(a) Enlist various building blocks of UML. What are the goals of UML? Discuss the advantages of using UML? In what sense UML is unified?

The building blocks of UML can be defined as −

- Things
- Relationships
- Diagrams

Things

Things are the most important building blocks of UML. Things can be −

- Structural
- Behavioral
- Grouping
- Annotational

Structural Things

Structural things define the static part of the model. They represent the physical and conceptual elements. Following are the brief descriptions of the structural things.

Class − Class represents a set of objects having similar responsibilities.

Interface − Interface defines a set of operations, which specify the responsibility of a class.

Collaboration −Collaboration defines an interaction between elements.

Use case −Use case represents a set of actions performed by a system for a specific goal.

Component −Component describes the physical part of a system.

Node − A node can be defined as a physical element that exists at run time.

Behavioral Things

A behavioral thing consists of the dynamic parts of UML models. Following are the behavioral things −

Interaction − Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.

State machine − State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change

Grouping Things

Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available −

Package − Package is the only one grouping thing available for gathering structural and behavioral things.

Annotational Things

Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. **Note -** It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.

Relationship

Relationship is another most important building block of UML. It shows how the elements are associated with each other and this association describes the functionality of an application.

There are four kinds of relationships available.

Dependency

Dependency is a relationship between two things in which change in one element also affects the other.

Association

Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.

Generalization

Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world of objects.

Realization

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.

UML Diagrams

UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system.

The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it complete.

UML includes the following nine diagrams, the details of which are described in the subsequent chapters.

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- State chart diagram
- Deployment diagram
- Component diagram

Goals of UML

- Since it is a general-purpose modeling language, it can be utilized by all the modelers.
- UML came into existence after the introduction of object-oriented concepts to systemize and consolidate the object-oriented development, due to the absence of standard methods at that time.
- The UML diagrams are made for business users, developers, ordinary people, or anyone who is looking forward to understand the system, such that the system can be software or non-software.
- Thus, it can be concluded that the UML is a simple modeling approach that is used to model all the practical systems.

Advantages:

- Provides standard for software development.
- Reducing of costs to develop diagrams of UML using supporting tools.
- Development time is reduced.
- The past faced issues by the developers are no longer exists.
- Has large visual elements to construct and easy to follow.

The unification in unified, is the unification of their methodologies.

(b) Discuss different views supported by UML diagrams and explain the significance of Packages? Prepare an object and state transition diagrams for priority queues or heaps storing numbers, where in the operations of the shift up and shift down are possible.

The different software diagrams according to the views they implement are:

1) User's view

This contains the diagrams in which the user's part of interaction with the software is defined. No internal working of the software is defined in this model. The diagrams contained in this view are:

- Use case Diagram

2) Structural view

In the structural view, only the structure of the model is explained. This gives an estimate of what the software consists of. However, internal working is still not defined in this model. The diagram that this view includes are:

- Class Diagrams
- Object Diagrams

3) Behavioral view

The behavioral view contains the diagrams which explain the behavior of the software. These diagrams are:

- Sequence Diagram
- Collaboration Diagram
- State chart Diagram
- Activity Diagram

4) Environmental view

The environmental view contains the diagram which explains the after deployment behavior of the software model. This diagram usually explains the user interactions and software effects on the system. The diagrams that the environmental model contain are:

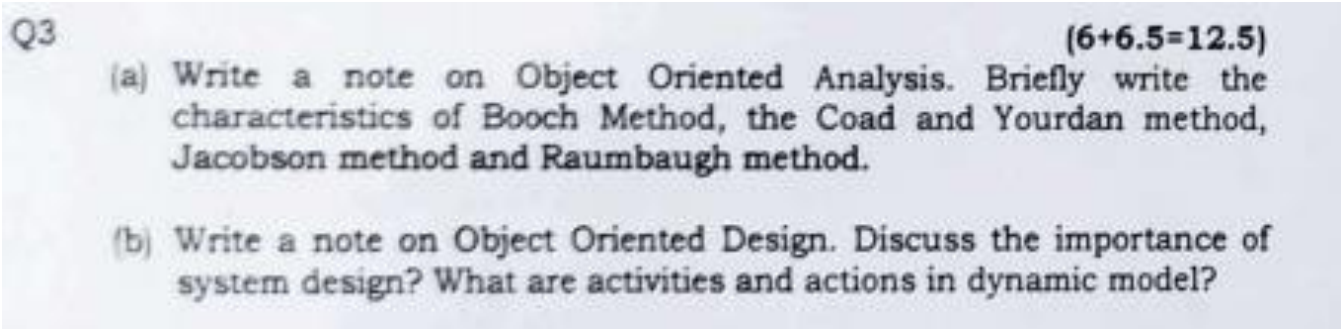
- Deployment diagram

5) Implementation view

The implementation view consists of the diagrams which represent the implementation part of the software. This view is related to the developer's views. This is the only view in which the internal workflow of the software is defined. The diagram that this view contains is as follows:

- Component Diagram

Packages in the Unified Modelling Language are used to group elements and provide namespaces for the grouped elements. A package can contain other packages, thus providing a hierarchical organization of packages.



