

## Practical – 5

**AIM:** Prepare the structural view: Draw Class diagram and object diagram.

- **Objectives:** to learn about UML class diagram and object diagram components.
- **Theory:**
  - **Class diagram:**
    - The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.
    - It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.
  - **Purpose of Class Diagrams**

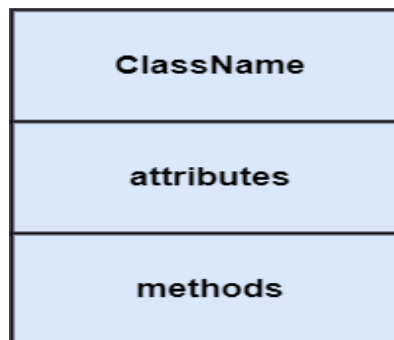
The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages. It is one of the most popular UML diagrams. Following are the purpose of class diagrams given below:

    1. It analyses and designs a static view of an application.
    2. It describes the major responsibilities of a system.
    3. It is a base for component and deployment diagrams.
    4. It incorporates forward and reverse engineering.
  - **Benefits of Class Diagrams**
    1. It can represent the object model for complex systems.
    2. It reduces the maintenance time by providing an overview of how an application is structured before coding.
    3. It provides a general schematic of an application for better understanding.
    4. It represents a detailed chart by highlighting the desired code, which is to be programmed.
    5. It is helpful for the stakeholders and the developers

## Vital components of a Class Diagram

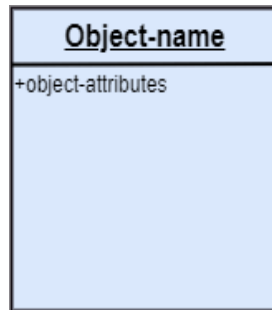
The class diagram is made up of three sections:

1. **Upper Section:** The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics. Some of the following rules that should be taken into account while representing a class are given below:
  1. Capitalize the initial letter of the class name.
  2. Place the class name in the center of the upper section.
  3. A class name must be written in bold format.
  4. The name of the abstract class should be written in italics format.
2. **Middle Section:** The middle section constitutes the attributes, which describe the quality of the class. The attributes have the following characteristics:
  1. The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~).
  2. The accessibility of an attribute class is illustrated by the visibility factors.
  3. A meaningful name should be assigned to the attribute, which will explain its usage inside the class.
3. **Lower Section:** The lower section contains methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.



### ○ Object diagram:

- Object diagrams are dependent on the class diagram as they are derived from the class diagram. It represents an instance of a class diagram. The objects help in portraying a static view of an object-oriented system at a specific instant.
- Both the object and class diagram are similar to some extent; the only difference is that the class diagram provides an abstract view of a system. It helps in visualizing a particular functionality of a system.



### ○ Purpose of Object Diagram

The object diagram holds the same purpose as that of a class diagram. The class diagram provides an abstract view which comprises of classes and their relationships, whereas the object diagram represents an instance at a particular point of time.

The object diagram is actually similar to the concrete (actual) system behavior. The main purpose is to depict a static view of a system.

## ● Background / Preparation:

### How to draw a Class Diagram?

The class diagram is used most widely to construct software applications. It not only represents a static view of the system but also all the major aspects of an application. A collection of class diagrams as a whole represents a system.

Some key points that are needed to keep in mind while drawing a class diagram are given below:

1. To describe a complete aspect of the system, it is suggested to give a meaningful name to the class diagram.
2. The objects and their relationships should be acknowledged in advance.
3. The attributes and methods (responsibilities) of each class must be known.
4. A minimum number of desired properties should be specified as more number of the unwanted property will lead to a complex diagram.
5. Notes can be used as and when required by the developer to describe the aspects of a diagram.
6. The diagrams should be redrawn and reworked as many times to make it correct before producing its final version.

- **How to draw an Object Diagram?**

1. All the objects present in the system should be examined before start drawing the object diagram.
2. Before creating the object diagram, the relation between the objects must be acknowledged.
3. The association relationship among the entities must be cleared already.
4. To represent the functionality of an object, a proper meaningful name should be assigned.
5. The objects are to be examined to understand its functionality.

- **How to draw an Object Diagram?**

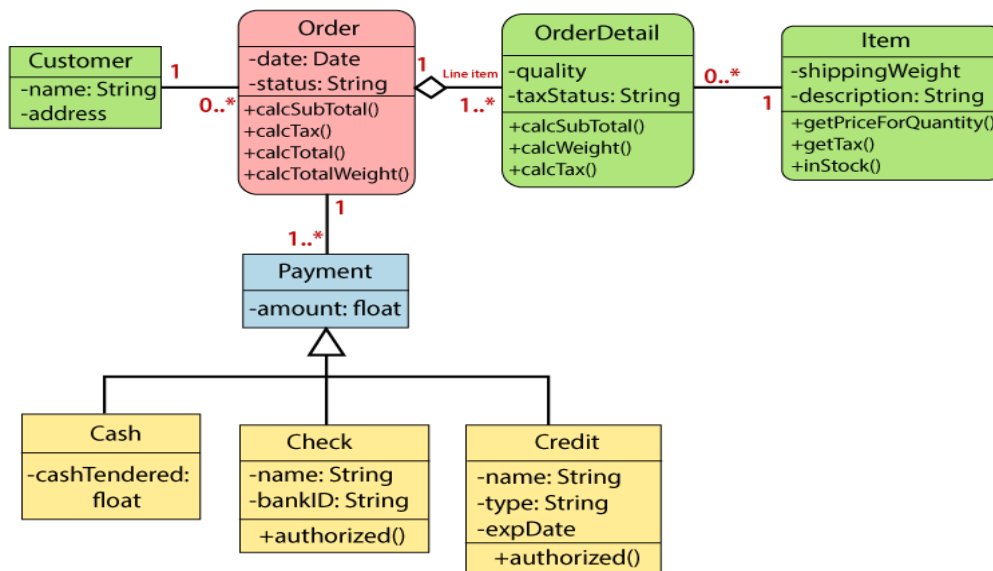
1. All the objects present in the system should be examined before start drawing the object diagram.
2. Before creating the object diagram, the relation between the objects must be acknowledged.
3. The association relationship among the entities must be cleared already.
4. To represent the functionality of an object, a proper meaningful name should be assigned.
5. The objects are to be examined to understand its functionality.

