MODULE-4
((User Interface Design, Coding & Testing)

Syllabus:

→Characteristics of a Good User Interface
→Basic Concepts, Types of User Interfaces
→Fundamentals of Component-based GUI Development
→A User Interface Design Methodology.
→Coding
→Code Review, Software Documentation
→Testing
→Unit Testing
→Black-box Testing
→White-Box Testing

1.Define User Interface, Explain about various types of User Interface Design

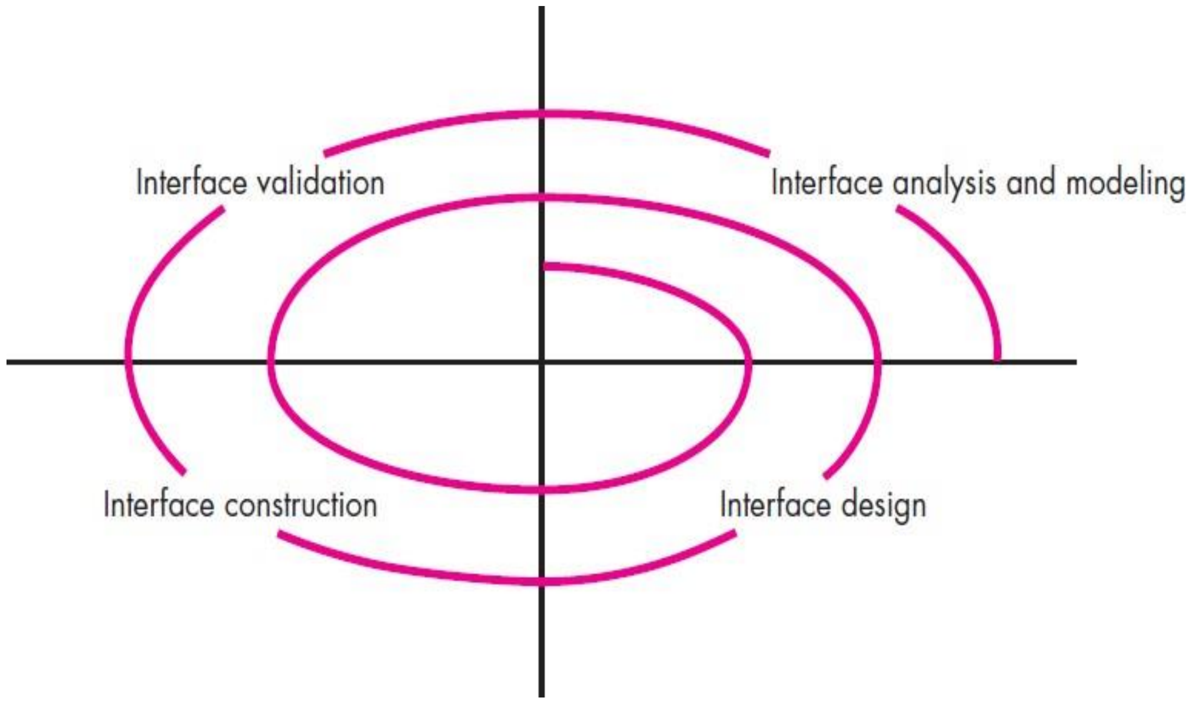
User interface is the front-end application view to which user interacts in order to use the software.
The software becomes more popular if its user interface is:
→Attractive
→Simple to use
→Responsive in short time
→Clear to understand
→Consistent on all interface screens

There are two types of User Interface:
1. Command Line/User Interface:
2. Graphical User Interface:

1. Command Line/User Interface: Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.

2. Graphical User Interface: Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.

The user interface design process:



The analysis and design process of a user interface is iterative and can be represented by a spiral model. The analysis and design process of user interface consists of four framework activities.

1. **User, task, environmental analysis and modeling:**
2. **Interface Design:**
3. **Interface construction and implementation:**
4. **Interface Validation:**

2.Discuss about the Golden Rules for User Interface Design ?

The Golden Rules:

a. Place the user in control.

b. Reduce the user's memory load.

c. Make the interface consistent.

1a) Define interaction modes in a way that does not force a user into unnecessary or undesired actions.

2a) Provide for flexible interaction.

3a) Allow user interaction to be interruptible and undoable.

4a) Streamline interaction as skill levels advance and allow the interaction to be customized.

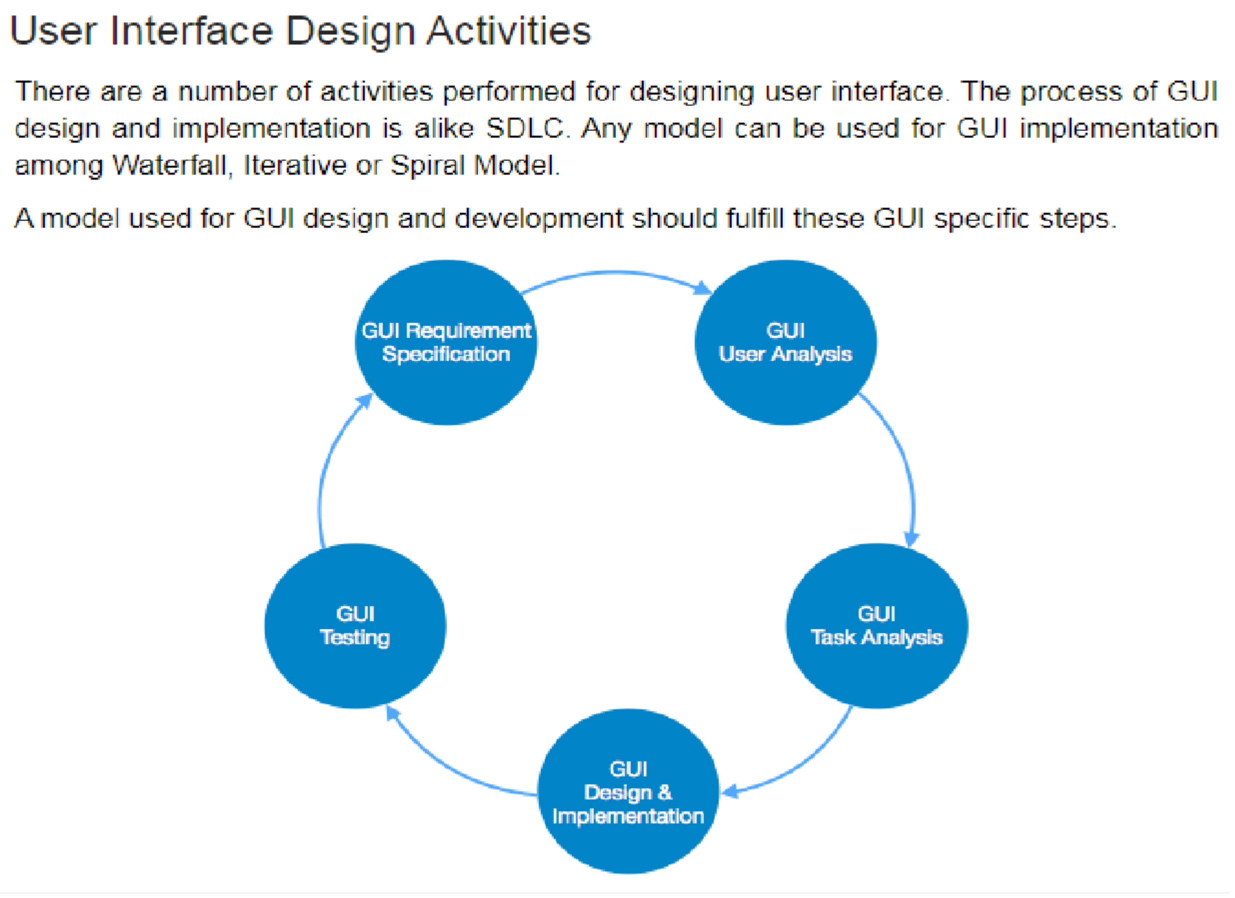
5a) Hide technical internals from the casual user.

6a) Design for direct interaction with objects that appear on the screen.

Reduce the user's memory load.
1b) Reduce demand on short-term memory.
2b) Establish meaningful defaults.
3b) Define shortcuts that are intuitive.
4b) The visual layout of the interface should be based on a real-world metaphor.
5b) Disclose information in a progressive fashion.
Make the interface consistent
Make the interface consistent.
1c) Allow the user to put the current task into a meaningful context.
2c) Maintain consistency across a family of applications.
3c) If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

3.Explain about User Interface Analysis and Design process ?