Q3 **(6+6.5=12.5)**

(a) Write a note on Object Oriented Analysis. Briefly write the characteristics of Booch Method, the Coad and Yourdan method, Jacobson method and Raumbaugh method.

(b) Write a note on Object Oriented Design. Discuss the importance of system design? What are activities and actions in dynamic model?

Object-Oriented Analysis (OOA): It is based on a set of basic principles, which are as follows-

1. The information domain is modelled.
2. Behaviour is represented.
3. The function is described.
4. Data, functional, and behavioural models are divided to uncover greater detail.
5. Early models represent the essence of the problem, while later ones provide implementation details.

Rumbaugh Methodology (OMT)

The Rumbaugh methodology also known as OMT (Object Modeling Technique) is an approach used to develop manageable object-oriented systems and host object oriented programming. The purpose is to allow for class attributes, methods, inheritance, and association to be easily expressed. OMT is used in the real world for software modeling and designing.

According to Rumbaugh, there are several main reasons to utilize this modeling approach. One is to simulate entities before constructing them

and another is to make communication with customers easier. Additionally, it help to reduce complexity through visualization.

OMT consists of four stages:

- Analysis
- Systems Design
- Object Design
- Implementation

Additionally, OMT is always broken down into three separate parts. These parts are the:

- An object model
- A dynamic model
- A functional model.

Booch Methodology

Booth's methodology focuses on object-oriented analysis and design and consists of five activities: conceptualization, analysis, design, evolution, and maintainance. The method is cyclical and accounts for incremental improvements that are made in the evolution of a product.

Note: Clouds are used to define classes.
Macro process:

- **Conceptualization:** Establish requirements taking into account the perspective of the customer
- **Analysis:** Develop a model by defining object classes, their attributes, methods, and inheritance. Include associations, the dynamic part of a model.
- **Design:** Develop a structure/architecture where logical and physical details are discussed
- **Evolution:** As it relates to the implementation
- **Maintainance:** Maintainance following the delivery of the product

Micro process

- Identify classes and objects
- Identify semantics as it relates to programming
- Identify interactions/relationships
- Note abstraction as it relates to interfaces and their implementation

Jacobson Methodology (OOSE)

The Jacobson methodology, also known as Object-Oriented Software Engineering (OOSE) or even Objectory, is a method used to plan, design, and implement object-oriented software. The method is broken down into five parts: a set of requirements, an analysis, a design, an implementation, and a testing model. Uniquely, the methodology or OOSE utilizes use cases in its design.

Components

OOSE can be broken down into five separate components as we previously mentioned: a set of requirements, an analysis, a design, an implementation, and a testing model. These five parts correspond with certain verbs we can associate them with. The requirements model is expresses, the analysis model is structures, the design model realizes, the implementation model implements, and the test model verifies.

Models:
Requirements: Create problem domain object diagram (as they satisfy requirements) and specifies use case diagrams
Analysis: Analysis diagrams (these are similar to the ones we covered in the other methods)
Design: State transition diagrams and interaction diagrams (these are similar to the ones we covered in the other methods)
Implementation: Implementation of the model
Testing Model: Testing of the model

Use Cases

Use cases help us understand the how we want to design our system; more specifically, they are scenarios for helping us understanding the requirements of our system.

