Software Testing

- High-quality associates with software many quality factors like the following:

  - **Portability:**

  A software is claimed to be transportable, if it may be simply created to figure in several package environments, in several machines, with alternative code merchandise, etc.

  - **Usability:**

  A software has smart usability if completely different classes of users (i.e. each knowledgeable and novice users) will simply invoke the functions of the merchandise.

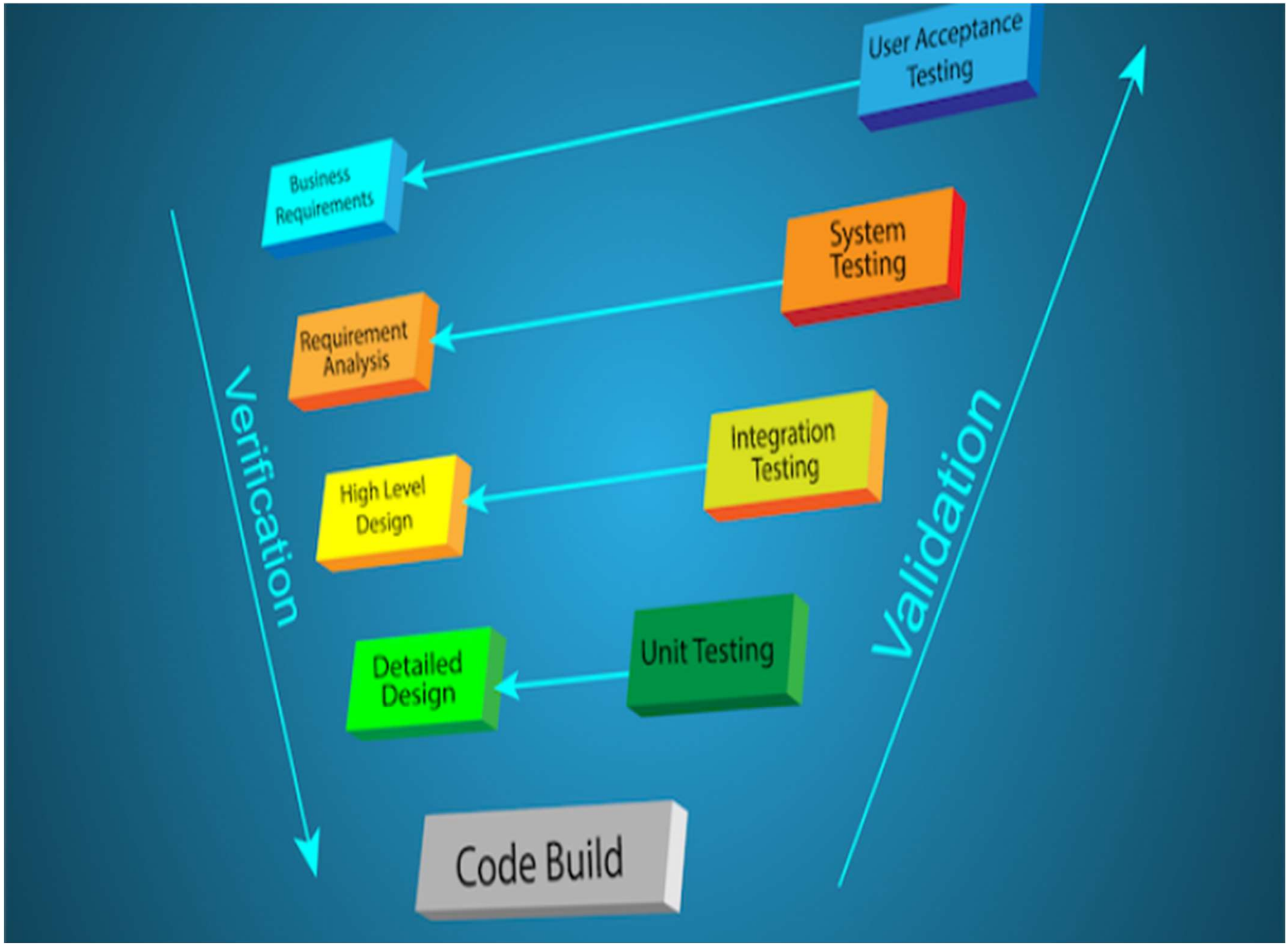
  - **Reusability:**

  A software has smart reusability if completely different modules of the merchandise will simply be reused to develop new merchandise.

