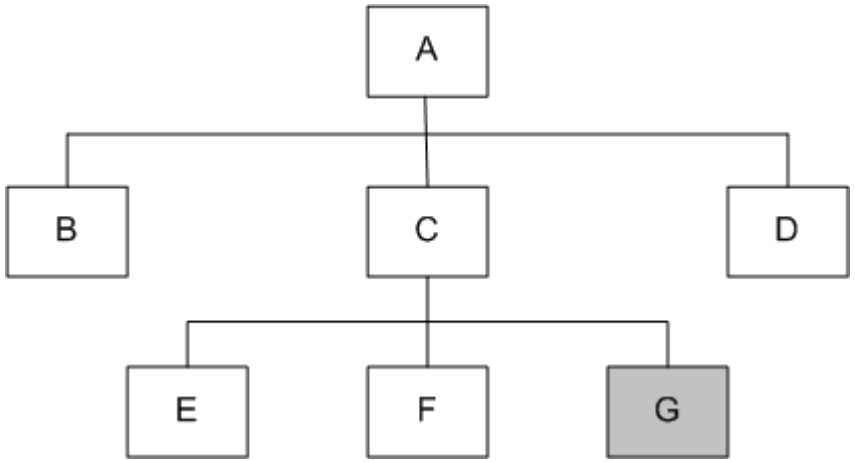


Figure 7.6: Top-down integration of modules A, B, C, D, E and F

TOP-DOWN

- **Advantages**
  - The SIT engineers continually observe system-level functions as the integration process continue
  - Isolation of interface errors becomes easier because of the incremental nature of the top-down integration
  - Test cases designed to test the integration of a module M are reused during the regression tests performed after integrating other modules
- **Disadvantages**
  - It may not be possible to observe meaningful system functions because of the absence of lower-level modules and the presence of stubs.
  - Test case selection and stub design become increasingly difficult when stubs lie far away from the top-level module.

BOTTOM-UP

- We design a test driver to integrate lowest-level modules E, F, and G
- Return values generated by one module are likely to be used in another module
- The test driver mimics module C to integrate E, F, and G in a limited way.
- The test driver is replaced with actual module , i.e., C.
- A new test driver is used
- At this moment, more modules such as B and D are integrated
- The new test driver mimics the behavior of module A
- Finally, the test driver is replaced with module A and further test are performed

