UNIT 4

UNIT 4 TESTING AND IMPLEMENTATION **9 Hrs.**

Levels - Software Testing Fundamentals - Types of s/w test - White box testing- Basis path testing - Black box testing - Control Structure testing- Regression testing strategies - Strategic approach and issues - UNIT testing - Integration testing - Validation testing - System testing and debugging. Case studies - Writing black box and white box testing-Coding Practices-Refactoring.

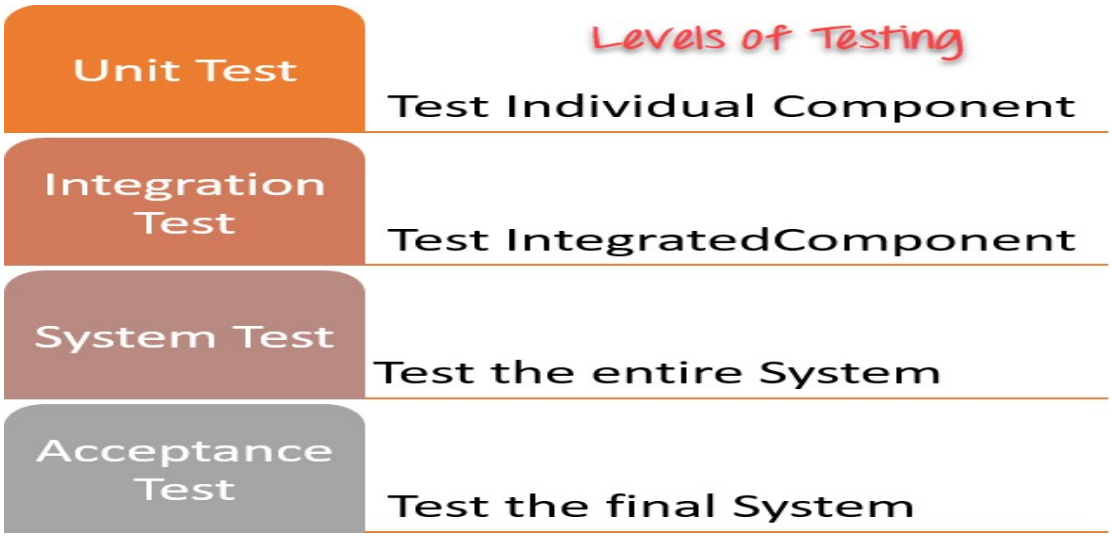
Levels:

Tests are grouped together based on where they are added in SDLC or the by the level of of detailing they contain. In general, there are four levels of testing: unit testing, integration testing, system testing, and acceptance testing. The purpose of Levels of testing is to make software testing systematic and easily identify all possible test cases at a particular level.

There are many different testing levels which help to check behavior and performance for software testing. These testing levels are designed to recognize missing areas and reconciliation between the development lifecycle states. In SDLC models there are characterized phases such as requirement gathering, analysis, design, coding or execution, testing, and deployment.

All these phases go through the process of software testing levels. There are mainly four testing levels are:

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing



UNIT 4

Each of these testing levels has a specific purpose. These testing level provide value to the software development lifecycle.

1) Unit testing:

A Unit is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately.

The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not. This kind of testing is performed by developers.

2) Integration testing:

Integration means combining. For Example, In this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing.

Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

3) System testing:

System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

4) Acceptance testing:

Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.

UNIT 4

Software Testing Fundamentals:

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. In this video we describe the fundamentals of software testing.

Different types of Software Testing processes are described below:

- **Unit-Testing**
  It is a method by which individual units of source code are tested to determine if they are fit for use.
- **Integration-Testing**
  Here individual software modules are combined and tested as a group.
- **Functionality-Testing**
  It is a type of black box testing that bases its test cases on the specifications of the software component under test.
- **Usability-Testing**
  It is a technique used to evaluate a product by testing it on users.
- **System-Testing**
  It is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.
- **Performance-Testing**
  It is testing that is performed, to determine how fast some aspect of a system performs under a particular workload.
- **Load-Testing**
  It refers to the practice of modeling the expected usage of a software program by simulating multiple users accessing the program concurrently.
- **Stress-Testing**
  It is a form of testing that is used to determine the stability of a given system or entity.

WHITE BOX TESTING

It is testing of a software solution's internal structure, design, and coding. In this type of testing, the code is visible to the tester. It focuses primarily on verifying the flow of inputs and outputs through the application, improving design and usability,

strengthening security. White box testing is also known as Clear Box testing, Open Box testing, Structural testing, Transparent Box testing, Code-Based testing, and Glass Box testing. It is usually performed by developers.