- **Correctness:**
  A software is correct if completely different needs as laid out in the SRS document are properly enforced.
- **Maintainability:**
  A software is reparable, if errors may be simply corrected as and once they show up, new functions may be simply added to the merchandise, and therefore the functionalities of the merchandise may be simply changed, etc

Issues considered to implement software testing strategies

- Specify the requirement before testing starts in a quantifiable manner.
- According to the categories of the user generate profiles for each category of user.
- Produce a robust software and it's designed to test itself.
- Should use the Formal Technical Reviews (FTR) for the effective testing.
- To access the test strategy and test cases FTR should be conducted.
- To improve the quality level of testing generate test plans from the users feedback.


User Acceptance Testing
Business Requirements
System Testing
Requirement Analysis
Integration Testing
High Level Design
Unit Testing
Detailed Design
Code Build
Verification
Validation

Test strategies for conventional software

Following are the four strategies for conventional software:

1) Unit testing
2) Integration testing
3) Regression testing
4) Smoke testing

Why Unit Testing?

- **Unit Testing** is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost Defect fixing during System Testing, Integration Testing and even Beta Testing after application is built. If proper unit testing is done in early development, then it saves time and money in the end.

Unit testing

- Unit testing focus on the smallest unit of software design, i.e module or software component.
- Test strategy conducted on each module interface to access the flow of input and output.
- The local data structure is accessible to verify integrity during execution.
- Boundary conditions are tested.
- In which all error handling paths are tested.
- An Independent path is tested.

Following figure shows the unit testing:

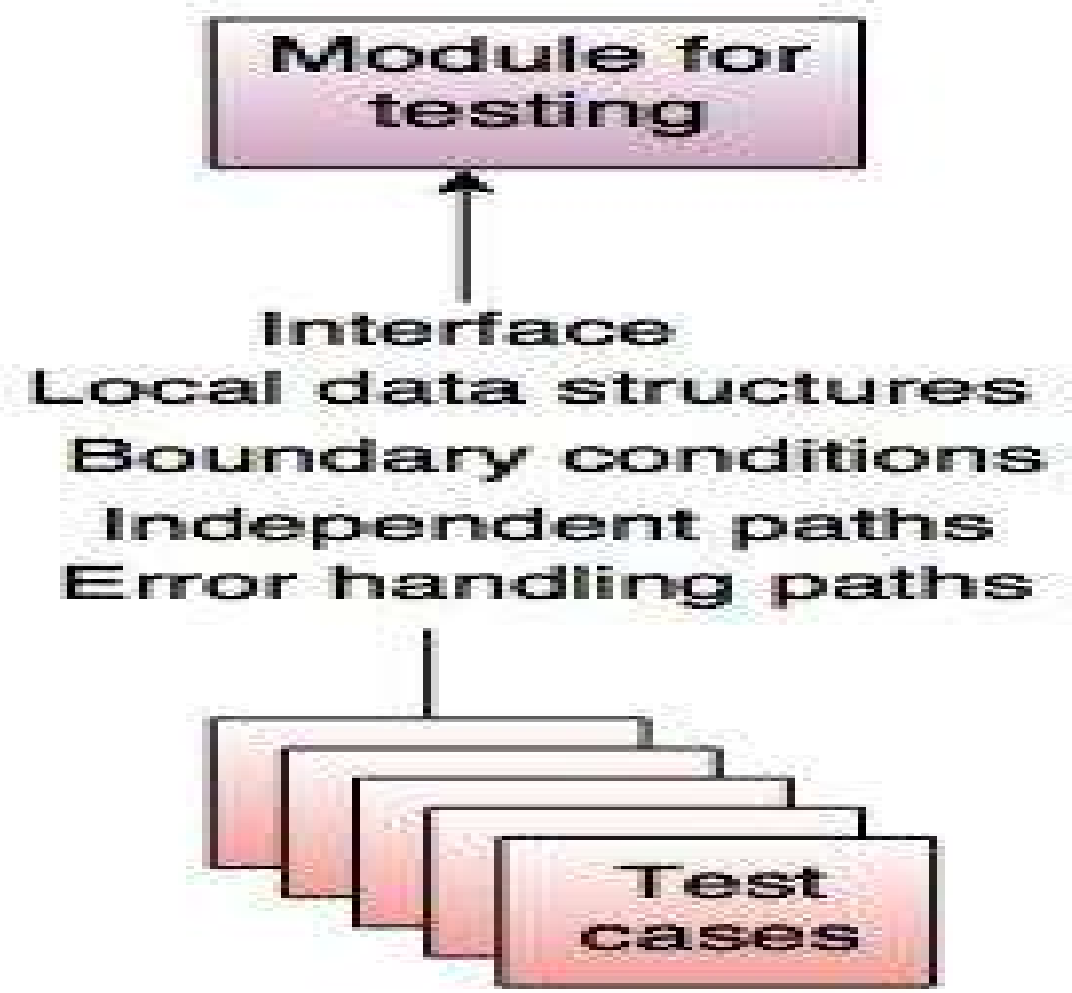


Fig. - Unit test

Unit test environment

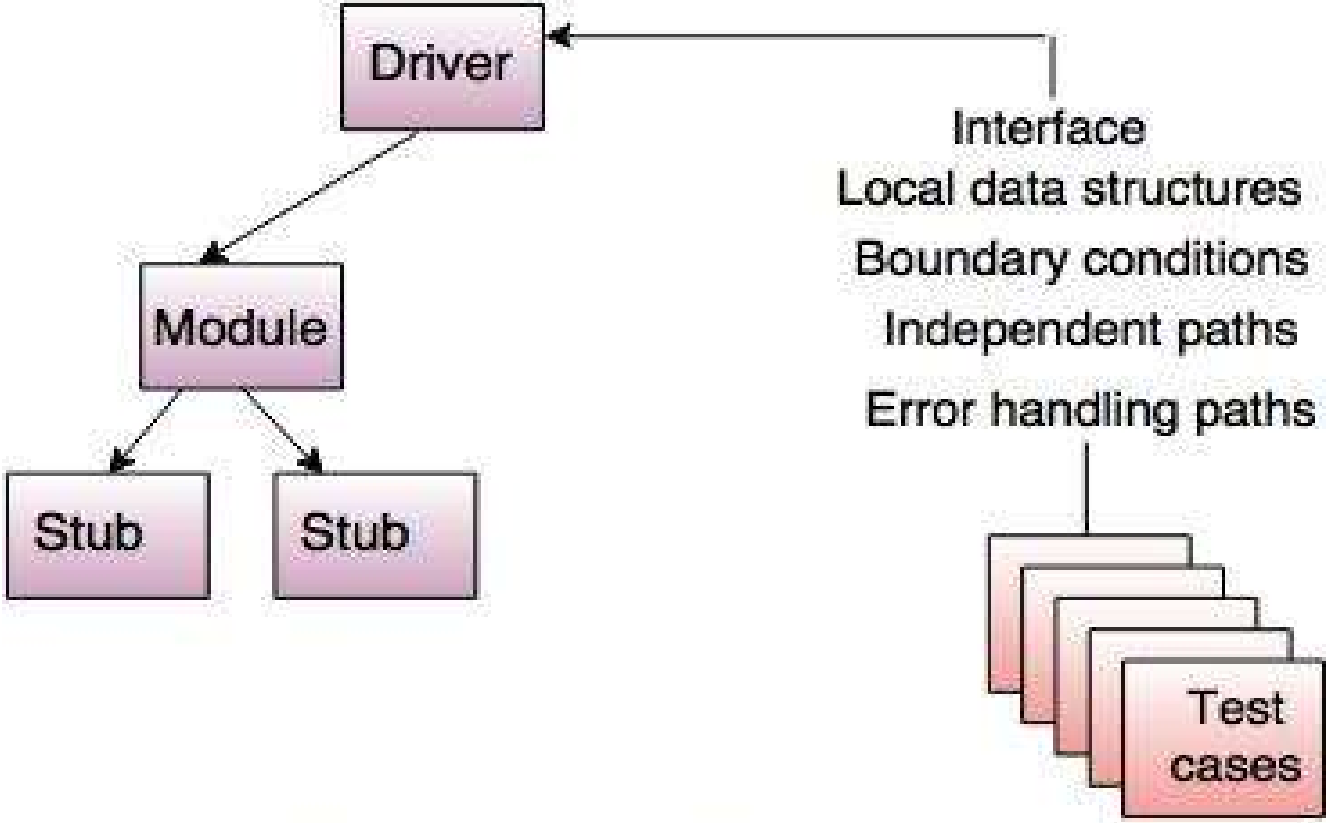
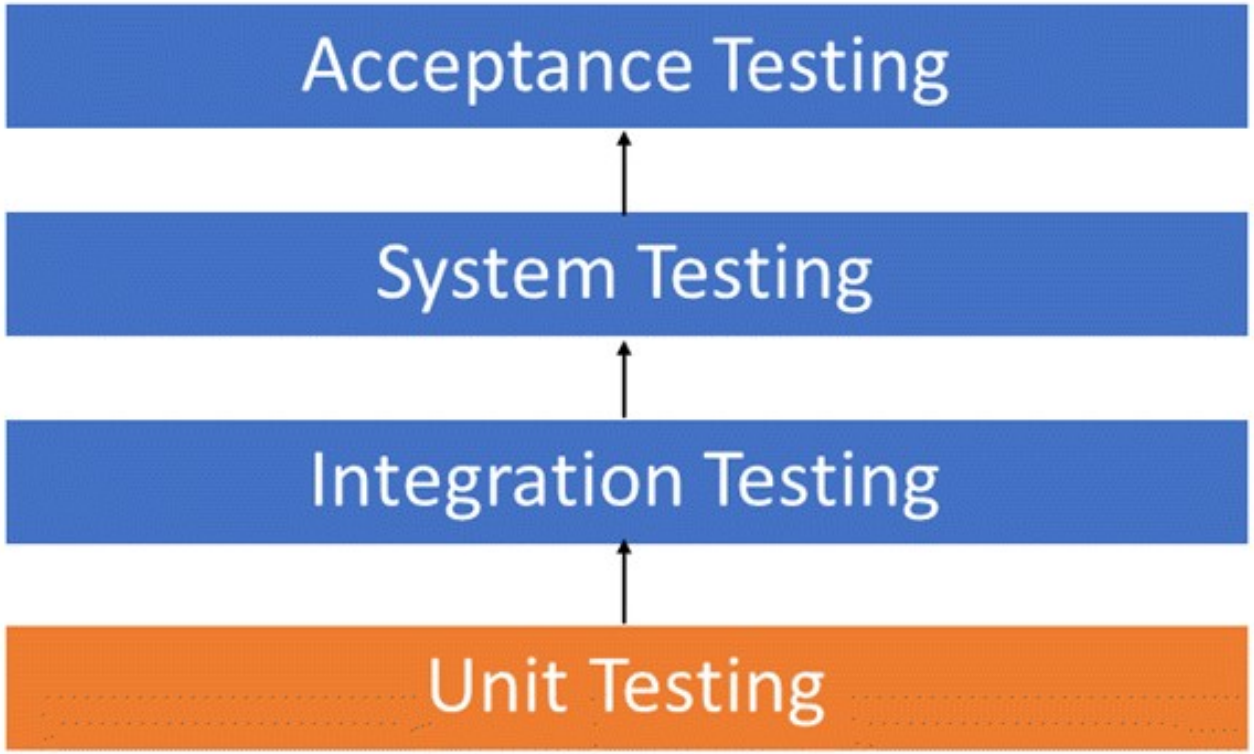


