UNIT - 5 : TESTING TACTICS, SOFTWARE MEASUREMENT AND ESTIMATION

WHAT IS SOFTWARE TESTING?

- Software testing is the act of examining the artifacts and the behavior of the software und
  verification.” But let’s break that down a bit.
- In simpler terms, think of software testing as a quality check for software. Just as you’d ins g
  it, software testing ensures that an application or system works correctly and is glitches-free.
- It’s all about ensuring the software does what it should and provides users with a seamless experience. By testing, we ensure potential issues are spotted and fixed before the software reaches your hands.
- Software testing is an **important process** in the **software development lifecycle** . It involves **verifying** and **validating** that a **software application** is free of bugs, meets the technical requirements set by its **design** and **development** , and satisfies user requirements efficiently and effectively.
- This process ensures that the application can handle all exceptional and boundary cases, providing a robust and reliable user experience. By systematically identifying and fixing issues, software testing helps deliver high-quality software that performs as expected in various scenarios.

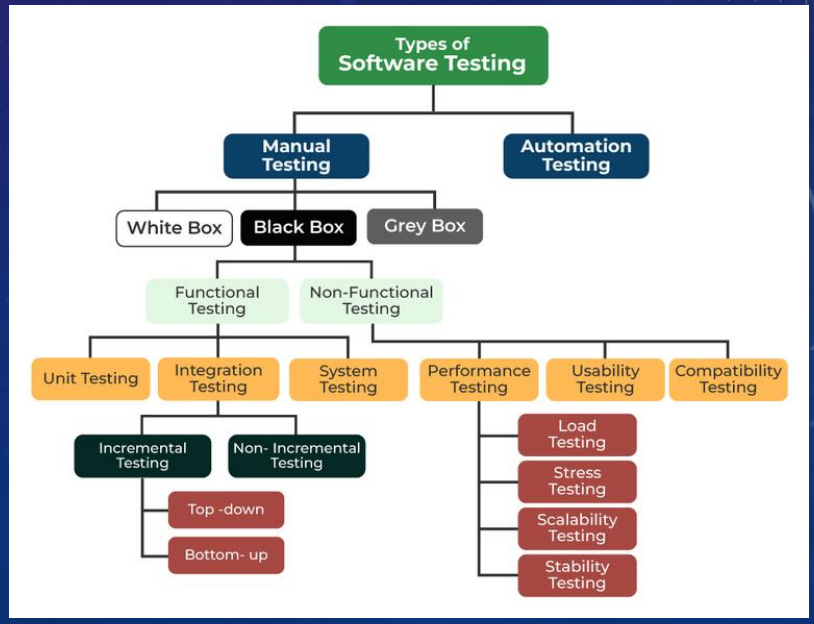
- **Software testing can be divided into two steps**
- **Verification:** It refers to the set of tasks that ensure that the software correctly implements a specific function. It means “Are we building the product right?”.
- **Validation:** It refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. It means “Are we building the right product?”.
- **Importance of Software Testing**
- **Defects can be identified early:** Software testing is important because if there are any bugs they can be identified early and can be fixed before the delivery of the software.
- **Improves quality of software:** Software Testing uncovers the defects in the software, and fixing them improves the quality of the software.
- **Increased customer satisfaction:** Software testing ensures reliability, security, and high performance which results in saving time, costs, and customer satisfaction.
- **Helps with scalability:** Software testing type non-functional testing helps to identify the scalability issues and the point where an application might stop working.
- **Saves time and money:** After the application is launched it will be very difficult to trace and resolve the issues, as performing this activity will incur more costs and time. Thus, it is better to conduct software testing at regular intervals during software development.

SOFTWARE TESTING CAN BE BROADLY CLASSIFIED INTO 3 TYPES:

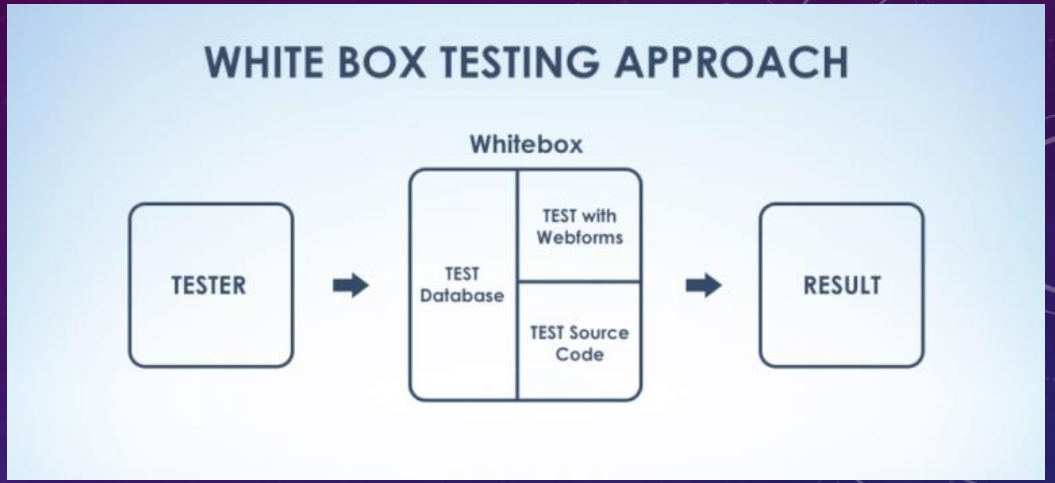
Functional testing : It is a type of software testing that validates the software systems against the functional requirements. It is performed to check whether the application is working as per the software's functional requirements or not. Various types of functional testing are Unit testing, Integration testing, System testing, Smoke testing, and so on.

