White Paper

INTEGRATION TESTING TECHNIQUES

Kratika Parmar

Contents

1 ABSTRACT

Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors or missing requirements in contrary to the actual desire or requirements. As integration testing is targeted at testing the interactions between modules, this testing has a set of methods and techniques.

2 INTRODUCTION

Integration testing is a logical extension of unit testing. In its simplest form, two units that have already been tested are combined into a component and the interface between them is tested. A component, in this sense, refers to an integrated aggregate of more than one unit. In a realistic scenario, many units are combined into components, which are in turn aggregated into even larger parts of the program. The idea is to test combinations of pieces and eventually expand the process to test your modules with those of other groups. Eventually all the modules making up a process are tested together. Beyond that, if the program is composed of more than one process, they should be tested in pairs rather than all at once.

Integration testing identifies problems that occur when units are combined. By using a test plan that requires you to test each unit and ensure the viability of each before combining units, you know that any errors discovered when combining units are likely related to the interface between units. This method reduces the number of possibilities to a far simpler level of analysis.

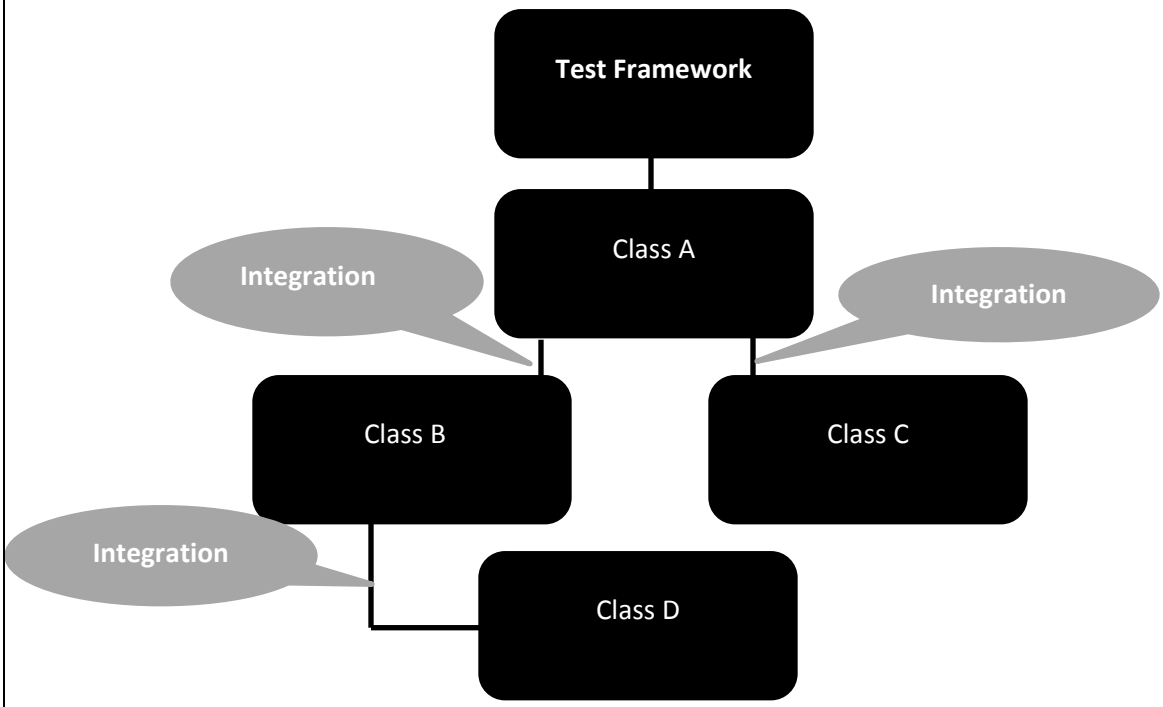

FIGURE 1

3 INTEGRATION TESTING TYPES

There are two groups of software integration strategies:
- Incremental software integration
- Non Incremental software integration