Fig. - Unit test environment

- Stubs and Drivers are two such elements that play a very crucial role while testing; they replace the modules that haven't been developed yet but are still needed in the testing of other modules against expected functionality and features.
- The Stubs and Drivers are considered as elements which are equivalent to to-do modules that could be replaced if modules are in their developing stage, missing or not developed yet, so that necessity of such modules could be met.
- Drivers and stubs simulate features and functionalities, and have ability to serve features that a module can provide. This reduces useless delay in testing and makes the testing process faster.
- Stubs are mainly used in Top-Down integration testing while the Drivers are used in Bottom-up integration testing, thus increasing the efficiency of testing process.

Unit testing levels



Unit Testing Techniques

- The **Unit Testing Techniques** are mainly categorized into three parts which are Black box testing that involves testing of user interface along with input and output, White box testing that involves testing the functional behaviour of the software application and Gray box testing that is used to execute test suites, test methods, test cases and performing risk analysis

.

- Code coverage techniques used in Unit Testing are listed below:
- Statement Coverage
- Decision Coverage
- Branch Coverage
- Condition Coverage
- Finite State Machine Coverage

Unit Test Example: Mock Objects

- Unit testing relies on mock objects being created to test sections of code that are not yet part of a complete application.
- Mock objects fill in for the missing parts of the program.
- For example, you might have a function that needs variables or objects that are not created yet. In unit testing, those will be accounted for in the form of mock objects created solely for the purpose of the unit testing done on that section of code.