Non-functional testing : It is a type of software testing that checks the application for non-functional requirements like performance, scalability, portability, stress, etc. Various types of non-functional testing are Performance testing, Stress testing, Usability Testing, and so on.

Maintenance testing **:** It is the process of changing, modifying, and updating the software to keep up with the customer's needs. It involves **regression testing** that verifies that recent changes to the code have not adversely affected other previously working parts of the software.



WHITE-BOX TESTING



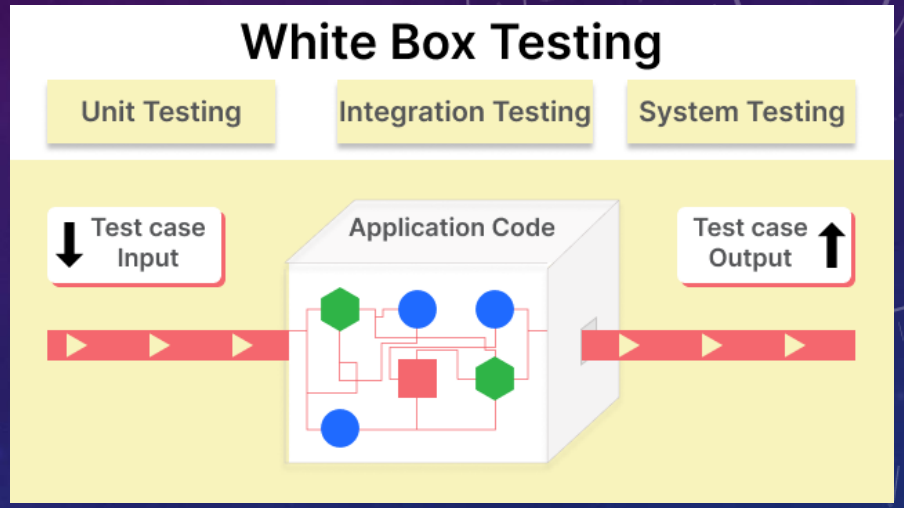
- **White box testing** techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing clear box testing or structural testing. White Box Testing is also known as transparent testing or open box testing.
- White box testing is a software testing technique that involves testing the internal structure and workings of a software application . The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

WHAT DOES WHITE BOX TESTING FOCUS ON

- White box testing is also known as structural testing or code-based testing, and it is used to test the software's internal logic, flow, and structure. The tester creates test cases to examine the code paths and logic flows to ensure they meet the specified requirements.
- Before we move in depth of the white box testing do you known that their are many different type of testing used in industry and some automation testing tools are there which automate the most of testing so if you wish to learn the latest industry level tools then you check-out our manual to automation testing course in which you will learn all these concept and tools
- White box testing uses detailed knowledge of a software's inner workings to create very specific test cases.
- **Path Checking:** Examines the different routes the program can take when it runs. Ensures that all decisions made by the program are correct, necessary, and efficient.
- **Output Validation:** Tests different inputs to see if the function gives the right output each time.
- **Security Testing:** Uses techniques like static code analysis to find and fix potential security issues in the software. Ensures the software is developed using secure practices.
- **Loop Testing:** Checks the loops in the program to make sure they work correctly and efficiently. Ensures that loops handle variables properly within their scope.
- **Data Flow Testing:** Follows the path of variables through the program to ensure they are declared, initialized, used, and manipulated correctly.

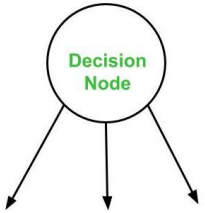
TYPES OF WHITE BOX TESTING

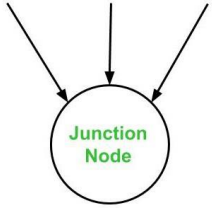
- White box testing can be done for different purposes. The three main types are:
- Unit Testing
- Integration Testing
- Regression Testing
- **Unit Testing**
- Checks if each part or function of the application works correctly.
- Ensures the application meets design requirements during development.
- **Integration Testing**
- Examines how different parts of the application work together.
- Done after unit testing to make sure components work well both alone and together.
- **Regression Testing**
- Verifies that changes or updates don't break existing functionality.
- Ensures the application still passes all existing tests after updates.
-