User Interface Analysis and Design:



User Interface Design Activities

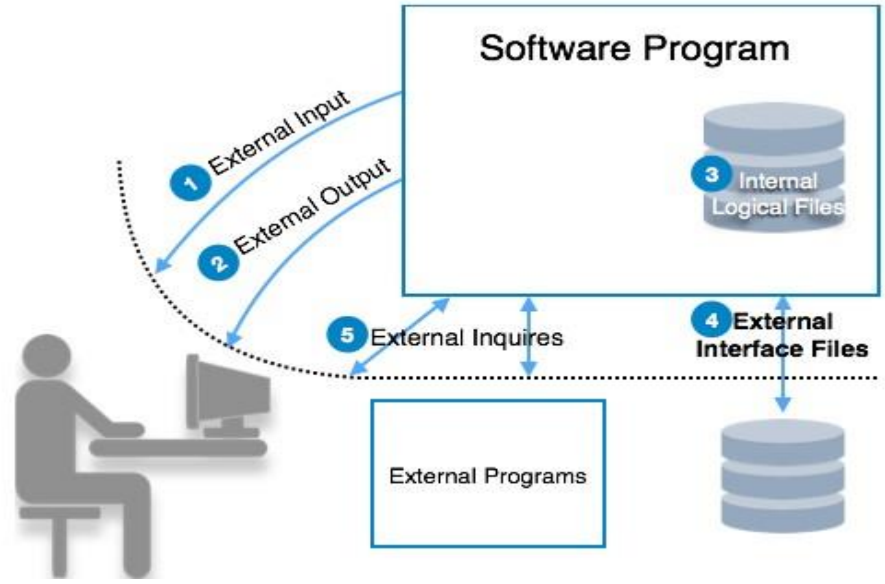
There are a number of activities performed for designing user interface. The process of GUI design and implementation is alike SDLC. Any model can be used for GUI implementation among Waterfall, Iterative or Spiral Model.

A model used for GUI design and development should fulfill these GUI specific steps.

Function Point

It is widely used to measure the size of software. Function Point concentrates on functionality provided by the system. Features and functionality of the system are used to measure the software complexity.

Function point counts on five parameters, named as External Input, External Output, Logical Internal Files, External Interface Files, and External Inquiry. To consider the complexity of software each parameter is further categorized as simple, average or complex.



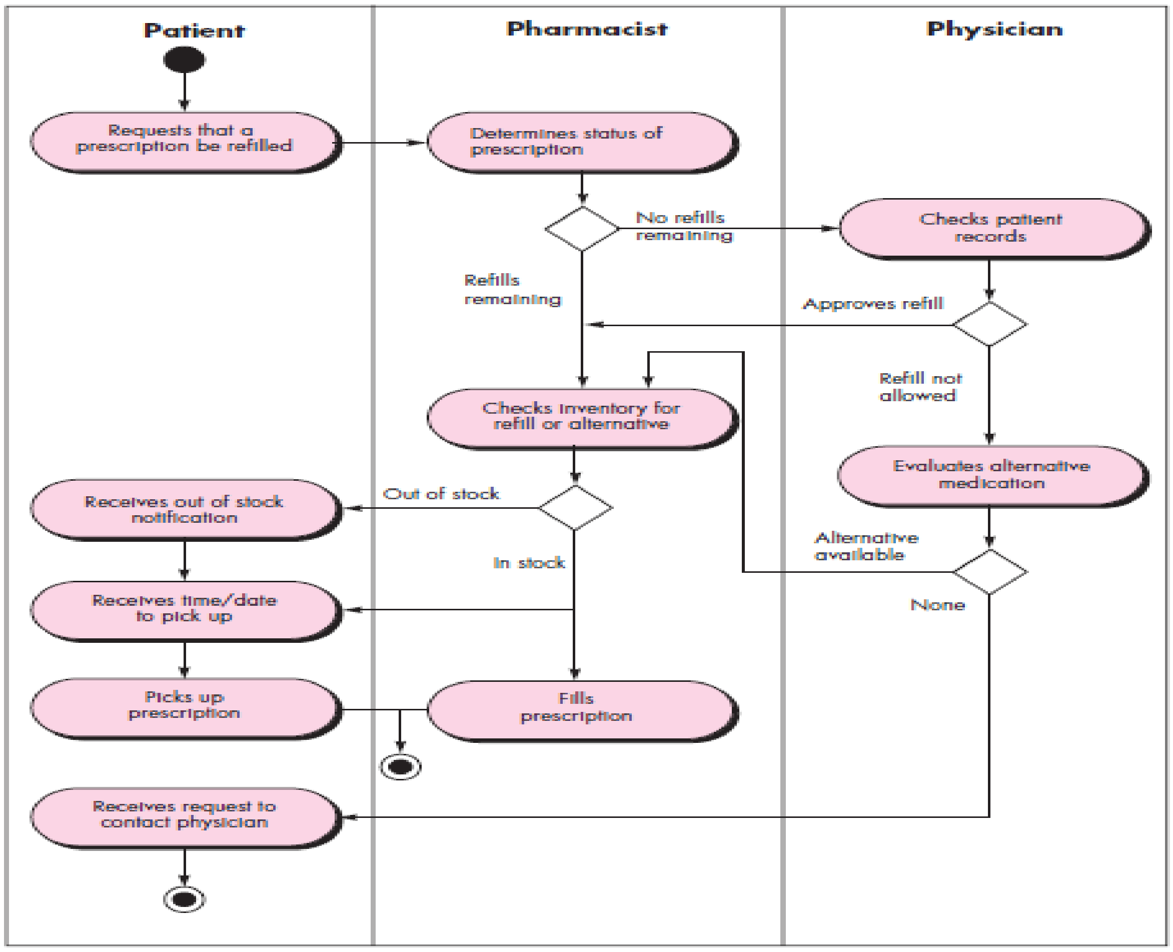
4.Discuss about Interface Analysis and Interface Design Steps ?

Interface Analysis:

→Interface Analysis is a business analysis elicitation technique that helps to identify interfaces between solutions/applications to determine the requirements for ensuring that the components interact with one another effectively.

→An interface is a shared boundary between two components. Most systems require connections with other applications, hardware and peripheral devices to function properly. Interface

→Analysis is a business analysis elicitation technique that helps to identify interfaces between solutions/applications to determine the requirements for ensuring that the components interact with one another effectively.


Patient
Pharmacist
Physician
Requests that a prescription be refilled
Determines status of prescription
No refills remaining
Checks patient records
Refills remaining
Approves refill
Refill not allowed
Checks inventory for refill or alternative
Evaluates alternative medication
Receives out of stock notification
Out of stock
In stock
Alternative available
None
Receives time/date to pick up
Picks up prescription
Fills prescription
Receives request to contact physician

User task: ***Requests that a prescription be refilled***
• *Provide identifying information.*
• *Specify name.*
• *Specify user-id.*
• *Specify PIN and password.*
• *Specify prescription number.*
• *Specify date refill is required.*

Interface Design Steps:
→Once interface analysis has been completed, all tasks required by the end user have been identified in detail and the interface design activity commences.
→Interface design is an iterative process. Each user interface design step occurs a number of times, elaborating and refining information.
1. Using information developed during interface analysis, define interface objects and actions (operations).

2. Define events (user actions) that will cause the state of the user interface to change. Model this behavior.
3. Depict each interface state as it will actually look to the end user.
4. Indicate how the user interprets the state of the system from information provided through the interface.

Homeowner tasks:
• *accesses the SafeHome system*
• *enters an* ***ID and password to allow remote access***
• *checks* ***system status***
• *arms or disarms SafeHome system*
• *displays* ***floor plan and sensor locations***
• *displays* ***zones on floor plan***
• *changes* ***zones on floor plan***
• *displays* ***video camera locations on floor plan***
• *selects* ***video camera for viewing***
• *views* ***video images (four frames per second)***
• *pans or zooms the* ***video camera***

WebApp Interface Design:
Web application interface design is, at its core, however, its focus is mainly on function. To compete with desktop applications, Web apps must offer simple, intuitive and responsive user interfaces that let their users get things done with less effort and time.
Great interfaces share eight qualities/characteristics:
1. **Clarity**: The interface avoids ambiguity by making everything clear through language, flow, hierarchy and metaphors for visual elements.
2. **Concision**: It's easy to make the interface clear by over-clarifying and labeling everything.

3. **Familiarity**: Something is familiar when you recall a previous encounter you've had with it. Even if someone uses an interface for the first time, certain elements can still be familiar.
4. **Responsiveness**: This means a couple of things. First, responsiveness means speed: a good interface should not feel sluggish.
5. **Consistency**: Keeping your interface consistent across your application is important because it allows users to recognize usage patterns.
6. **Aesthetics**: While you don't need to make an interface attractive for it to do its job, making something look good will make the time your users spend using your application more enjoyable.

