

Unit 1. Fundamentals of Testing

1.1. Testing Concepts:

Software Testing was established as a complete process rather than a single phase in Software Development Life Cycle (SDLC).

Testing is performed after coding.

1.1.1 Software Testing Terminologies:

Error:

- A human action that produces an incorrect result.
- Whenever a development team member makes a mistake in any phase of coding errors are produced.
- An error causes a bug and a bug in turn causes failures.
- Error description is the section of a defect report where the tester describes
 - the test steps he/she performed,
 - what the outcome was,
 - what result he/she expected,
 - And any additional information that will assist in troubleshooting.
- Experience-based test design technique where the tester develops test cases based on his/her skill and intuition, and experience with similar systems and technologies.

Fault:

- It is a condition that causes the software to fail to perform its required function.
- The tester must give description of the fault in code that causes a failure.

Failure:

- It is the inability of a system or component to perform required function according to its specification.
- Failure occurs when there is a deviation of the component or system under test from its expected result.

Bug:

- While testing the application or executing the test cases, the test engineer may not get the expected result as per the requirement.
- The bug can occur for the following reasons:
 - Wrong coding: Suppose if we take the Gmail application where we click on the "Inbox" link, and it navigates to the "Draft" page, this is happening because of the wrong coding which is done by the developer, that's why it is a bug.
 - Missing coding: If we take the above example and open the inbox link, we see that it is not there only, which means the feature is not developed.
 - Extra coding: Suppose we have one application form wherein the Name field, the First name, and Last name textbox are needed to develop according to the client's requirement. But, the developers also develop the "Middle name" textbox, which is not needed according to the client's requirements.

Testing:

- Testing is the process of evaluating a system or its component(s) with the intent to find 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 requirements.

Test case:

- A structured test script that describes how a function or feature should be tested, including test steps, expected results preconditions and post conditions.
- Having a test case helps the tester recall the functionality when it comes back for testing after a long gap. Also, if any tester is assigned to test a product already in place then the test cases will help in making sure that the new tester does not miss any scenario.

Test suite:

- With a growing amount of test cases, the need for categorizing them increases as well.
- You don't want to throw all your books on a pile, you'll organize them on shelves, for better accessibility. In a way, test suites are very similar to shelves, as they group test cases together.
- A test suite allows you to categorize test cases in such a way that they match your planning and analysis needs.

Test Data:

- Test Data in Software Testing is the input given to a software program during test execution.
- Test data is used for both positive testing to verify that functions produce expected results for given inputs and for negative testing to test software ability to handle unusual, exceptional or unexpected inputs.
- In a test case where you add a customer to the system the test data might be customer name and address.
- Test data might exist in a separate test data file or in a database.

Test Result:

- Reporting test execution results is very important part of testing, whenever test execution cycle is complete, tester should make a

complete test results report which includes the Test Pass/Fail status of the test cycle.

- If manual testing is done then the test pass/fail result should be captured in an excel sheet and if automation testing is done using automation tool then the HTML or XML reports should be provided to developers.

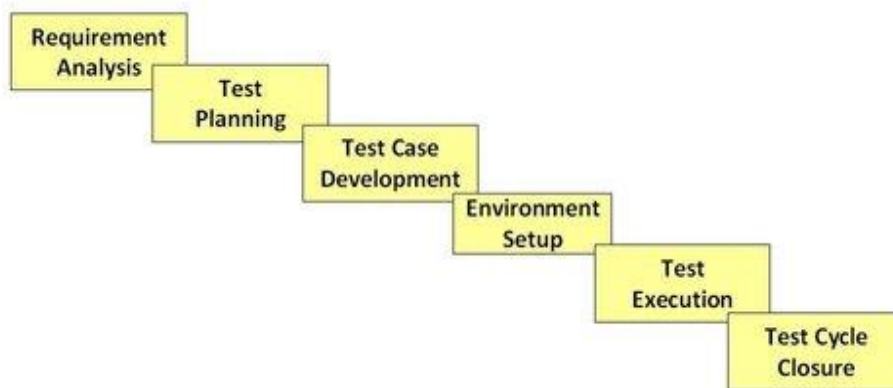
Test Reports:

- A document that summarizes the process and outcome of testing activities at the conclusion of a test period.
- Contains the test manager's recommendations, which in turn are based on the degree to which the test activities attained its objectives. Also called test summary report.