It is one of two parts of the **Box Testing** approach to software testing. Its counterpart, **Blackbox testing**, involves testing from an external or end-user type perspective. On the other hand, Whitebox testing is based on the inner workings of an application and revolves around internal testing.

The term "WhiteBox" was used because of the see-through box concept. The clear box or WhiteBox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "Black Box Testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested.

White box testing involves the testing of the software code for the following:

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object, and function on an individual basis

The testing can be done at system, integration and unit levels of software development. One of the basic goals of white box testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the expected output, you have encountered a bug.

Perform White Box Testing

To give you a simplified explanation of white box testing, we have divided it into **two basic steps**. This is what testers do when testing an application using the white box testing technique:

Step 1) Understand the Source Code

The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of

an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.

Step 2) CREATE TEST CASES AND EXECUTE

The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include Manual Testing, trial, and error testing and the use of testing tools as we will explain further on in this article.

WhiteBox Testing Example

Consider the following piece of code

```
Printme (int a, int b) {---------------------------- Printme is a function
   int result = a+ b;
   If (result> 0)
         Print ("Positive", result)
   Else
         Print ("Negative", result)
   }---------------------------------------- End of the source code
```

The goal of WhiteBox testing is to verify all the decision branches, loops, statements in the code.

To exercise the statements in the above code, WhiteBox test cases would be

- A = 1, B = 1
- A = -1, B = -3

White Box Testing Techniques

UNIT 4

A major White box testing technique is Code Coverage analysis. Code Coverage analysis eliminates gaps in a Test Case suite. It identifies areas of a program that are not exercised by a set of test cases. Once gaps are identified, you create test cases to verify untested parts of the code, thereby increasing the quality of the software product

There are automated tools available to perform Code coverage analysis. Below are a few coverage analysis techniques

Statement Coverage:- This technique requires every possible statement in the code to be tested at least once during the testing process of software engineering.

Branch Coverage - This technique checks every possible path (if-else and other conditional loops) of a software application.

Apart from above, there are numerous coverage types such as Condition Coverage, Multiple Condition Coverage, Path Coverage, Function Coverage etc. Each technique has its own merits and attempts to test (cover) all parts of software code. Using Statement and Branch coverage you generally attain 80-90% code coverage which is sufficient.

Types of White Box Testing

White box testing encompasses several testing types used to evaluate the usability of an application, block of code or specific software package. There are listed below --

- **Unit Testing:** It is often the first type of testing done on an application. Unit Testing is performed on each unit or block of code as it is developed. Unit Testing is essentially done by the programmer. As a software developer, you develop a few lines of code, a single function or an object and test it to make sure it works before continuing Unit Testing helps identify a majority of bugs, early in the software development lifecycle. Bugs identified in this stage are cheaper and easy to fix.
- **Testing for Memory Leaks**: Memory leaks are leading causes of slower running applications. A QA specialist who is experienced at detecting memory leaks is essential in cases where you have a slow running software application.

Apart from above, a few testing types are part of both black box and white box testing. They are listed as below

- **White Box Penetration Testing:** In this testing, the tester/developer has full information of the application's source code, detailed network information, IP addresses involved and all server information the application runs on. The aim is to attack the code from several angles to expose security threats
- **White Box Mutation Testing**: Mutation testing is often used to discover the best coding techniques to use for expanding a software solution.

White Box Testing Tools

Below is a list of top white box testing tools.

- Parasoft Jtest
- EclEmma
- NUnit
- PyUnit
- HTMLUnit
- CppUnit

Advantages of White Box Testing

- Code optimization by finding hidden errors.
- White box tests cases can be easily automated.
- Testing is more thorough as all code paths are usually covered.
- Testing can start early in SDLC even if GUI is not available.

Disadvantages of WhiteBox Testing

- White box testing can be quite complex and expensive.
- Developers who usually execute white box test cases detest it. The white box testing by developers is not detailed can lead to production errors.
- White box testing requires professional resources, with a detailed understanding of programming and implementation.
- White-box testing is time-consuming, bigger programming applications take the time to test fully

UNIT 4

Basis Path Testing:

Path Testing:

Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path. It helps to determine all faults lying within a piece of code. This method is designed to execute all or selected path through a computer program.