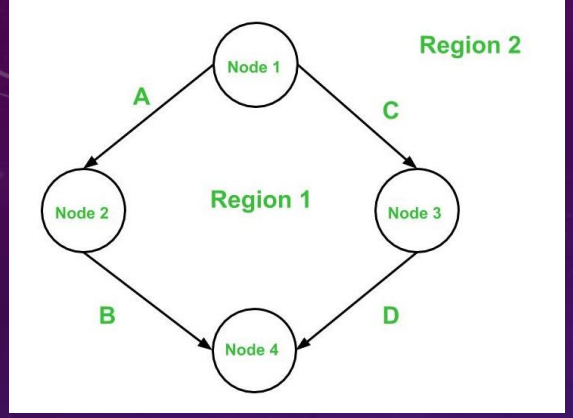


BASIS PATH TESTING

- **Basis Path Testing** is a white-box testing technique based on the control structure of a program or a module. Using this structure, a control flow graph is prepared and the various possible paths present in the graph are executed as a part of testing. Therefore, by definition, Basis path testing is a technique of selecting the paths in the control flow graph, that provide a basis set of execution paths through the program or module. Since this testing is based on the control structure of the program, it requires complete knowledge of the program's structure. To design test cases using this technique, four steps are followed :
- Construct the Control Flow Graph
- Compute the Cyclomatic Complexity of the Graph
- Identify the Independent Paths
- Design Test cases from Independent Paths
- **Junction Node –** a node with more than one arrow entering it.
- **Decision Node –** a node with more than one arrow leaving it.
- **Region –** area bounded by edges and nodes (area outside the graph is also counted as a region.).
-

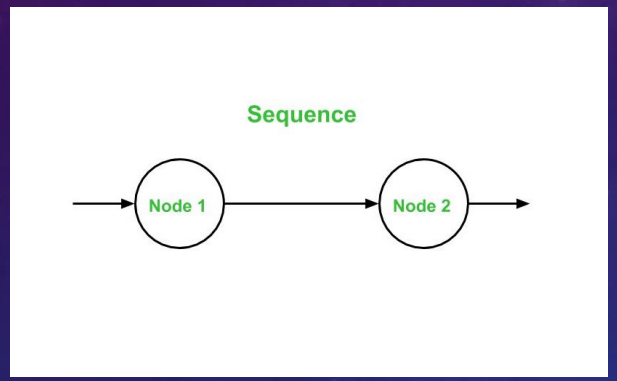




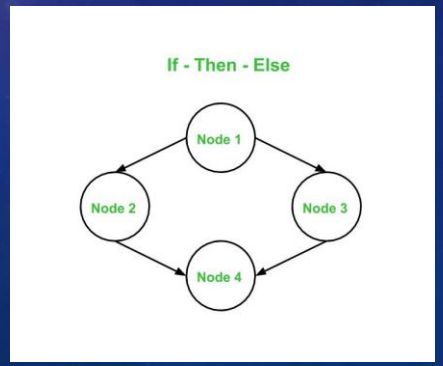


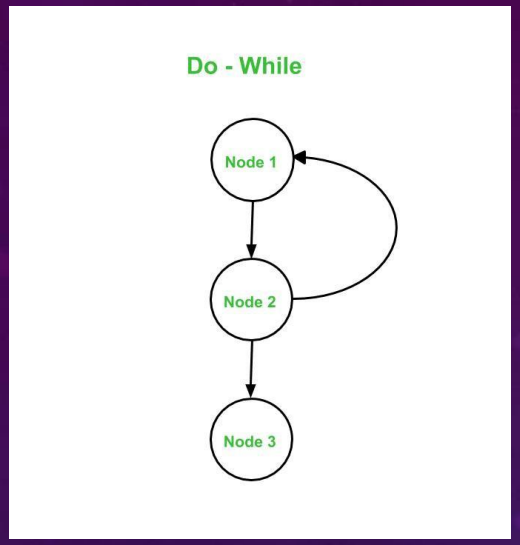
Below are the **notations** used while constructing a flow graph