1.1.2 Testing life cycle, Test Exit criteria

Software Testing Life Cycle (STLC) is a sequence of different activities performed during the software testing process.

Phases of STLC:



Requirement Analysis:

- Requirement Analysis is the first step of Software Testing Life Cycle (STLC).

- In this phase quality assurance team understands the requirements like what is to be tested.
- If anything is missing or not understandable then quality assurance team meets with the developers to better understand the detail knowledge of requirement.

Test Planning:

- Test Planning is most efficient phase of software testing life cycle where all testing plans are defined.
 - Test tool selection
 - Test effort estimation
 - Resource planning and determining roles and responsibilities.
 - Training requirement
- This phase gets started once the requirement gathering phase is completed.

Test Case Development:

- The test case development phase gets started once the test planning phase is completed.
- In this phase testing team note down the detailed test cases. Testing team also prepare the required test data for the testing.
- When the test cases are prepared then they are reviewed by quality assurance team.

Test Environment Setup:

- Test Environment Setup decides the software and hardware conditions under which a work product is tested.
- In this process the testing team is not involved. either the developer or the customer creates the testing environment.
- The test team is required to do a readiness check (smoke testing) of the given environment.

- Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
- Setup test Environment and test data.

Test Execution:

- After the test case development and test environment setup test execution phase gets started.
- Testing team start executing test cases based on prepared test cases in the earlier step.
- Document test results, and logs defects for failed cases
- Retest the Defect fixes.
- Test cases updated with results
- Defect reports

Test Closure:

- This is the last stage of STLC in which the process of testing is analysed.

Test Exit criteria:

- Exit criterion is used to determine whether a given test activity has been completed or NOT. Exit criteria can be defined for all of the test activities right from planning, specification and execution.
- Exit criterion should be part of test plan and decided in the planning stage.
- Exit criteria includes following
 - Verify if All tests planned have been run.
 - Verify if the level of requirement coverage has been met.
 - Verify if there are NO Critical or high severity defects that are left outstanding.
 - Verify if all high risk areas are completely tested.
 - Verify if software development activities are completed within the projected cost.

- Verify if software development activities are completed within the projected timelines.

1.1.4. Testing and Debugging:

Testing	Debugging
Testing is the process to find bugs and errors in a software.	Debugging is the process to correct the bugs found during testing.
It is the process to identify the failure of implemented code.	It is the process to give the solution to code failure.
Testing is the display of errors.	Debugging is a process to eliminate the error.
Testing is done by the tester.	Debugging is done by either programmer or developer.
Testing is initiated after the code is written.	Debugging takes place with the execution of a test case.
There is no need of design knowledge in the testing process.	Debugging can't be done without proper design knowledge.

1.1.5 Test Driven Development

Test Driven Development (TDD) is software development approach in which test cases are developed to specify and validate what the code will do.

In simple terms, test cases for each functionality are created and tested first and if the test fails then the new code is written in order to pass the test and making code simple and bug-free.

TDD instructs developers to write new code only if an automated test has failed.

The simple concept of TDD is to write and correct the failed tests before writing new code (before development). This helps to avoid duplication