Incremental software integration

- **Top-Down:** The top-down approach to integration testing requires the highest-level modules be test and integrated first. This allows high-level logic and data flow to be tested early in the process and it tends to minimize the need for drivers. However, the need for stubs complicates test management and low-level utilities are tested relatively late in the development cycle. Another disadvantage of top-down integration testing is its poor support for early release of limited functionality.
- **Bottom-Up:** The bottom-up approach requires the lowest-level units be tested and integrated first. These units are frequently referred to as utility modules. By using this approach, utility modules are tested early in the development process and the need for stubs is minimized. The downside, however, is that the need for drivers complicates test management and high-level logic and data flow are tested late. Like the top-down approach, the bottom-up approach also provides poor support for early release of limited functionality.
- **Sandwich Testing**: Also referred to as Hybrid Integration Testing, requires testing along functional data and control-flow paths. First, the inputs for functions are integrated in the bottom-up pattern discussed above. The outputs for each function are then integrated in the top-down manner. The primary advantage of this approach is the degree of support for early release of limited functionality. It also helps minimize the need for stubs and drivers. The potential weaknesses of this approach are significant, however, in that it can be less systematic than the other two approaches, leading to the need for more regression testing.

Non Incremental software integration:

- **Big band integration approach**: In big bang approach most or all of the developed modules are coupled together to form a complete system and then used for integration testing. Big Bang Integration Testing is an integration testing strategy wherein all units are linked at once, resulting in a complete system. When this type of testing strategy is adopted, it is difficult to isolate any errors found, because attention is not paid to verifying the interfaces across individual units.

3.1 Stubs and Drivers in Integration testing

Stubs are dummy modules that are always distinguish as "called programs", or you can say that is handle in integration testing (top down approach), it used when sub programs are under construction. Stubs are considered as the dummy modules that always simulate the low level modules.

Drivers are also considered as the form of dummy modules which are always distinguished as "calling programs", that is handled in bottom up integration testing, it is only used when main programs are under construction. Drivers are considered as the dummy modules that always simulate the high level modules.

Example of Stubs and Drivers:

For Example we have 3 modules login, home, and user module. Login module is ready and need to test it, but we call functions from home and user (which is not ready). To test at a selective module we write a short dummy piece of a code which simulates home and user, which will return values for Login, this piece of dummy code is always called **Stubs** and it is used in a **top down integration.**

Considering the same Example above: If we have Home and User modules get ready and Login module is not ready, and we need to test Home and User modules Which return values from Login module, So to extract the values from Login module We write a Short Piece of Dummy code for login which returns value for home and user, So these pieces of code is always called **Drivers** and it is used in **Bottom Up Integration.**

3.2 Top- Down Integration Testing

Top-down integration testing is an integration testing technique used in order to simulate the behavior of the lower-level modules that are not yet integrated. Stubs are the modules that act as temporary replacement for a called module and give the same output as that of the actual product.

The replacement for the 'called' modules is known as 'Stubs' and is also used when the software needs to interact with an external system.

