



the next phase may be an evolutionary development that includes developing a more detailed prototype for solving the risks.

The **risk-driven** feature of the spiral model allows it to accommodate any mixture of a specification-oriented, prototype-oriented, simulation-oriented, or another type of approach. An essential element of the model is that each period of the spiral is completed by a review that includes all the products developed during that cycle, including plans for the next cycle. The spiral model works for development as well as enhancement projects.

## Advantages

- High amount of risk analysis
- Useful for large and mission-critical projects.

## Disadvantages

- Can be a costly model to use.
- Risk analysis needed highly particular expertise
- Doesn't work well for smaller projects.

b) Differentiate between object oriented analysis and structured analysis.

Structured Analysis	Object-Oriented Analysis
<p><b>The main focus is on the process and procedures of the system.</b></p> <p><b>It uses System Development Life Cycle (SDLC) methodology for different purposes like planning, analyzing, designing, implementing, and supporting an information system.</b></p>	<p><b>The main focus is on data structure and real-world objects that are important.</b></p> <p><b>It uses Incremental or Iterative methodology to refine and extend our design.</b></p>
<p><b>It is suitable for well-defined projects with stable user requirements.</b></p> <p><b>Risk while using this analysis technique is high and reusability is also low.</b></p>	<p><b>It is suitable for large projects with changing user requirements.</b></p> <p><b>Risk while using this analysis technique is low and reusability is also high.</b></p>

<p><b>Structuring requirements include DFDs (Data Flow Diagram), Structured Analysis, ER (Entity Relationship) diagram, CFD (Control Flow Diagram), Data Dictionary, Decision table/tree, and the State transition diagram.</b></p> <p><b>This technique is old and is not preferred usually.</b></p>	<p><b>Requirement engineering includes the Use case model (find Use cases, Flow of events, Activity Diagram), the Object model (find Classes and class relations, Object interaction, Object to ER mapping), State chart Diagram, and deployment diagram.</b></p> <p><b>This technique is new and is mostly preferred.</b></p>
---	--

c) Elaborate the difference goals of UML.

## Goals of UML

- Since it is a general-purpose modelling 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 modelling approach that is used to model all the practical systems.

d) Differentiate between black box testing and white Box testing.

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.

S. No.	Black Box Testing	White Box 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.
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.	<b>Example:</b> Search something on google by using keywords	<b>Example:</b> By input to check and verify loops
16.	<b>Black-box test design techniques-</b> <ul style="list-style-type: none"> <li>• Decision table testing</li> <li>• All-pairs testing</li> <li>• Equivalence partitioning</li> <li>• Error guessing</li> </ul>	<b>White-box test design techniques-</b> <ul style="list-style-type: none"> <li>• Control flow testing</li> <li>• Data flow testing</li> <li>• Branch testing</li> </ul>
17.	<b>Types of Black Box Testing:</b> <ul style="list-style-type: none"> <li>• Functional Testing</li> <li>• Non-functional testing</li> <li>• Regression Testing</li> </ul>	<b>Types of White Box Testing:</b> <ul style="list-style-type: none"> <li>• Path Testing</li> <li>• Loop Testing</li> <li>• Condition testing</li> </ul>
18.	It is less exhaustive as compared to white box testing.	It is comparatively more exhaustive than black box testing.

(a) Explain with suitable examples the different types of requirements problems that should be identified and resolved during the requirements analysis activity. (6.5)

## **Problem 1: Customers don't (really) know what they want**

To solve this problem, you should:

- Ensure that you spend sufficient time at the start of the project on understanding the objectives, deliverables and scope of the project.
- Make visible any assumptions that the customer is using, and critically evaluate both the likely end-user benefits and risks of the project.
- Attempt to write a concrete vision statement for the project, which encompasses both the specific functions or user benefits it provides and the overall business problem it is expected to solve.
- Get your customer to read, think about and sign off on the completed software requirements specification, to align expectations and ensure that both parties have a clear understanding of the deliverable.