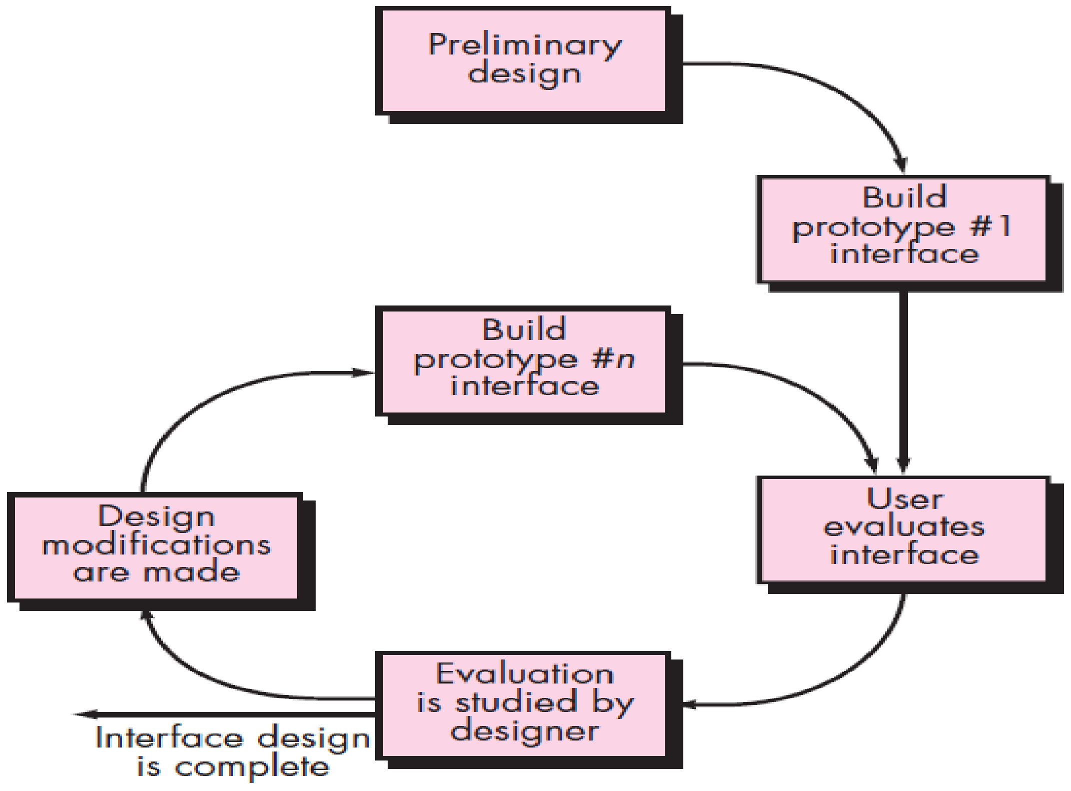
7. **Efficiency**: Time is money, and a great interface should make the user more productive through shortcuts and good design. After all, this is one of the core benefits of technology.
8. **Forgiveness**: Everyone makes mistakes, and how your application handles those mistakes will be a test of its overall quality.

Interface Design Principles and Guidelines:
Design guidelines are sets of recommendations on how to apply design principles to provide a positive user experience. Designers use such guidelines to judge how to adopt principles.
1.Anticipation→ A WebApp should be designed so that it anticipates the user's next move.
2.Communication→The interface should communicate the status of any activity initiated by the user.
***3.*Consistency→ The** *use of navigation controls, menus, icons, and aesthetics(color, shape, layout) should be consistent throughout the WebApp.*
4. Controlled autonomy→The interface should facilitate user movement throughout the WebApp, but it should do so in a manner that enforces navigation conventions.
5. Efficiency→ The design of the WebApp and its interface should optimize the user's work efficiency, not the efficiency of the developer who designs and builds it or the client server environment.
6. Flexibility→ The interface should be flexible enough to enable some users to accomplish tasks directly and others to explore the WebApp in a somewhat random fashion.
7. Focus→ The WebApp interface (and the content it presents) should stay focused on the user task(s) at hand. In all hypermedia there is a tendency to route the user to loosely related content..

Design Evaluation:
→Software design and estimation play the key role for software development process.
→Different methods are used for architecture design and detailed design evaluation.
→For architectural design stage a technique that allows selecting and evaluating suite of architectural patterns is proposed.
→It allows us to consistently evaluate the impact of specific patterns to software characteristics with a given functionality.

1. The length and complexity of the requirements model or written specification of the system and its interface provide an indication of the amount of learning required by users of the system.
2. The number of user tasks specified and the average number of actions per task provide an indication of interaction time and the overall efficiency of the system.
3. The number of actions, tasks, and system states indicated by the design model imply the memory load on users of the system.
4. Interface style, help facilities, and error handling protocol provide a general indication of the complexity of the interface and the degree to which it will be accepted by the user.

The interface design evaluation cycle:


Preliminary design
Build prototype #1 interface
Build prototype #n interface
Design modifications are made
User evaluates interface
Evaluation is studied by designer
Interface design is complete

6.Discuss about fundamentals of component-based GUI ?

Component-based GUI Development:

Component-based development techniques consist of non-conventional development routines, including component evaluation, component retrieval, etc.
It is important that the CBD is carried out within a middleware infrastructure that supports the process.
The key goals of CBD are as follows:

1. Save time and money when building large and complex systems: Developing complex software systems with the help of off-the-shelf components helps reduce software development time substantially. Function points or similar techniques can be used to verify the affordability of the existing method.

2.Enhance the software quality: The component quality is the key factor behind the enhancement of software quality.

3. Detect defects within the systems: The CBD strategy supports fault detection by testing the components; however, finding the source of defects is challenging in CBD.

Advantages of CBD include:

1. Minimized delivery:
Search in component catalogs
Recycling of pre-fabricated components

2. Improved efficiency:
Developers concentrate on application development

3. Improved quality:
Component developers can permit additional time to ensure quality

4. Minimized expenditures

Additional Advantages:
Ease of deployment − As new compatible versions become available, it is easierto replace existing versions with no impact on the other components or the system as a whole.
Reduced cost − The use of third-party components allows you to spread the cost of development and maintenance.
Ease of development − Components implement well-known interfaces to provide defined functionality, allowing development without impacting other parts of the system.
Reusable − The use of reusable components means that they can be used to spread the development and maintenance cost across several applications or systems.
Modification of technical complexity − A component modifies the complexity through the use of a component container and its services.

Reliability − The overall system reliability increases since the reliability of each individual component enhances the reliability of the whole system via reuse.
System maintenance and evolution − Easy to change and update the implementation without affecting the rest of the system.
Independent − Independency and flexible connectivity of components.Independent development of components by different group in parallel. Productivity for the software development and future software development.

The specific routines of CBD are:
1. Component development
2. Component publishing
3. Component lookup as well as retrieval
4. Component analysis
5. Component assembly

User Interface Design Methodology:
Structure: Design should organize the user interface purposefully, in the meaningful and usual based on precise, consistent models that are apparent and recognizable to users, putting related things together and separating unrelatedthings, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.
Simplicity: The design should make the simple, common task easy,communicating clearly and directly in the user's language, and providing good shortcuts that are meaningfully related to longer procedures.
Visibility: The design should make all required options and materials for a given function visible without distracting the user with extraneous or redundant data.
Feedback: The design should keep users informed of actions or interpretation, changes of state or condition, and bugs or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.
Tolerance: The design should be flexible and tolerant, decreasing the cost of errors and misuse by allowing undoing and redoing while also preventing bugs wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.
6.Explain about a) Coding b) Code review ?
Coding:
The coding is the process of transforming the design of a system into a computer language format.

This coding phase of software development is concerned with software translating design specification into the source code.
It is necessary to write source code & internal documentation so that conformance of the code to its specification can be easily verified.
Coding is done by the coder or programmers who are independent people than the designer. The goal is not to reduce the effort and cost of the coding phase, but to cut to the cost of a later stage.
The cost of testing and maintenance can be significantly reduced with efficient coding.

Goals of Coding:
1. To translate the design of system into a computer language format: The coding is the process of transforming the design of a system into a computer language format, which can be executed by a computer and that perform tasks as specified by the design of operation during the design phase.
2. To reduce the cost of later phases: The cost of testing and maintenance can be significantly reduced with efficient coding.
3. Making the program more readable: Program should be easy to read and understand. It increases code understanding having readability and understandability as a clear objective of the coding activity can itself help in producing more maintainable software.