Any software program includes, multiple entry and exit points. Testing each of these points is a challenging as well as time-consuming. In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

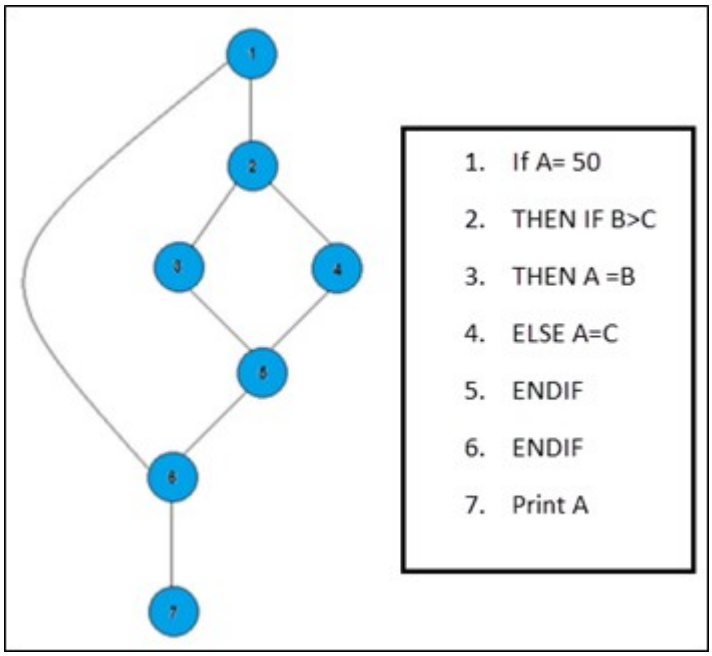
Basis Path Testing?

The basis path testing is same, but it is based on a White Box Testing method, that defines test cases based on the flows or logical path that can be taken through the program. In software engineering, Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases. It is a hybrid of branch testing and path testing methods.

The objective behind basis path in software testing is that it defines the number of independent paths, thus the number of test cases needed can be defined explicitly (maximizes the coverage of each test case).

Here we will take a simple example, to get a better idea what is basis path testing include

UNIT 4



In the above example, we can see there are few conditional statements that is executed depending on what condition it suffice. Here there are 3 paths or condition that need to be tested to get the output,

- **Path 1**: 1,2,3,5,6, 7
- **Path 2**: 1,2,4,5,6, 7
- **Path 3**: 1, 6, 7

Steps for Basis Path testing

The basic steps involved in basis path testing include

- Draw a control graph (to determine different program paths)
- Calculate Cyclomatic complexity (metrics to determine the number of independent paths)
- Find a basis set of paths
- Generate test cases to exercise each path

Advantages of Basic Path Testing

- It helps to reduce the redundant tests
- It focuses attention on program logic
- It helps facilitates analytical versus arbitrary case design

- Test cases which exercise basis set will execute every statement in a program at least once

Black Box Testing:

It is defined as a testing technique in which functionality of the Application Under Test (AUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on software requirements and specifications. In BlackBox Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.

The above Black-Box can be any software system you want to test. For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

BlackBox Testing Steps:

Here are the generic steps followed to carry out any type of Black Box Testing.

- Initially, the requirements and specifications of the system are examined.
- Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly. Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
- Tester determines expected outputs for all those inputs.
- Software tester constructs test cases with the selected inputs.
- The test cases are executed.
- Software tester compares the actual outputs with the expected outputs.
- Defects if any are fixed and re-tested.

Types of Black Box Testing

UNIT 4

There are many types of Black Box Testing but the following are the prominent ones -

- **Functional testing** - This black box testing type is related to the functional requirements of a system; it is done by software testers.
- **Non-functional testing** - This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.
- **Regression testing** - Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Tools used for Black Box Testing:

Tools used for Black box testing largely depends on the type of black box testing you are doing.

- For Functional/ Regression Tests you can use - QTP, Selenium
- For Non-Functional Tests, you can use - LoadRunner, Jmeter

Black Box Testing Techniques

Following are the prominent Test Strategy amongst the many used in Black box Testing

- **Equivalence Class Testing:** It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.
- **Boundary Value Testing:** Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is most suitable for the systems where an input is within certain ranges.
- **Decision Table Testing**: A decision table puts causes and their effects in a matrix. There is a unique combination in each column.