## **Problem 2: Requirements change during the course of the project**

To solve this problem, you should:

- Have a clearly defined process for receiving, analyzing and incorporating change requests, and make your customer aware of his/her entry point into this process.

- Set milestones for each development phase beyond which certain changes are not permissible — for example, disallowing major changes once a module reaches 75 percent completion.
- Ensure that change requests (and approvals) are clearly communicated to all stakeholders, together with their rationale, and that the master project plan is updated accordingly.

### **Problem 3: Customers have unreasonable timelines**

To solve this problem, you should:

- Convert the software requirements specification into a project plan, detailing tasks and resources needed at each stage and modeling best-case, middle-case and worst-case scenarios.
- Ensure that the project plan takes account of available resource constraints and keeps sufficient time for testing and quality inspection.
- Enter into a conversation about deadlines with your customer, using the figures in your draft plan as supporting evidence for your statements. Assuming that your plan is reasonable, it's quite likely that the ensuing negotiation will be both productive and result in a favorable outcome for both parties.

### **Problem 4: Communication gaps exist between customers, engineers and project managers**

To solve this problem, you should:

- Take notes at every meeting and disseminate these throughout the project team.
- Be consistent in your use of words. Make yourself a glossary of the terms that you're going to use right at the start, ensure all stakeholders have a copy, and stick to them consistently.

## Problem 5: The development team doesn't understand the politics of the customer's organization

To solve this problem, you should:

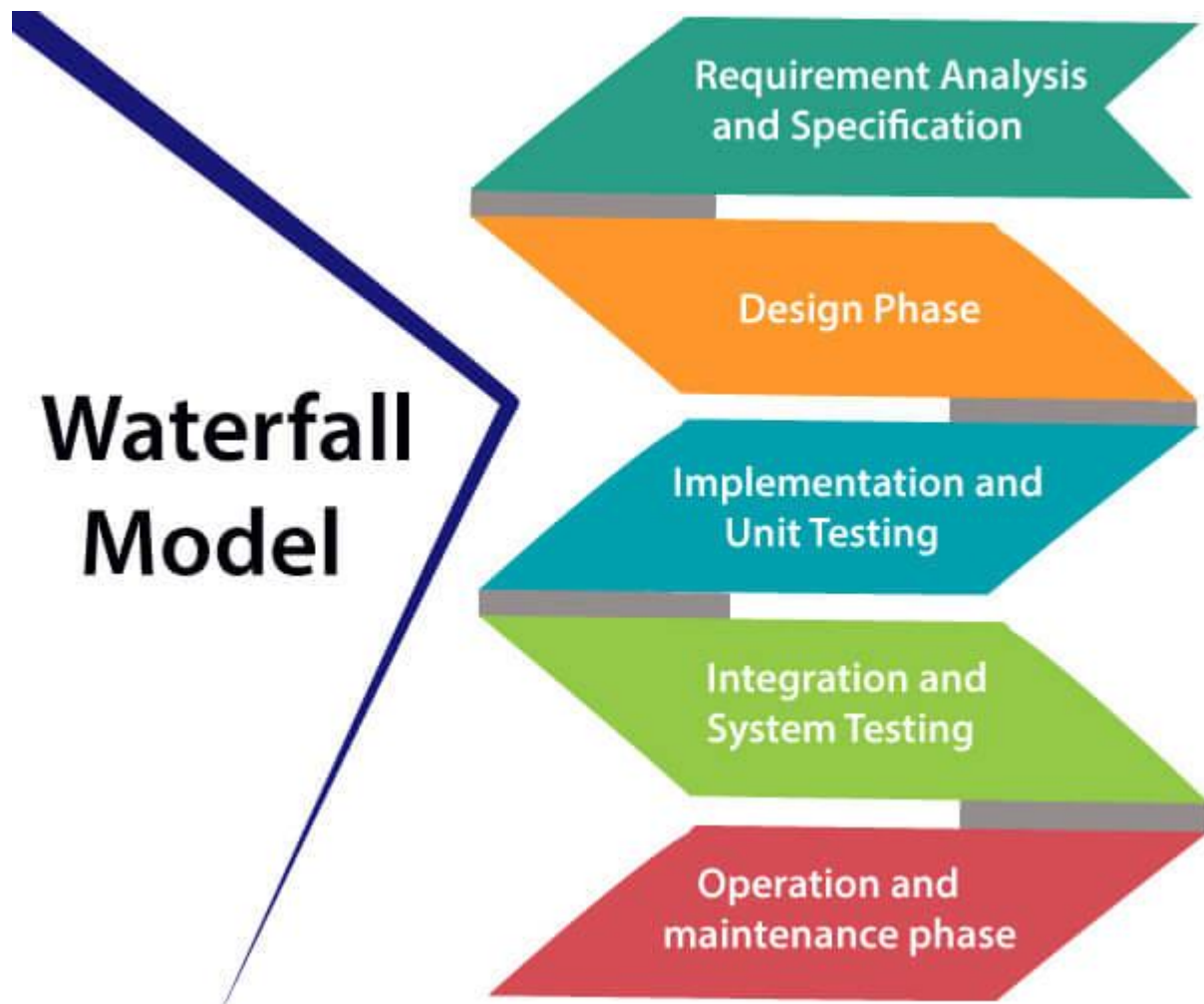
- Review your existing network and identify both the information you need and who is likely to have it.
- Cultivate allies, build relationships and think systematically about your social capital in the organization.
- Persuade opponents within your customer's organization by framing issues in a way that is relevant to their own experience.
- Use initial points of access/leverage to move your agenda forward.

