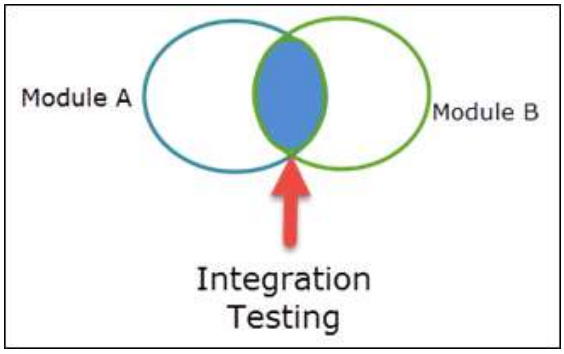
Learning Outcomes

At the end of this module, the student should be able to:

1. Define and describe system integration testing.
2. Understand the need for system integration testing
3. Enumerate the procedures in system integration testing.
4. Differentiate system integration testing with system testing and user acceptance testing.

System Integration Testing

System Integration Testing is defined as a type of software testing carried out in an integrated hardware and software environment to verify the behavior of the complete system. It is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirement. It is the overall testing of the whole system which is composed of many sub-systems. It also verifies a software system's coexistence with others and tests the interface between modules of the software application. In this type of testing, modules are first tested individually and then combined to make a system. The main objective of SIT is to ensure that all software module dependencies are functioning properly and the data integrity is preserved between distinct modules of the whole system.



For Example, software and/or hardware components are combined and tested progressively until the entire system has been integrated.

Need for System Integration Testing

In Software Engineering, System Integration Testing is done because,

- It helps to detect defect early
- Earlier feedback on the acceptability of the individual module will be available
- Scheduling of defect fixes is flexible, and it can be overlapped with development
- Correct data flow
- Correct control flow
- Correct timing
- Correct memory usage
- Correct with software requirements

The Granularity of SIT

SIT can be conducted at three different levels of granularity:
(i) Intra-System Testing: This is a low level of integration testing that aims at fusing the modules together to build a unified system.
(ii) Inter-System Testing: This is high-level testing that needs interfacing independently tested systems.
(iii) Pairwise Testing: Here, only two inter-connected subsystems in the whole system are tested at a time. This aims at ensuring that the two sub-systems can function well when combined together presuming that the other sub-systems are already working fine.

Procedures of System Integration Testing

It's a systematic technique for constructing the program structure while conducting tests to uncover errors associated with interfacing.

- All modules are integrated in advance, and the entire program is tested as a whole. But during this process, a set of errors is likely to be encountered.
- Correction of such errors is difficult because isolation causes is complicated by the vast expansion of the entire program. Once these errors are rectified and corrected, a new one will appear, and the process continues seamlessly in an endless loop. To avoid this situation, another approach is used, Incremental Integration.
- There are some incremental methods like the integration tests are conducted on a system based on the target processor. The methodology used is Black Box Testing. Either bottom-up or top-down integration can be used.
- Test cases are defined using the high-level software requirements only.

Data-Driven Method

The simplest way to perform SIT is through data-driven method. It requires minimum usage of software testing tools.
First, data exchange (data import and data export) happens between the system components and then the behavior of each data field within the individual layer is examined. Once the software is integrated, there are three main states of data flow as mentioned below:

#1: Data state within the Integration Layer

- ✓ The integration layer acts as an interface between the data import and export. Performing SIT at this layer requires some basic knowledge on certain technology like schema (XSD), XML, WSDL, DTD, and EDI.
- ✓ The performance of data exchange can be examined at this layer through the below steps:
  1. Validate the data properties within this layer against BRD/ FRD/ TRD (Business requirement document/ Functional requirement Document/ Technical requirement document).
  2. Cross check the web service request using XSD and WSDL.
  3. Run some unit tests and validate the data mappings and requests.
  4. Review the middleware logs.

