Software Testing and Quality Assurance

Theory and Practice

System Integration Testing

Dr. Mohammad Ahmad

Outline

- 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
- Software and Hardware Integration
- Hardware Design Verification Tests
- Hardware and Software Compatibility Matrix
- Test Plan for System Integration
- Off-the-self Component Integration
- Off-the-shelf Component Testing
- Built-in 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

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 development

Different Types of Interfaces

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

- Construction
- Inadequate functionality
- Location of functionality
- Changes in functionality
- Added functionality
- Misuse of interface
- Misunderstanding of interface
- Data structure alteration
- Inadequate error processing
- Additions to error processing
- Inadequate post-processing
- Inadequate interface support
- Initialization/value errors
- Validation od data constraints
- Timing/performance problems
- Coordination changes
- 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 which requires interfacing independently tested systems
- 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

Common approaches to perform system integration testing

- Incremental
- Top-down
- Bottom-up
- Sandwich
- Big-bang

Pre-requisite

A module must be available to be integrated

A module is said to *available* for combining with other modules when the module's *check-in request form* is ready

Author:	<Name of the person requesting this check-in>
Today's date:	<yy-mm-dd>
Check-in request date:	<yy-mm-dd>
Category (identify all that apply):	<New Feature: (Y, N); Enhancement: (Y, N); Defect: (Y, N); If yes: Defect numbers: Are any of these major defects: (Y, N) Are any of these moderate defects: (Y, N) >
Short description of check-in:	<Describe in a short paragraph the feature, the enhancement, or the defects fixes to be checked-in.>
Number of files to be checked-in:	<Give the number of files to be checked-in. Include the file names, if possible.>
Code reviewer names:	<Provide the names of the code reviewers.>
Command line interface changes made:	<(Y, N); If yes, were they Documented? (Y, N) Reviewed? (Y, N, Pending)>
Does this check-in involve changes to global header?	<(Y, N); If yes, include the header file names.>
Does this check-in involve changes in output logging?	<(Y, N); If yes, were they documented? (Y, N)>
Unit test description:	<Description of the unit tests conducted>
Comments:	<Any other comments and issues>

Table 7.1: Check-in request form.

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 am 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

- 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 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 deman, and cost to test this build?

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

Top-down

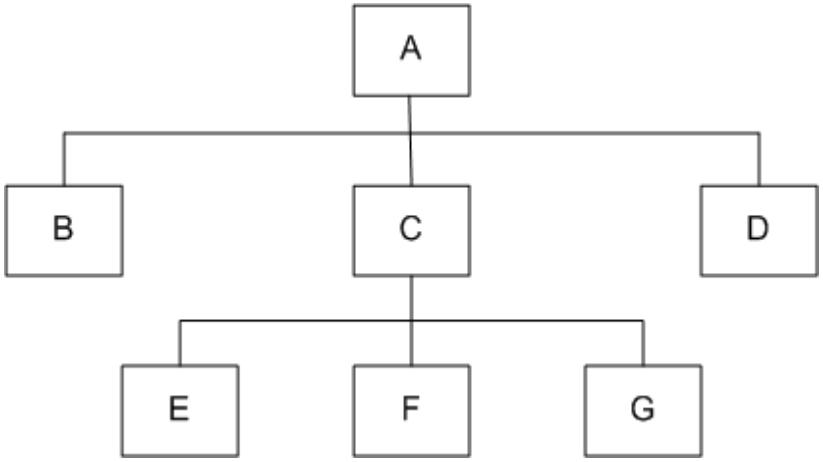

Figure 7.1: A module hierarchy with three levels and seven modules

- 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

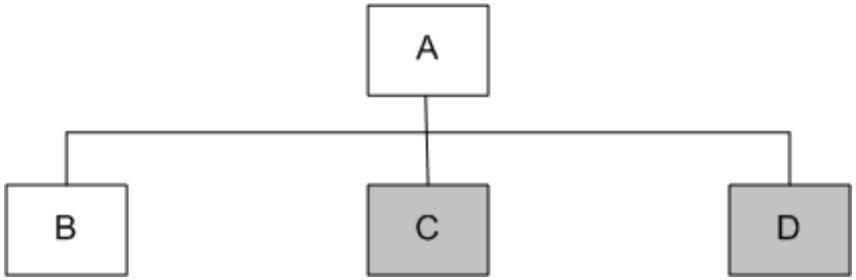

Figure 7.2: Top-down integration of modules A and B

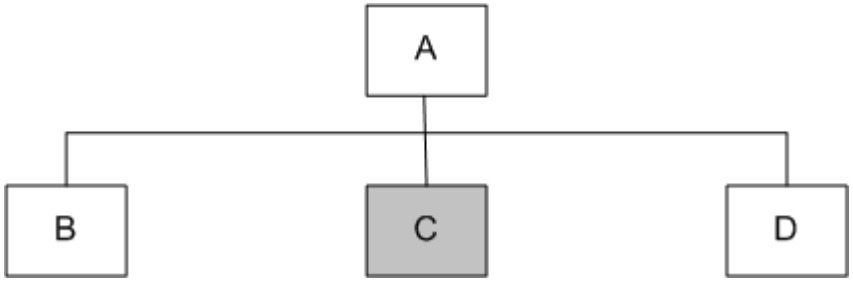

Figure 7.3: Top-down integration of modules A, B and D

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.