- **Tools / Material Needed:**

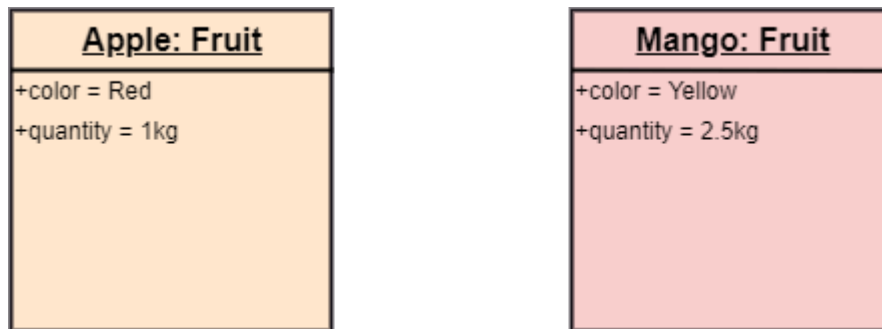
- **Hardware:**
- **Software:**

- **Procedure / Steps:**

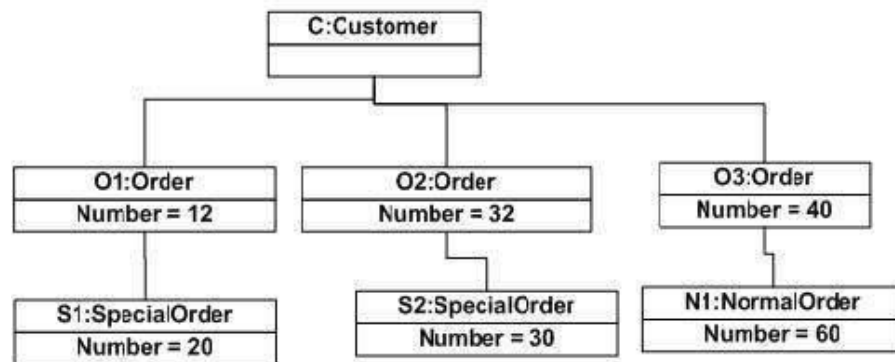
- **Example of sales order system**



- **Example of Object Diagram**



**Object diagram of an order management system**



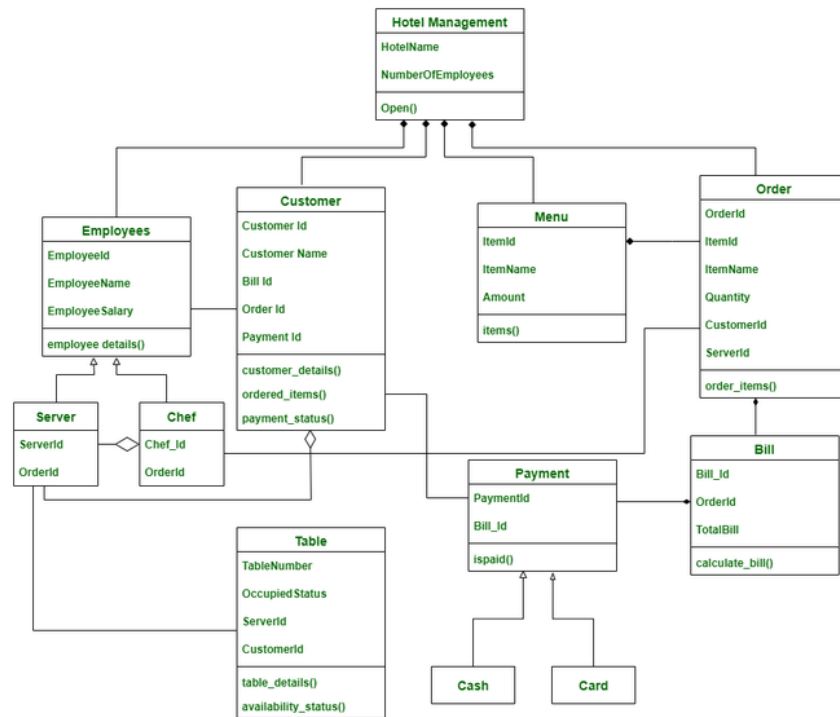
**Quiz:**

- 1) How to create domain model?
- 2) When to Use: Class Diagrams.
- 3) Define Object diagram.