Top-Down Flow diagram

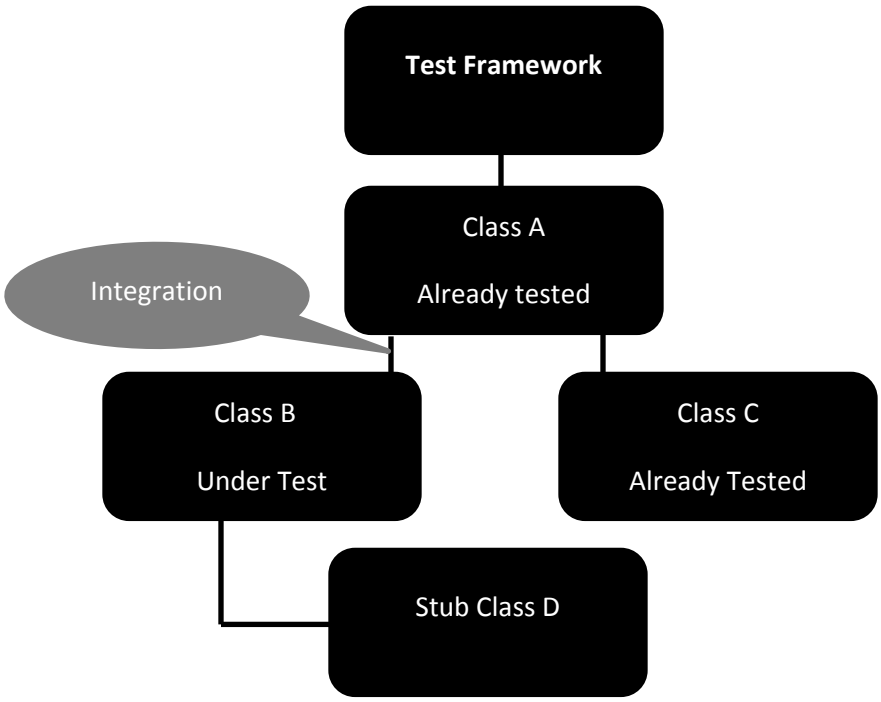

FIGURE 2

Stub - Flow Diagram:

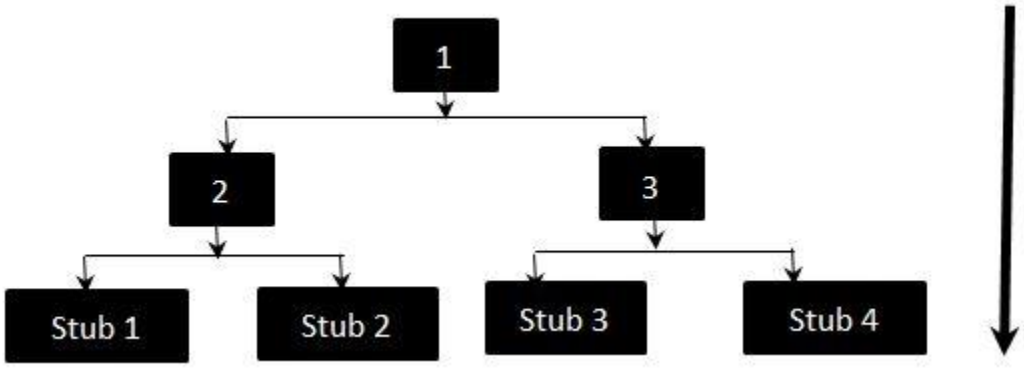

FIGURE 3

The above diagrams clearly states that Modules 1, 2 and 3 are available for integration, whereas, below modules are still under development that cannot be integrated at this point of time. Hence, Stubs are used to test the modules. The order of Integration will be:

1,2 -> 1,3 -> 2,Stub 1 -> 2,Stub 2 -> 3,Stub 3 -> 3,Stub 4

Testing Approach:

+ Firstly, the integration between the modules 1,2 and 3

+ Test the integration between the module 2 and stub 1,stub 2

+ Test the integration between the module 3 and stub 3,stub 4

Advantages:

- Advantageous if major flaws occur toward the top of the program.
- Once the I/O functions are added, representation of test cases is easier.
- Early skeletal Program allows demonstrations and boosts morale.

Disadvantages:

- Stub modules must be produced
- Stub Modules are often more complicated than they first appear to be.
- Before the I/O functions are added, representation of test cases in stubs can be difficult.
- Test conditions may be impossible, or very difficult, to create.
- Observation of test output is more difficult.
- Allows one to think that design and testing can be overlapped.
- Induces one to defer completion of the testing of certain modules.

3.3 Bottom-Up Integration Testing

Each component at lower hierarchy is tested individually and then the components that rely upon these components are tested.