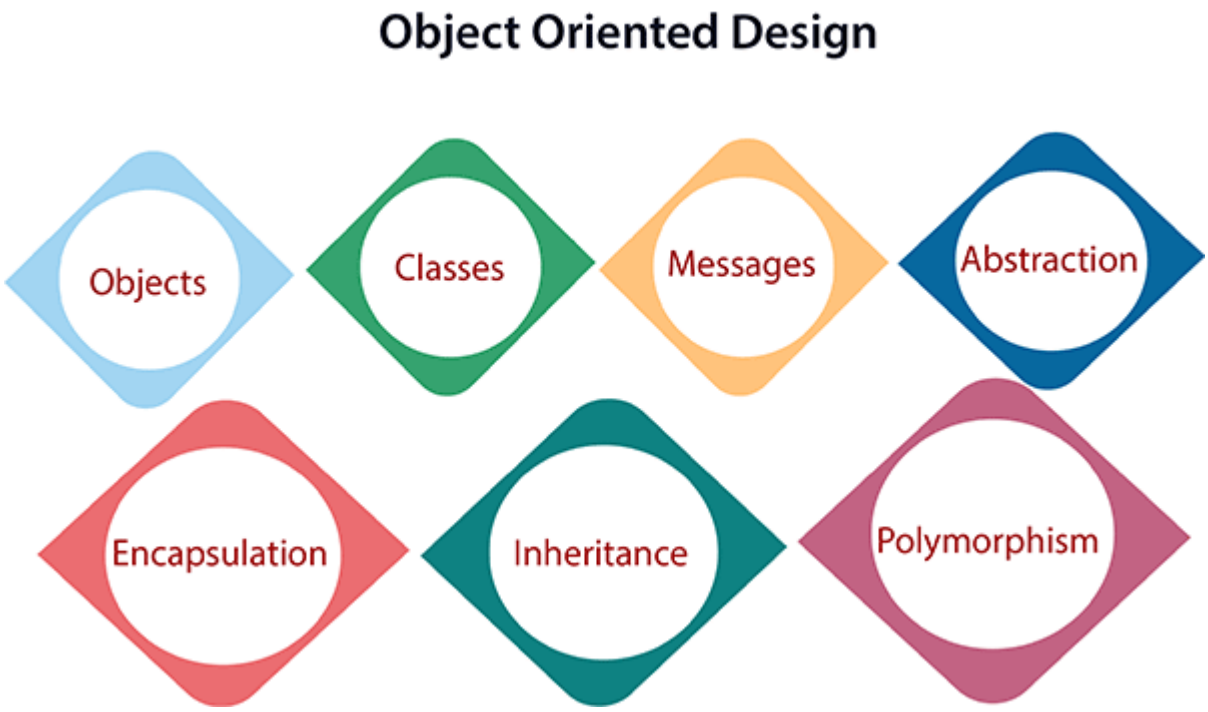
The Coad-Yourdon method

Coad-Yourdon methodology has its primary strength in system analysis. Their methodology is based on a technique called "SOSAS", which stands for the five steps that help make up the analysis part of their methodology. The first step in system analysis is called "Subjects", which are basically data flow diagrams for objects. The second step is called "Objects", where they identify the object classes and the class hierarchies. The third step is called "Structures", where they decompose structures into two types, classification structures and composition structures. Classification structures handle the inheritance connection between related classes, while composition structures handle all of the other connections among classes. The next step in analysis is called "Attributes", and the final step is called "Services", where all of the behaviours or methods for each class are identified.

(b) Write a note on Object Oriented Design. Discuss the importance of system design? What are activities and actions in dynamic model?

Object-Oriented Design

In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data. For example, in a Library Automation Software, each library representative may be a separate object with its data and functions to operate on these data. The tasks defined for one purpose cannot refer or change data of other objects. Objects have their internal data which represent their state. Similar objects create a class. In other words, each object is a member of some class. Classes may inherit features from the superclass.



The different terms related to object design are:

1. **Objects:** All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.
2. **Classes:** A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.
3. **Messages:** Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.
4. **Abstraction** In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.
5. **Encapsulation:** Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.
6. **Inheritance:** OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate superclasses.This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.
7. **Polymorphism:** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.

Benefits/Advantages of learning System Design

1. Improved System Performance

2. Increased Efficiency

3. Improved Communication

4. Reduced Risk

5. Improved Problem-Solving Skills

6. Increased Confidence

7. Greater Understanding of Your Code

8. Ability to Design Better Systems

9. Improved Career Prospects

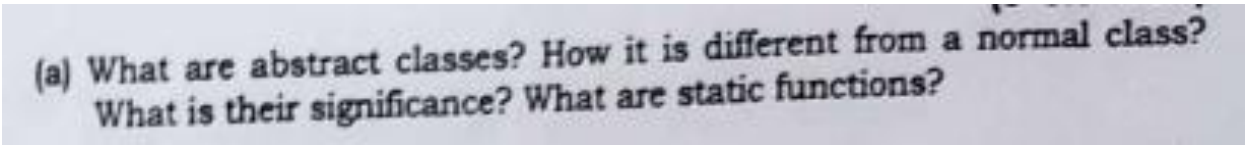
10. Ability to Create Scalable Systems

Activity

Activity is an operation upon the states of an object that requires some time period. They are the ongoing executions within a system that can be interrupted. Activities are shown in activity diagrams that portray the flow from one activity to another.