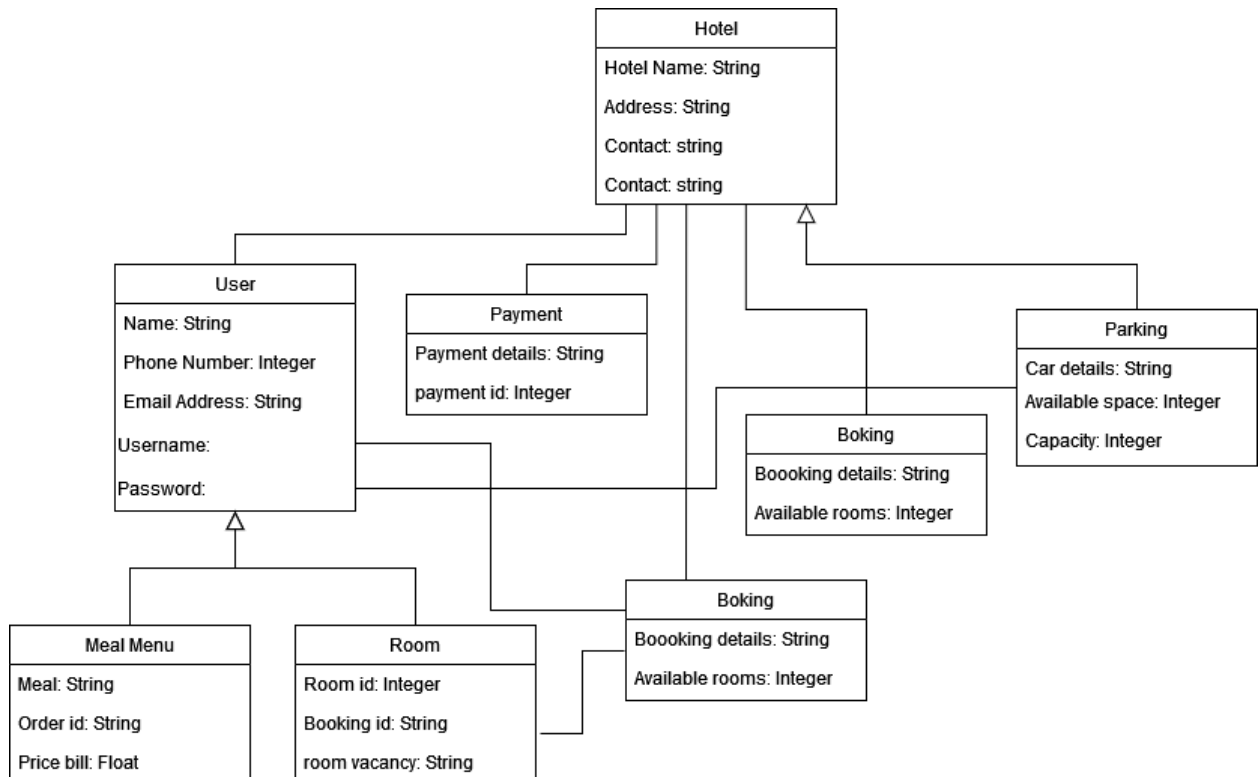
**Suggested Reference:**

- 1) Domain Analysis.
- 2) Domain Analysis Using Textual Analysis Approach
- 3) Domain model
- 4) Business Modeling – The Domain Mode
- 5) I. Y. Song, K. Yano, J. Trujillo, and S. Luján-Mora. "A Taxonomic Class Modeling Methodology for Object-Oriented Analysis", In Information Modeling Methods and Methodologies, Advanced Topics in Databases Series, Ed. (J Krostige, T. Halpin, K. Siau), Idea Group Publishing, 2004, pp. 216-240.

## Class Diagram for Hotel Management System:



## Object Diagram for Hotel Management System:



**Rubric wise marks obtained:**

<b>Rubrics</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>Total</b>
<b>Marks</b>	Complete implementation as asked	Complete implementation as asked  Problem analysis	Complete implementation as asked  Problem analysis  Development of the Solution	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding  Correct answer to all questions	

**Signature of Faculty**

## Practical – 6

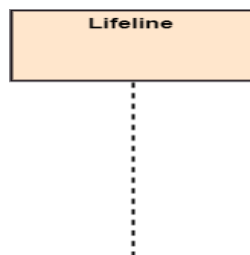
**AIM:** Prepare the behavioral view: Draw Sequence diagram and Collaboration diagram.

- **Objectives:** To explore and use various UML components of sequence diagram and collaboration diagram
- **Theory:**
  - **Sequence diagram:**
    - The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time. In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.
  - **Purpose of a Sequence Diagram**
    1. To model high-level interaction among active objects within a system.
    2. To model interaction among objects inside a collaboration realizing a use case.
    3. It either models generic interactions or some certain instances of interaction.

### Notations of a Sequence Diagram

#### *Lifeline*

An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram.



#### *Actor*

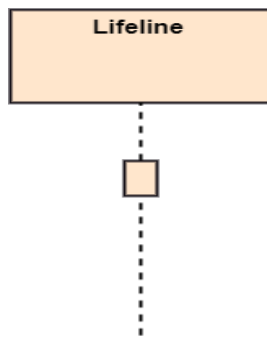
A role played by an entity that interacts with the subject is called as an actor. It is out of the scope of the system. It represents the role, which involves human users and external hardware or subjects. An actor may or may not represent a physical entity, but it purely depicts the role of an entity. Several distinct roles can be played by an actor or vice versa.





### *Activation*

It is represented by a thin rectangle on the lifeline. It describes that time period in which an operation is performed by an element, such that the top and the bottom of the rectangle is associated with the initiation and the completion time, each respectively.

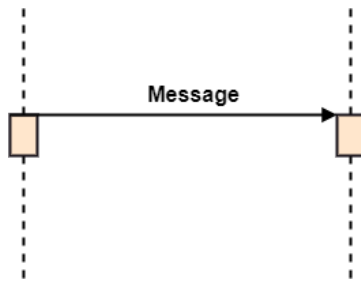


### *Messages*

The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of the sequence diagram is formed by messages and lifelines.

Following are types of messages enlisted below:

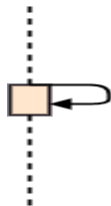
- **Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.



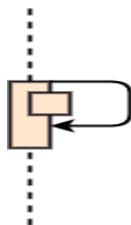
**Return Message:** It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.



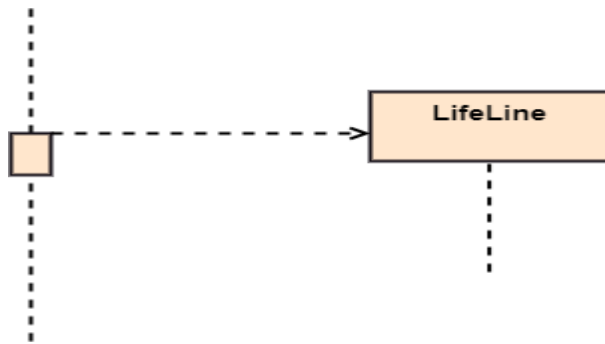
**Self Message:** It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.



**Recursive Message:** A self message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self message as it represents the recursive calls.



**Create Message:** It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.



**Destroy Message:** It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.



**Duration Message:** It describes a communication particularly between the lifelines of an interaction, which portrays the time passage of the message while modeling a system.

- **Collaboration diagram:**

- The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

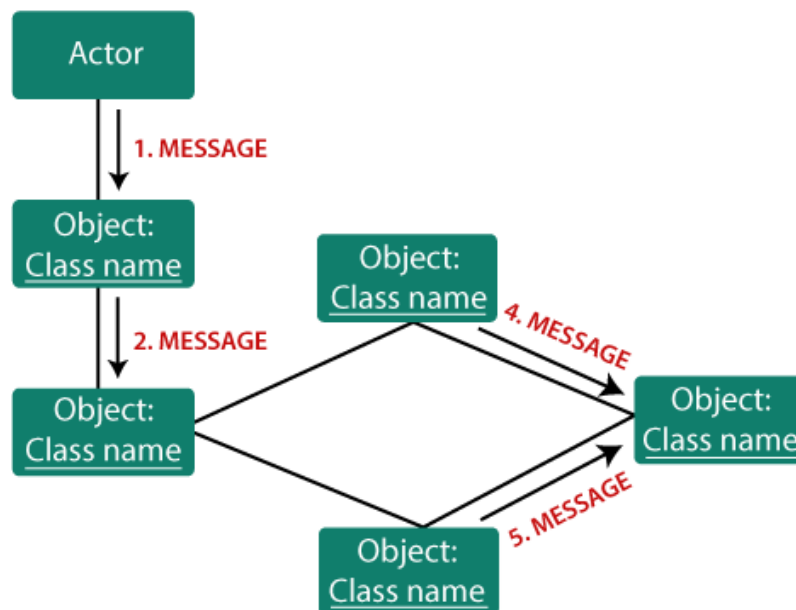
### Notations of a Collaboration Diagram

Following are the components of a component diagram that are enlisted below:

1. **Objects:** The representation of an object is done by an object symbol with its name and class underlined, separated by a colon. In the collaboration diagram, objects are utilized in the following ways:
  - The object is represented by specifying their name and class.
  - It is not mandatory for every class to appear.
  - A class may constitute more than one object.