(b) Define waterfall model and iterative waterfall model.

## Waterfall model

**1. Requirement analysis and specification phase:** The aim of this phase is to understand the exact requirements of the customer and to document them properly. Both the customer and the software developer work together so as to document all the functions, performance, and interfacing requirement of the software. It describes the "what" of the system to be produced and not "how." In this phase, a large document called **Software Requirement Specification (SRS)** document is created which contained a detailed description of what the system will do in the common language.





**2. Design Phase:** This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).

**3. Implementation and unit testing:** During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD.

During testing, the code is thoroughly examined and modified. Small modules are tested in isolation initially. After that these modules are tested by writing some overhead code to check the interaction between these modules and the flow of intermediate output.

**4. Integration and System Testing:** This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out. The better output will lead to satisfied customers, lower maintenance costs, and accurate results.

Unit testing determines the efficiency of individual modules. However, in this phase, the modules are tested for their interactions with each other and with the system.

**5. Operation and maintenance phase:** Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.

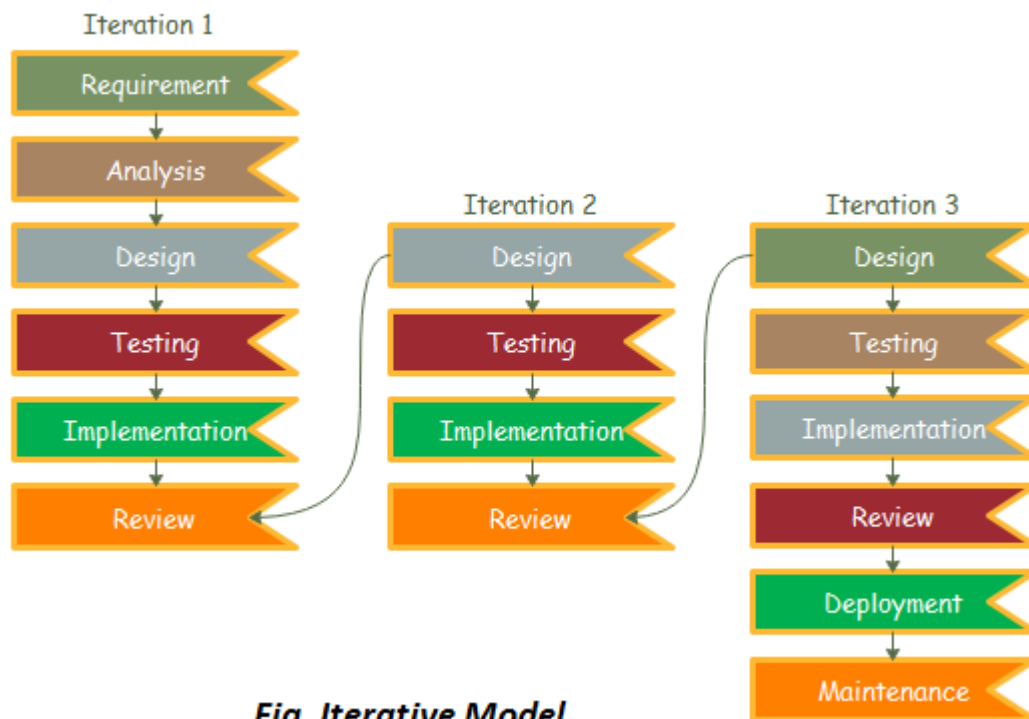
## Advantages of Waterfall model

- This model is simple to implement also the number of resources that are required for it is minimal.
- The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
- The start and end points for each phase is fixed, which makes it easy to cover progress.
- The release date for the complete product, as well as its final cost, can be determined before development.
- It gives easy to control and clarity for the customer due to a strict reporting system.

## Disadvantages of Waterfall model

- In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.
- This model cannot accept the changes in requirements during development.
- It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.
- Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

## Iterative Model



**Fig. Iterative Model**

**The various phases of Iterative model are as follows:**

**1. Requirement gathering & analysis:** In this phase, requirements are gathered from customers and check by an analyst whether requirements will fulfil or not. Analyst checks that need will achieve within budget or not. After all of this, the software team skips to the next phase.

**2. Design:** In the design phase, team design the software by the different diagrams like Data Flow diagram, activity diagram, class diagram, state transition diagram, etc.