Action

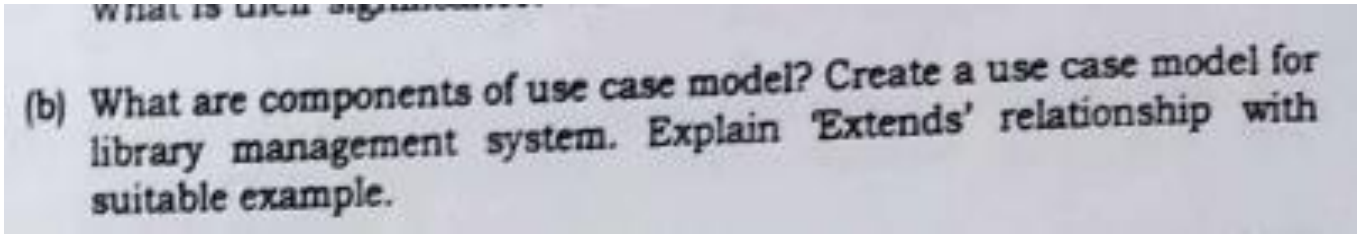
An action is an atomic operation that executes as a result of certain events. By atomic, it is meant that actions are un-interruptible, i.e., if an action starts executing, it runs into completion without being interrupted by any event. An action may operate upon an object on which an event has been triggered or on other objects that are visible to this object. A set of actions comprise an activity.


(a) What are abstract classes? How it is different from a normal class? What is their significance? What are static functions?

Abstract Class	Concrete Class
An abstract class is declared using abstract modifier.	A concrete class is not declared using abstract modifier.
An abstract class cannot be directly instantiated using the new keyword.	A concrete class can be directly instantiated using the new keyword.
An abstract class may or may not contain abstract methods.	A concrete class cannot contain an abstract method.

Abstract Class	Concrete Class
An abstract class cannot be declared as final.	A concrete class can be declared as final.
Implement an interface is possible by not providing implementations of all of the interface's methods. For this a child class is needed.	Easy implementation of all of the methods in the interface.

If the keyword **static** is prefixed before the function name, the function is called a **static function**. It is often called a **method**.


(b) What are components of use case model? Create a use case model for library management system. Explain 'Extends' relationship with suitable example.

There are various components of the basic model:

1. Actor
2. Use Case
3. Associations

Actor

Usually, actors are people involved with the system defined on the basis of their roles. An actor can be anything such as human or another external system.

Use Case

The use case defines how actors use a system to accomplish a specific objective. The use cases are generally introduced by the user to meet the objectives of the activities and variants involved in the achievement of the goal.

Associations

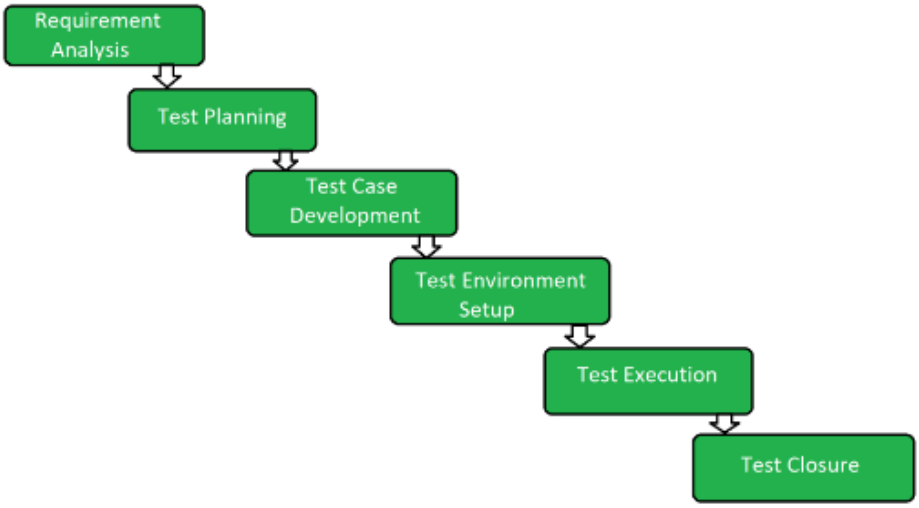
Associations are another component of the basic model. It is used to define the associations among actors and use cases they contribute in. This association is called communicates-association.