Characteristics of Programming Language:
For implementing our design into code, we require a high-level functional language. A programming language should have the following characteristics:
Readability: A good high-level language will allow programs to be written in some methods that resemble a quite-English description of the underlying functions. The coding may be done in an essentially self-documenting way.
Portability: High-level languages, being virtually machine-independent, should be easy to develop portable software.
Generality: Most high-level languages allow the writing of a vast collection of programs, thus relieving the programmer of the need to develop into an expert in many diverse languages.
Brevity: Language should have the ability to implement the algorithm with less amount of code. Programs mean in high-level languages are often significantly shorter than their low-level equivalents.

Error checking: A programmer is likely to make many errors in the development of a computer program. Many high-level languages invoke a lot of bugs checking both at compile-time and run-time.

Cost: The ultimate cost of a programming language is a task of many of its characteristics.
Quick translation: It should permit quick translation.
Efficiency: It should authorize the creation of an efficient object code.
Modularity: It is desirable that programs can be developed in the language as several separately compiled modules, with the appropriate structure for ensuring self-consistency among these modules.
Widely available: Language should be widely available, and it should be feasible to provide translators for all the major machines and all the primary operating systems.

A coding standard lists several rules to be followed during coding, such as the way variables are to be named, the way the code is to be laid out, error return conventions, etc.
Coding Standards
General coding standards refers to how the developer writes code, so here we will discuss some essential standards regardless of the programming language being used.
The following are some representative coding standards:
1. Indentation: Proper and consistent indentation is essential in producing easy to read and maintainable programs.Indentation should be used to:
→Emphasize the body of a control structure such as a loop or a select statement.
→Emphasize the body of a conditional statement
→Emphasize a new scope block
2. Inline comments: Inline comments analyze the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.
3. Rules for limiting the use of global: These rules file what types of data can be declared global and what cannot.

4. Structured Programming: Structured (or Modular) Programming methods shall be used. "GOTO" statements shall not be used as they lead to code, which is hard to read and maintain, except as outlined line in the FORTRAN Standards and Guidelines.
5. Naming conventions for global variables, local variables, and constant identifiers: A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters.
6. Error return conventions and exception handling system: Different functions in a program report the way error conditions are handled should be standard within

an organization. For example, different tasks while encountering an error condition should either return a 0 or 1 consistently.

Coding Guidelines:

General coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain. Most of the examples use the C language syntax, but the guidelines can be tested to all languages.

The following are some representative coding guidelines recommended by many software development organizations.

1. Line Length: It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.

2. Spacing: The appropriate use of spaces within a line of code can improve readability.

Example:

Bad:
```
cost=price+(price*sales_tax)
fprintf(stdout ,"The total cost is %5.2f\n",cost);
```
Better:
```
cost = price + ( price * sales_tax )
fprintf (stdout,"The total cost is %5.2f\n",cost);
```
3. The code should be well-documented: As a rule of thumb, there must be at least one comment line on the average for every three-source line.

4. The length of any function should not exceed 10 source lines: A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs.

5. Do not use goto statements: Use of goto statements makes a program unstructured and very tough to understand.

6. Inline Comments: Inline comments promote readability.

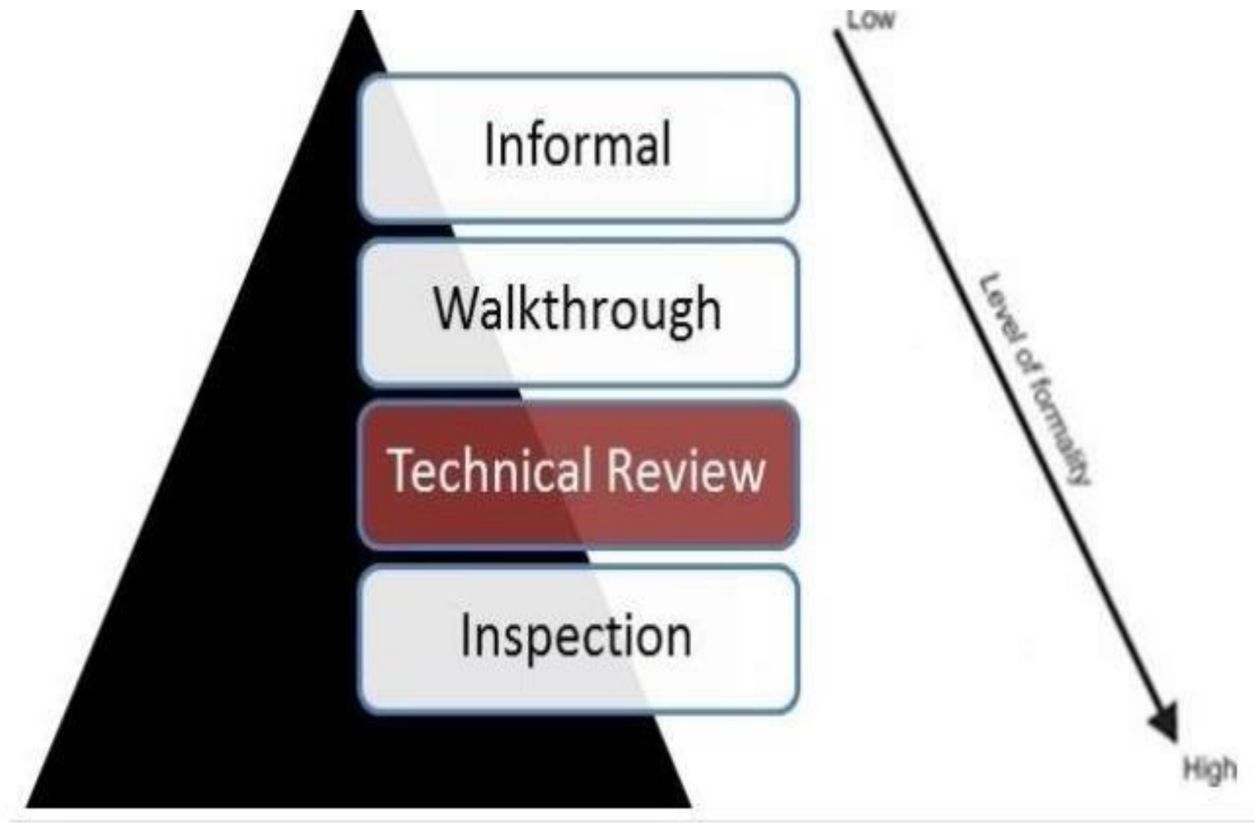
7. Error Messages: Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

Code Review:

Code Review is a systematic examination, which can find and remove the vulnerabilities in the code such as memory leaks and buffer overflows.

→Technical reviews are well documented and use a well-defined defect detection process that includes peers and technical experts.

→It is ideally led by a trained moderator, who is NOT the author.
→This kind of review is usually performed as a peer review without management participation.
→Reviewers prepare for the review meeting and prepare a review report with a list of findings.
→Technical reviews may be quite informal or very formal and can have a number of purposes but not limited to discussion, decision making, evaluation of alternatives, finding defects and solving technical problems.



Walkthrough:
Members of the development team is guided by author and other interested parties and the participants ask questions and make comments about defects.

Technical Review:
A team of highly qualified individuals examines the software product for its client's use and identifies technical defects from specifications and standards.

Inspection:
In inspection the reviewers follow a well-defined process to find defects.

Informal reviews:
It take place between two or three people. The review conference is scheduled at their convenience. This meeting is generally scheduled during the free time of the team members. There is no planning for the meeting.

7.Explain about c) Software Documentation d) Testing ?

Software Documentation:
→It is a written piece of text that is often accompanied with a software program. This makes the life of all the members associated with the project more easy.
→It may contain anything from API documentation, build notes or just helpcontent. It is a very critical process in software development.
→It is a primarily an integral part of any computer code development method. Moreover, the computer code practitioners are a unit typically concerned with the worth, degree of usage and quality of the actual documentation throughout development and its maintenance throughout the total method.
→For example before development of any software product requirements are documented which is called as Software Requirement Specification (SRS). Requirement gathering is considered as an stage of Software Development Life Cycle (SDLC).

Importance of software documentation:
For a programmer reliable documentation is always a must the presence keeps track of all aspects of an application and helps in keeping the software updated.
Advantage of software documentation:
1. The presence of documentation helps in keeping the track of all aspects of an application and also improves on the quality of the software product .
2. The main focus are based on the development , maintenance and knowledge transfer to other developers.
3. Helps development teams during development.
4. Helps end-users in using the product.
5. Improves overall quality of software product
6. It cuts down duplicative work.
7. Makes easier to understand code.
8. Helps in establishing internal co-ordination in work.