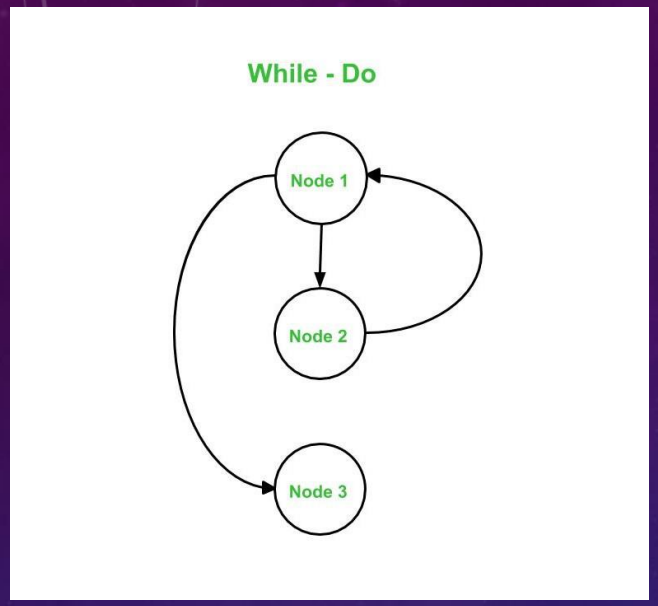
Sequential Statements –

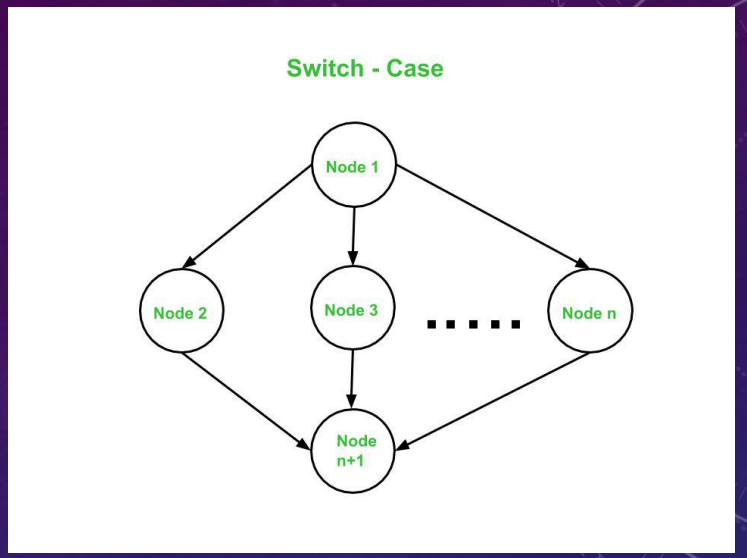


If – Then – Else









- **Do – While**

While – Do

Switch – Case

1.Maintenance Testing – When a software is modified, it is still necessary to test the changes made in the software which as a result, requires path testing.

2.Unit Testing – When a developer writes the code, he or she tests the structure of the program or module themselves first. This is why basis path testing requires enough knowledge about the structure of the code.

3.Integration Testing – When one module calls other modules, there are high chances of Interface errors. In order to avoid the case of such errors, path testing is performed to test all the paths on the interfaces of the modules.

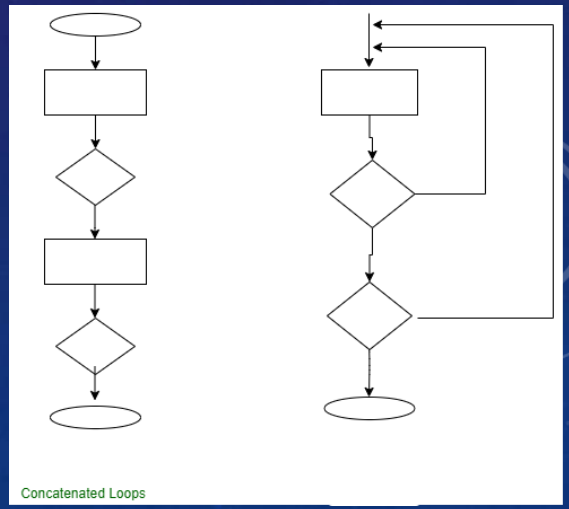
4.Testing Effort – Since the basis path testing technique takes into account the complexity of the software (i.e., program or module) while computing the cyclomatic complexity, therefore it is intuitive to note that testing effort in case of basis path testing is directly proportional to the complexity of the software or program.

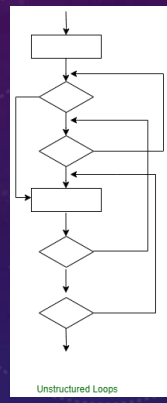
CONTROL STRUCTURE TESTING

- **Condition Testing :** Condition testing is a test cased design method, which ensures that the logical condition and decision statements are free from errors. The errors present in logical conditions can be incorrect boolean operators, missing parenthesis in a booleans expression, error in relational operators, arithmetic expressions, and so on. The common types of logical conditions that are tested using condition testing are-
- A relation expression, like E1 op E2 where ‘E1’ and ‘E2’ are arithmetic expressions and ‘OP’ is an operator.
- A simple condition like any relational expression preceded by a NOT (~) operator. For example, (~E1) where ‘E1’ is an arithmetic expression and ‘a’ denotes NOT operator.
- A compound condition consists of two or more simple conditions, Boolean operator, and parenthesis. For example, (E1 & E2)|(E2 & E3) where E1, E2, E3 denote arithmetic expression and ‘&’ and ‘|’ denote AND or OR operators.
- A Boolean expression consists of operands and a Boolean operator like ‘AND’, OR, NOT. For example, ‘A|B’ is a Boolean expression where ‘A’ and ‘B’ denote operands and | denotes OR operator.
- **2. Data Flow Testing :** The data flow test method chooses the test path of a program based on the locations of the definitions and uses all the variables in the program. The data flow test approach is depicted as follows suppose each statement in a program is assigned a unique statement number and that theme function cannot modify its parameters or global variables. For example, with S as its statement number.