(a) Differentiate between testing and debugging? Explain the Testing Life Cycle. Write a note on Object Oriented testing strategies?

Testing can be done by insiders as well as outsiders.	Debugging is done only by insiders. An outsider can't do debugging.
Testing can be manual or automated.	Debugging is always manual. Debugging can't be automated.
It is based on different testing levels i.e. unit testing, integration testing, system testing, etc.	Debugging is based on different types of bugs.
Testing is a stage of the software development life cycle (SDLC).	Debugging is not an aspect of the software development life cycle, it occurs as a consequence of testing.
Testing is composed of the validation and verification of software.	While debugging process seeks to match symptoms with cause, by that it leads to error correction.
Testing is initiated after the code is written.	Debugging commences with the execution of a test case.
Testing process based on various levels of testing-system testing, integration testing, unit testing, etc.	Debugging process based on various types of bugs is present in a system.

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

Phases of STLC:

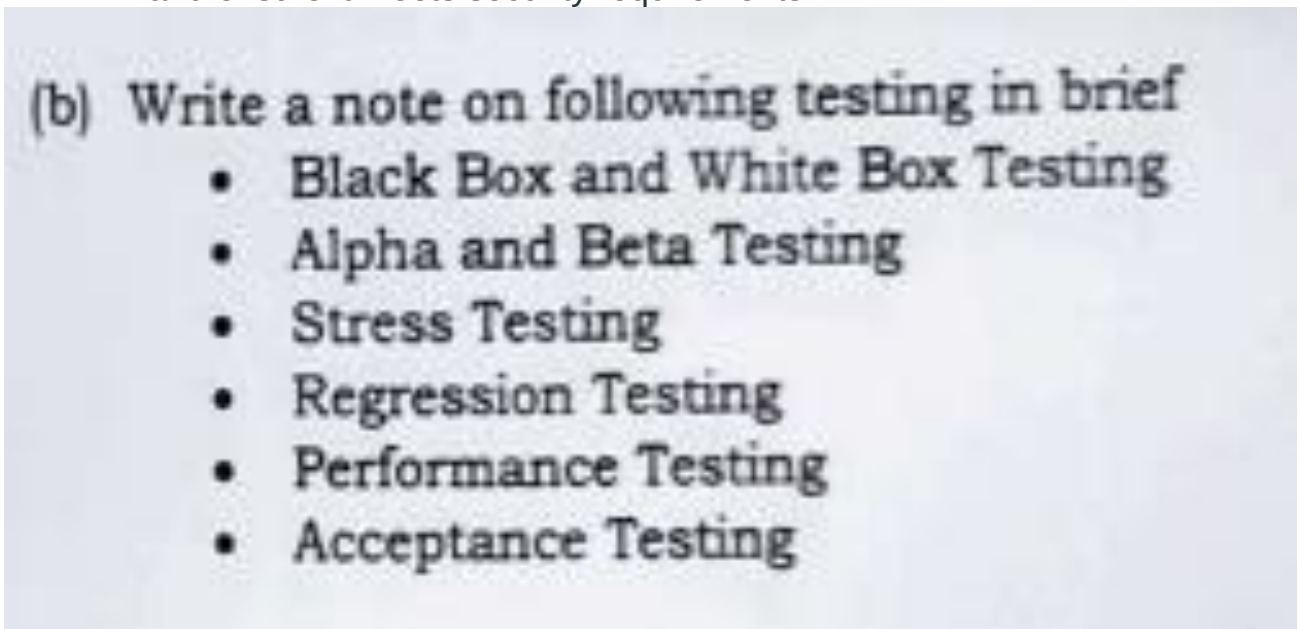


1. **Requirement Analysis:**
   Requirement Analysis is the first step of Software Testing Life Cycle (STLC). In this phase quality assurance team understands the requirements like what is to be tested. If anything is missing or not understandable then quality assurance team meets with the stakeholders to better understand the detail knowledge of requirement.
2. **Test Planning:**
   Test Planning is most efficient phase of software testing life cycle where all testing plans are defined. In this phase manager of the testing team calculates estimated effort and cost for the testing work. This phase gets started once the requirement gathering phase is completed.
3. **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.
4. **Test Environment Setup:**
   Test environment setup is the vital part of the STLC. Basically test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process the testing team is not involved. either the developer or the customer creates the testing environment.