**3. Implementation:** In the implementation, requirements are written in the coding language and transformed into computer programmes which are called Software.

**4. Testing:** After completing the coding phase, software testing starts using different test methods. There are many test methods, but the most common are white box, black box, and grey box test methods.

**5. Deployment:** After completing all the phases, software is deployed to its work environment.

**6. Review:** In this phase, after the product deployment, review phase is performed to check the behaviour and validity of the developed product. And if there are any error found then the process starts again from the requirement gathering.

**7. Maintenance:** In the maintenance phase, after deployment of the software in the working environment there may be some bugs, some errors or new updates are required. Maintenance involves debugging and new addition options.

## Advantage (Pros) of Iterative Model:

1. Testing and debugging during smaller iteration is easy.
2. A Parallel development can plan.
3. It is easily acceptable to ever-changing needs of the project.
4. Risks are identified and resolved during iteration.
5. Limited time spent on documentation and extra time on designing.

## Disadvantage (Cons) of Iterative Model:

1. It is not suitable for smaller projects.
2. More Resources may be required.
3. Design can be changed again and again because of imperfect requirements.
4. Requirement changes can cause over budget.
5. Project completion date not confirmed because of changing requirements.

b) Explain Object Oriented Methodology with suitable example. What are functional and non-functional requirement. Explain with example. (6.5)

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies "What should the software system do?"	It places constraints on "How should the software system fulfill the functional requirements?"
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g., Architect, Technical leaders and software developers.

Functional Requirements	Non Functional Requirements
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
Usually easy to define.	Usually more difficult to define.
<b>Example</b> <b>1)</b> Authentication of user whenever he/she logs into the system. <b>2)</b> System shutdown in case of a cyber attack. <b>3)</b> A Verification email is sent to user whenever he/she registers for the first time on some software system.	<b>Example</b> <b>1)</b> Emails should be sent with a latency of no greater than 12 hours from such an activity. <b>2)</b> The processing of each request should be done within 10 seconds <b>3)</b> The site should load in 3 seconds when the number of simultaneous users are > 10000

Q4. (a) What is the aim of the analysis model? What are the various dimensions of the analysis model? Describe various strategies for allocating the functionality. (6.5)

<https://www.geeksforgeeks.org/analysis-modelling-in-software-engineering/>

(b) Explain component diagram.