Unit Testing Tools

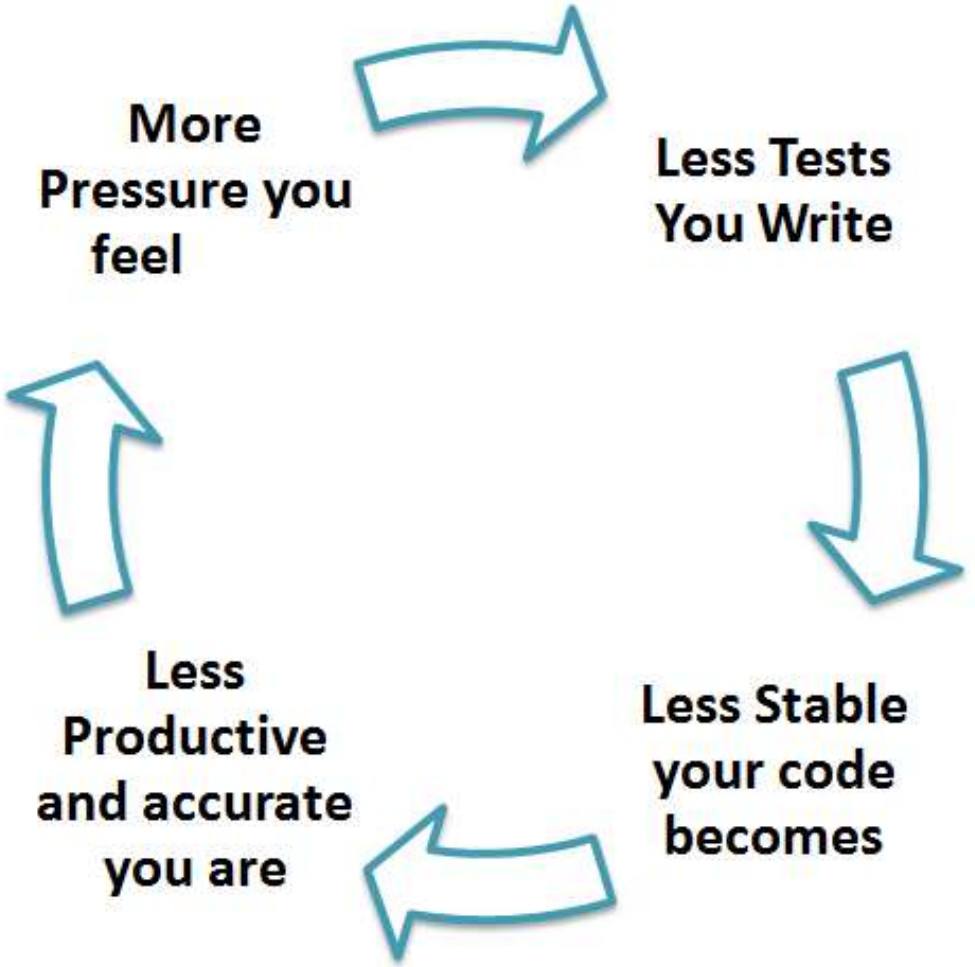
- several automated unit test software available to assist with unit testing.
- Junit: Junit is a free to use testing tool used for Java programming language. It provides assertions to identify test method. This tool test data first and then inserted in the piece of code.
- NUnit: NUnit is widely used unit-testing framework use for all .net languages. It is an open source tool which allows writing scripts manually. It supports data-driven tests which can run in parallel.

- JMockit: JMockit is open source Unit testing tool. It is a code coverage tool with line and path metrics. It allows mocking API with recording and verification syntax. This tool offers Line coverage, Path Coverage, and Data Coverage.
- EMMA: EMMA is an open-source toolkit for analyzing and reporting code written in Java language. Emma support coverage types like method, line, basic block. It is Java-based so it is without external library dependencies and can access the source code.
- PHPUnit: PHPUnit is a unit testing tool for PHP programmer. It takes small portions of code which is called units and test each of them separately. The tool also allows developers to use pre-define assertion methods to assert that a system behave in a certain manner.