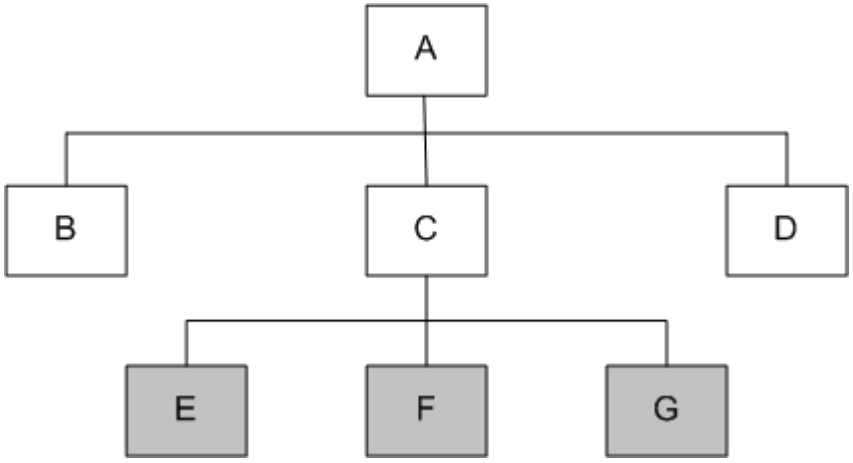

Figure 7.4: Top-down integration of modules A, B, D and C

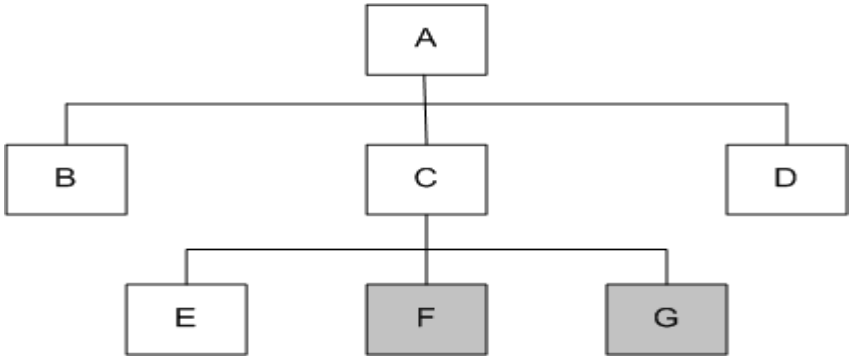

Figure 7.5: Top-down integration of modules A, B, C, D and E

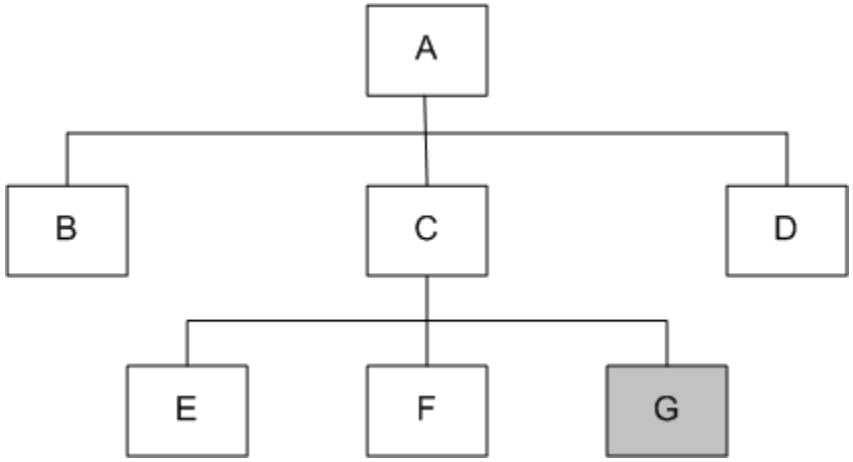

Figure 7.6: Top-down integration of modules A, B, C, D, E and F

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 an 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 is 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