#2: Data state within the Database Layer

✓ Performing SIT at this layer requires a basic knowledge of SQL and stored procedures.
✓ The performance of data exchange at this layer can be examined through the below steps:
1. Check if all the data from the integration layer has reached successfully at the database layer and has been committed.
2. Validate the table and column properties against BRD/ FRD/ TRD.
3. Validate the constraints and data validation rules applied in the database as per business specifications.
4. Check stored procedures for any processing data.
5. Review server logs.

#3: Data state within the Application Layer

✓ SIT can be performed at this layer through the below steps:
1. Check if all the required fields are visible in the UI.
2. Execute some positive and negative test cases and validate the data properties.

Note: There can be a lot of combinations corresponding to data import and data export. You will need to execute SIT for the best combinations considering the time available to you.

Entry and Exit Criteria for Integration Testing

Usually while performing Integration Testing, ETVX (Entry Criteria, Task, Validation, and Exit Criteria) strategy is used.

Entry Criteria: Completion of Unit Testing

Inputs:

- Software Requirements Data
- Software Design Document
- Software Verification Plan
- Software Integration Documents

Activities:

- Based on the High and Low-level requirements create test cases and procedures
- Combine low-level modules builds that implement a common functionality
- Develop a test harness
- Test the build
- Once the test is passed, the build is combined with other builds and tested until the system is integrated as a whole.
- Re-execute all the tests on the target processor-based platform, and obtain the results

Exit Criteria:

❖ Successful completion of the integration of the Software module on the target Hardware
❖ Correct performance of the software according to the requirements specified

Outputs:

- ✓ Integration test reports
- ✓ Software Test Cases and Procedures [SVCP].

Hardware Software Integration Testing

Hardware Software Integration Testing is a process of testing Computer Software Components (CSC) for high-level functionalities on the target hardware environment. The goal of hardware/software integration testing is to test the behavior of developed software integrated on the hardware component.

Requirement based Hardware-Software Integration Testing

The aim of requirements-based hardware/software integration testing is to make sure that the software in the target computer will satisfy the high-level requirements.

Typical errors revealed by this testing method includes:

- Hardware/software interfaces errors
- Violations of software partitioning.
- Inability to detect failures by built-in test
- Incorrect response to hardware failures
- Error due to sequencing, transient input loads and input power transients
- Feedback loops incorrect behaviour
- Incorrect or improper control of memory management hardware
- Data bus contention problem
- Incorrect operation of mechanism to verify the compatibility and correctness of field loadable software

Hardware and software Integration deals with the verification of the high-level requirements. All tests at this level are conducted on the target hardware.

- Black box testing is the primary testing methodology used at this level of testing.
- Define test cases from the high-level requirements only
- A test must be executed on production standard hardware (on target)

Things to consider when designing test cases for HW/SW Integration:

- Correct acquisition of all data by the software
- Scaling and range of data as expected from hardware to software
- Correct output of data from software to hardware
- Data within specifications (normal range)
- Data outside specifications (abnormal range)
- Boundary data
- Interrupts processing
- Timing

- Correct memory usage (addressing, overlaps, etc.)
- State transitions

System Testing VS System Integration Testing

Differences between System Testing and SIT:

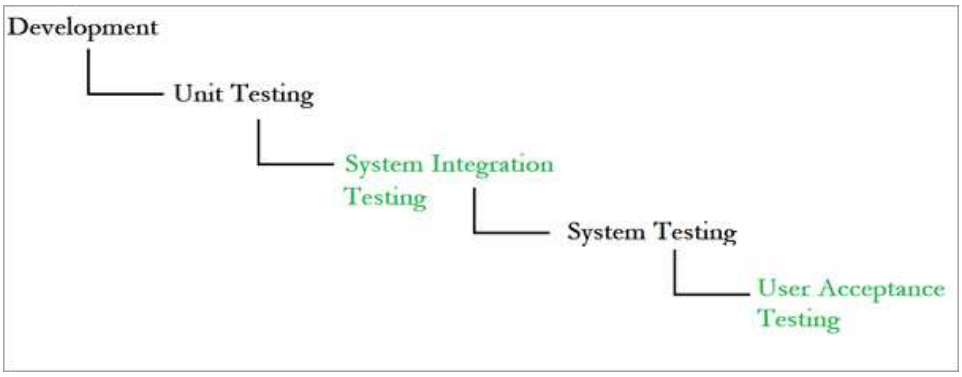
SIT (System Integration Testing)	System Testing
SIT is mainly done to check how individual modules interact with each other when integrated into a system as a whole.	System testing is mainly done to check if the whole system is working as expected with reference to the specified requirements.
It is conducted after unit testing and will be done each time when a new module is added to the system.	It is conducted at the final level i.e. after the completion of integration testing and just before delivering the system for UAT.
It is a low-level testing.	It is a high-level testing.
SIT test cases focus on the interface between the system components.	Test cases, in this case, focus on simulating the real-life scenarios.