Test Driven Development (TDD) & Unit Testing

- Unit testing in TDD involves an extensive use of testing frameworks. Below we look at some of what TDD brings to the world of unit testing:
- Tests are written before the code
- Rely heavily on testing frameworks
- All classes in the applications are tested
- Quick and easy integration is made possible

Myths by their very nature are false assumptions. These assumptions lead to a vicious cycle as follows



- Truth is Unit testing increase the speed of development.
- Programmers think that Integration Testing will catch all errors and do not execute the unit test.
- Once units are integrated, very simple errors which could have very easily found and fixed in unit tested take a very long time to be traced and fixed.

Unit Testing Advantage

- Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit API.
- Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly (i.e. Regression testing).
- The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.
- Due to the modular nature of the unit testing, we can test parts of the project without waiting for others to be completed

Unit Testing Disadvantages

- Unit testing can’t be expected to catch every error in a program. It is not possible to evaluate all execution paths even in the most trivial programs
- Unit testing by its very nature focuses on a unit of code. Hence it can’t catch integration errors or broad system level errors.

It’s recommended unit testing be used in conjunction with other testing activities.

Unit Testing Best Practices

- Unit Test cases should be independent. In case of any enhancements or change in requirements, unit test cases should not be affected.
- Test only one code at a time.
- Follow clear and consistent naming conventions for unit tests.
- In case of a change in code in any module, ensure there is a corresponding unit Test Case for the module, and the module passes the tests before changing the implementation
- Bugs identified during unit testing must be fixed before proceeding to the next phase in SDLC.
- Adopt a “test as your code” approach. The more code you write without testing, the more paths you have to check for errors.

Keep on a straight path with proper unit testing.

Summary

- UNIT TESTING is defined as a type of software testing where individual units or components of a software are tested.
- As you can see, there can be a lot involved in unit testing. It can be complex or rather simple depending on the application being tested and the testing strategies, tools and philosophies used.
- Certainly, Unit testing is always necessary on some level.

Stubs

- Stubs are developed by software developers to use them in place of modules, if the respective modules aren't developed, missing in developing stage, or are unavailable currently while Top-down testing of modules.
- A Stub simulates module which has all the capabilities of the unavailable module.
- Stubs are used when the lower-level modules are needed but are unavailable currently.

Stubs are divided into four basic categories based on what they do :

- Shows the traced messages,
- Shows the displayed message if any,
- Returns the corresponding values that are utilized by modules,
- Returns the value of the chosen parameters(arguments) that were used by the testing modules.

Drivers

- Drivers serve the same purpose as stubs, but drivers are used in Bottom-up integration testing and are also more complex than stubs.
- Drivers are also used when some modules are missing and unavailable at time of testing of a specific module because of some unavoidable reasons, to act in absence of required module.
- Drivers are used when both high-level modules and lower-level modules are missing.

- **Ex :** Suppose, to test a website whose corresponding primary modules are, where each of them is interdependent on each other, as follows:
- **Module-A :** Login page website,
- **Module-B :** Home page of the website
- **Module-C :** Profile setting
- **Module-D :** Sign-out page
- It’s always considered good practice to begin development of all modules parallelly because as soon as each gets developed they can be integrated and could be tested further as per their corresponding interdependencies order with a module.
- But in some cases, if any one of them is in developing stage or not available in the testing process of a specific module, stubs or drivers could be used instead.

- Assume **Module-A** is developed. As soon as it's developed, it undergoes testing, but it requires **Module-B,** which isn't developed yet.
- So in this case, we can use the **Stubs or Drivers** that simulate all features and functionality that might be shown by actual **Module-B.**
- So, we can conclude that Stubs and drivers are used to fulfil the necessity of unavailable modules. Similarly, the Stubs or Drivers may be used in place of **Module-C** and **Module-D** if they are too not available.

Do both drivers and Stubs serve the **same functionality?**

Yes,.

we can say both serve the same feature and are used in the absence of a **module(M_1)** that has interdependencies with an other **module(M_2)** that is need to be test.

so we use drivers or stubs in order to fulfil **module(M_1)'s** unavailability's and to serve its functionality.

Difference between Stubs and Drivers

S.No.	Stubs	Drivers
1.	Stubs are used in Top-Down Integration Testing.	Drivers are used in Bottom-Up Integration Testing.
2.	Stubs are basically known as a “called programs” and are used in the Top-down integration testing.	While, drivers are the “calling program” and are used in bottom-up integration testing.
3.	Stubs are similar to the modules of the software, that are under development process.	While drivers are used to invoking the component that needs to be tested.