UNIT 4

Black Box Testing	White Box Testing
the main focus of black box testing is on the validation of your functional requirements.	White Box Testing (Unit Testing) validates internal structure and working of your software code
Black box testing gives abstraction from code and focuses on testing effort on the software system behavior.	To conduct White Box Testing, knowledge of underlying programming language is essential. Current day software systems use a variety of programming languages and technologies and its not possible to know all of them.
Black box testing facilitates testing communication amongst modules	White box testing does not facilitate testing communication amongst modules

Black Box Testing and Software Development Life Cycle (SDLC)

Black box testing has its own life cycle called Software Testing Life Cycle (STLC) and it is relative to every stage of Software Development Life Cycle of Software Engineering.

- **Requirement** - This is the initial stage of SDLC and in this stage, a requirement is gathered. Software testers also take part in this stage.
- **Test Planning & Analysis** - Testing Types applicable to the project are determined. A Test Plan is created which determines possible project risks and their mitigation.
- **Design** - In this stage Test cases/scripts are created on the basis of software requirement documents
- **Test Execution**- In this stage Test Cases prepared are executed. Bugs if any are fixed and re-tested.

Control structure testing

Control structure testing is a part of white box testing. it includes following methods:

1) Condition testing

2) Loop testing

3) Data validation testing.

4) Branch testing/Path testing.

Control structure testing is a group of white-box testing methods.

Condition Testing

- It is a test case design method.
- It works on logical conditions in program module.
- It involves testing of both relational expressions and arithmetic expressions.
- If a condition is incorrect, then at least one component of the condition is incorrect.

Types of errors in condition testing are

- boolean operator errors
- boolean variable errors
- boolean parenthesis errors
- relational operator errors
- arithmetic expression errors

Simple condition: Boolean variable or relational expression, possibly proceeded by a NOT operator.

Compound condition: It is composed of two or more simple conditions, Boolean operators and parentheses.

Boolean expression: It is a condition without Relational expressions.

Data Flow Testing:

Data flow testing method is effective for error protection because it is based on the relationship between statements in the program according to the definition and uses of variables.

UNIT 4

Test paths are selected according to the location of definitions and uses of variables in the program.

It is unrealistic to assume that data flow testing will be used extensively when testing a large system, However, it can be used in a targeted fashion for areas of software that are suspect.

Loop Testing :

- Loop testing method concentrates on validity of the loop structures.

- Loops are fundamental to many algorithms and need thorough testing.

- Loops can be defined as simple, concatenated, nested, and unstructured.

Simple loops :

- The following set of tests can be applied to simple loops, where n is the maximum number of allowable passes through the loop.

- Skip the loop entirely.

- Only one pass through the loop.

- Two passes through the loop.

- M passes through the loop where $m < n$

- $n - 1$, n, $n + 1$ passes through the loop.

Nested loops : If we were to extend the test approach for simple loops to nested loops, the number of possible tests would grow geometrically as the level of nesting increases. This would result in an impractical number of tests.

1. Start at the innermost loop. Set all other loops to minimum values.

2. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iter

3.Conducting tests for the next loop, but keeping all other outer loops at minimum values and other nested loops to “typical” values.

4. Continue until all loops have been tested.

Concatenated loops: Concatenated loops can be tested using the approach defined for simple loops, if each of the loops is independent of the other. However, if two loops are concatenated and the loop counter for loop 1 is used as the initial value for loop 2, then the loops are not independent

Unstructured loops: Whenever possible, this class of loops should be redesigned to reflect the use of the structured programming constructs

Validation Testing:

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

Path Testing:

Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path. It helps to determine all faults lying within a piece of code. This method is designed to execute all or selected path through a computer program.

Any software program includes, multiple entry and exit points. Testing each of these points is a challenging as well as time-consuming. In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

Unit Testing:

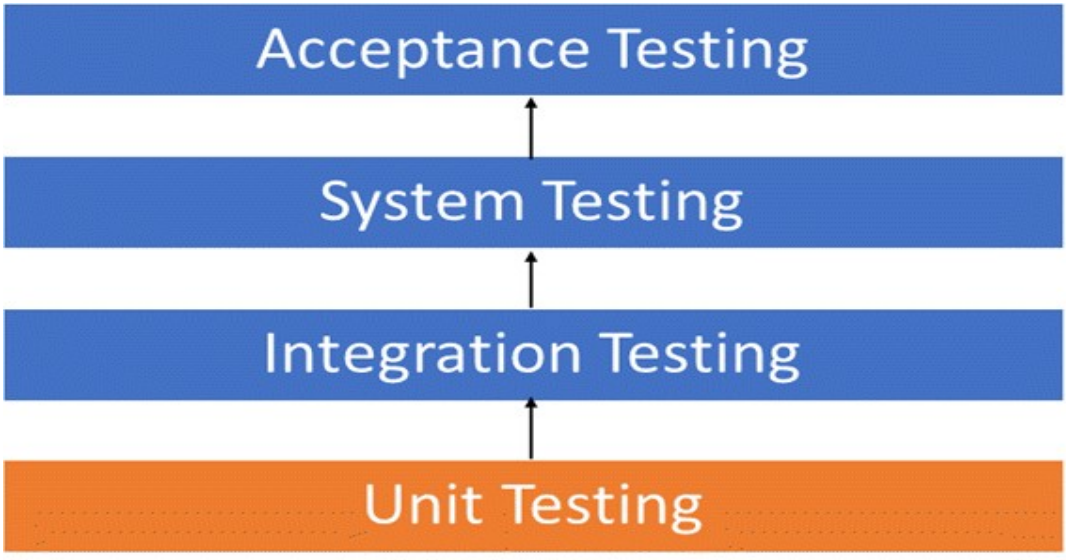
It is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs

as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing. Unit testing is a WhiteBox testing technique that is usually performed by the developer. Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing.

Need of Unit Testing:

Sometimes software developers attempt to save time by doing minimal unit testing. This is a myth because skipping on unit testing leads to higher Defect fixing costs during System Testing, Integration Testing and even Beta Testing after the application is completed. Proper unit testing done during the development stage saves both time and money in the end. Here, are key reasons to perform unit testing.



Unit Testing Levels

1. Unit tests help to fix bugs early in the development cycle and save costs.
2. It helps the developers to understand the code base and enables them to make changes quickly
3. Good unit tests serve as project documentation

4. Unit tests help with code re-use. Migrate both your code **and** your tests to your new project. Tweak the code until the tests run again.

Need of Unit Testing:

Unit Testing is of two types

- Manual
- Automated

Unit testing is commonly automated but may still be performed manually. Software Engineering does not favor one over the other but automation is preferred. A manual approach to unit testing may employ a step-by-step instructional document.

Under the automated approach-

- A developer writes a section of code in the application just to test the function. They would later comment out and finally remove the test code when the application is deployed.
- A developer could also isolate the function to test it more rigorously. This is a more thorough unit testing practice that involves copy and paste of code to its own testing environment than its natural environment. **Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data spaces** in the product. These dependencies can then be eliminated.
- A coder generally uses a UnitTest Framework to develop automated test cases. Using an automation framework, the developer codes criteria into the test to verify the correctness of the code. During execution of the test cases, the framework logs failing test cases. Many frameworks will also automatically flag and report, in summary, these failed test cases. Depending on the severity of a failure, the framework may halt subsequent testing.
- The workflow of Unit Testing is 1) Create Test Cases 2) Review/Rework 3) Baseline 4) Execute Test Cases.

Unit Testing Techniques

Code coverage techniques used in united testing are listed below:

- Statement Coverage

- Decision Coverage
- Branch Coverage
- Condition Coverage
- Finite State Machine Coverage

Unit Testing 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

There are several automated tools available to assist with unit testing. We will provide a few examples below:

1. 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.
2. 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.
3. 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.
4. 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.

5. 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.

Those are just a few of the available unit testing tools. There are lots more, especially for C languages and Java, but you are sure to find a unit testing tool for your programming needs regardless of the language you use.

Test Driven Development (TDD) & Unit Testing

Unit testing in TDD involves an extensive use of testing frameworks. A unit test framework is used in order to create automated unit tests. Unit testing frameworks are not unique to TDD, but they are essential to it. Below we look at some of what TDD brings to the world of unit testing:

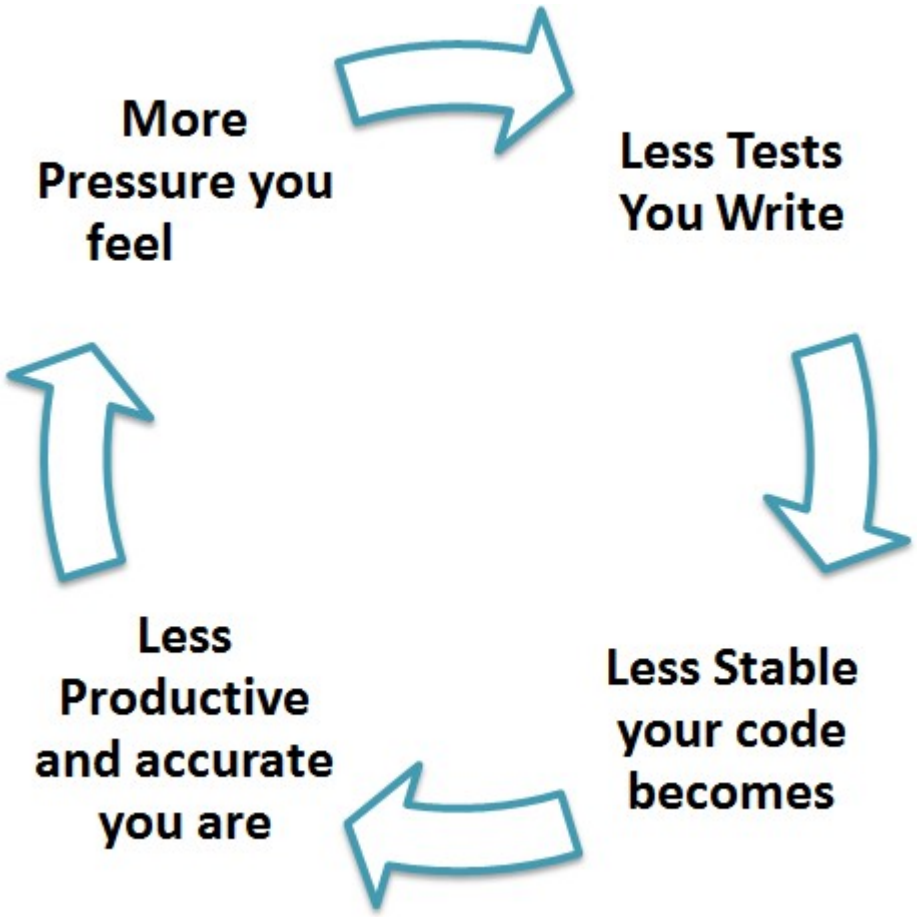
- Tests are written before the code
- Rely heavily on testing frameworks
- All classes in the applications are tested
- Quick and easy integration is made possible

Unit Testing Myth

Myth: It requires time, and I am always overscheduled My code is rock solid! I do not need unit tests.

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

UNIT 4



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.

Advantages:

- 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.

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.

Regression Testing:

It is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.

Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

Need of Regression Testing

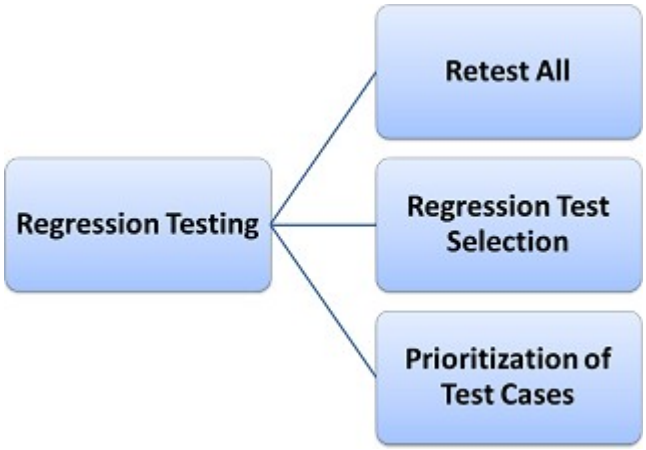
Regression Testing is required when there is a

- Change in requirements and code is modified according to the requirement
- New feature is added to the software
- Defect fixing
- Performance issue fix

Regression Testing procedure:

Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of existing features. These modifications

may cause the system to work incorrectly. Therefore, Regression Testing becomes necessary. Regression Testing can be carried out using the following techniques:



Retest All

- This is one of the methods for Regression Testing in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.

Regression Test Selection

- Instead of re-executing the entire test suite, it is better to select part of the test suite to be run
- Test cases selected can be categorized as 1) Reusable Test Cases 2) Obsolete Test Cases.
- Re-usable Test cases can be used in succeeding regression cycles.
- Obsolete Test Cases can't be used in succeeding cycles.

Prioritization of Test Cases

- Prioritize the test cases depending on business impact, critical & frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite.

Selecting test cases for regression testing

UNIT 4

It was found from industry data that a good number of the defects reported by customers were due to last minute bug fixes creating side effects and hence selecting the Test Case for regression testing is an art and not that easy. Effective Regression Tests can be done by selecting the following test cases -

- Test cases which have frequent defects
- Functionalities which are more visible to the users
- Test cases which verify core features of the product
- Test cases of Functionalities which has undergone more and recent changes
- All Integration Test Cases
- All Complex Test Cases
- Boundary value test cases
- A sample of Successful test cases
- A sample of Failure test cases

Regression Testing Tools

If your software undergoes frequent changes, regression testing costs will escalate.