- In the collaboration diagram, firstly, the object is created, and then its class is specified.
  - To differentiate one object from another object, it is necessary to name them.
2. **Actors:** In the collaboration diagram, the actor plays the main role as it invokes the interaction. Each actor has its respective role and name. In this, one actor initiates the use case.
  3. **Links:** The link is an instance of association, which associates the objects and actors. It portrays a relationship between the objects through which the messages are sent. It is represented by a solid line. The link helps an object to connect with or navigate to another object, such that the message flows are attached to links.
  4. **Messages:** It is a communication between objects which carries information and includes a sequence number, so that the activity may take place. It is represented by a labeled arrow, which is placed near a link. The messages are sent from the sender to the receiver, and the direction must be navigable in that particular direction. The receiver must understand the message.

### Components of a collaboration diagram

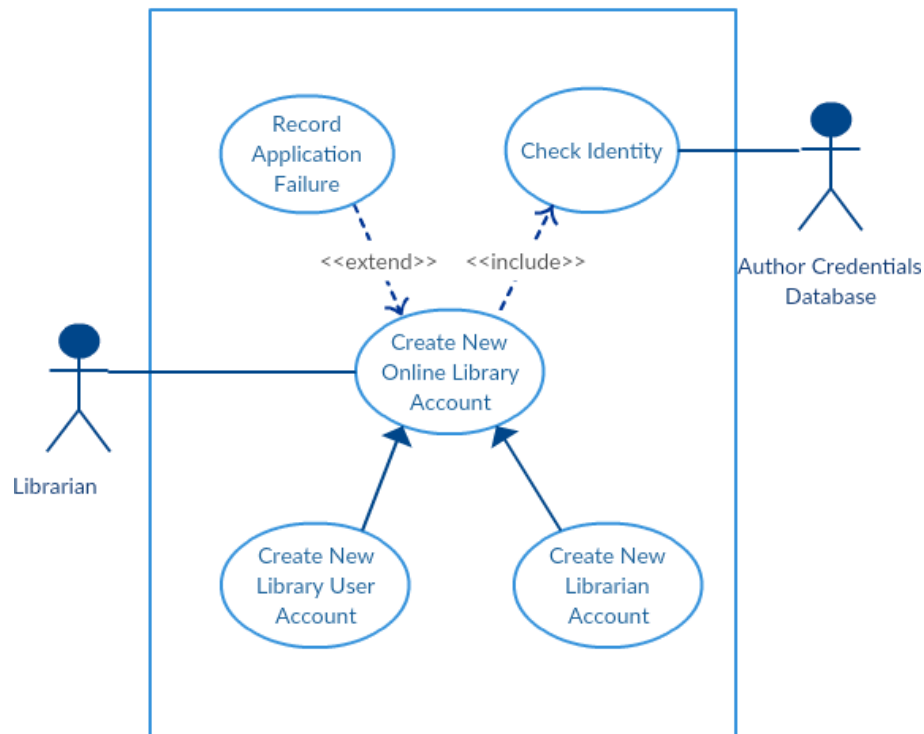


### • Background / Preparation:

- *How to Draw a Sequence Diagram*
  - A sequence diagram represents the scenario or flow of events in one single use case. The message flow of the sequence diagram is based on the narrative of the particular use case.
  - Then, before you start drawing the sequence diagram or decide what interactions

should be included in it, you need to draw the use case diagram and ready a comprehensive description of what the particular use case

#### Online Library Management System



From the above use case diagram example of ‘Create New Online Library Account’, we will focus on the use case named ‘Create New User Account’ to draw our sequence diagram example.

Before drawing the sequence diagram, it’s necessary to identify the objects or actors that would be involved in creating a new user account. These would be;

- Librarian
- Online Library Management system
- User credentials database
- Email system

Once you identify the objects, it is then important to write a detailed description on what the use case does. From this description, you can easily figure out the interactions (that should go in the sequence diagram) that would occur between the objects above, once the use case is executed.

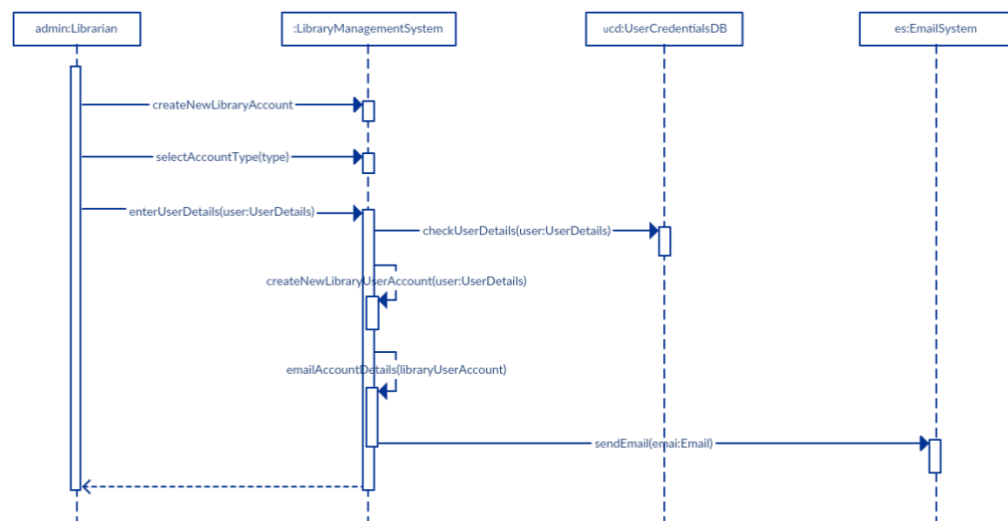
Here are the steps that occur in the use case named ‘Create New Library User Account’.

- The librarian request the system to create a new online library account

- The librarian then selects the library user account type
- The librarian enters the user's details
- The user's details are checked using the user Credentials Database
- The new library user account is created
- A summary of the of the new account's details are then emailed to the user

From each of these steps, you can easily specify what messages should be exchanged between the objects in the sequence diagram. Once it's clear, you can go ahead and start drawing the sequence diagram.