4.	Stubs are basically used in the unavailability of low-level modules.	While drivers are mainly used in place of high-level modules and in some situation as well as for low-level modules.
5.	Stubs are taken into use to test the feature and functionality of the modules.	Whereas the drivers are used if the main module of the software isn’t developed for testing.
6.	The stubs are taken into concern if testing of upper-levels of the modules are done and the lower-levels of the modules are under developing process.	The drivers are taken into concern if testing of lower-levels of the modules are done and the upper-levels of the modules ae under developing process

7.	Stubs are used when lower-level of modules are missing or in a partially developed phase, and we want to test the main module.	Drivers are used when higher-level of modules are missing or in a partially developed phase, and we want to test the lower(sub)-module.

Difference between stub and driver

- Stub- Stub is considered as subprogram. It is a simple main program. Stub does not accept test case data.
- Driver - accepts test case data. It replace the modules of the program into subprograms and are tested by the next driver. Pass the data to the tested components and print the returned result.

2) Integration testing

- Integration testing is used for the construction of software architecture.

  There are two approaches of incremental testing are:
  i) Non incremental integration testing
  ii) Incremental integration testing

  i) Non incremental integration testing Combines all the components in advanced.
- A set of error is occurred then the correction is difficult because isolation cause is complex.
- **ii) Incremental integration testing** The programs are built and tested in small increments.
- The errors are easier to correct and isolate.
- Interfaces are fully tested and applied for a systematic test approach to it.

- **Following are the incremental integration strategies:**
  a. Top-down integration
  b. Bottom-up integration

a. Top-down integration

- It is an incremental approach for building the software architecture.
- It starts with the main control module or program.
- Modules are merged by moving downward through the control hierarchy.

Following figure shows the top down integration.

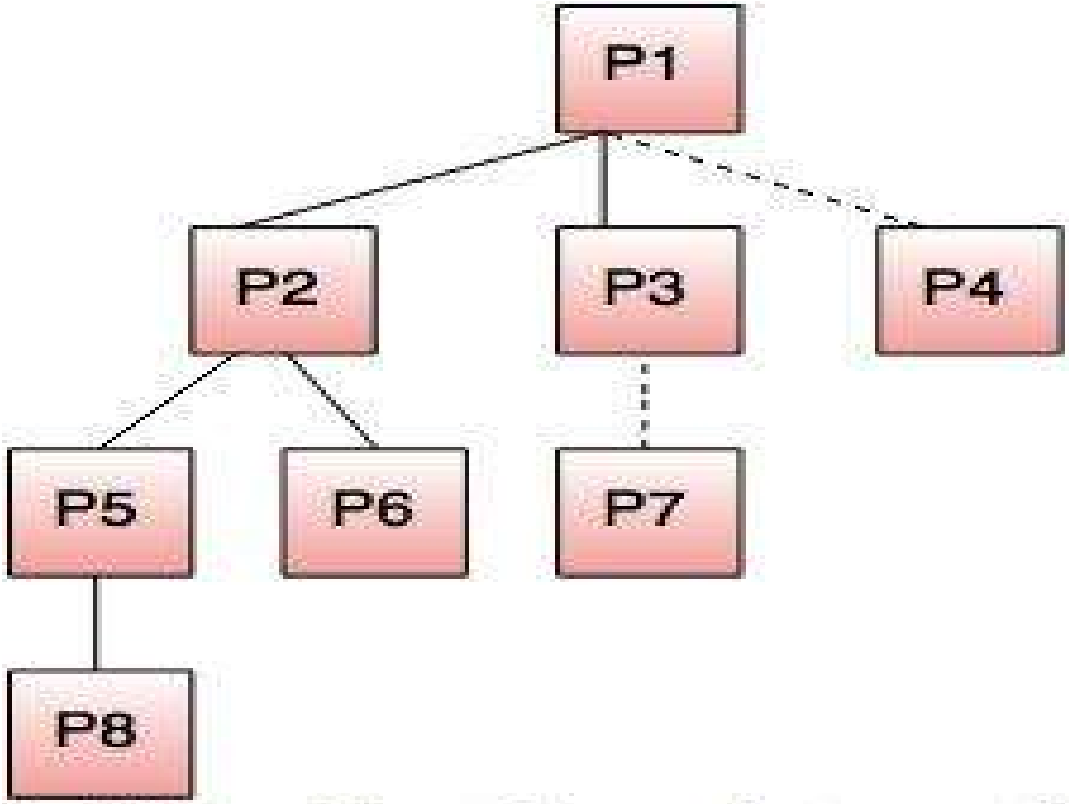


Fig. - Top-down integration

Problems with top-down approach of testing

- **Following are the problems associated with top-down approach of testing as follows:** Top-down approach is an incremental integration testing approach in which the test conditions are difficult to create.
- A set of errors occur, then correction is difficult to make due to the isolation of cause.
- The programs are expanded into various modules due to the complications.
- If the previous errors are corrected, then new get created and the process continues. This situation is like an infinite loop

b. Bottom-up integration

- In bottom up integration testing the components are combined from the lowest level in the program structure.

