Why We Use Stubs And Drivers?

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 is given below:-

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.

Real Life Example:

Suppose we have to test the integration between 2 modules **A** and **B**and we have developed only module **A** while Module **B** is yet in development stage.
So in such case we can not do integration test for module A but, if we prepare a dummy module, having similar features like **B** then using that we can do integration testing.

Our main aim in this is to test Module A & not Module B so that we can save time otherwise we have to wait till the module B is actually developed. Hence this dummy module B is called as **Stub**.
Now module B cannot send/receive data from module A directly/automatically so, in such case we have to transfer data from one module to another module by some external features. This external feature used is called **Driver**.
At a time both stub and driver are essential.

Conclusion:-

So it is fine from the above example that **Stubs** act "called" functions in top down integration. **Drivers** are "calling" Functions in bottom up integration.

Approaches/Methodologies/Strategies of Integration Testing:

The Software Industry uses variety of strategies to execute Integration testing , viz.

- Big Bang Approach :
- Incremental Approach: which is further divided into following
  - Top Down Approach
  - Bottom Up Approach
  - Sandwich Approach - Combination of Top Down and Bottom Up

Below are the different strategies, the way they are executed and their limitations as well advantages.

Big Bang Approach:

Here all component are integrated together at once, and then tested.

Advantages:

- Convenient for small systems.

Disadvantages:

- Fault Localization is difficult.
- Given the sheer number of interfaces that need to be tested in this approach, some interfaces links to be tested could be missed easily.
- Since the integration testing can commence only after "all" the modules are designed, testing team will have less time for execution in the testing phase.
- Since all modules are tested at once, high risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

Incremental Approach:

In this approach, testing is done by joining two or more modules that are logically related. Then the other related modules are added and tested for the proper functioning. Process continues until all of the modules are joined and tested successfully.

This process is carried out by using dummy programs called Stubs and Drivers. Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.

Stub: Is called by the Module under Test.

Driver: Calls the Module to be tested.

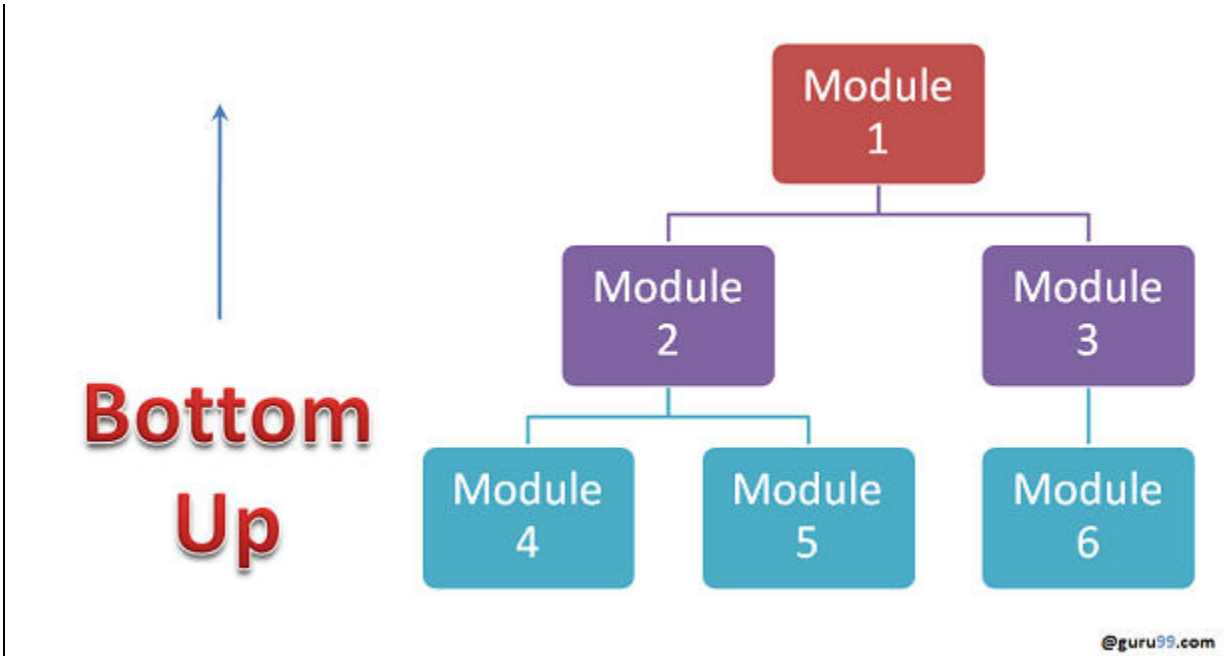
Incremental Approach in turn is carried out by two different Methods:

- Bottom Up
- Top Down

Bottom up Integration

In the bottom up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing

Diagrammatic Representation:



Advantages:

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

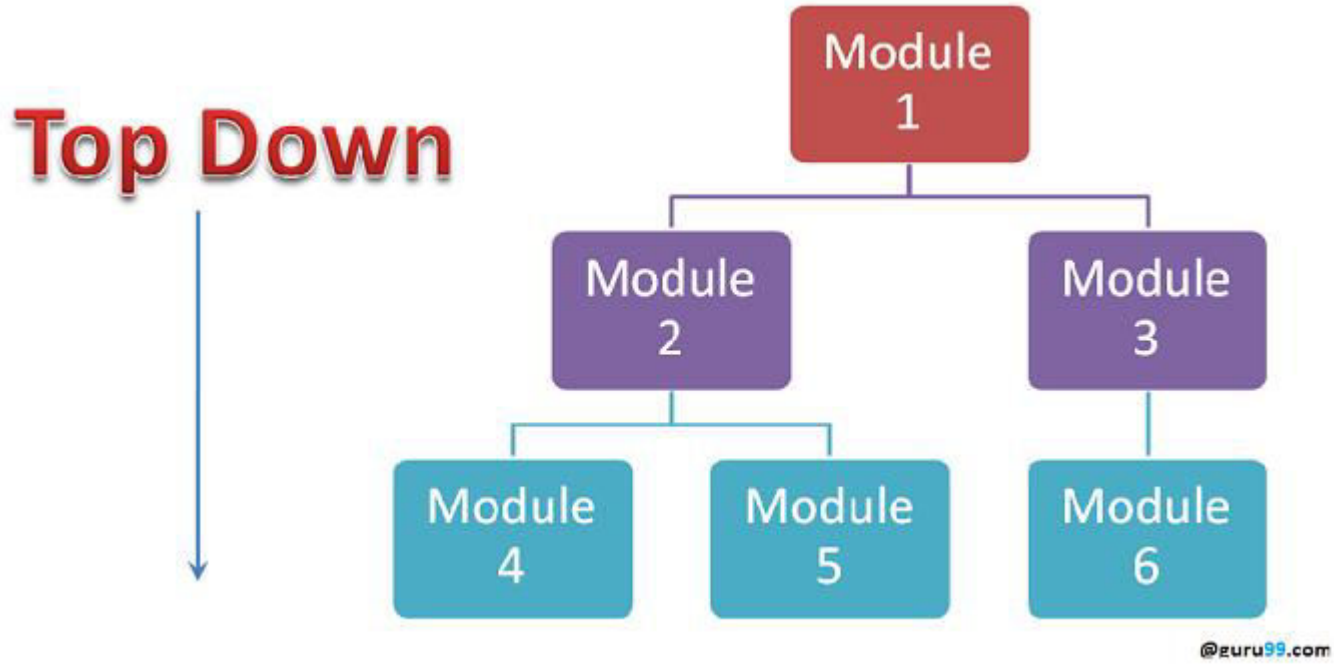
- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- Early prototype is not possible

Top down Integration:

In Top to down approach, testing takes place from top to down following the control flow of the software system.

Takes help of stubs for testing.

Diagrammatic Representation:



Advantages:

- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.

Let's conclude some difference between Stubs and Driver:

Stubs	Driver
Used in Top down approach	Used in Bottom up approach
Top most module is tested first	Lowest modules are tested first.
Stimulates the lower level of components	Stimulates the higher level of components
Dummy program of lower level components	Dummy program for Higher level component

Only change is Constant in this world, so we have another approach called "**Sandwich testing**" which combines the features of both Top down and bottom up approach. When we test huge programs like Operating systems, we have to have some more techniques which is efficient and boosts more confidence. Sandwich testing plays a very important role here, where both, the Top down and bottom up testing are started simultaneously.

Static Analysis vs Dynamic Analysis in Software Testing

What is Static Analysis?

Static analysis involves no dynamic execution of the software under test and can detect possible defects in an early stage, before running the program.

Static analysis is done after coding and before executing unit tests.

Static analysis can be done by a machine to automatically "walk through" the source code and detect noncomplying rules. The classic example is a compiler which finds lexical, syntactic and even some semantic mistakes.

Static analysis can also be performed by a person who would review the code to ensure proper coding standards and conventions are used to construct the program. This is often called Code Review and is done by a peer developer, someone other than the developer who wrote the code.

Static analysis is also used to force developers to not use risky or buggy parts of the programming language by setting rules that must not be used.

When developers performs code analysis, they usually look for

- Lines of code
- Comment frequency
- Proper nesting
- Number of function calls
- Cyclomatic complexity
- Can also check for unit tests

Quality attributes that can be the focus of static analysis:

- Reliability
- Maintainability
- Testability
- Re-usability
- Portability
- Efficiency

What are the Advantages of Static Analysis?

The main advantage of static analysis is that it finds issues with the code before it is ready for integration and further testing.