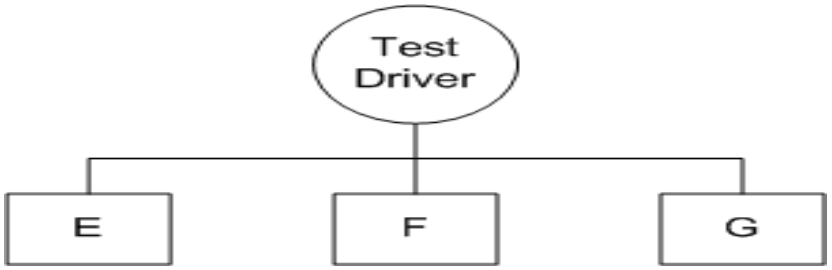

Figure 7.8: Bottom-up integration of module E, F, and G

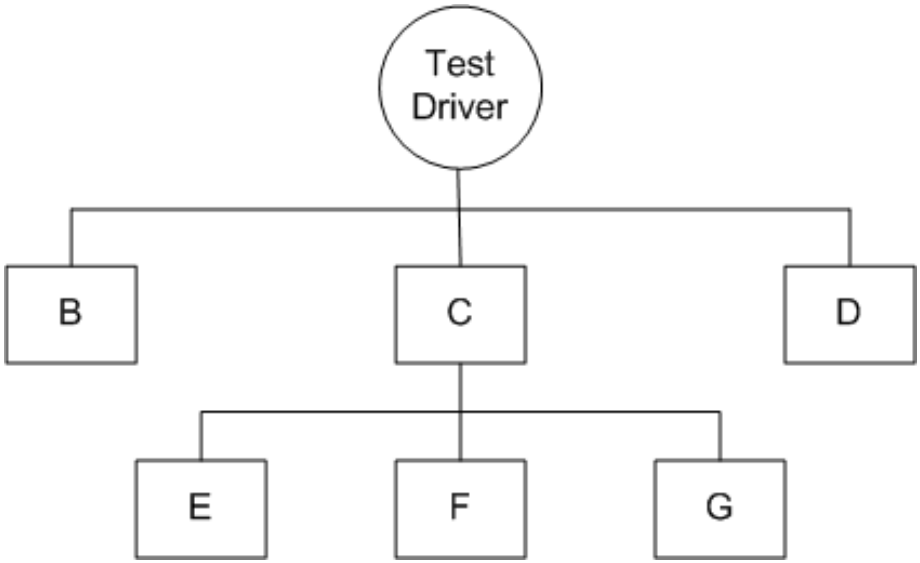

Figure 7.9: Bottom-up integration of module B, C, and D with F, F, and G

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 decision 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

Software and Hardware Integration

- Integration is often done in an iterative manner
- A software image with a minimal number of core modules is loaded on a prototype hardware
- A small number of tests are performed to ensure that all the desired software modules are present in the build
- Next, additional tests are run to verify the essential functionalities
- The process of assembling the build, loading on the target hardware, and testing the build continues until the entire product has been integrated

Hardware Design Verification Tests

A hardware engineering process consists of **four phases**

- Planning and specification
- Design, prototype implementation, and testing
- Integration with the software system
- Manufacturing, distribution and field service

A hardware **Design Verification Test** (DVT) plan is prepared and executed by the hardware group before the integration with software system

Hardware Design Verification Tests

The main hardware tests are as follows:

- Diagnostic Test
- Electrostatic Discharge Test
- Electromagnetic Emission Test
- Electrical Test
- Thermal Test
- Environment Test
- Equipment Handling and Packaging Test
- Acoustic Test
- Safety Test
- Reliability Test

Hardware and Software Compatibility Matrix

- H/W and s/w compatibility information is maintained in the form of a compatibility matrix
- It documents different revisions of the h/w and s/w that will be used for official release of the product
- An Engineering Change Order (ECO) is a formal document that describes a change to the hardware and software
- An ECO document includes the hardware software compatible matrix
- It is distributed to the operation, customer support and sales teams of the organization

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

Entry Criteria
Software functional and design specifications must be written, reviewed and approved.
Code reviewed and approved.
Unit test plan for each module is written, reviewed, and executed.
100% unit tests passed.
All the check-in request form must be completed, submitted, and approved.
Hardware design specification written, reviewed, and approved.
Hardware design verification test written, reviewed, and executed.
100% design verification tests passed.
Hardware/software integration test plan written, reviewed, and executed.
100% hardware/software integration tests passed.

Table 7.4: A framework for system integration entry criteria

Test Plan for System Integration

Exit Criteria
All code completed and frozen and no more modules to be integrated.
100% system integration tests passed.
No major defect is outstanding.
All the moderate defects found in SIT phase have been fixed and re-tested.
Not more than 25 minor defects are outstanding.
Two weeks system uptime in system integration test environment without any anomalies, i.e., crashes.
System integration tests results are documented.

Table 7.5: A framework for system integration exit criteria

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

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 a functionality not provided by the vendor
  - Tailoring does not involve modifying the source code of the component

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 use 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:** This kind of tests are used to determine the tolerance of a software system when the COTS component is functioning correctly

Built-in Testing

- Testability is incorporated into software components

- Testing and maintenance can be self-contained
  - Normal mode
    - The built-in test capabilities are transparent to the component user
    - The component does not differ from other non-built-in testing enabled components
  - Maintenance mode
    - The component user can test the component with the help of its built-in testing features
    - The component user can invoke the respective methods of the component, which execute the test, evaluate autonomously its results, and output the test summary