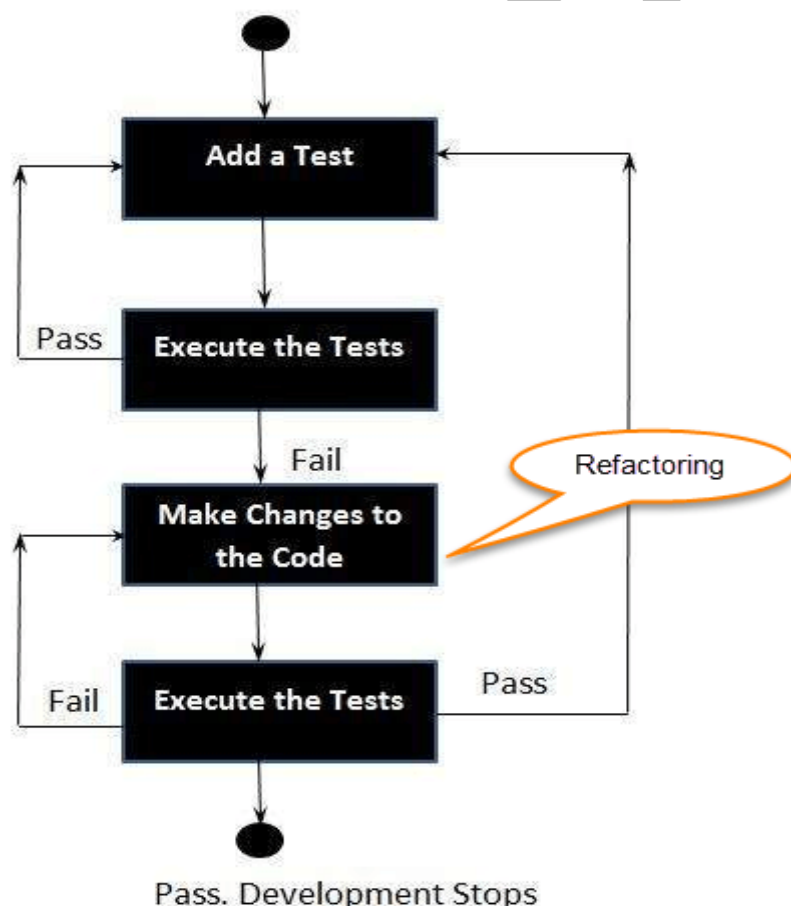
of code as we write a small amount of code at a time in order to pass tests.

Test-Driven development is a process of developing and running automated test before actual development of the application. Hence, TDD sometimes also called as Test First Development.

How to perform TDD Test

Following steps define how to perform TDD test,

1. Add a test.
2. Run all tests and see if any new test fails.
3. Write some code.
4. Run tests and Refactor code.
5. Repeat.



Benefits of TDD:

- Much less debug time
- Code proven to meet requirements
- Tests become Safety Net
- Near zero defects
- Shorter development cycles

1.2 Testing practices:

1.2.1 Overview of testing types:

1. Ad-hoc testing:

Ad hoc Testing is an informal or unstructured software testing type that aims to break the testing process in order to find possible defects or errors at an early possible stage.

Ad hoc testing is done randomly and it is usually an unplanned activity which does not follow any documentation and test design techniques to create test cases.

It is done randomly on any part of application to find defects.

Ad-hoc testing can be achieved with the Software testing technique called Error Guessing. Error guessing can be done by the people having enough experience on the system to "guess" the most likely source of errors.

Ad hoc testing can be performed when there is limited time to do elaborative testing.

Ad hoc testing will be effective only if the tester is knowledgeable of the System Under Test.

Types of Ad-hoc testing:

1. **Buddy Testing:** Two buddies, one from development team and one from test team mutually work on identifying defects in the same module.

Buddy testing helps the testers develop better test cases while development team can also make design changes early. This kind of testing happens usually after completing the unit testing.

2. **Pair Testing:** Two testers are assigned the same modules and they share ideas and work on the same systems to find defects.

One tester executes the tests while another tester records the notes on their findings.

3. **Monkey Testing:** A software testing technique in which the tester enters any random inputs into the software application without predefined test cases and checks the behaviour of the software application, whether it crashes or not. The purpose of Monkey testing is to find the bugs and errors in the software application using experimental techniques.

In Software Engineering, Ad-hoc Testing saves lot of time as it doesn't require elaborate test planning, documentation and Test Case design.