Bottom-Up Integration - Flow Diagram

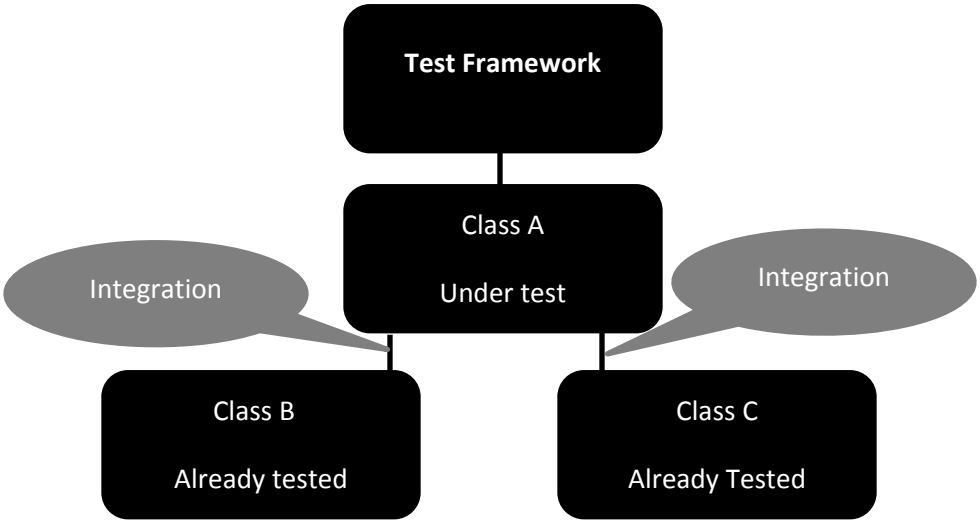

FIGURE 4

Stub - Flow Diagram:

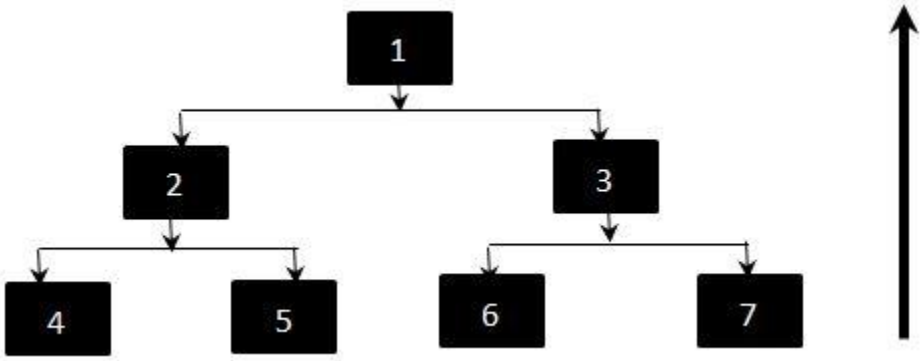

FIGURE 5

The order of Integration by Top-down approach will be:

4,2 -> 5,2 -> 6,3 -> 7,3 -> 2,1 -> 3,1

Testing Approach:

+ Firstly, Test 4,5,6,7 individually using drivers.

+ Test 2 such that it calls 4 and 5 separately. If an error occurs we know that the problem is in one of the modules.

+ Test 1 such that it calls 3 and If an error occurs we know that the problem is in 3 or in the interface between 1 and 3

Advantages:

- Advantageous if major flaws occur toward the bottom of the program.
- Test conditions are easier to create.
- Observation of test results is easier.

Disadvantages:

- Driver Modules must be produced.
- The program as an entity does not exist until the last module is added.

3.4 Sandwich/Hybrid Integration testing

Sandwich testing is also known as Hybrid approach in Integration testing. This approach is the combination of bottom-up & Top-Down Approach. In this approach the usage of Stubs (Called Program) and Drivers (Calling Program) are used where ever the programs are incomplete

In Sandwich testing / Hybrid Integration Testing, we exploit the advantages of Top-down and Bottom-up approaches. As the name suggests, we make use of both the Integration techniques.