Disadvantage of software documentation :
1. The documenting code is time consuming.
2. The software development process often takes place under time pressure, due to which many times the documentation updates don't match the updated code .
3. The documentation has no influence on the performance of the an application.
4. Documenting is not so fun, its sometime boring to a certain extent.

Types Of Software Documentation :

1. **Requirement Documentation :**

It is the description of how the software shall perform and which environment setup would be appropriate to have the best out of it. These are generated while the software is under development and is supplied to the tester groups too.

2. Architectural Documentation :

Architecture documentation is a special type of documentation that concerns the design. It contains very little code and is more focused on the components of the system, their roles and working. It also shows the data flows throughout thesystem.

3. Technical Documentation :

These contain the technical aspects of the software like API, algorithms etc. It is prepared mostly for the software devs.

4. End-user Documentation :

As the name suggests these are made for the end user. It contains support resources for the end user.

8.Define Testing? Explain the importance of Testing Process ?

Testing:

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of thesoftware product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

Testing is the process of executing a program to find errors. To make our software perform well it should be error-free. If testing is done successfully it will removeall the errors from the software.

Testing is the process of executing a program to find errors. To make our software perform well it should be error-free. If testing is done successfully it will removeall the errors from the software.

Principles of Testing:

(i) All the tests should meet the customer requirements.

(ii) To make our software testing should be performed by a third party.

(iii) Exhaustive testing is not possible. As we need the optimal amount of testing

based on the risk assessment of the application.
(iv) All the tests to be conducted should be planned before implementing .
(v) It follows the Pareto rule(80/20 rule) which states that 80% of errors come from 20% of program components.
(vi) Start testing with small parts and extend it to large parts.

Types of Testing:

1. Unit Testing

It focuses on the smallest unit of software design. In this, we test an individual unit or group of interrelated units. It is often done by the programmer by using sample input and observing its corresponding outputs.

Example:

a) In a program we are checking if the loop, method, or function is working fine
b) b) Misunderstood or incorrect, arithmetic precedence.
c) c) Incorrect initialization

2. Integration Testing

The objective is to take unit-tested components and build a program structure that has been dictated by design. Integration testing is testing in which a group of components is combined to produce output.
Integration testing is of four types: (i) Top-down (ii) Bottom-up (iii) Sandwich (iv) Big-Bang

Example:

(a) Black Box testing:- It is used for validation. In this, we ignore internal working mechanisms and focus on **what is the output?**.
(b) White box testing:- It is used for verification. In this, we focus on internal mechanisms i.e. **how the output is achieved?**

3. Regression Testing

Every time a new module is added leads to changes in the program. This type of testing makes sure that the whole component works properly even after adding components to the complete program.

Example:

In school, record suppose we have module staff, students and finance combining these modules and checking if on integration of these modules works fine in regression testing.

4. Smoke Testing

This test is done to make sure that the software under testing is ready or stable for further testing. It is called a smoke test as the testing of an initial pass is done to check if it did not catch the fire or smoke in the initial switch on.

Example:

If the project has 2 modules so before going to the module make sure that module 1 works properly.

5. Alpha Testing

This is a type of validation testing. It is a type of *acceptance testing* which is done before the product is released to customers. It is typically done by QA

Example:

When software testing is performed internally within the organization

6. Beta Testing

The beta test is conducted at one or more customer sites by the end-user of the software. This version is released for a limited number of users for testing in a real-time environment

Example:

When software testing is performed for the limited number of people

7. System Testing

This software is tested such that it works fine for the different operating systems. It is covered under the black box testing technique. In this, we just focus on the required input and output without focusing on internal working.

In this, we have security testing, recovery testing, stress testing, and performance

Example:

This includes functional as well as nonfunctional testing

8. Stress Testing

In this, we give unfavorable conditions to the system and check how they perform in those conditions.

Example:

(a) Test cases that require maximum memory or other resources are executed

(b) Test cases that may cause thrashing in a virtual operating system

(c) Test cases that may cause excessive disk requirement

9. Performance Testing

It is designed to test the run-time performance of software within the context of an integrated system. It is used to test the speed and effectiveness of the program. It is also called load testing. In it we check, what is the performance of the system in the given load.

Example:

Checking several processor cycles.

10. Object-Oriented Testing

This testing is a combination of various testing techniques that help to verify and validate object-oriented software. This testing is done in the following manner:

* Testing of Requirements,
* Design and Analysis of Testing,
* Testing of Code,

* Integration testing,
* System testing,
* User Testing.

11. Acceptance Testing

Acceptance testing is done by the customers to check whether the delivered products perform the desired tasks or not, as stated in requirements.

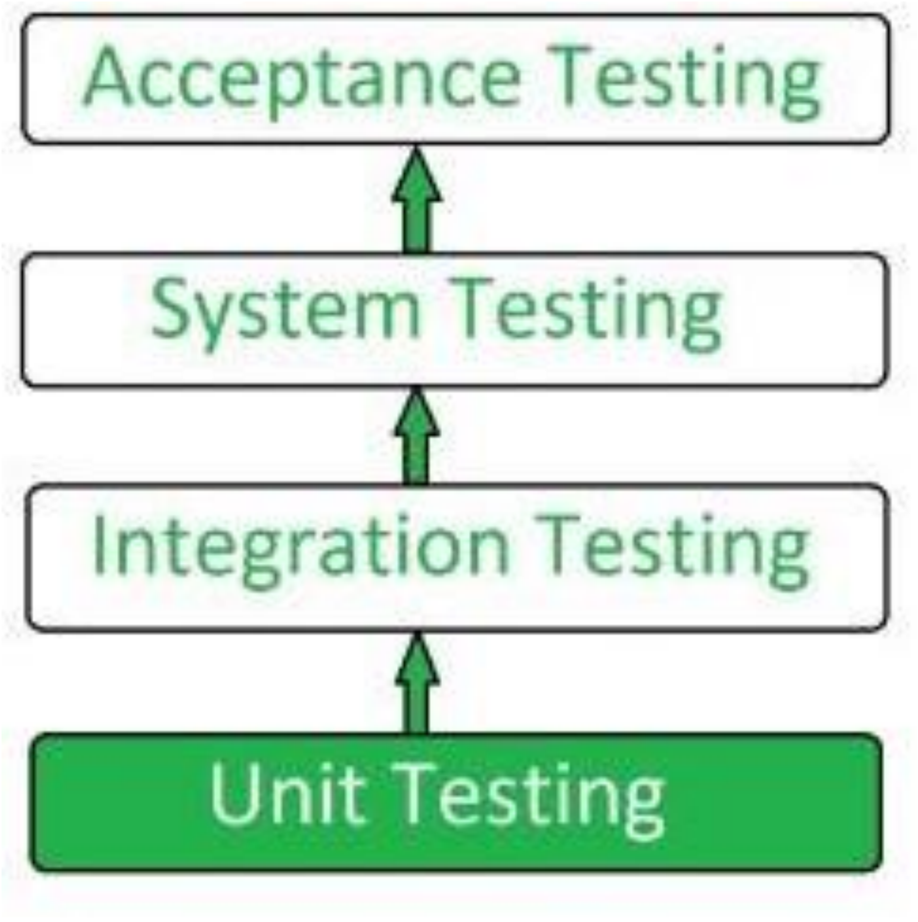
Unit Testing:

Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures, and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent module is tested to determine if there is an issue by the developer himself.

Objective of Unit Testing:

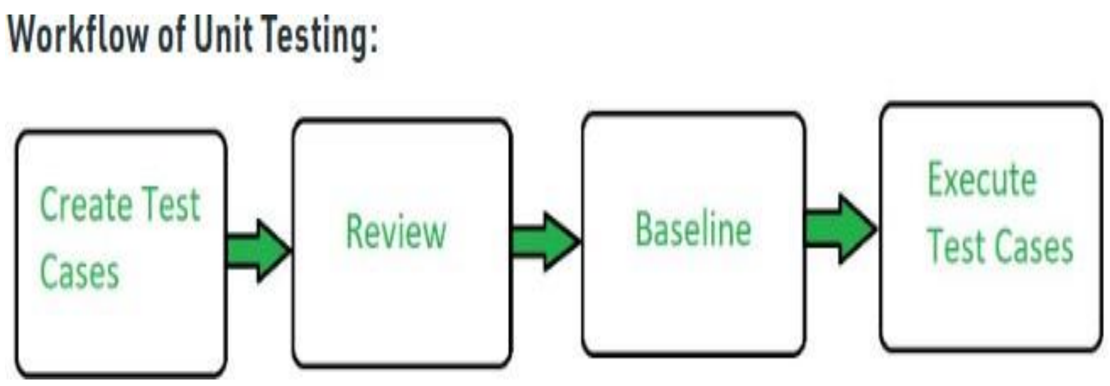
The objective of Unit Testing is:

1. To isolate a section of code.
2. To verify the correctness of the code.
3. To test every function and procedure.
4. To fix bugs early in the development cycle and to save costs.
5. To help the developers to understand the code base and enable them to make changes quickly.
6. To help with code reuse.