2. Gorilla Testing:

Gorilla Testing is a Software testing technique wherein a module of the program is repeatedly tested to ensure that it is working correctly and there is no bug in that module.

Gorilla Testing is a testing technique in which testers, sometimes developers also join hands with testers to test a particular module thoroughly in all aspects.

Gorilla Testing is performed on specifically few selective modules with few test cases.

Objective of Gorilla testing is to check whether the module is working properly or not.

4. Random testing and Systematic testing:

Random testing is a black-box software testing technique where programs are tested by generating random, independent inputs. Results of the output are compared against software specifications to verify that the test output is pass or fail.

Example:

Consider the following c++ program

```
int myAbs(int x) {  
    if (x > 0) {  
        return x;  
    }  
    else {  
        return x; // bug: should be '-x'  
    }  
}
```

Now the random tests for this function could be {123, 36, -35, 48, 0}. Only the value '-35' triggers the bug.

If there is no reference implementation to check the result, the bug still could go unnoticed.

5. Static testing and Dynamic Testing

Static testing	Dynamic Testing
It is performed in the early stage of the software development.	It is performed at the later stage of the software development.
In static testing whole code is not executed.	In dynamic testing whole code is executed.
Static testing prevents the defects.	Dynamic testing finds and fixes the defects.

Static testing is performed before code deployment.	Dynamic testing is performed after code deployment.
Static testing is less costly.	Dynamic testing is highly costly.
Static Testing involves checklist for testing process.	Dynamic Testing involves test cases for testing process.
It includes walkthroughs, code review, inspection etc.	It involves functional and non-functional testing.
It generally takes shorter time.	It usually takes longer time as it involves running several test cases.
It can discover variety of bugs.	It expose the bugs that are explorable through execution hence discover only limited type of bugs.
Static Testing may complete 100% statement coverage in comparably less time.	While dynamic testing only achieves less than 50% statement coverage.
Example: Verification	Example: Validation

6. Functional, Non-functional and Behavioural Testing:

Functional testing:

Functional testing is a type of testing which verifies that each function of the software application operates in conformance with the requirement specification.

Every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results.

This testing involves checking of User Interface, APIs, Database, security, client/ server applications and functionality of the Application Under Test.

The testing can be done either manually or using automation.

Non-functional testing:

Non-functional testing is a type of testing to check non-functional aspects (performance, usability, reliability, etc.) of a software application.

It is explicitly designed to test the readiness of a system as per non-functional parameters which are never addressed by functional testing.

A good example of non-functional test would be to check how many people can simultaneously login into a software.

Non-functional testing is equally important as functional testing and affects client satisfaction.

Usability testing:

Usability testing, a non-functional testing technique that is a measure of how easily the system can be used by end users.

It is difficult to evaluate and measure but can be evaluated based on the below parameters:

- Level of Skill required to learn/use the software. It should maintain the balance for both novice and expert user.
- Time required to get used to in using the software.
- The measure of increase in user productivity if any.
- Assessment of a user's attitude towards using the software.

Configuration Testing:

Configuration Testing is a software testing technique in which the software application is tested with multiple combinations of software and hardware in order to evaluate the functional requirements and find out optimal configurations under which the software application works without any defects or flaws.

Example of a Desktop Application:

Let's consider a 3 tier Desktop application which is developed using Asp.Net and consists of Client, Business Logic Server and Database Server where each component supports below-mentioned platforms.

- Client Platform - Windows XP, Window7 OS, windows 8 OS, etc.
- Database –SQL Server 2008, SQL Server 2012, etc.

Configuration testing is not only restricted to Software but also applicable for Hardware which is why it is also referred as a Hardware configuration testing, where we test different hardware devices like Printers, Scanners, Web cams, etc. that support the application under test.

Configuration testing is not only restricted to Software but also applicable for Hardware which is why it is also referred as a Hardware configuration testing, where we test different hardware devices like Printers, Scanners, Web cams, etc. that support the application under test.

7. Compatibility Testing:

Compatibility Testing is a type of Software testing to check whether your software is capable of running on different hardware, operating systems, applications, network environments or Mobile devices.