FIGURE 6

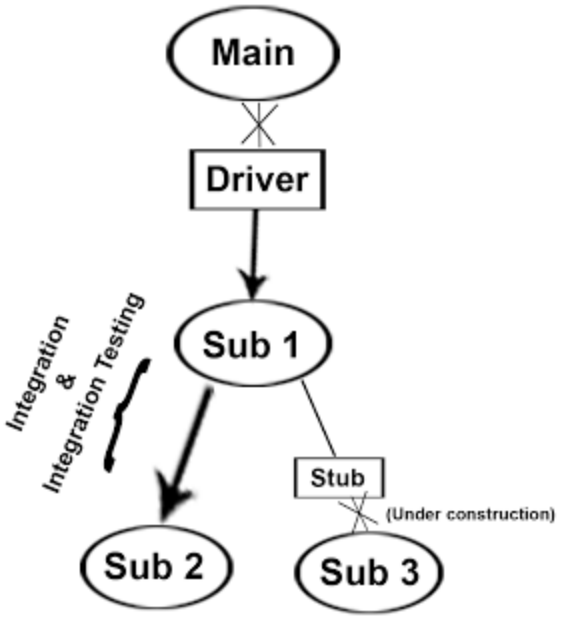

FIGURE 7

Sandwich Testing - Features

- It is viewed as three layers; - The Main Target Layer, a layer above the target layer and a layer below the target layer.
- Testing is mainly focused for the middle level target layer and is selected on the basis of system characteristics and the structure of the code.
- Hybrid Integration testing can be adopted if the customer wants to work on a working version of the application as soon as possible aimed at producing a basic working system in the earlier stages of the development cycle.

3.5 Big Bang Testing

In **big bang** approach most or all of the developed modules are coupled together to form a complete system and then used for integration testing. Big Bang Integration Testing is an integration testing strategy wherein all units are linked at once, resulting in a complete system. When this type of testing strategy is adopted, it is difficult to isolate any errors found, because attention is not paid to verifying the interfaces across individual units.

In big bang Integration testing, individual modules of the programs are not integrated until everything is ready. This approach is seen mostly in inexperienced programmers who rely on 'Run it and see' approach. In this approach, the program is integrated without any formal integration testing, and then run to ensures that all the components are working properly.

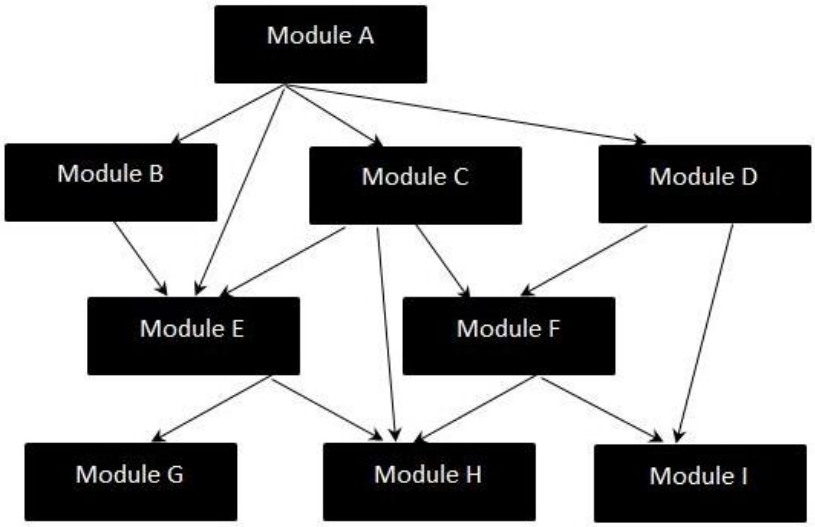

FIGURE 8

Advantages of Big-Bang Testing:

- Testing is done as a last phase of development life cycle in form of system testing, time for writing test case and defining test data at unit level may be saved.
- No Stubs/Drivers is required to be designed and coded in this approach. the cost involved is very less as it does not involved much creation of test artifacts.
- Big Bang approach is very fast. It may not give adequate confidence to users as all permutations and combinations cannot be tested in this testing. Even test log may not be maintained

Disadvantages of Big-Bang Testing

- Problem found in this approach are hard to debug, Many times defects found in random testing cannot be reproduced as one may not remember all steps followed in testing.
- It is difficult to say system interface are working correctly and will work in all cases.
- Location of defect may not be found easily. In this event if we can reproduce the defect, it can be very difficult to locate problematic areas for correcting them.
- Testers conduct testing based on few test cases by historic approach and certify whether the system works/ do not work.