  The bottom-up integration is implemented in following steps: The low level components are merged into clusters which perform a specific software sub function.
- A control program for testing(driver) coordinate test case input and output.
- After these steps are tested in cluster.
- The driver is removed and clusters are merged by moving upward on the program structure.

Following figure shows the bottom up integration:

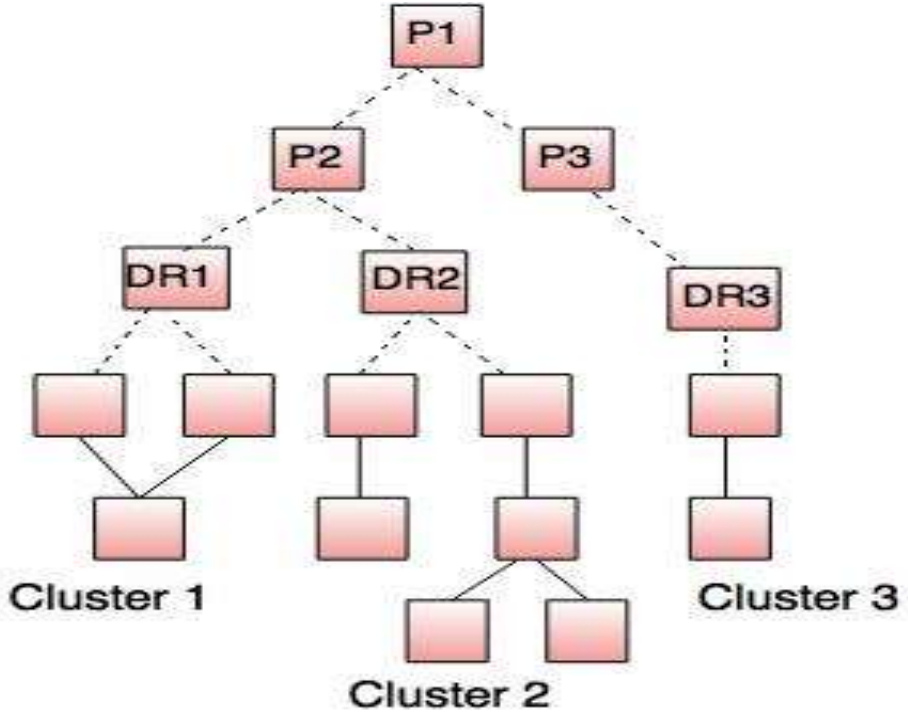


Fig. - Bottom-up integration

3) Regression testing

- In regression testing the software architecture changes every time when a new module is added as part of integration testing.

4) smoke testing

- The developed software component are translated into code and merge to complete the product.

Regression Testing

- Regression testing

Regression testing is used to check defects generated to other modules by making the changes in existing programs.

In regression tested components are tested again to verify the errors..

Regression testing needs extra manpower because the cost of the project increases.

Testers conduct the regression testing

Smoke testing

- At the time of developing a software product smoke testing is used
- It permit the software development team to test projects on a regular basis.
- Smoke testing does not need an extra manpower because it does not affect the cost of project.
- Developer conducts smoke testing just before releasing the product.

Alpha and Beta testing

Alpha testing

- Alpha testing is executed at developers end by the customer.
- It handles the software project and applications.
- It is not open to market and the public.
- Alpha testing does not have any different name.
- Alpha testing is not able to test the errors because the developer does not known the type of user.
- In alpha testing, developer modifies the codes before release the software without user feedback.

.

Beta testing

- Beta testing is executed at end-user sites in the absence of a developer.
- It usually handles software product
- It is always open to the market and the public.
- Beta testing is also known as the field testing.
- In beta testing, the developer corrects the errors as users report the problems.
- In beta testing, developer modifies the code after getting the feedback from user