Backward Compatibility Testing:

- Backward Compatibility Testing is a technique to verify the behaviour and compatibility of the developed hardware or software with their older versions of the hardware or software.
- Backward compatibility testing is much predictable as all the changes from the previous versions are known.

Forward Compatibility Testing:

- Forward Compatibility Testing is a process to verify the behaviour and compatibility of the developed hardware or software with the newer versions of the hardware or software.
- Forward compatibility testing is a bit hard to predict as the changes that will be made in the newer versions are not known.

1.2.2. Black box testing:

Black box testing is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products.

Tester has no knowledge of internal code structure, implementation details and internal paths.

Equivalence partitioning:

It is often seen that many type of inputs work similarly so instead of giving all of them separately we can group them together and test only one input of each group.

if a test case in one class results in some error, other members of class would also result into same error.

The technique involves two steps:

1. Identification of equivalence class – Partition any input domain into minimum two sets: valid values and invalid values.
 - For example, if the valid range is 0 to 100 then select one valid input like 49 and one invalid like 104.
2. Generating test cases –
 - To each valid and invalid class of input assign unique identification number.
 - Write test case covering all valid and invalid test case considering that no two invalid inputs mask each other.

Example

Input conditions are valid between 1 to 10 and 20 to 30

Hence there are five equivalence classes:

--- to 0 (invalid)

1 to 10 (valid)

11 to 19 (invalid)

20 to 30 (valid)

31 to --- (invalid)

You select values from each class, i.e.,

-2, 3, 15, 25, 45

Boundary Value Analysis (BVA):

It is generally seen that a large number of errors occur at the boundaries of the defined input values rather than the centre. It is also known as BVA and gives a selection of test cases which exercise bounding values.

This software testing technique base on the principle that, if a system works well for these particular values then it will work perfectly well for all values which comes between the two boundary values.

If an input condition is restricted between values x and y, then the test cases should be designed with values x and y as well as values which are above and below x and y.

Example:

Input condition is valid between 1 to 10

Boundary values 0,1,2 and 9,10,11 must be tested.

Decision Table Testing:

A Decision Table is a tabular representation of inputs versus rules/cases/test conditions.

It is a very effective tool used for both complex software testing and requirements management.

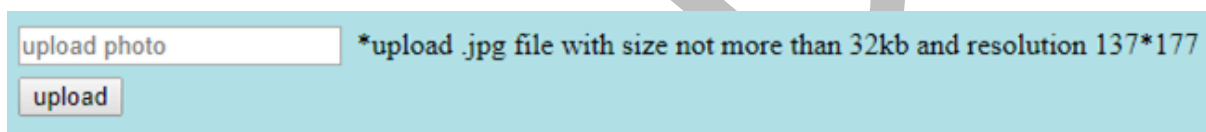
Decision table helps to check all possible combinations of conditions for testing and testers can also identify missed conditions easily.

Decision table testing is a software testing technique used to test system behaviour for different input combinations.

This is a systematic approach where the different input combinations and their corresponding system Output are captured in a tabular form.

Example:

Consider a dialogue box which will ask the user to upload photo with certain conditions.



If any of the conditions fail the system will throw corresponding error message stating the issue and if all conditions are met photo will be updated successfully.

Conditions	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
Format	.jpg	.jpg	.jpg	.jpg	Not .jpg	Not .jpg	Not .jpg
Size	Less than 32kb	Less than 32kb	>= 32kb	>= 32kb	Less than 32kb	Less than 32kb	>= 32kb
resolution	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177	137*177
Output	Photo uploaded	Error message resolution mismatch	Error message size mismatch	Error message size and resolution mismatch	Error message for format mismatch	Error message format and resolution mismatch	Error message for format and size mismatch

For this condition, we can create 7 different test cases and ensure complete coverage based on the above table.

The main disadvantage is that when the number of input increases the table will become more complex.

1.2.3. White box testing:

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design.

Tester analyse the internal structures the used data structures, internal design, code structure and the working of the software.