In such cases, Manual execution of test cases increases test execution time as well as costs.

Automation of regression test cases is the smart choice in such cases.

The extent of automation depends on the number of test cases that remain re-usable for successive regression cycles.

Following are the most important tools used for both functional and regression testing in software engineering.

Ranorex Studio: all-in-one regression test automation for desktop, web, and mobile apps with built-in Selenium WebDriver. Includes a full IDE plus tools for codeless automation.

Selenium: This is an open source tool used for automating web applications. Selenium can be used for browser-based regression testing.

Quick Test Professional (QTP): HP Quick Test Professional is automated software designed to automate functional and regression test cases. It uses VBScript language for automation. It is a Data-driven, Keyword based tool.

UNIT 4

Rational Functional Tester (RFT): IBM's rational functional tester is a Java tool used to automate the test cases of software applications. This is primarily used for automating regression test cases and it also integrates with Rational Test Manager.

Regression Testing and Configuration Management

Configuration Management during Regression Testing becomes imperative in Agile Environments where a code is being continuously modified. To ensure effective regression tests, observe the following :

- Code being regression tested should be under a configuration management tool
- No changes must be allowed to code, during the regression test phase. Regression test code must be kept immune to developer changes.
- The database used for regression testing must be isolated. No database changes must be allowed

Difference between Re-Testing and Regression Testing:

Retesting means testing the functionality or bug again to ensure the code is fixed. If it is not fixed,Defect needs to be re-opened. If fixed, Defect is closed.

Regression testing means testing your software application when it undergoes a code change to ensure that the new code has not affected other parts of the software.

Challenges in Regression Testing:

Following are the major testing problems for doing regression testing:

- With successive regression runs, test suites become fairly large. Due to time and budget constraints, the entire regression test suite cannot be executed
- Minimizing the test suite while achieving maximum Test coverage remains a challenge
- Determination of frequency of Regression Tests, i.e., after every modification or every build update or after a bunch of bug fixes, is a challenge.

Validation Testing:

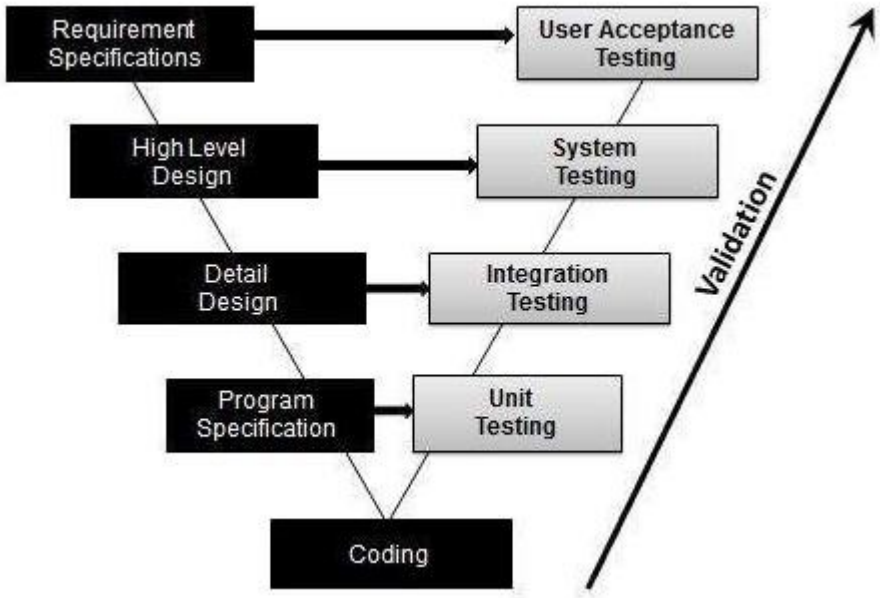
UNIT 4

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

Validation Testing - Workflow:

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.



Activities:

- Unit Testing
- Integration Testing
- System Testing
- User Acceptance Testing

System Testing:

UNIT 4

It is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

System Testing is Blackbox

Two Category of Software Testing

- Black Box Testing
- White Box Testing

System test falls under the **black box testing** category of software testing.