<https://www.javatpoint.com/uml-component-diagram>

(a) Define software testing. What is the purpose of integration testing? How is it done? What are various testing levels on which testing is done? Explain with examples. (6.5)

Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.

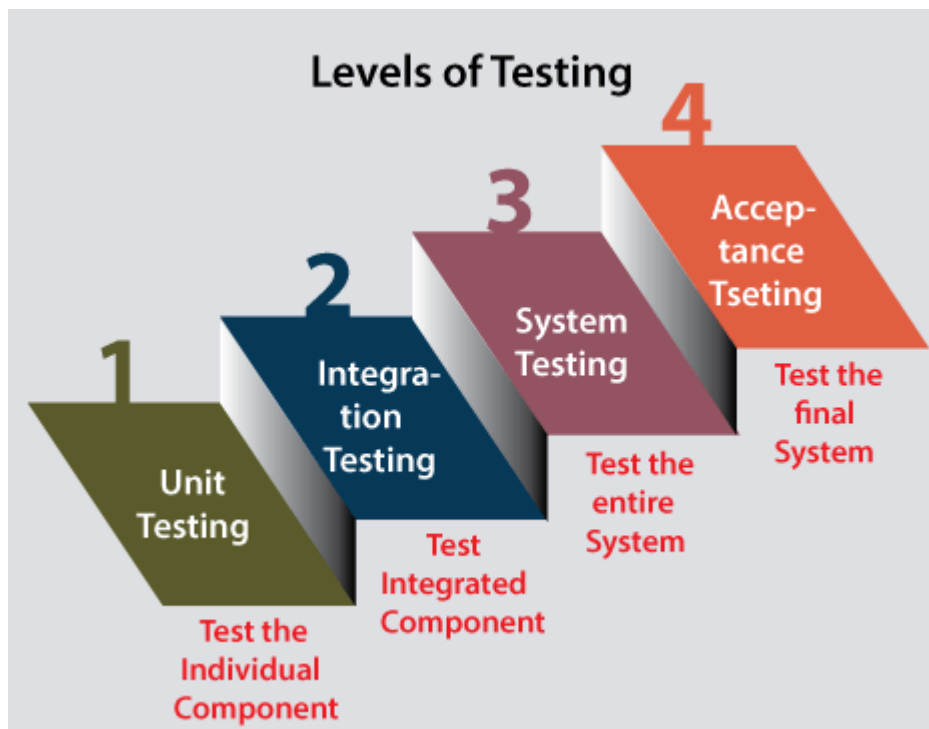
## Reason Behind Integration Testing

Although all modules of software application already tested in unit testing, errors still exist due to the following reasons:

1. Each module is designed by individual software developer whose programming logic may differ from developers of other modules so; integration testing becomes essential to determine the working of software modules.
2. To check the interaction of software modules with the database whether it is an erroneous or not.
3. Requirements can be changed or enhanced at the time of module development. These new requirements may not be tested at the level of unit testing hence integration testing becomes mandatory.
4. Incompatibility between modules of software could create errors.
5. To test hardware's compatibility with software.
6. If exception handling is inadequate between modules, it can create bugs.

In [software testing](#), we have four different levels of testing, which are as discussed below:

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



## Level1: Unit Testing

**Unit testing** is the first level of software testing, which is used to test if software modules are satisfying the given requirement or not.

The first level of testing involves **analysing each unit or an individual component** of the software application.

Unit testing is also the first level of **functional testing**. The primary purpose of executing unit testing is to validate unit components with their performance.

A unit component is an individual function or regulation of the application, or we can say that it is the smallest testable part of the software. The reason of performing the unit testing is to test the correctness of inaccessible code.

Unit testing will help the test engineer and developers in order to understand the base of code that makes them able to change defect causing code quickly. The developers implement the unit.

## Level2: Integration Testing

The second level of software testing is the **integration testing**. The integration testing process comes after **unit testing**.