In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing.

Black-box testing, involves testing from an external or end-user type perspective. On the other hand, White box testing is based on the inner workings of an application.

Code coverage testing techniques:

Code Coverage testing is determining how much code is being tested. It can be calculated using the formula:

$$\text{Code Coverage} = \frac{\text{Number of executed statements}}{\text{Total Number statements}} * 100$$

1.Statement Coverage Testing:

- In this technique, the aim is to traverse all statement at least once. Hence, each line of code is tested.
- It is used to calculate the total number of executed statements in the source code out of total statements present in the source code.
- Since all lines of code are covered, helps in pointing out faulty code.

2.Branch Coverage Testing:

- Branch Coverage Testing is performed on each and every part of the code, where branching occurs. For instance, the conditional statements and the loop statements in the program, which gives more than one possible result when executed.
- Steps to implement Branch coverage testing
 1. The first step in the implementation of Branch Coverage Testing is identification of branches.
 2. The next step is to make a list of the results or outcomes of each branch in the code. A branch can possibly have two or three outcomes when the branch is found to be an 'if' conditional, and more than that if the branch is found to be a 'switch case' conditional statement. And so, it is essential to not miss any potential branch or the branch's result in this process.
 3. The final step is to validate the test execution on all the branches and fetch the results. These results should match the expected results.
- If there is any deviation found, it simply means that functionality is defected and needs to be recoded to match the requirement created by the client.

3. Decision Coverage Testing:

- It is a white box testing technique which reports the true or false outcomes of each Boolean expression of the source code.
- The goal of decision coverage testing is to cover and validate all the accessible source code by checking and ensuring that each branch of every possible decision point is executed at least once.

Data flow testing:

Data flow testing is a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of variables or data objects.

Dataflow Testing focuses on the points at which variables receive values and the points at which these values are used.

Data Flow testing helps us to pinpoint any of the following issues:

- A variable that is declared but never used within the program.
- A variable that is used but never declared.
- A variable that is defined multiple times before it is used.
- Deallocating a variable before it is used.

Example:

Consider the following programme.

1. read x, y;
2. if(x>y)
3. a = x+1
- else
4. a = y-1
5. print a;

Define/use of variables of above example:

Variable	Defined at node	Used at node
x	1	2, 3
y	1	2, 4
a	3, 4	5

1.2.4. Levels of testing:

There are many different testing levels which help to check behaviour and performance for software testing.

The purpose of Levels of testing is to make software testing systematic and easily identify all possible test cases at a particular level.

1. Unit Testing:

- 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. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.
- 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.

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.
- A typical software project consists of multiple software modules, coded by different programmers.
- The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated.

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.
- Acceptance Testing arises once software has undergone Unit, Integration and System testing
- Developers might have built software based on requirements document by their own understanding and further required changes during development may not be effectively communicated to them, so for testing whether the final product is accepted by client/end-user, user acceptance testing is needed.

1.2.5. Smoke testing, Sanity Testing and Regression Testing:

Smoke testing:

Smoke Testing is a software testing technique performed post software build to verify that the critical functionalities of software are working fine.

It is executed before any detailed functional or non-functional tests are executed. The main purpose of smoke testing is to reject a software application with defects so that QA team does not waste time testing broken software application.

In Smoke Testing, the test cases chose to cover the most important functionality or component of the system.

The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system are working fine.

For Example, a typical smoke test would be - Verify that the application launches successfully, check that the GUI is responsive ... etc.

Sanity testing:

Sanity testing is a kind of Software Testing performed after receiving a software build, with minor changes in code, or functionality, to ascertain that the bugs have been fixed and no further issues are introduced due to these changes.

The goal is to determine that the proposed functionality works roughly as expected.

If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.

The objective is "not" to verify thoroughly the new functionality but to determine that the developer has applied some rationality (sanity) while producing the software.

For instance, if your scientific calculator gives the result of $2 + 2 = 5$, Then, there is no point testing the advanced functionalities like $\sin 30 + \cos 50$.