The sequence diagram below shows how the objects in the online library management system interact with each other to perform the function 'Create New Library User Account'.



- **Tools / Material Needed:**

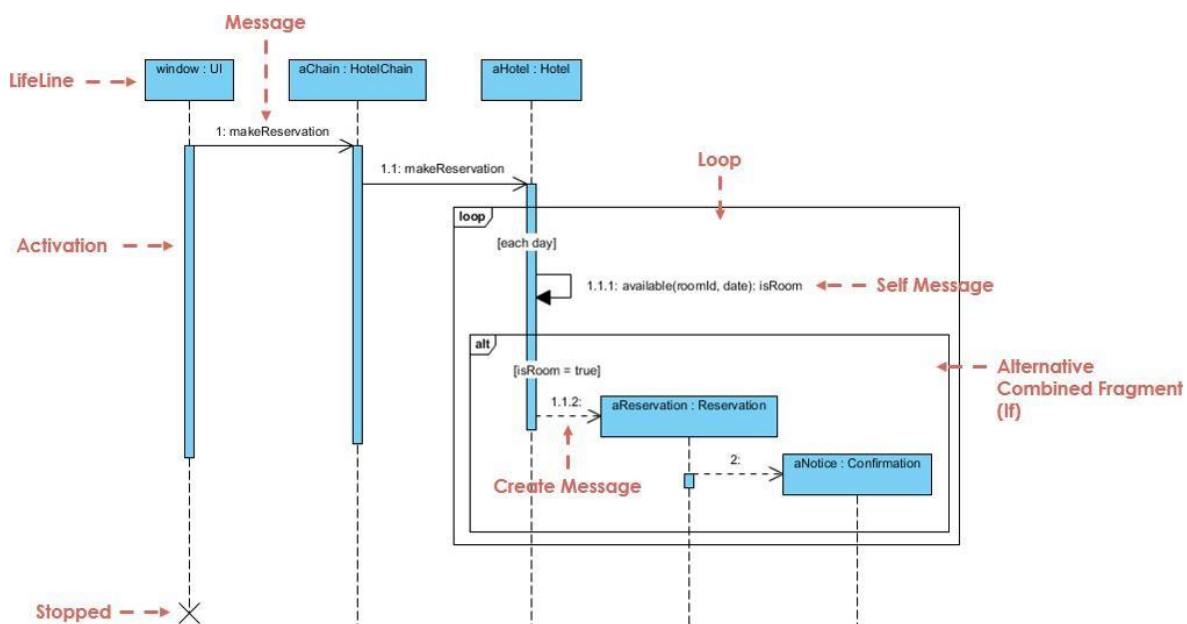
- **Hardware:**
- **Software:**

- **Procedure / Steps:**

- Sequence Diagram Example: Hotel System

Sequence Diagram is an interaction diagram that details how operations are carried out -- what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Below is a sequence diagram for making a hotel reservation. The object initiating the sequence of messages is a Reservation window.

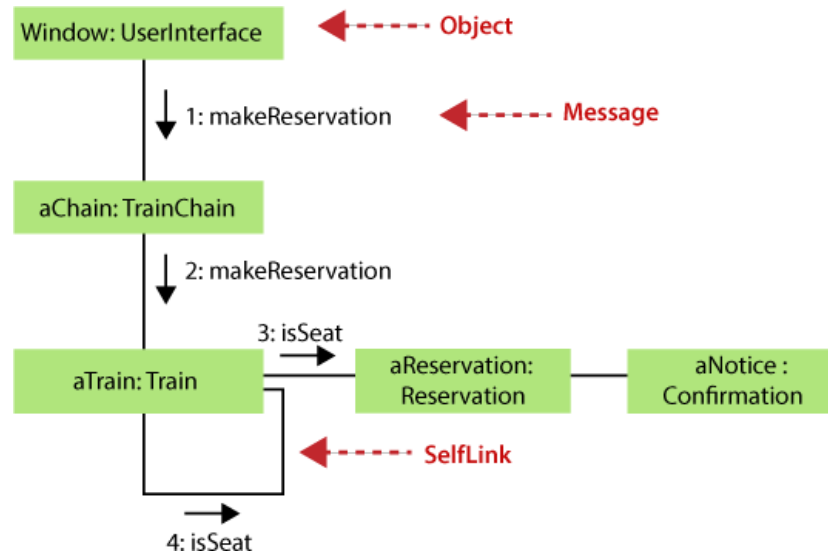


Note That: Class and object diagrams are static model views. Interaction diagrams are dynamic. They describe how objects collaborate.

### ○ Steps for creating a Collaboration Diagram

- Determine the behavior for which the realization and implementation are specified.
- Discover the structural elements that are class roles, objects, and subsystems for performing the functionality of collaboration.
  - Choose the context of an interaction: system, subsystem, use case, and operation.
- Think through alternative situations that may be involved.
  - Implementation of a collaboration diagram at an instance level, if needed.
  - A specification level diagram may be made in the instance level sequence diagram for summarizing alternative situations.