It is mainly used to test the **data flow from one module or component to other modules**.

In integration testing, the **test engineer** tests the units or separate components or modules of the software in a group.

The primary purpose of executing the integration testing is to identify the defects at the interaction between integrated components or units.

When each component or module works separately, we need to check the data flow between the dependent modules, and this process is known as **integration testing**.

We only go for the integration testing when the functional testing has been completed successfully on each application module.

In simple words, we can say that **integration testing** aims to evaluate the accuracy of communication among all the modules.

## Level3: System Testing

The third level of software testing is **system testing**, which is used to test the software's functional and non-functional requirements.

It is **end-to-end testing** where the testing environment is parallel to the production environment. In the third level of software testing, **we will test the application as a whole system**.

To check the end-to-end flow of an application or the software as a user is known as **System testing**.

In system testing, we will go through all the necessary modules of an application and test if the end features or the end business works fine, and test the product as a complete system.



In simple words, we can say that System testing is a sequence of different types of tests to implement and examine the entire working of an integrated software computer system against requirements.

## Level4: Acceptance Testing

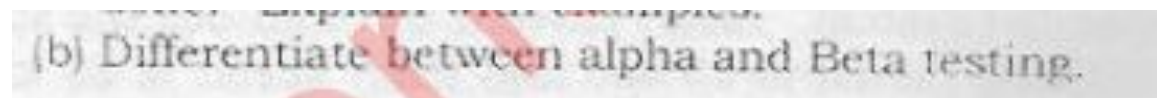
The **last and fourth level** of software testing is **acceptance testing**, which is used to evaluate whether a specification or the requirements are met as per its delivery.

The software has passed through three testing levels (**Unit Testing, Integration Testing, System Testing**). Some minor errors can still be identified when the end-user uses the system in the actual scenario.

In simple words, we can say that Acceptance testing is the **squeezing of all the testing processes that are previously done**.

The acceptance testing is also known as **User acceptance testing (UAT)** and is done by the customer before accepting the final product.

Usually, UAT is done by the domain expert (customer) for their satisfaction and checks whether the application is working according to given business scenarios and real-time scenarios.



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 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	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.

Q7. a) Differentiate between a package, component diagram and a deployment diagram. (6)

## Benefits of a package diagram

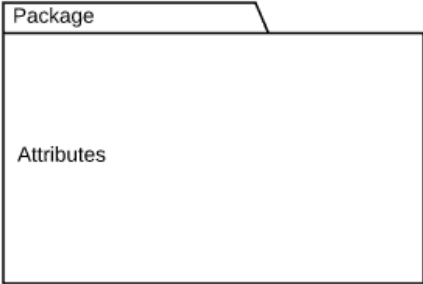

A well-designed package diagram provides numerous benefits to those looking to create a visualization of their UML system or project.

- They provide a clear view of the hierarchical structure of the various UML elements within a given system.
- These diagrams can simplify complex class diagrams into well-ordered visuals.
- They offer valuable high-level visibility into large-scale projects and systems.

- Package diagrams can be used to visually clarify a wide variety of projects and systems.
- These visuals can be easily updated as systems and projects evolve.

## Basic components of a package diagram

Each diagram includes only two symbols:

Symbol Image	Symbol Name	Description
	Package	Groups common elements based on data, behavior, or user interaction
	Dependency	Depicts the relationship between one element (package, named element, etc) and another

The purpose of deployment diagrams can be described as –

- Visualize the hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe the runtime processing nodes.

Before drawing a deployment diagram, the following artifacts should be identified –

- Nodes
- Relationships among nodes

An efficient deployment diagram controls the following parameters –

- Performance
- Scalability

- Maintainability
- Portability

The purpose of the component diagram can be summarized as –

- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

Before drawing a component diagram, the following artifacts are to be identified clearly –

- Files used in the system.
- Libraries and other artifacts relevant to the application.
- Relationships among the artifacts.