Sanity testing is done to check the bugs identified in smoke testing have been fixed or not.

Regression Testing:

Regression testing 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.

Regression testing is needed, when a new feature is added to the software application and for defect fixing as well as performance issue fixing.

We need to first debug the code to identify the bugs. Once the bugs are identified, required changes are made to fix it, then the regression testing is done by selecting relevant test cases from the test suite that covers both modified and affected parts of the code.

1.2.6. Practices for static testing:

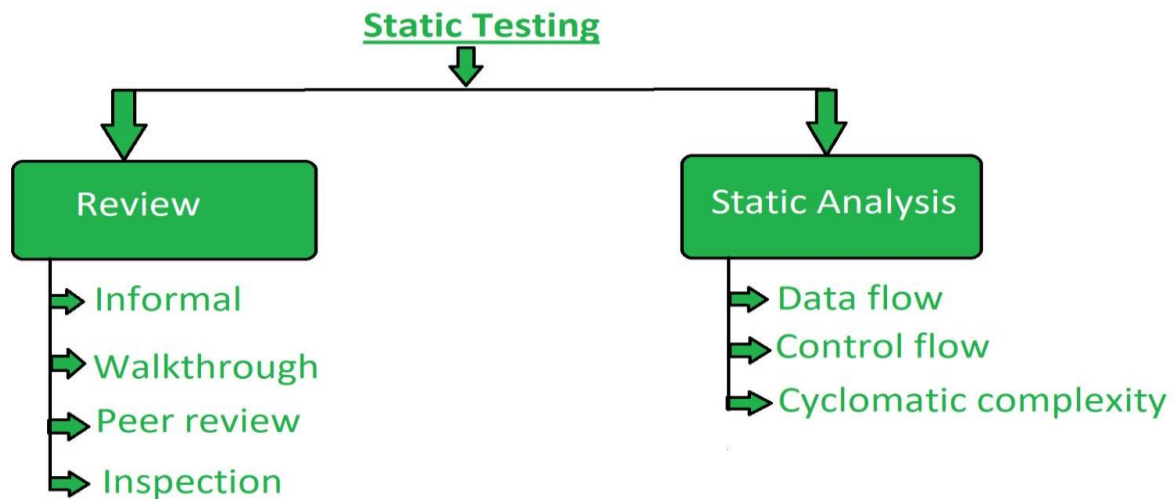
Static Testing is a type of a Software Testing method which is performed to check the defects in software without actually executing the code of the software application. Whereas in Dynamic Testing checks the code is executed to detect the defects.

Static testing is performed in early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily.

The errors that can't not be found using Dynamic Testing, can be easily found by Static Testing.

Static Testing Techniques:

There are mainly two type techniques used in Static Testing:



Review:

A review in a Static Testing is a process or meeting conducted to find the potential defects in the design of any program.

Another significance of review is that all the team members get to know about the progress of the project and sometimes the diversity of thoughts may result in excellent suggestions.

1. Informal:

In informal review the creator of the documents put the contents in front of audience and everyone gives their opinion and thus defects are identified in the early stage.

2. Walkthrough:

It is basically performed by experienced person or expert to check the defects so that there might not be problem further in the development or testing phase.

3. Peer review:

Peer review means checking documents of one-another to detect and fix the defects. It is basically done in a team of colleagues.

4. Inspection:

Inspection is basically the verification of document the higher authority like the verification of software requirement specifications (SRS).

Static Analysis:

Static Analysis includes the evaluation of the code quality that is written by developers.

Different tools are used to do the analysis of the code and comparison of the same with the standard.

It also helps in following identification of following defects:

- Unused variables
- Dead code
- Infinite loops
- Variable with undefined value
- Wrong syntax

Data Flow: It involves gathering information about the possible set of values.

Control Flow: Control flow is basically how the statements or instructions are executed.

Cyclomatic Complexity: Cyclomatic complexity is the measurement of the complexity of the program that is basically related to the number of independent paths in the control flow graph of the program.