System Integration Testing VS User Acceptance Testing

Here is the difference between SIT and UAT:

SIT (System Integration Testing)	UAT (User Acceptance Testing)
This testing is from the perspective of interfacing between modules.	This testing is from the perspective of user requirements.
SIT is done by developers and testers.	UAT is done by customers and end users.
Done after unit testing and before system testing.	This is the last level of testing and is done after system testing.
Generally, the issues found in SIT would be related to data flow, control flow, etc.	The issues found in UAT would generally be like the features that are not working as per the user requirements.

Levels of Testing (flow from Unit testing to UAT)

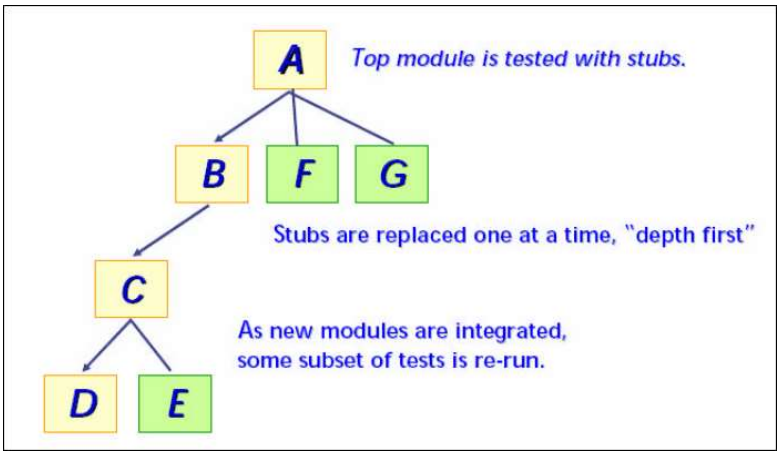


Incremental Approach of System Integration Testing

Incremental testing is a way of integration testing. In this type of testing method, you first test each module of the software individually and then continue testing by appending other modules to it then another and so on.

Top-Down Approach

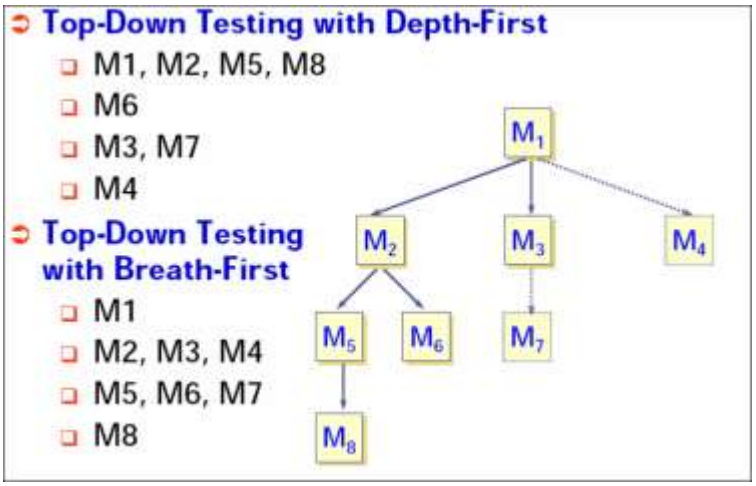
In this type of approach, individual start by testing only the user interface, with the underlying functionality simulated by stubs, then you move downwards integrating lower and lower layers as shown in the image below.