4 COMPARISIONS OF INTEGRATION TESTING APPROCHS

Testing techniques → / Characteristics ↓	Top-down approach	Bottom-up approach	Sandwich approach	Big Bang approach
Integration	Early	Early	Early	Late
Time to basic working program	Early	Late	Early	Late
Component driver needed	No	Yes	Yes	Yes
Stubs needed	Yes	No	Yes	Yes
Work parallelism at beginning	Low	Medium	Medium	High
Ability to test particular path	Hard	Easy	Medium	Easy
Ability to plan and control sequence	Hard	Easy	Hard	Easy

5 CASE STUDIES

Case study is done for Banking Integration for Account Management and Bank analyzer

The **Transactional banking system** (Accounts Management System) application component is responsible for the management and processing of the statutory reporting figures. It provides information about the receivables and payables between a bank and third parties. The accounts, balances, and payment items are managed in the Account Management system

Analytical system (Bank Analyzer) supports overall bank controlling by calculating, evaluating, and analyzing financial products. Bank Analyzer provides a consistent view of a bank's operational data and enables you to process data promptly so that you are always in a position to provide current financial and risk information

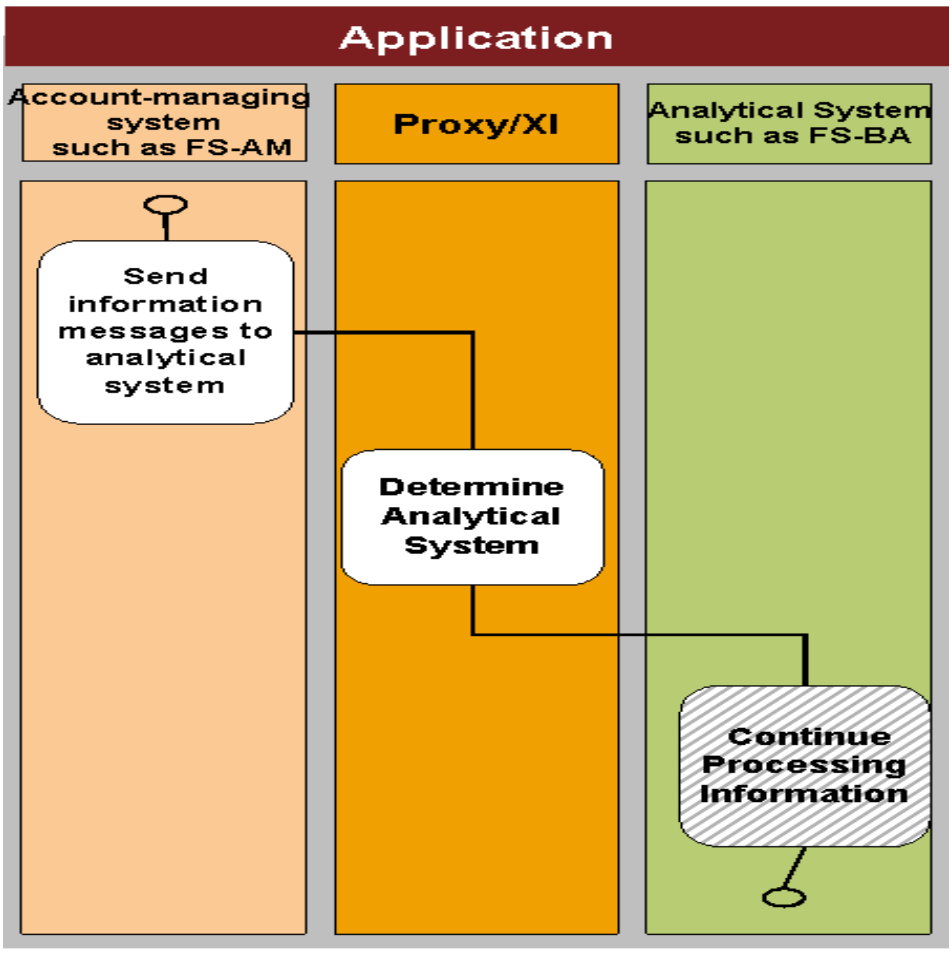

FIGURE 9

This Integration enables application components to communicate with each other by means of a proxy and to send information about account contracts the account-managing system provides account contract master data, posting data, settlement items, and accrual items. Using an appropriate method it calls the proxy to send the data to the connected analytical system

This integration scenario supports the "checking account", "deposit account"', and "loan" financial transactions, and the "payment", "settlement date", and "accruals and deferrals" business transactions.

IOA (Integration of analytic) Scenario:

- Create Bank and Business Partner in Transactional Banking System
- Distribute Data in Analytical System So that Legal Entity would be created and mapped with Bank and Business partner would be replicated.
- Create saving Scheme account and post payment item in Transactional Banking system
- Distribute data and check Financial and business transactions and financial position balance in Analytical system

- Execute Settlement in Transactional Banking system
- Distribute Data and check business transactions and financial position balance in Analytical system
- Prepare send General Ledger in Analytical system

Testing above scenario with different integration testing techniques:

5.1 Testing via Top- Down testing technique

According to Top-Down testing the product will be testing assuming that bottom part is not integrated, we will test Transactional Banking system assuming Analytical system as Stub and not yet integrated.