- If statement S is an if loop statement, them its DEF set is empty and its USE set depends on the state of statement S. The definition of the variable X at statement S is called the line of statement S' if the statement is any way from S to statement S' then there is no other definition of X. A definition use (DU) chain of variable X has the form [X, S, S'], where S and S' denote statement numbers, X is in DEF(S) and USE(S'), and the definition of X in statement S is line at statement S'. A simple data flow test approach requires that each DU chain be covered at least once. This approach is known as the DU test approach. The DU testing does not ensure coverage of all branches of a program. However, a branch is not guaranteed to be covered by DU testing only in rare cases such as then in which the other construct does not have any certainty of any variable in its later part and the other part is not present. Data flow testing strategies are appropriate for choosing test paths of a program containing nested if and loop statements. **3. Loop Testing :** Loop testing is actually a white box testing technique. It specifically focuses on the validity of loop construction. Following are the types of loops.
- **Simple Loop –** The following set of test can be applied to simple loops, where the maximum allowable number through the loop is n.
  - Skip the entire loop.
  - Traverse the loop only once.
  - Traverse the loop two times.
  - Make p passes through the loop where p<n.
  - Traverse the loop n-1, n, n+1 times.

- **Concatenated Loops –** If loops are not dependent on each other, contact loops can be tested using the approach used in simple loops. if the loops are interdependent, the steps are followed in nested loops.

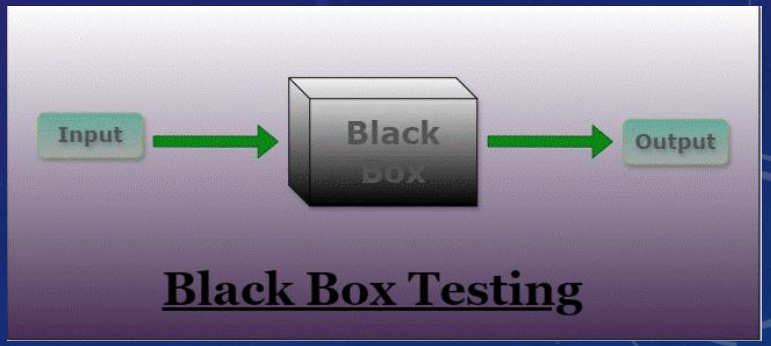




- **Nested Loops –** Loops within loops are called as nested loops. when testing nested loops, the number of tested increases as level nesting increases. The following steps for testing nested loops are as follows-
  - Start with inner loop. set all other loops to minimum values.
  - Conduct simple loop testing on inner loop.
  - Work outwards.
  - Continue until all loops tested.
- **Unstructured loops –** This type of loops should be redesigned, whenever possible, to reflect the use of unstructured the structured programming constructs.

BLACK BOX TESTING

- Black Box Testing is an important part of making sure software works as it should. Instead of peeking into the code, testers check how the software behaves from the outside, just like users would. This helps catch any issues or bugs that might affect how the software works.
- This simple guide gives you an overview of what Black Box Testing is all about and why it matters in software development
- Black-box testing is a type of software testing in which the tester is not concerned with the software's internal knowledge or implementation details but rather focuses on validating the functionality based on the provided specifications or requirements.
- **Types Of Black Box Testing**
- The following are the several categories of black box testing:
- Functional Testing
- Regression Testing
- Nonfunctional Testing (NFT)



- **Functional Testing**
- Functional testing is defined as a type of testing that verifies that each function of the software application works in conformance with the requirement and specification.
- This testing is not concerned with the source code of the application. Each functionality of the software application is tested by providing appropriate test input, expecting the output, and comparing the actual output with the expected output.
- This testing focuses on checking the user interface, APIs, database, security, client or server application, and functionality of the Application Under Test. Functional testing can be manual or automated. It determines the system's software functional requirements.
- **Regression Testing**
- Regression Testing is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure that no new errors have been introduced in the software after the modifications have been made.
- Regression means the return of something and in the software field, it refers to the return of a bug. It ensures that the newly added code is compatible with the existing code.
- In other words, a new software update has no impact on the functionality of the software. This is carried out after a system maintenance operation and upgrades.

- **Nonfunctional Testing**
- Non-functional testing is a software testing technique that checks the non-functional attributes of the system.
- Non-functional testing is defined as a type of software testing to check non-functional aspects of a software application.
- It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.
- Non-functional testing is as important as functional testing.
- Non-functional testing is also known as NFT. This testing is not functional testing of software. It focuses on the software's performance, usability, and scalability.
- **Advantages of Black Box Testing**
- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user's or client's point of view.
- Test cases are easily reproducible.
- It is used to find the ambiguity and contradictions in the functional specifications

- **Disadvantages of Black Box Testing**
- There is a possibility of repeating the same tests while implementing the testing process.
- Without clear functional specifications, test cases are difficult to implement.
- It is difficult to execute the test cases because of complex inputs at different stages of testing.
- Sometimes, the reason for the test failure cannot be detected.
- Some programs in the application are not tested.
- It does not reveal the errors in the control structure.
- Working with a large sample space of inputs can be exhaustive and consumes a lot of time.
- **Black Box and White Box Testing**
- Black box testing is a testing technique in which the internal workings of the software are not known to the tester. The tester only focuses on the input and output of the software. Whereas, White box testing is a testing technique in which the tester has knowledge of the internal workings of the software, and can test individual code snippets, algorithms and methods.
- **Testing objectives:** Black box testing is mainly focused on testing the functionality of the software, ensuring that it meets the requirements and specifications. White box testing is mainly focused on ensuring that the internal code of the software is correct and efficient.
- **Knowledge level** : Black box testing does not require any knowledge of the internal workings of the software, and can be performed by testers who are not familiar with programming languages.