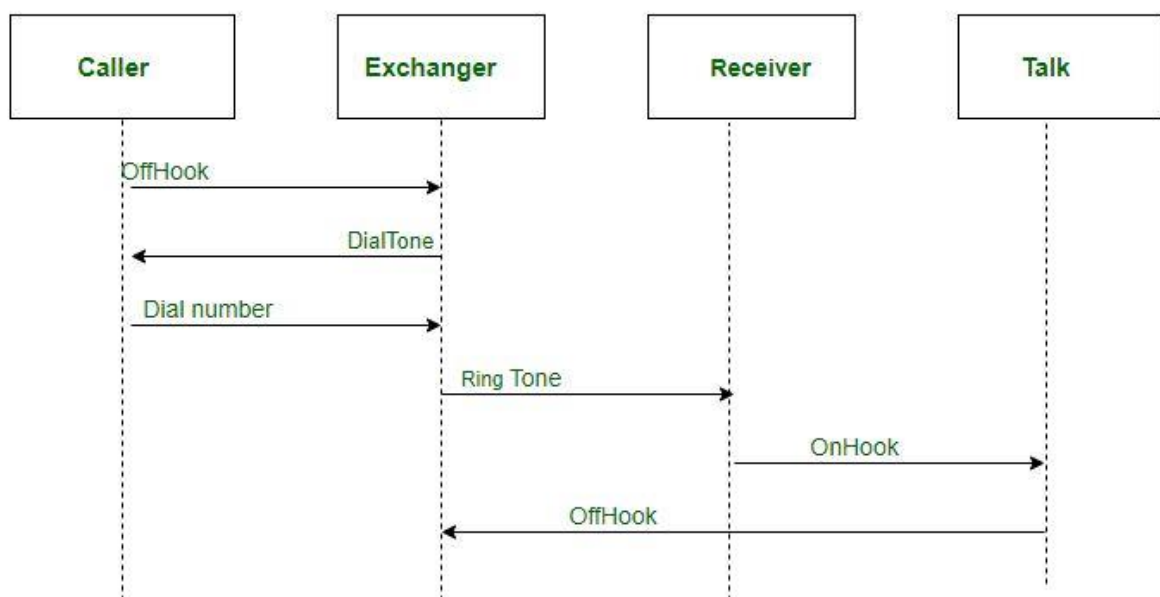
Component diagrams can be used to –

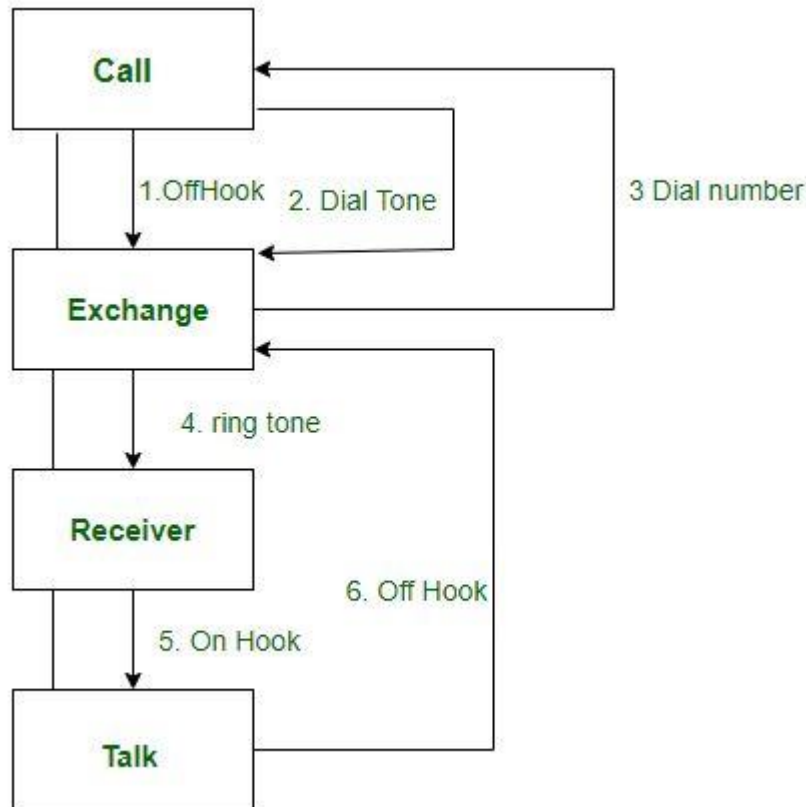
- Model the components of a system.
- Model the database schema.
- Model the executables of an application.
- Model the system's source code.