Types of Unit Testing:
There are 2 types of Unit Testing:

1. Manual Testing.
2. Automated Testing.



Unit Testing Techniques:

There are 3 types of Unit Testing Techniques. They are

1. Black Box Testing: This testing technique is used in covering the unit tests for input, user interface, and output parts.

2. White Box Testing: This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.

3. Gray Box Testing: This technique is used in executing the relevant test cases, test methods, test functions, and analyzing the code performance for the modules.

Advantages of Unit Testing:

1. Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
2. Unit testing allows the programmer to refine code and make sure the module works properly.
3. Unit testing enables testing parts of the project without waiting for others to be completed.

Disadvantages of Unit Testing:

1. The process is time-consuming for writing the unit test cases.
2. Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.
3. Unit Testing is not efficient for checking the errors in the UI(User Interface) part of the module.
4. It requires more time for maintenance when the source code is changed frequently.
5. It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.

9 Differentiate between White-Box Testing and Black-Box Testing ?

Black-box Testing:

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.

In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

BlackBox
Input
Output

Generic steps of black box testing:

→The black box test is based on the specification of requirements, so it is examined in the beginning.

→In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.

→In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.

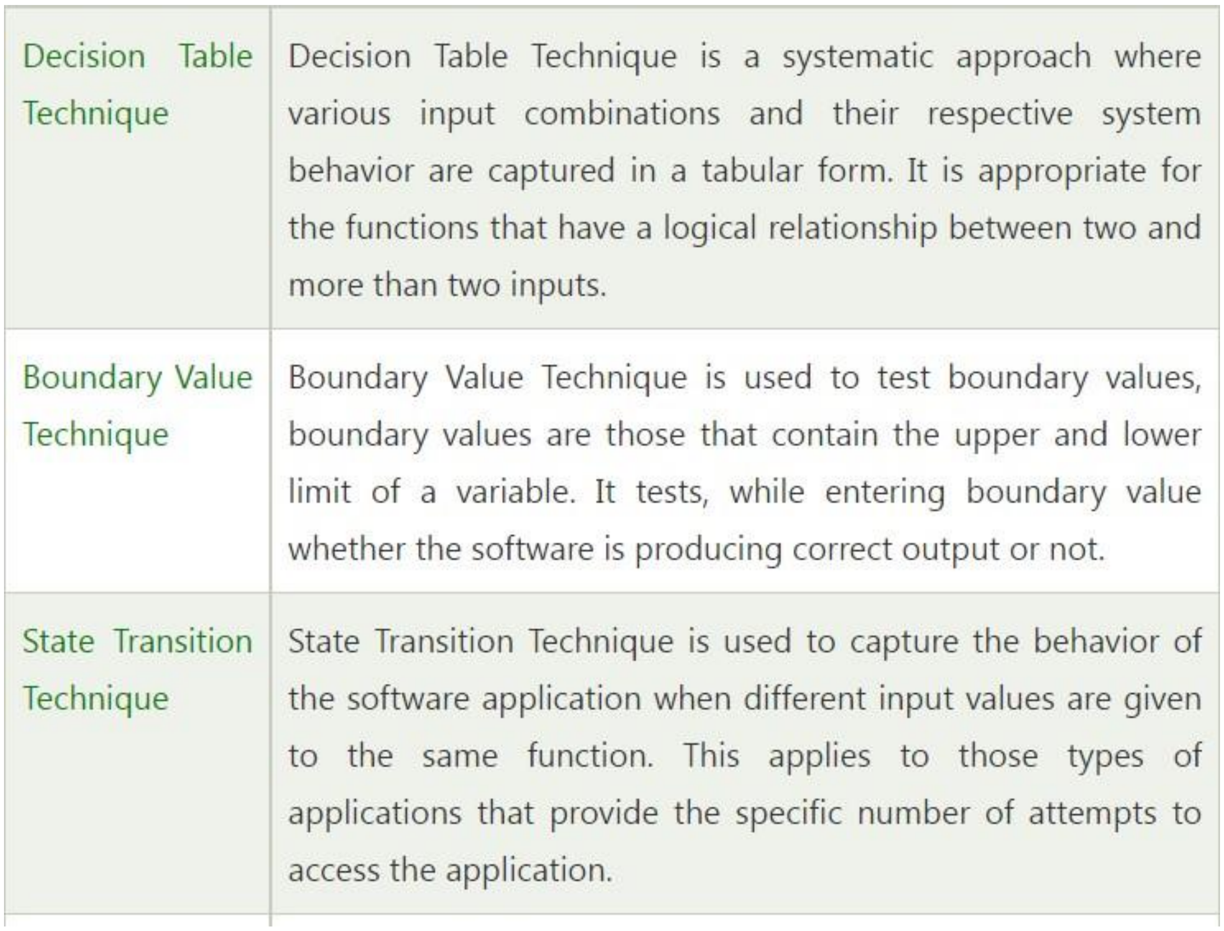
→The fourth phase includes the execution of all test cases.

→In the fifth step, the tester compares the expected output against the actual output.

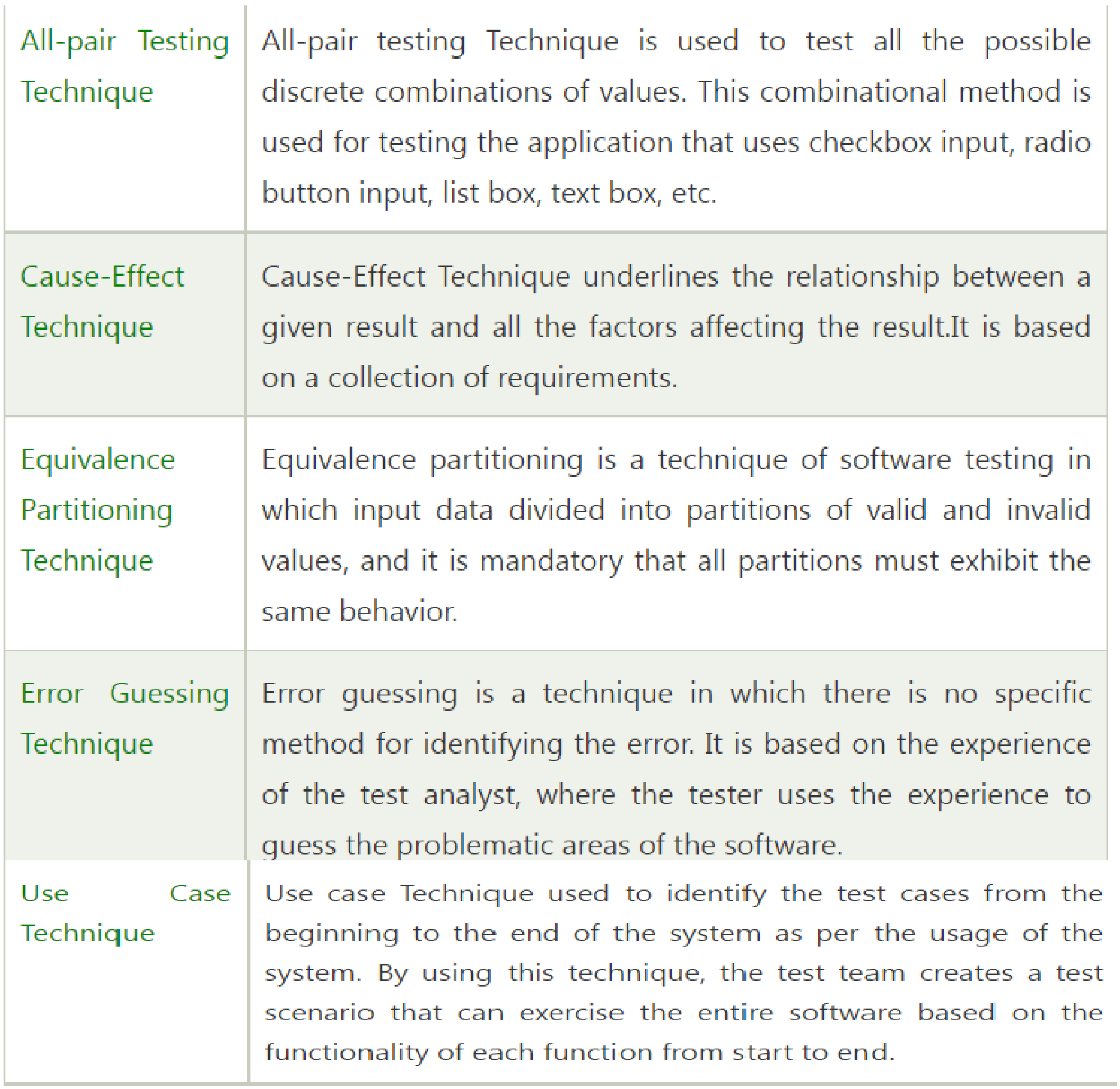
→In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