Testing same scenario via Top-Down Testing technique:

- Create Bank and Business Partner in Transactional Banking system
- Create saving account Scheme account and post payment item in Transactional Banking system assuming FT and BT created in Analytical system.
- Execute Settlement in Transactional Banking system and assuming data sent in analytical system.

5.2 Testing via Bottom-Up Testing technique:

In Bottom-Up approach the top part will be considered as driver and not yet integrated, so we will assume Transactional Banking system as Driver and testing will be done in Analytical system. Test case for Testing same scenario via Bottom-up Testing technique:

- Create Legal Entity Providing Dummy Bank details in Analytical System.
- Create Financial and business transactions and check financial position balance in Analytical system
- Prepare and send general ledger in Analytical system

5.3 Testing via Big-Bang testing technique:

In Big-Bang testing technique come in picture when all systems are integrated and working, So here we will test Scenario assuming Transactional Banking system is integrated with Analytical system. Test case of scenario with Big-bang testing:

- Create Bank and Business Partner in Transactional Banking System
- Distribute Data in Analytical System So that Legal Entity would be created and mapped with Bank and Business partner would be replicated.
- Create saving Scheme account and post payment item in Transactional Banking system
- Distribute data and check Financial and business transactions and financial position balance in Analytical system
- Execute Settlement in Transactional Banking system
- Distribute Data and check business transactions and financial position balance in Analytical system

- Prepare send General Ledger in Analytical system

6 SUMMARY

The integration of multiple applications is increasingly common and with it come new challenges to creating reliable products. It is no longer sufficient to simply test the application you have created, now we must test the way that application interacts with other applications that may be used alongside it. Integration testing is actually composed of different types of tests, but its objective is to ensure that the interaction of two or more components produces results that satisfy functional testing requirements. Integration Testing resembles actual usage more than Unit Tests do therefore, functional deficiencies are more likely to be detected at this level of testing than they are with Unit Testing. Here Unit is defined as the smallest testable part of an application. Integration testing can expose problems with the interfaces among program components before trouble occurs in real-world program execution. In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application.

Copyright