- **Testing methods:** Black box testing uses methods like equivalence partitioning, boundary value analysis, and error guessing to create test cases. Whereas, white box testing uses methods like control flow testing, data flow testing and statement coverage.
- **Scope** : Black box testing is generally used for testing the software at the functional level. White box testing is used for testing the software at the unit level, integration level and system level.
- **Grey Box Testing**
- Gray Box Testing is a software testing technique that is a combination of the Black Box Testing technique and the White Box Testing technique.
- In the Black Box Testing technique, the tester is unaware of the internal structure of the item being tested and in White Box Testing the internal structure is known to the tester.
- The internal structure is partially known in Gray Box Testing.
- This includes access to internal data structures and algorithms to design the test cases.
- Gray Box Testing is named so because the software program is like a semitransparent or gray box inside which the tester can partially see.
- It commonly focuses on context-specific errors related to web systems.
- **Objectives of Gray Box Testing**
- **To provide combined advantages of both black box testing and white box testing.**
- **To combine the input of developers as well as testers.**
- **To improve overall product quality.**

SIZE ORIENTED METRICS

- Size-oriented metrics play a fundamental role in software development by measuring and comparing software project sizes based on various factors. This article explores the concept of size-oriented metrics, their advantages, and disadvantages, and provides an example of how they are applied in software organizations.
- **What is Size Oriented Metrics?**
- **Size-oriented metrics** are derived by normalizing quality and productivity Point Metrics measures by considering the **the** size of the software that has been produced. The organization builds a simple record of size measure for the software projects. It is built on past experiences of organizations. It is a direct measure of software. This metric **measure** is one of the simplest and earliest metrics that is used for computer programs to measure size. Size Oriented Metrics are also used for measuring and comparing the productivity of programmers. It is a direct measure of a Software. The size measurement is based on lines of code computation. The lines of code are defined as one line of text in a source file. While counting lines of code, the simplest standard is:
- Don’t count blank lines
- Don’t count comments
- Count everything else
- The size-oriented measure is not a universally accepted method

ADVANTAGES OF SIZE-ORIENTED METRICS

- This measure is dependent upon programming language.
- This method is well designed upon programming language.
- It does not accommodate non-procedural languages.
- Sometimes, it is very difficult to estimate LOC in early stage of development.
- Though it is simple to measure but it is very hard to understand it for users.
- It cannot measure size of specification as it is defined on code.
- **Conclusion**
- In conclusion, size-oriented measures are a useful tool for making software because they are easy to use, standardised, and can be used to estimate. They do, however, have some problems, such as being dependent on the computer language and possibly having trouble with early-stage estimates. Companies can make better choices and improve their software development processes if they know these measures and how to use them.

FUNCTION ORIENTED METRICS

- Function-oriented metrics are used to measure software characteristics based on the functionality provided to the user. These metrics focus on the functional aspects of software systems rather than on the implementation details or the code itself. This article focuses on discussing function-oriented metrics in detail.
- **What are Function-Oriented Metrics?**
- Function-oriented metrics is a method that was developed by Albrecht in 1979 for IBM (International Business Machine). He suggested a measure known as Function points that are derived using an empirical relationship based on countable measures of software's information or requirements domain and assessments of the complexity of software.
- Function-oriented metrics are used to assess the software size and complexity based on the functionality provided by the software to its users.
- Function-Oriented Metrics are also known as Function Point Models.
- These metrics focus on the software's behavior and the services it provides rather than its internal structure or code complexity.
- They are valuable for estimating resources, scheduling, and monitoring project progress.
- The effectiveness of function-oriented metrics depends upon the type of software, development methodology, and project context.

- **Purpose of Function-Oriented Metrics**
- Function-Oriented metrics have several important purposes:
- **Size Estimation:** They provide a standard method to measure the size of software based on the functionality instead of Lines of Code (LOC) or any other implementation details.
- **Project Planning and Management:** They aid in project planning by providing insights into the complexity and scope of the project.
- **Cost Estimation:** They aid in estimating the size of the software in terms of functionality, thus contributing to cost estimation for software development.
- **Quality Assessment:** Function-oriented metrics can indirectly assess the software quality by identifying the potential risks and complexities associated with the functional requirements.
- **Productivity Measurement:** Fucntion-oriented metrics help measure and compare the productivity of the organization across different projects or teams.
-