Techniques Used in Black Box Testing:

1. Decision Table Technique
2. Boundary Value Technique
3. State Transition Technique
4. All-pair testing Technique
5. Cause-Effect Technique
6. Equivalence partitioning is a technique
7. Error guessing technique
8. Use case Technique



Decision Table Technique	Decision Table Technique is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form. It is appropriate for the functions that have a logical relationship between two and more than two inputs.
Boundary Value Technique	Boundary Value Technique is used to test boundary values, boundary values are those that contain the upper and lower limit of a variable. It tests, while entering boundary value whether the software is producing correct output or not.
State Transition Technique	State Transition Technique is used to capture the behavior of the software application when different input values are given to the same function. This applies to those types of applications that provide the specific number of attempts to access the application.

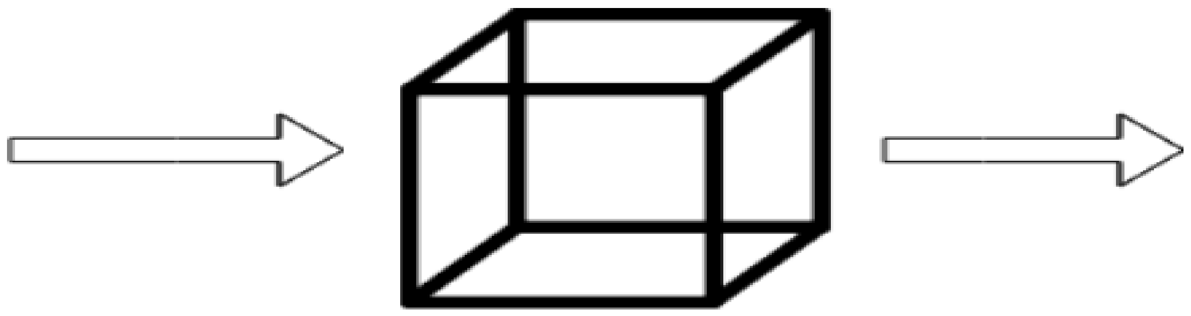


All-pair Testing Technique	All-pair testing Technique is used to test all the possible discrete combinations of values. This combinational method is used for testing the application that uses checkbox input, radio button input, list box, text box, etc.
Cause-Effect Technique	Cause-Effect Technique underlines the relationship between a given result and all the factors affecting the result.It is based on a collection of requirements.
Equivalence Partitioning Technique	Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.
Error Guessing Technique	Error guessing is a technique in which there is no specific method for identifying the error. It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software.
Use Case Technique	Use case Technique used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end.

White-Box Testing:
white box testing which also known as glass box is **testing, structural testing, clear box testing, open box testing and transparent box testing**. It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs.
It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.
Developers do white box testing. In this, the developer will test every line of the code of the program.

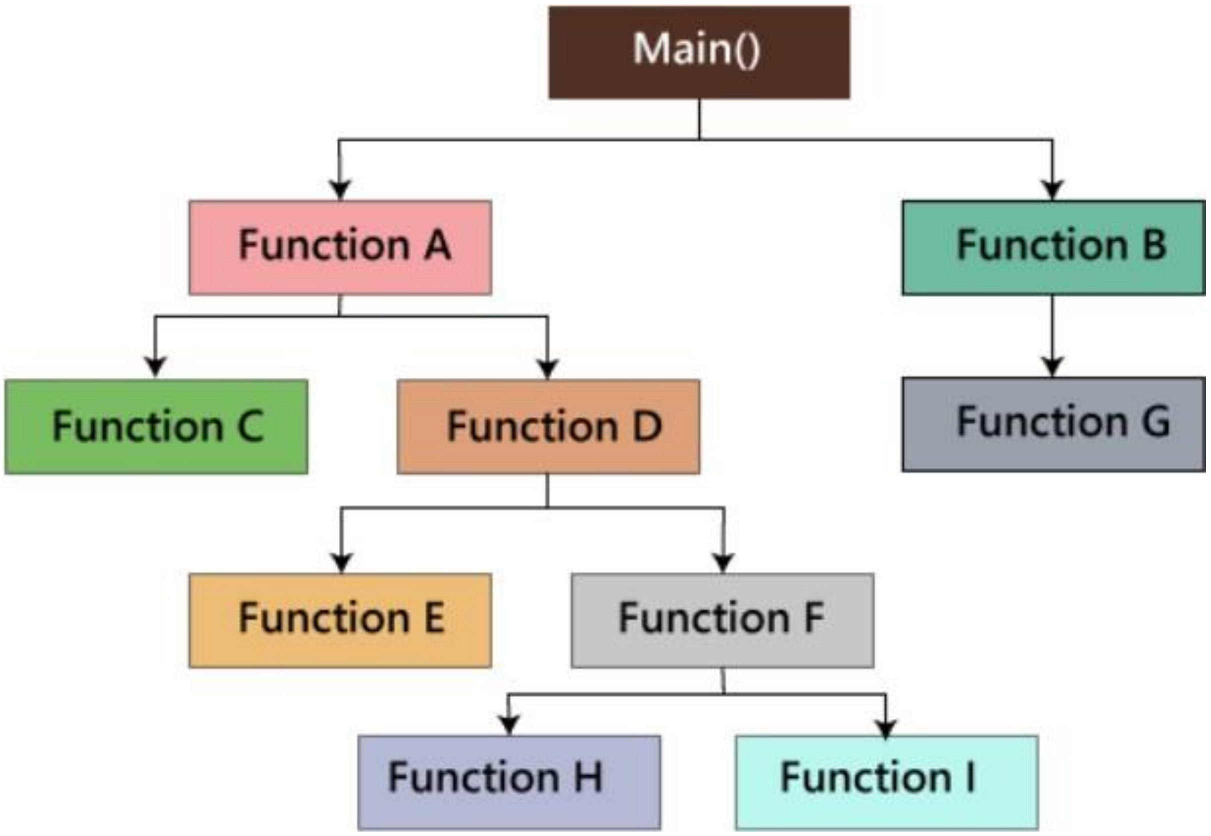
The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.
The developer fixes the bugs and does one round of white box testing and sends it to the testing team. Here, fixing the bugs implies that the bug is deleted, and the particular feature is working fine on the application.
The test engineers will not include in fixing the defects for the following reasons:
a. Fixing the bug might interrupt the other features. Therefore, the test engineer should always find the bugs, and developers should still be doing the bug fixes.
b. If the test engineers spend most of the time fixing the defects, then they may be unable to find the other bugs in the application.

The white box testing contains various tests, which are as follows:
1. Path testing
2. Loop testing
3. Condition testing
4. Testing based on the memory perspective
5. Test performance of the program
In the path testing, we will write the flow graphs and test all independent paths. Here writing the flow graph implies that flow graphs are representing the flow of the program and also show how every program is added with one another as we can see in the below image:



Whitebox Testing

In the path testing, we will write the flow graphs and test all independent paths. Here writing the flow graph implies that flow graphs are representing the flow of the program and also show how every program is added with one another as we can see in the below image:



In the loop testing, we will test the loops such as while, for, and do-while, etc. and also check for ending condition if working correctly and if the size of the conditions is enough.

For example: we have one program where the developers have given about 50,000 loops.

```
{
while(50,000)
……
……
}
```

In Condition testing, we will test all logical conditions for both **true** and **false** values; that is, we will verify for both **if** and **else** condition.