b) Differentiate between a sequence diagram and a collaboration diagram with the help of a suitable example. (6.5)

Sequence Diagrams	Collaboration Diagrams
<p>The sequence diagram represents the UML, which is used to visualize the sequence of calls in a system that is used to perform a specific functionality.</p>	<p>The collaboration diagram also comes under the UML representation which is used to visualize the organization of the objects and their interaction.</p>

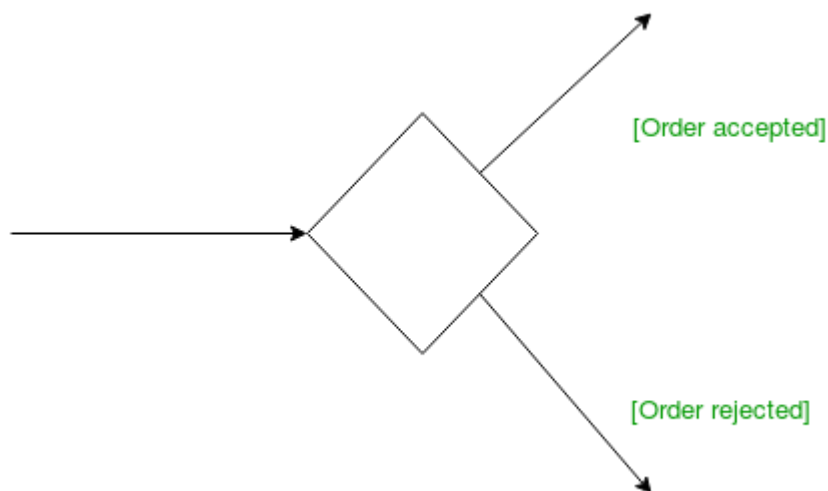
Sequence Diagrams	Collaboration Diagrams
<p>The sequence diagram is used to represent the sequence of messages that are flowing from one object to another.</p>	<p>The collaboration diagram is used to represent the structural organization of the system and the messages that are sent and received.</p>
<p>The sequence diagram is used when time sequence is main focus.</p>	<p>The collaboration diagram is used when object organization is main focus.</p>
<p>The sequence diagrams are better suited of analysis activities.</p>	<p>The collaboration diagrams are better suited for depicting simpler interactions of the smaller number of objects.</p>





c) Swim lanes & Guard conditions in Activity diagram.

**Guards** – A Guard refers to a statement written next to a decision node on an arrow sometimes within square

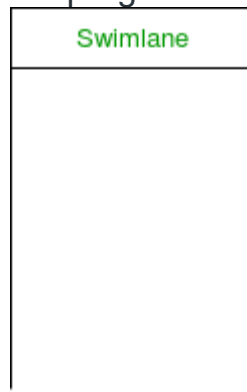


brackets.

– guards being used next to a decision node The statement must be true for the control to shift along a particular direction. Guards help us know the constraints and conditions which determine the flow of a process.

**Figure**

**Swimlanes** – We use swimlanes for grouping related activities in one column. Swimlanes group related activities into one column or one row. Swimlanes can be vertical and horizontal. Swimlanes are used to add modularity to the activity diagram. It is not mandatory to use swimlanes. They usually give more clarity to the activity diagram. It's similar to creating a function in a program. It's not mandatory to do so, but, it is a recommended



practice.

**Figure** – swimlanes notation We use a rectangular column to represent a swimlane as shown in the figure above. For example – Here different set of activities are executed based on if the number is odd or even. These activities are grouped into a swimlane.