- Starting with the main control module, the modules are integrated by moving downward through the control hierarchy
- Sub-modules to the main control module are incorporated into the structure either in a breadth-first manner or depth-first manner.
- Depth-first integration integrates all modules on a major control path of the structure as displayed in the following diagram:

The module integration process is done in the following manner:

- The main control module is used as a test driver, and the stubs are substituted for all modules directly subordinate to the main control module.
- The subordinate stubs are replaced one at a time with actual modules depending on the approach selected (breadth first or depth first).
- Tests are executed as each module is integrated.
- On completion of each set of tests, another stub is replaced with a real module on completion of each set of tests
- To make sure that new errors have not been introduced Regression Testing may be performed.
- The process continues from step2 until the entire program structure is built. The top-down strategy sounds relatively uncomplicated, but in practice, logistical problems arise.
- The most common of these problems occur when processing at low levels in the hierarchy is required to adequately test upper levels.
- Stubs replace low-level modules at the beginning of top-down testing and, therefore no significant data can flow upward in the program structure.

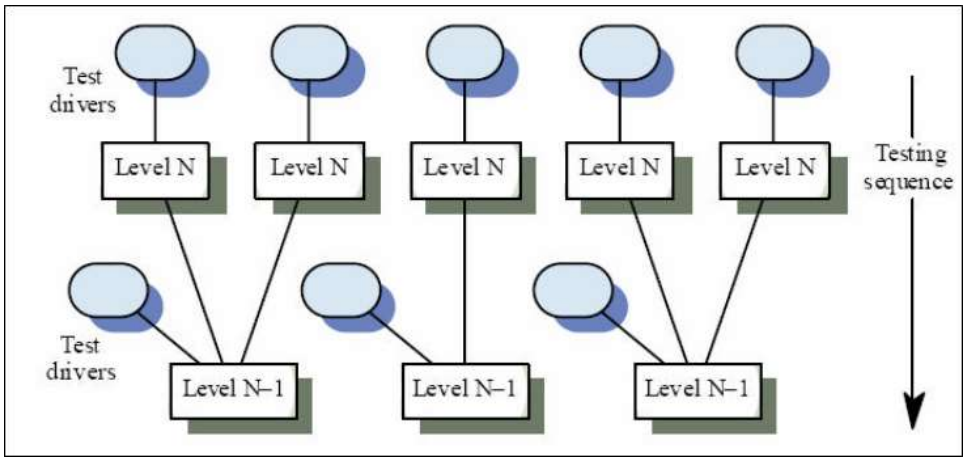


Challenges tester might face:

- Delay many tests until stubs are replaced with actual modules.
- Develop stubs that perform limited functions that simulate the actual module.
- Integrate the software from the bottom of the hierarchy upward.

Bottom-Up Approach

Bottom-up integration begins construction and testing with modules at the lowest level in the program structure. In this process, the modules are integrated from the bottom to the top.
In this approach processing required for the modules subordinate to a given level is always available and the need for the stubs is eliminated.



This integration test process is performed in a series of four steps

1. Low-level modules are combined into clusters that perform a specific software sub-function.
2. A driver is written to coordinate test case input and output.
3. The cluster or build is tested.
4. Drivers are removed, and clusters are combined moving upward in the program structure.

As integration moves upward, the need for separate test drivers lessons. In fact, if the top two levels of program structure are integrated top-down, the number of drivers can be reduced substantially, and integration of clusters is greatly simplified. Integration follows the pattern illustrated below. As integration moves upward, the need for separate test drivers lessons.

Big Bang Approach

In this approach, all modules are not integrated until and unless all the modules are ready. Once they are ready, all modules are integrated and then its executed to know whether all the integrated modules are working or not.
In this approach, it is difficult to know the root cause of the failure because of integrating everything at once.
Also, there will be a high chance of occurrence of the critical bugs in the production environment. This approach is adopted only when integration testing has to be done at once.

Reference:
https://www.guru99.com/
Next Generation Application Integration