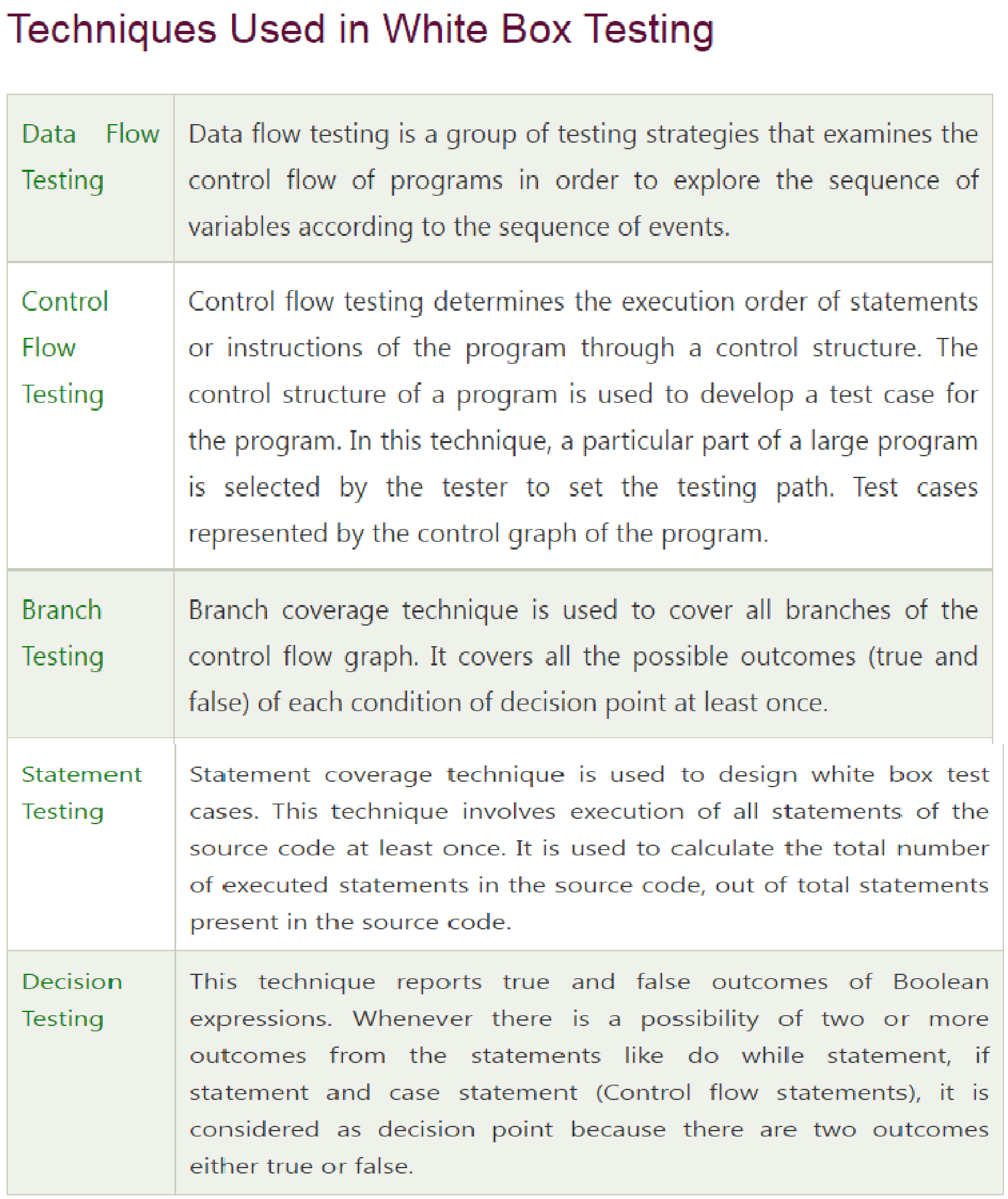
For example:

```
if(condition) - true
{
…..
……
……
}
else - false
{
…..
……
……
}
```

Generic steps of white box testing:
1. Design all test scenarios, test cases and prioritize them according to high priority number.
2. This step involves the study of code at runtime to examine the resource utilization, not accessed areas of the code, time taken by various methods and operations and so on.
3. In this step testing of internal subroutines takes place. Internal subroutines such as nonpublic methods, interfaces are able to handle all types of data appropriately or not.
4. This step focuses on testing of control statements like loops and conditional statements to check the efficiency and accuracy for different data inputs.
5. In the last step white box testing includes security testing to check all possible security loopholes by looking at how the code handles security.

Reasons for white box testing:
→It identifies internal security holes.
→To check the way of input inside the code.
→Check the functionality of conditional loops.
→To test function, object, and statement at an individual level. **Advantages of White box testing:**
1. White box testing optimizes code so hidden errors can be identified.
2. Test cases of white box testing can be easily automated.
3. This testing is more thorough than other testing approaches as it covers all code paths.
4. It can be started in the SDLC phase even without GUI.
Disadvantages of White box testing:
1. White box testing is too much time consuming when it comes to large-scale programming applications.
2. White box testing is much expensive and complex.
3. It can lead to production error because it is not detailed by the developers.

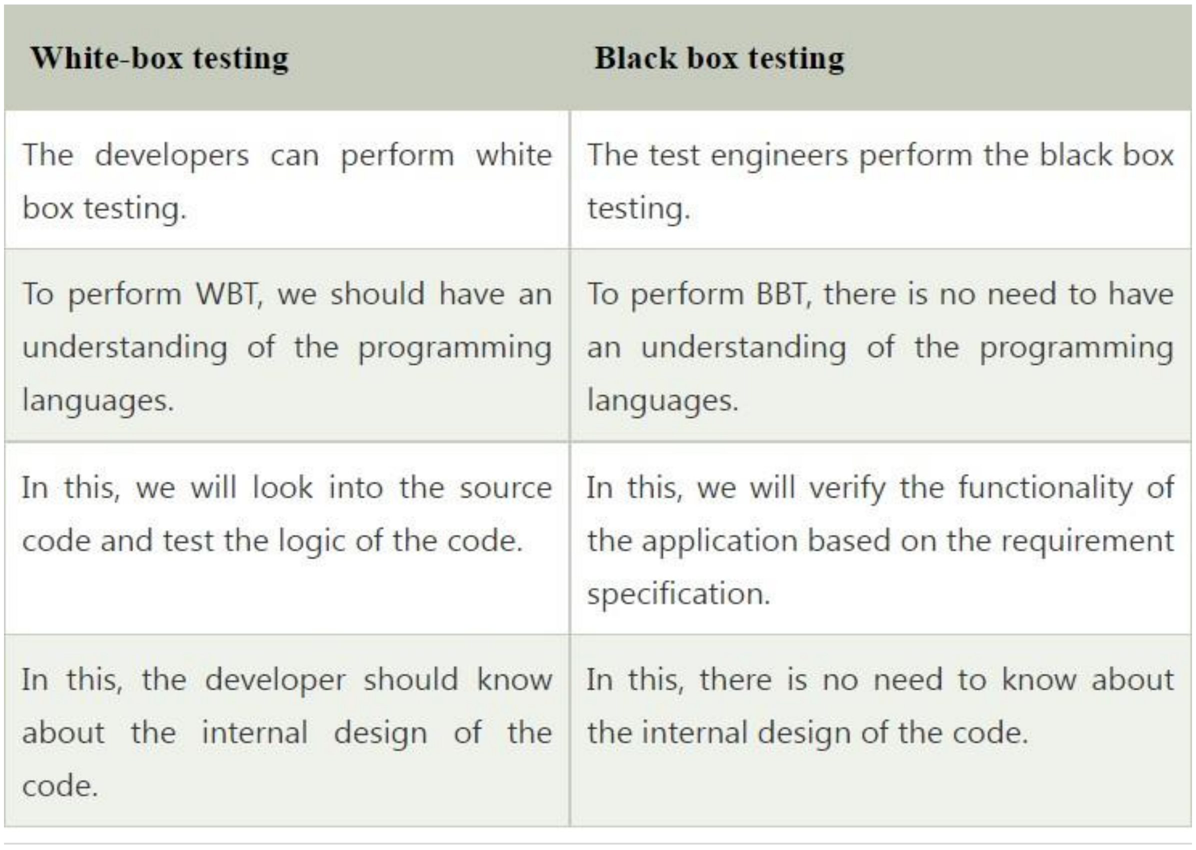
4. White box testing needs professional programmers who have a detailed knowledge and understanding of programming language and implementation.



Techniques Used in White Box Testing

Data Flow Testing	Data flow testing is a group of testing strategies that examines the control flow of programs in order to explore the sequence of variables according to the sequence of events.
Control Flow Testing	Control flow testing determines the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. Test cases represented by the control graph of the program.
Branch Testing	Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.
Statement Testing	Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code, out of total statements present in the source code.
Decision Testing	This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

Difference between white-box testing and black-box testing:



White-box testing	Black box testing
The developers can perform white box testing.	The test engineers perform the black box testing.
To perform WBT, we should have an understanding of the programming languages.	To perform BBT, there is no need to have an understanding of the programming languages.
In this, we will look into the source code and test the logic of the code.	In this, we will verify the functionality of the application based on the requirement specification.
In this, the developer should know about the internal design of the code.	In this, there is no need to know about the internal design of the code.