○ **Example of a Collaboration Diagram**



**Quiz:**

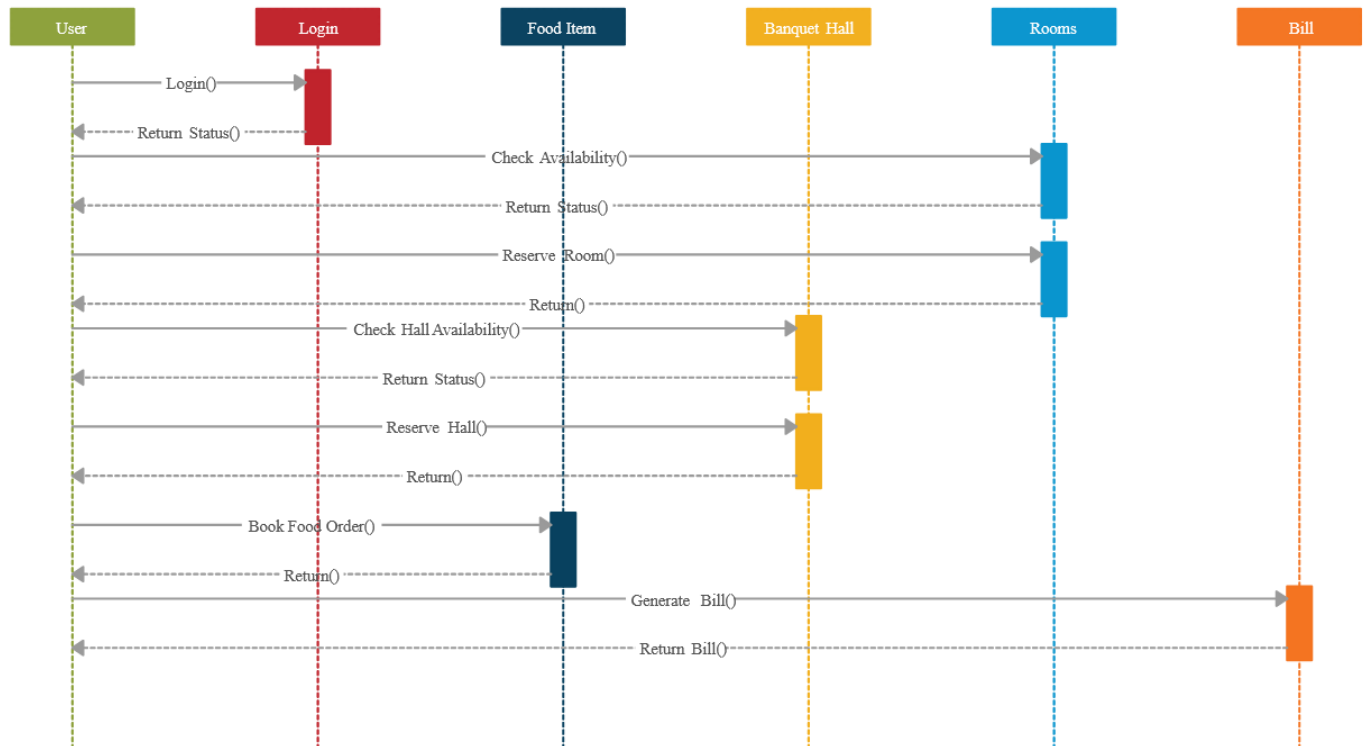
- 1) Difference between Sequence diagram and Collaboration diagram.
- 2) In a sequence diagram, what does a box depict? What does a dashed line depict? What does an arrow between boxes depict?
- 3) What does an X over a lifeline indicate?

**Suggested Reference:**

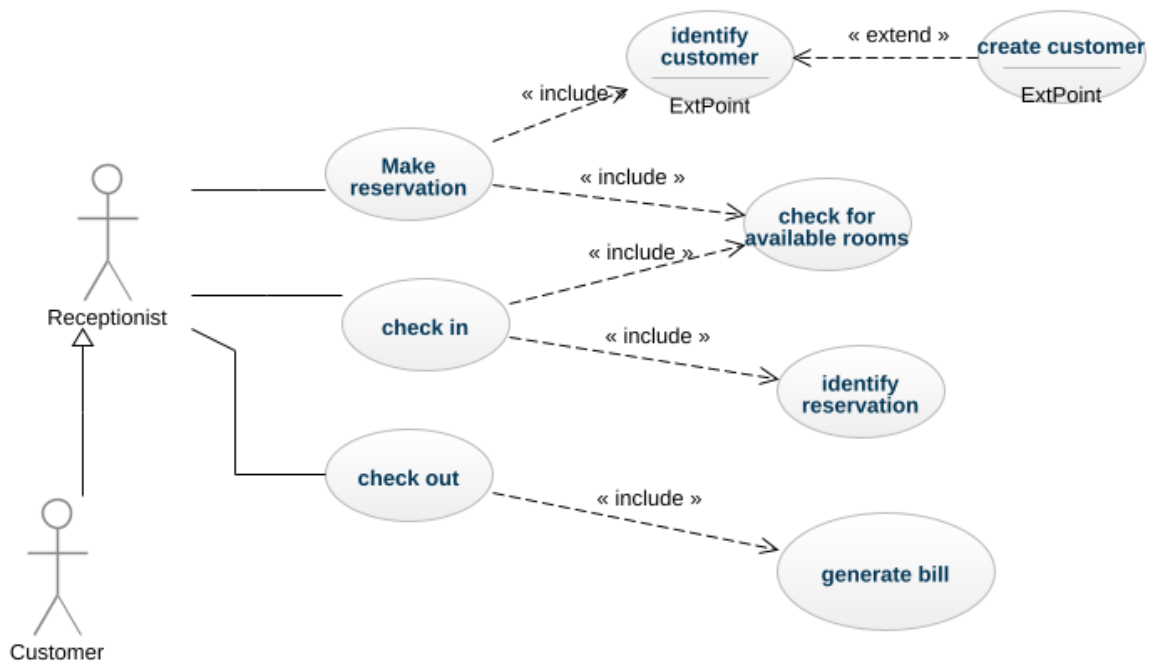
- 1) Booch, G. et al. The Unified Modeling Language User Guide. Chapters 15, 18, 27. Addison-Wesley.
- 2) Jacobson, I. et al. Object-Oriented Software Engineering: A Use-Case Driven Approach. Addison-Wesley.
- 3) Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modelling Language. Chapter 5. Addison Wesley.



## Sequence Diagram for Hotel Management System:



## Collaboration diagram For Hotel Management System:



**Rubric wise marks obtained:**

<b>Rubrics</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>Total</b>
<b>Marks</b>	Complete implementation as asked	Complete implementation as asked  Problem analysis	Complete implementation as asked  Problem analysis  Development of the Solution	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding  Correct answer to all questions	

**Signature of Faculty:**

## Practical – 7

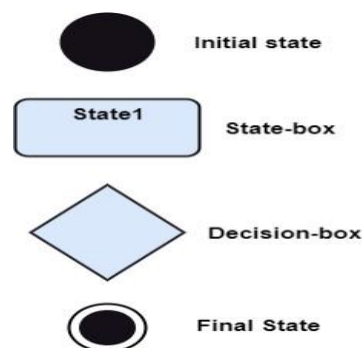
**AIM:** Prepare the behavioral view: Draw State-chart diagram and Activity diagram.

- **Objectives:**
  - To explore various components of UML state-chart diagram and use them.
  - To explore various elements of activity diagram and use them.
- **Theory:**
  - **State Machine Diagram:**
    - The state machine diagram is also called the Statechart or State Transition diagram, which shows the order of states underwent by an object within the system. It captures the software system's behavior. It models the behavior of a class, a subsystem, a package, and a complete system.
    - It tends out to be an efficient way of modeling the interactions and collaborations in the external entities and the system. It models event-based systems to handle the state of an object. It also defines several distinct states of a component within the system. Each object/component has a specific state.