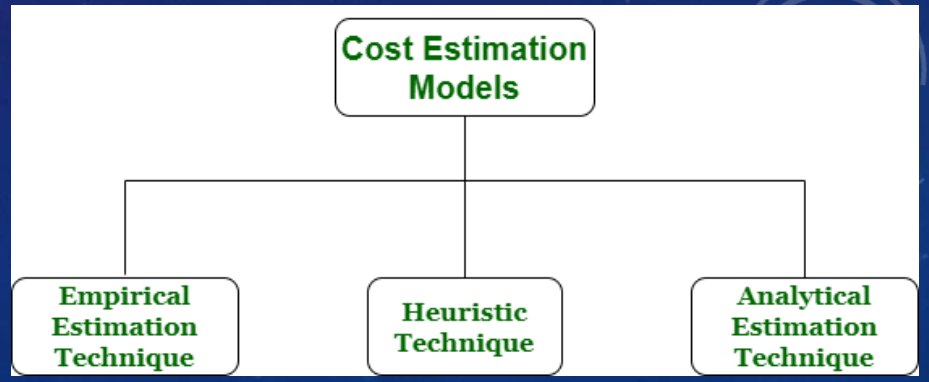
METRICS FOR SOFTWARE QUALITY

- In Software Engineering, Software Measurement is done based on some Software Metrics where these software metrics are referred to as the measure of various characteristics of a Software.
- In Software engineering Software Quality Assurance (SAQ) assures the quality of the software. A set of activities in SAQ is continuously applied throughout the software process. Software Quality is measured based on some software quality metrics.
- There is a number of metrics available based on which software quality is measured. But among them, there are a few most useful metrics which are essential in software quality measurement. They are –
- Code Quality
- Reliability
- Performance
- Usability
- Correctness
- Maintainability
- Integrity
- Security

- **Now let's understand each quality** metric **in detail –**
- **1. Code Quality –** Code quality metrics measure the quality of code used for software project development. Maintaining the software code quality by writing Bug-free and semantically correct code is very important for good software project development. In code quality, both Quantitative metrics like the number of lines, complexity, functions, rate of bugs generation, etc, and Qualitative metrics like readability, code clarity, efficiency, and maintainability, etc are measured.
- **2. Reliability –** Reliability metrics express the reliability of software in different conditions. The software is able to provide exact service at the right time or not checked. Reliability can be checked using Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR).
- **3. Performance –** Performance metrics are used to measure the performance of the software. Each software has been developed for some specific purposes. Performance metrics measure the performance of the software by determining whether the software is fulfilling the user requirements or not, by analyzing how much time and resource it is utilizing for providing the service.
- **4. Usability –** Usability metrics check whether the program is user-friendly or not. Each software is used by the end-user. So it is important to measure that the end-user is happy or not by using this software.
- **5. Correctness –** Correctness is one of the important software quality metrics as this checks whether the system or software is working correctly without any error by satisfying the user. Correctness gives the degree of service each function provides as per developed.
- **6. Maintainability –** Each software product requires maintenance and up-gradation. Maintenance is an expensive and time-consuming process. So if the software product provides easy maintainability then we can say software quality is up to mark. Maintainability metrics include the time required to adapt to new features/functionality, Mean Time to Change (MTTC), performance in changing environments, etc

EMPIRICAL ESTIMATION MODELS

- **Cost estimation** simply means a technique that is used to find out the cost estimates. The cost estimate is the financial spend that is done on the efforts to develop and test software in Software Engineering. Cost estimation models are some mathematical algorithms or parametric equations that are used to estimate the cost of a product or a project. Various techniques or models are available for cost estimation, also known as Cost Estimation Models.
- **Empirical Estimation Technique –** Empirical estimation is a technique or model in which empirically derived formulas are used for predicting the data that are a required and essential part of the software project planning step. These techniques are usually based on the data that is collected previously from a project and also based on some guesses, prior experience with the development of similar types of projects, and assumptions. It uses the size of the software to estimate the effort. In this technique, an educated guess of project parameters is made. Hence, these models are based on common sense. However, as there are many activities involved in empirical estimation techniques, this technique is formalized. For example Delphi technique and Expert Judgement technique.



- **Heuristic Technique –** Heuristic word is derived from a Greek word that means “to discover”. The heuristic technique is a technique or model that is used for solving problems, learning, or discovery in the practical methods which are used for achieving immediate goals. These techniques are flexible and simple for taking quick decisions through shortcuts and good enough calculations, most probably when working with complex data. But the decisions that are made using this technique are necessary to be optimal. In this technique, the relationship among different project parameters is expressed using mathematical equations. The popular heuristic technique is given by Constructive Cost Model (COCOMO). This technique is also used to increase or speed up the analysis and investment decisions.
- **Analytical Estimation Technique –** Analytical estimation is a type of technique that is used to measure work. In this technique, firstly the task is divided or broken down into its basic component operations or elements for analyzing. Second, if the standard time is available from some other source, then these sources are applied to each element or component of work. Third, if there is no such time available, then the work is estimated based on the experience of the work. In this technique, results are derived by making certain basic assumptions about the project. Hence, the analytical estimation technique has some scientific basis. Halstead’s software science is based on an analytical estimation model.