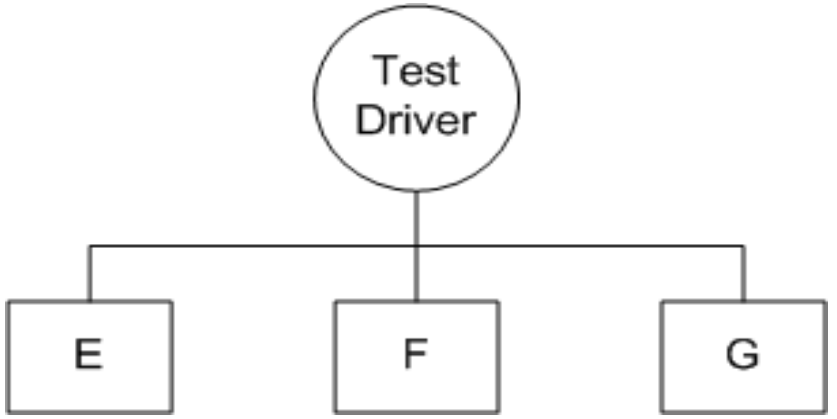


Figure 7.8: Bottom-up integration of module E, F, and G

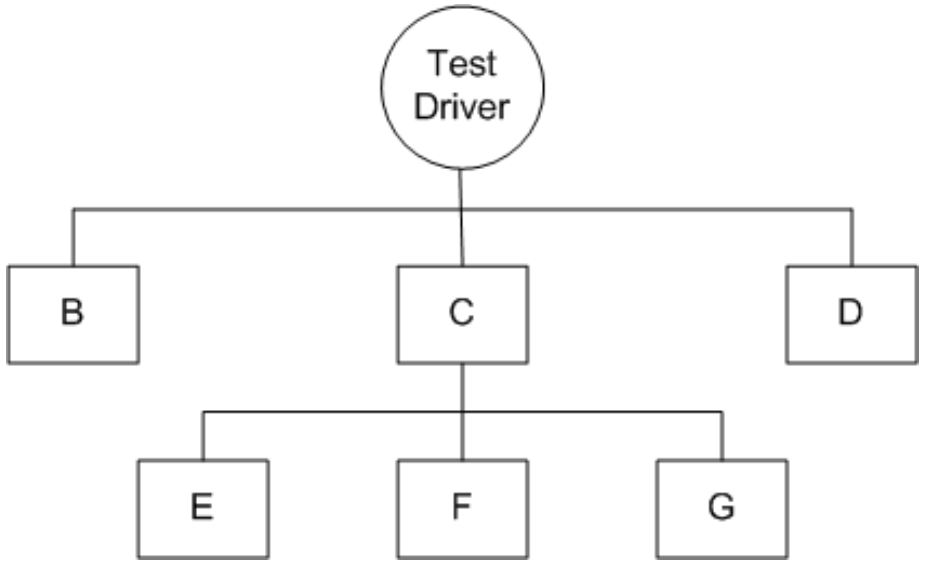


Figure 7.9: Bottom-up integration of module B, C, and D with F, F, and G

BOTTOM-UP

- **Advantages**
  - One designs the behavior of a test driver by simplifying the behavior of the actual module
  - If the low-level modules and their combined functions are often invoked by other modules, then it is more useful to test them first so that meaningful effective integration of other modules can be done
- **Disadvantages**
  - Discovery of major faults are detected towards the end of the integration process, because major design decisions are embodied in the top-level modules
  - Test engineers can not observe system-level functions from a partly integrated system. In fact, they can not observe system-level functions until the top-level test driver is in place

BIG-BANG AND SANDWICH

- **Big-bang Approach**
  - First all the modules are individually tested
  - Next all those modules are put together to construct the entire system which is tested as a whole
- **Sandwich Approach**
  - In this approach a system is integrated using a mix of top-down, bottom-up, and big-bang approaches
  - A hierarchical system is viewed as consisting of three layers
  - The bottom-up approach is applied to integrate the modules in the bottom-layer
  - The top layer modules are integrated by using top-down approach
  - The middle layer is integrated by using the big-bang approach after the top and the bottom layers have been integrated

TEST PLAN FOR SYSTEM INTEGRATION

1. Scope of Testing
2. Structure of the Integration Levels a. Integration test phases b. Modules or subsystems to be integrated in each phase c. Building process and schedule in each phase d. Environment to be set up and resources required in each phase
3. Criteria for Each Integration Test Phase n a. Entry criteria b. Exit criteria c. Integration Techniques to be used d. Test configuration set-up
4. Test Specification for Each Integration Test Phase a. Test case id# b. Input data c. Initial condition d. Expected results e. Test procedure How to execute this test? How to capture and interpret the results?
5. Actual Test Results for Each Integration Test Phase
6. References
7. Appendix

Table 7.3: A framework for System Integration Test (SIT) Plan.

TEST PLAN FOR SYSTEM INTEGRATION

Categories of System Integration Tests:

Interface integrity
- Internal and external interfaces are tested as each module is integrated

Functional validity
- Tests to uncover functional errors in each module after it is integrated

End-to-end validity
- Tests are designed to ensure that a completely integrated system works together from end-to-end

Pairwise validity
- Tests are designed to ensure that any two systems work properly when connected by a network

Interface stress
- Tests are designed to ensure that the interfaces can sustain the load

System endurance
- Tests are designed to ensure that the integrated system stay up for weeks without any crashes

OFF-THE-SELF COMPONENT INTEGRATION

Organization occasionally purchase off-the-self (OTS) components from vendors and integrate them with their own components

Useful set of components that assists in integrating actual components:

Wrapper: It is a piece of code that one builds to isolate the underlying components from other components of the system.