Static code analysis advantages:

- It can find weaknesses in the code at the exact location.
- It can be conducted by trained software assurance developers who fully understand the code.

- Source code can be easily understood by other or future developers
- It allows a quicker turn around for fixes
- Weaknesses are found earlier in the development life cycle, reducing the cost to fix.
- Less defects in later tests
- Unique defects are detected that cannot or hardly be detected using dynamic tests
    - Unreachable code
    - Variable use (undeclared, unused)
    - Uncalled functions
    - Boundary value violations

Static code analysis limitations:

- It is time consuming if conducted manually.
- Automated tools produce false positives and false negatives.
- There are not enough trained personnel to thoroughly conduct static code analysis.
- Automated tools can provide a false sense of security that everything is being addressed.
- Automated tools only as good as the rules they are using to scan with.
- It does not find vulnerabilities introduced in the runtime environment.

What is Dynamic Analysis?

In contrast to Static Analysis, where code is not executed, dynamic analysis is based on the **system execution**, often using tools.

Dynamic program analysis is the analysis of computer software that is performed with executing programs built from that software on a real or virtual processor (analysis

performed without executing programs is known as static code analysis). Dynamic program analysis tools may require loading of special libraries or even recompilation of program code.

The most common dynamic analysis practice is executing Unit Tests against the code to find any errors in code.

Dynamic code analysis advantages:

- It identifies vulnerabilities in a runtime environment.
- It allows for analysis of applications in which you do not have access to the actual code.
- It identifies vulnerabilities that might have been false negatives in the static code analysis.
- It permits you to validate static code analysis findings.
- It can be conducted against any application.

Dynamic code analysis limitations:

- Automated tools provide a false sense of security that everything is being addressed.
- Cannot guarantee the full test coverage of the source code
- Automated tools produce false positives and false negatives.
- Automated tools are only as good as the rules they are using to scan with.
- It is more difficult to trace the vulnerability back to the exact location in the code, taking longer to fix the problem.

Data Dictionary:

Data Dictionary acts as an automated or a manual, active or a passive file which has the ability to store the definitions of the data elements and the data characteristics. The Data Dictionary is actually the repository of the information about the data. Data Dictionary defines each of the data elements and also gives it a name for the easy access.

Data Dictionary acts as the core or the hub of the database management system, is the third component of the database management system. The Data Dictionary provides with the following information –

1. The name of the data item.
2. The description of the data item.
3. The sources of the data.
4. The impact analysis.
5. Keywords that are used for the categorization and the search of the data item descriptions.

Functions of the Data Dictionary

1. Defines the data element.
2. Helps in the scheduling.
3. Helps in the control.
4. Permits the various users who know which data is available and how can it be obtained.
5. Helps in the identification of the organizational data irregularity.
6. Acts as a very essential data management tool.
7. Provides with a good standardization mechanism.
8. Acts as the corporate glossary of the ever growing information resource.
9. Provides the report facility, the control facility along with the extract facility

Advantage of Data Dictionary

There is lot of advantages of data dictionary. Some of main points are

- It gives the well structured and clear information about the database. One can analyze the requirement, any redundancy like duplicate columns, tables etc. Since it provides a good documentation on each object, it helps to understand the requirement and design to the great extent.
- It is very helpful for the administrator or any new DBA to understand the database. Since it has all the information about the database, DBA can easily able to track any chaos in the database.

- Since database is a very huge, and will have lots of tables, views, constraints, indexes etc, it will be difficult for anyone to remember. Data dictionary helps user by providing all the details in it.

Disadvantages of Data Dictionary

- Creating a new data dictionary is a very big task. It will take years to create one.
- It should be well designed in advance to take all the advantages of it. Otherwise, it will create problems throughout its life.
- The cost of data dictionary will be bit high as it includes its initial build and hardware charges as well as cost of maintenance.
- Non technical users will not understand what the columns in the data dictionary views are. It meant only for technical users.

Differentiate between Open and Closed Systems

An open system is one that interacts with its environment and thus exchanges information, material, or energy with the environment, including random and undefined inputs. Open systems are adaptive in nature as they tend to react with the environment in such a way organizing', in the sense that they change their continued existence.

Such systems are 'self organizing', because they change their organization in response to changing conditions. A closed system is one, which doesn't interact with its environment. Such systems, in business world, are rare. Thus the systems that are relatively isolated from the environment but not completely closed are termed closed systems.

What is Alpha Testing?

This is a form of internal acceptance testing performed mainly by in-house software QA and testing teams. Alpha testing is the last testing done by test teams at development site after the acceptance testing and before releasing the software for beta test. Alpha testing can also be done by potential users or customers of the application. But still this is a form of in-house acceptance testing.

What is Beta Testing?

This is a testing stage followed by internal full alpha test cycle. This is the final testing phase where companies release the software for few external user groups outside the company test teams or employees. This initial software version is called as beta version. Most companies gather user feedback in this release.