Following are the types of a state machine diagram that are given below:

- **Behavioral state machine**
  - The behavioral state machine diagram records the behavior of an object within the system. It depicts an implementation of a particular entity. It models the behavior of the system.
- **Protocol state machine**
  - It captures the behavior of the protocol. The protocol state machine depicts the change in the state of the protocol and parallel changes within the system. But it does not portray the implementation of a particular component.
- **Notation of a State Machine Diagram**

Following are the notations of a state machine diagram enlisted below:



1. **Initial state:** It defines the initial state (beginning) of a system, and it is represented by a black filled circle.
2. **Final state:** It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.
3. **Decision box:** It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.
4. **Transition:** A change of control from one state to another due to the occurrence of some event is termed as a transition. It is represented by an arrow labeled with an event due to which the change has ensued.
5. **State box:** It depicts the conditions or circumstances of a particular object of a class at a specific point of time. A rectangle with round corners is used to represent the state box.

- **Background / Preparation:**

- **Types of State**

The UML consist of three states:

1. **Simple state:** It does not constitute any substructure.
2. **Composite state:** It consists of nested states (substates), such that it does not contain more than one initial state and one final state. It can be nested to any level.
3. **Submachine state:** The submachine state is semantically identical to the composite state, but it can be reused.

**When to use an Activity Diagram?**

An activity diagram can be used to portray business processes and workflows. Also, it used for modeling business as well as the software. An activity diagram is utilized for the followings:

1. To graphically model the workflow in an easier and understandable way.
2. To model the execution flow among several activities.
3. To model comprehensive information of a function or an algorithm employed within the system.
4. To model the business process and its workflow.
5. To envision the dynamic aspect of a system.
6. To generate the top-level flowcharts for representing the workflow of an application.
7. To represent a high-level view of a distributed or an object-oriented system.

- **Tools / Material Needed:**

- **Hardware:**
- **Software:**

- **Procedure / Steps:**

- **How to Draw a State Machine Diagram?**

The state machine diagram is used to portray various states underwent by an object. The change in one state to another is due to the occurrence of some event. All of the possible states of a particular component must be identified before drawing a state machine diagram.

The primary focus of the state machine diagram is to depict the states of a system. These states are essential while drawing a state transition diagram. The objects, states, and events due to which the state transition occurs must be acknowledged before the implementation of a state machine diagram.

Following are the steps that are to be incorporated while drawing a state machine diagram:

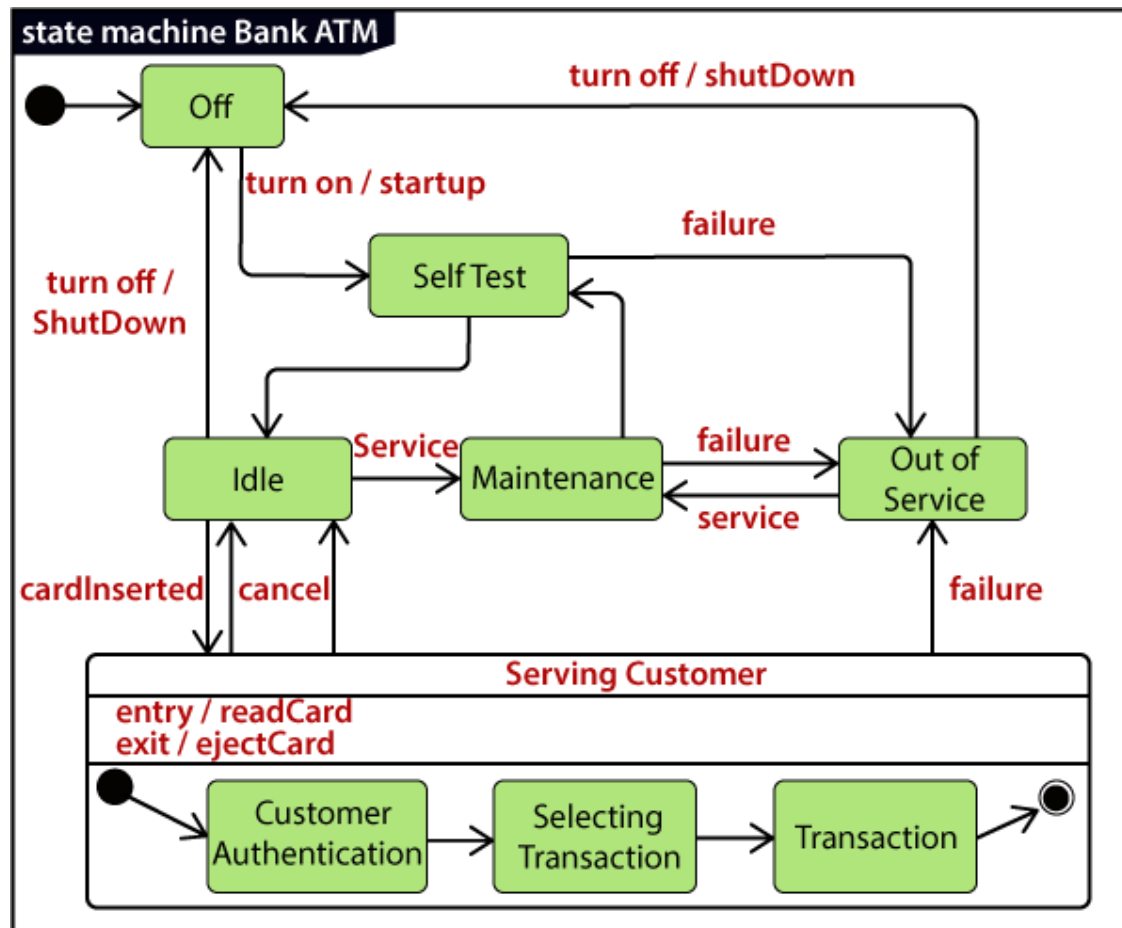
1. A unique and understandable name should be assigned to the state transition that describes the behavior of the system.
2. Out of multiple objects, only the essential objects are implemented.
3. A proper name should be given to the events and the transitions.

- **Example of a State Machine Diagram**

An example of a top-level state machine diagram showing Bank Automated TellerMachine (ATM) is given below.

Initially, the ATM is turned off. After the power supply is turned on, the ATM starts performing the startup action and enters into the **Self Test** state. If the test fails, the ATM will enter into the **Out Of Service** state, or it will undergo a **triggerless transition** to the **Idle** state. This is the state where the customer waits for the interaction.