QUALITY MANAGEMENT

- **What is Software Quality?**
- Software Quality shows how good and reliable a product is. To convey an associate degree example, think about functionally correct software. It performs all functions as laid out in the SRS document. But, it has an associate degree virtually unusable program. even though it should be functionally correct, we tend not to think about it to be a high-quality product.
- Another example is also that of a product that will have everything that the users need but has an associate degree virtually incomprehensible and not maintainable code. Therefore, the normal construct of quality as “fitness of purpose” for code merchandise isn’t satisfactory.

- **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. 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:** 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
- **Reliability:** Software is more reliable if it has fewer failures. Since software engineers do not deliberately plan for their software to fail, reliability depends on the number and type of mistakes they make. Designers can improve reliability by ensuring the software is easy to implement and change, by testing it thoroughly, and also by ensuring that if failures occur, the system can handle them or can recover easily.
- **Efficiency.** The more efficient software is, the less it uses of CPU-time, memory, disk space, network bandwidth, and other resources. This is important to customers in order to reduce their costs of running the software, although with today's powerful computers, CPU time, memory and disk usage are less of a concern than in years gone by.

SOFTWARE QUALITY MANAGEMENT SYSTEM

- Software Quality Management System contains the methods that are used by the authorities to develop products having the desired quality.
- Some of the methods are:
- **Managerial Structure:** Quality System is responsible for managing the structure as a whole. Every Organization has a managerial structure.
- **Individual Responsibilities:** Each individual present in the organization must have some responsibilities that should be reviewed by the top management and each individual present in the system must take this seriously.
- **Quality System Activities:** The activities which each quality system must have been
  - Project Auditing.
  - Review of the quality system.
  - It helps in the development of methods and guidelines.

FORMAL TECHNICAL REVIEWS

- Formal Technical Review (FTR) is a software quality control activity performed by software engineers. It is an organized, methodical procedure for assessing and raising the standard of any technical paper, including software objects. Finding flaws, making sure standards are followed, and improving the product or document under review's overall quality are the main objectives of a formal technical review (FTR). Although FTRs are frequently utilized in software development, other technical fields can also employ the concept.
- **Objectives of formal technical review (FTR)**
- **Detect Identification:** Identify defects in technical objects by finding and fixing mistakes, inconsistencies, and deviations.
- **Quality Assurance:** To ensure high-quality deliverables, and confirm compliance with project specifications and standards.
- **Risk Mitigation:** To stop risks from getting worse, proactively identify and manage possible threats.
- **Knowledge Sharing:** Encourage team members to work together and build a common knowledge base.
- **Consistency and Compliance:** Verify that all procedures, coding standards, and policies are followed.
- **Learning and Training:** Give team members the chance to improve their abilities through learning opportunities.

- **The Review Meeting**
- Each review meeting should be held considering the following constraints- *Involvement of people*:
- Between 3, 4, and 5 people should be involved in the review.
- preparation should occur, but it should be very short that is at the most 2 hours of work for every person.
- The short duration of the review meeting should be less than two hours. Given these constraints, it should be clear that an FTR focuses on specific (and small) parts of the overall software.
- At the end of the review, all attendees of FTR must decide what to do.
- Accept the product without any modification.
- Reject the project due to serious error (Once corrected, another app needs to be reviewed), or
- Accept the product provisional (minor errors are encountered and should be corrected, but no additional review will be required).
- The decision was made, with all FTR attendees completing a sign indicating their participation in the review and their agreement with the findings of the review team.

- **Review reporting and record-keeping:**
- During the FTR, the reviewer actively records all issues that have been raised.
- At the end of the meeting all these issues raised are consolidated and a review list is prepared.
- Finally, a formal technical review summary report is prepared.
- **A review summary report answers three questions:**
- What was reviewed?
- Who reviewed it?
- What were the findings and conclusions?
- **Review Guidelines**
- Guidelines for the conducting of formal technical reviews should be established in advance. These guidelines must be distributed to all reviewers, agreed upon, and then followed. An unregistered review can often be worse than a review that does not minimum set of guidelines for FTR.
- **Review the product, not the manufacturer (producer):** FTR includes people and egos. while properly conducted FTR will give all participants a good feeling of accomplishment, and when conducted improperly, the FTR team will be pointed out clearly. the meeting tone will be only loose and constructive, the review team leader will ensure that the proper way of communication will be happening or not into the team or it will get out of control.
- **Set an agenda and maintain it:** one of the measure keys in the meeting is drift. FTR will always keep on track and schedule. the review leader is making and maintaining the meeting schedule and should not be afraid that many people drift sets in.
- **Limit debate and rebuttal:** whenever the issue is raised by the reviewer there will not be an impact on a universal agreement. rather than spending time with the questions, then issues will be further discussed off-line.
- **Enunciate problem areas, but don’t attempt to solve every problem noted:** A review is not a problem-solving session. The solution to a problem can often be accomplished for the producer alone or with the help of only one other individual. problem-solving should be postponed for a while until after the review meeting.