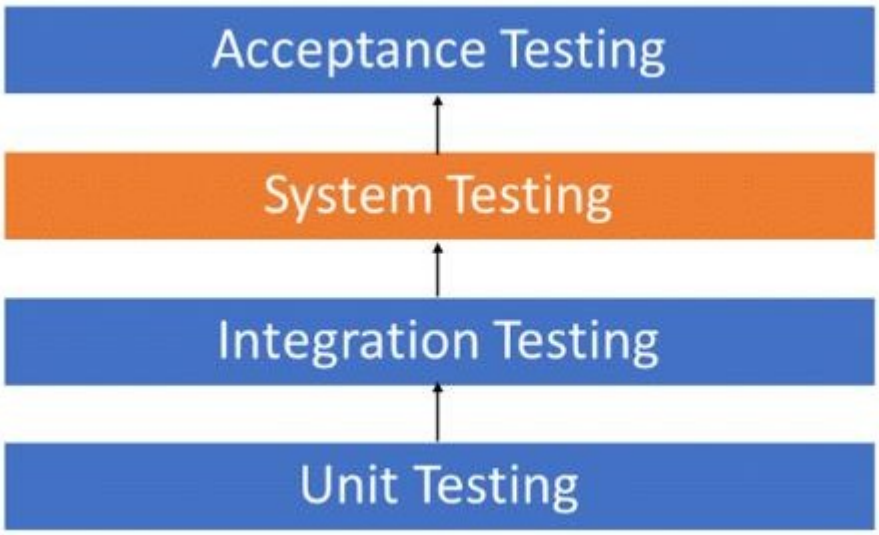
White box testing is the testing of the internal workings or code of a software application. In contrast, black box or System Testing is the opposite. System test involves the external workings of the software from the user's perspective.

System Testing involves testing the software code for following

- Testing the fully integrated applications including external peripherals in order to check how components interact with one another and with the system as a whole. This is also called End to End testing scenario.
- Verify thorough testing of every input in the application to check for desired outputs.
- Testing of the user's experience with the application.

That is a very basic description of what is involved in system testing. You need to build detailed test cases and test suites that test each aspect of the application as seen from the outside without looking at the actual source code.

UNIT 4



As with almost any software engineering process, software testing has a prescribed order in which things should be done. The following is a list of software testing categories arranged in chronological order. These are the steps taken to fully test new software in preparation for marketing it:

- Unit testing performed on each module or block of code during development. Unit Testing is normally done by the programmer who writes the code.

- Integration testing done before, during and after integration of a new module into the main software package. This involves testing of each individual code module. One piece of software can contain several modules which are often created by several different programmers. It is crucial to test each module's effect on the entire program model.

- System testing done by a professional testing agent on the completed software product before it is introduced to the market.

- Acceptance testing - beta testing of the product done by the actual end users.

Different Types of System Testing

Below we have listed types of system testing a large software development company would typically use

1. **Usability Testing**- mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives
2. **Load Testing**- is necessary to know that a software solution will perform under real-life loads.
3. **Regression Testing**- involves testing done to make sure none of the changes made over the course of the development process have caused new bugs. It also makes sure no old bugs appear from the addition of new software modules over time.
4. **Recovery testing** - is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes.
5. **Migration testing**- is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.
6. **Functional Testing** - Also known as functional completeness testing, Functional Testing involves trying to think of any possible missing functions. Testers might make a list of additional functionalities that a product could have to improve it during functional testing.
7. **Hardware/Software Testing** - IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the interactions between the hardware and software during system testing.

Types of System Testing Should Testers Use:

There are over 50 different types of system testing. The specific types used by a tester depend on several variables. Those variables include:

- Who the tester works for - This is a major factor in determining the types of system testing a tester will use. Methods used by large companies are different than that used by medium and small companies.

- Time available for testing - Ultimately, all 50 testing types could be used. Time is often what limits us to using only the types that are most relevant for the software project.
- Resources available to the tester - Of course some testers will not have the necessary resources to conduct a testing type. For example, if you are a tester working for a large software development firm, you are likely to have expensive automated testing software not available to others.
- Software Tester's Education- There is a certain learning curve for each type of software testing available. To use some of the software involved, a tester has to learn how to use it.
- Testing Budget - Money becomes a factor not just for smaller companies and individual software developers but large companies as well.

Debugging:

It is a systematic process of spotting and fixing the number of bugs, or defects, in a piece of software so that the software is behaving as expected. Debugging is harder for complex systems in particular when various subsystems are tightly coupled as changes in one system or interface may cause bugs to emerge in another.

Debugging is a developer activity and effective debugging is very important before testing begins to increase the quality of the system. Debugging will not give confidence that the system meets its requirements completely but testing gives confidence.