5. **Test Execution:**
   After the test case development and test environment setup test execution phase gets started. In this phase testing team start executing test cases based on prepared test cases in the earlier step.
6. **Test Closure:**
   This is the last stage of STLC in which the process of testing is analysed.

The following are common testing strategies:

1. **Black box testing** – Tests the functionality of the software without looking at the internal code structure.
2. **White box testing** – Tests the internal code structure and logic of the software.
3. **Unit testing** – Tests individual units or components of the software to ensure they are functioning as intended.
4. **Integration testing** – Tests the integration of different components of the software to ensure they work together as a system.
5. **Functional testing** – Tests the functional requirements of the software to ensure they are met.
6. **System testing** – Tests the complete software system to ensure it meets the specified requirements.
7. **Acceptance testing** – Tests the software to ensure it meets the customer's or end-user's expectations.
8. **Regression testing** – Tests the software after changes or modifications have been made to ensure the changes have not introduced new defects.
9. **Performance testing** – Tests the software to determine its performance characteristics such as speed, scalability, and stability.
10. **Security testing** – Tests the software to identify vulnerabilities and ensure it meets security requirements.



S. No.	Black Box Testing	White Box Testing
1.	It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
2.	Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
3.	It is mostly done by software testers.	It is mostly done by software developers.
4.	No knowledge of implementation is needed.	Knowledge of implementation is required.
5.	It can be referred to as outer or external software testing.	It is the inner or the internal software testing.
6.	It is a functional test of the software.	It is a structural test of the software.
7.	This testing can be initiated based on the requirement specifications document.	This type of testing of software is started after a detail design document.
8.	No knowledge of programming is required.	It is mandatory to have knowledge of programming.
9.	It is the behavior testing of the software.	It is the logic testing of the software.
10.	It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
11.	It is also called closed testing.	It is also called as clear box testing.
12.	It is least time consuming.	It is most time consuming.

S. No.	Black Box Testing	White Box Testing
13.	It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
14.	Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
15.	**Example:** Search something on google by using keywords	**Example:** By input to check and verify loops
16.	**Black-box test design techniques-** • Decision table testing • All-pairs testing • Equivalence partitioning • Error guessing	**White-box test design techniques-** • Control flow testing • Data flow testing • Branch testing
17.	**Types of Black Box Testing:** • Functional Testing • Non-functional testing • Regression Testing	**Types of White Box Testing:** • Path Testing • Loop Testing • Condition testing
18.	It is less exhaustive as compared to white box testing.	It is comparatively more exhaustive than black box testing.

Alpha Testing	Beta Testing
Alpha testing involves both the white box and black box testing.	Beta testing commonly uses black-box testing.
Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.

Alpha Testing	Beta Testing
Alpha testing is performed at the developer's site.	Beta testing is performed at the end-user of the product.
Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.
Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
Alpha testing requires a testing environment or a lab.	Beta testing doesn't require a testing environment or lab.
Alpha testing may require a long execution cycle.	Beta testing requires only a few weeks of execution.
Developers can immediately address the critical issues or fixes in alpha testing.	Most of the issues or feedback collected from the beta testing will be implemented in future versions of the product.
Multiple test cycles are organized in alpha testing.	Only one or two test cycles are there in beta testing.

Stress Testing

Stress testing is testing an application's stability and response time by applying load, which is more than the designed number of users for an application.

For example, your application handles 1000 users at a time with a response time of 4 seconds, then stress testing can be done by applying a load of more than 1000 users. Test the application with 1100,1200,1300 users and notice the response time. The goal is to verify the stability of an application under stress.

Regression Testing

Regression testing is testing of unchanged features of the application to make sure that any bug fixes, adding new features, deleting, or updating existing features, are not impacting the working application.

To find out regression scope is an important part in Regression Testing. To find out regression scope, Tester needs to find out the area of application where changes happened and the Impact of those changes on the entire application. It is difficult to cover the whole regression test suite in every release, so Automation Testing Tools are used in regression testing.

Performance Testing
Performance testing is testing of an application's stability and response time by applying load. The word stability means the ability of the application to withstand in the presence of load. Response time is how quickly an application is available to users. Performance testing is done with the help of tools. Loader.IO, JMeter, LoadRunner, etc. are good tools available in the market.

Acceptance Testing

Acceptance testing is a type of testing where client/business/customer test the software with real time business scenarios.

The client accepts the software only when all the features and functionalities work as expected. This is the last phase of testing, after which the software goes into production. This is also called User Acceptance Testing (UAT).