Glue: A glue component provides the functionality to combine different components

Tailoring: Components tailoring refers to the ability to enhance the functionality of a component

- Tailoring is done by adding some elements to a component to enrich it with functionality not provided by the vendor
- Tailoring does not involve modifying the source code of the component
- For example, executing some “script” when some event occurs.

OFF-THE-SHELF COMPONENT TESTING

OTS components produced by the vendor organizations are known as **commercial off-the-shelf** (COTS) components

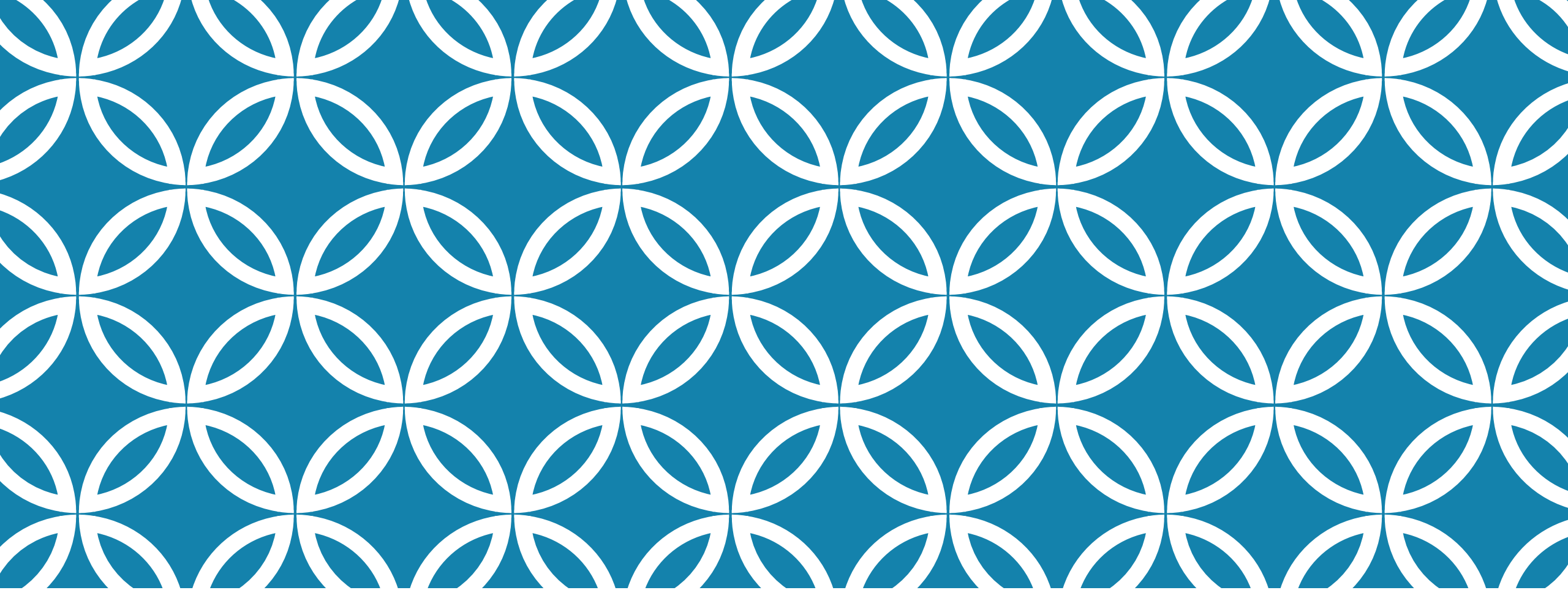
A COTS component is defined as: "*A unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*"

Three types of testing techniques are used to determine the suitability of a COTS component:

Black-box component testing: This is used to determine the quality of the component

System-level fault injection testing: This is used to determine how well a system will tolerate a failing component

Operational system testing: These kinds of tests are used to determine the tolerance of a software system when the COTS component is functioning correctly to determine if the OTS component is a good match for the system.



THANK YOU!!