Whenever the customer inserts the bank or credit card in the ATM's card reader, the ATM state changes from **Idle** to **Serving Customer**, the entry action **readCard** is performed after entering into **Serving Customer** state. Since the customer can cancel the transaction at any instant, so the transition from **Serving Customer** state back to the **Idle** state could be triggered by **cancel** event.



Here the **Serving Customer** is a composite state with sequential substates that are **Customer Authentication**, **Selecting Transaction**, and **Transaction**.

**Customer Authentication** and **Transaction** are the composite states itself is displayed by a hidden decomposition indication icon. After the transaction is finished, the **Serving Customer** encompasses a triggerless transition back to the **Idle** state. On leaving the state, it undergoes the exit action **ejectCard** that discharges the customer card.

#### ○ **Activity Diagram:**

- In UML, the activity diagram is used to demonstrate the flow of control within the system rather than the implementation. It models the concurrent and sequential activities.
- The activity diagram helps in envisioning the workflow from one activity to another. It put emphasis on the condition of flow and the order in which it occurs. The flow can

be sequential, branched, or concurrent, and to deal with such kinds of flows, the activity diagram has come up with a fork, join, etc.

- It is also termed as an object-oriented flowchart. It encompasses activities composed of a set of actions or operations that are applied to model the behavioral diagram.

- **Components of an Activity Diagram**

Following are the component of an activity diagram:

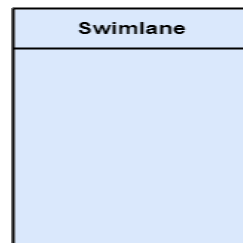
- **Activities**

- The categorization of behavior into one or more actions is termed as an activity. In other words, it can be said that an activity is a network of nodes that are connected by edges. The edges depict the flow of execution. It may contain action nodes, control nodes, or object nodes.
- The control flow of activity is represented by control nodes and object nodes that illustrates the objects used within an activity. The activities are initiated at the initial node and are terminated at the final node.



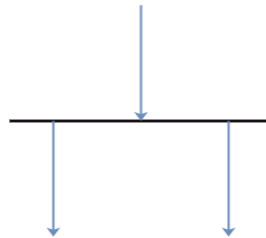
- **Activity partition /swimlane**

- The swimlane is used to cluster all the related activities in one column or one row. It can be either vertical or horizontal. It used to add modularity to the activity diagram. It is not necessary to incorporate swimlane in the activity diagram. But it is used to add more transparency to the activity diagram.



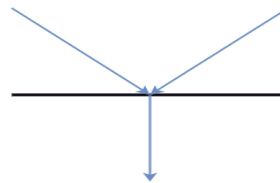
- **Forks**

- Forks and join nodes generate the concurrent flow inside the activity. A fork node consists of one inward edge and several outward edges. It is the same as that of various decision parameters. Whenever a data is received at an inward edge, it gets copied and split crossways various outward edges. It split a single inward flow into multiple parallel flows.



- **Join Nodes**

- Join nodes are the opposite of fork nodes. A Logical AND operation is performed on all of the inward edges as it synchronizes the flow of input across one single output (outward) edge.



- **Pins**

- It is a small rectangle, which is attached to the action rectangle. It clears out all the messy and complicated thing to manage the execution flow of activities. It is an object node that precisely represents one input to or output from the action.

- **Notation of an Activity diagram**

Activity diagram constitutes following notations:

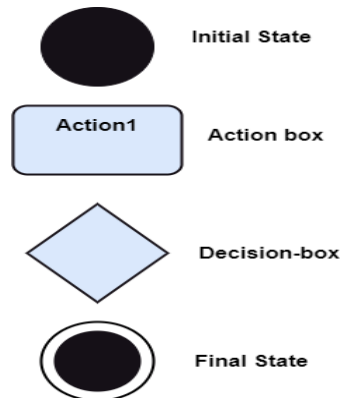
**Initial State:** It depicts the initial stage or beginning of the set of actions.

**Final State:** It is the stage where all the control flows and object flows end.

**Decision Box:** It makes sure that the control flow or object flow will follow only one path.

**Action Box:** It represents the set of actions that are to be performed.





### ○ **How to draw an Activity Diagram?**

An activity diagram is a flowchart of activities, as it represents the workflow among various activities. They are identical to the flowcharts, but they themselves are not exactly the flowchart. In other words, it can be said that an activity diagram is an enhancement of the flowchart, which encompasses several unique skills.

Since it incorporates swimlanes, branching, parallel flows, join nodes, control nodes, and forks, it supports exception handling. A system must be explored as a whole before drawing an activity diagram to provide a clearer view of the user. All of the activities are explored after they are properly analyzed for finding out the constraints applied to the activities. Each and every activity, condition, and association must be recognized.

After gathering all the essential information, an abstract or a prototype is built, which is then transformed into the actual diagram.

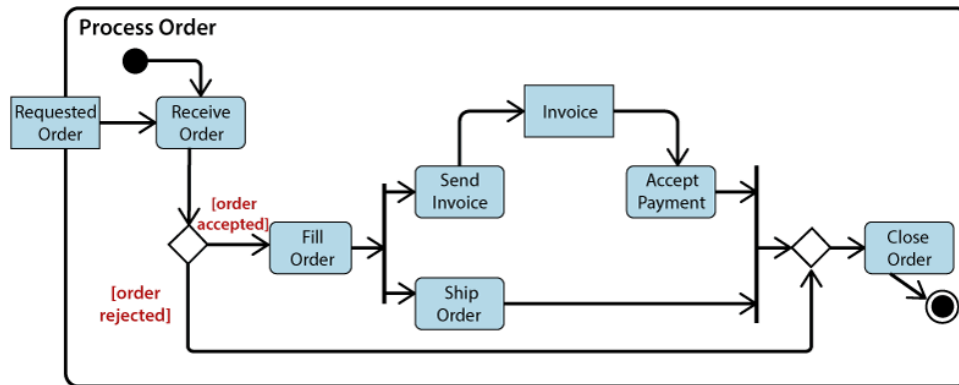
Following are the rules that are to be followed for drawing an activity diagram:

1. A meaningful name should be given to each and every activity.
2. Identify all of the constraints.
3. Acknowledge the activity associations.

### ○ **Example of an Activity Diagram**

An example of an activity diagram showing the business flow activity of order processing is given below.

Here the input parameter is the Requested order, and once the order is accepted, all of the required information is then filled, payment is also accepted, and then the order is shipped. It permits order shipment before an invoice is sent or payment is completed.



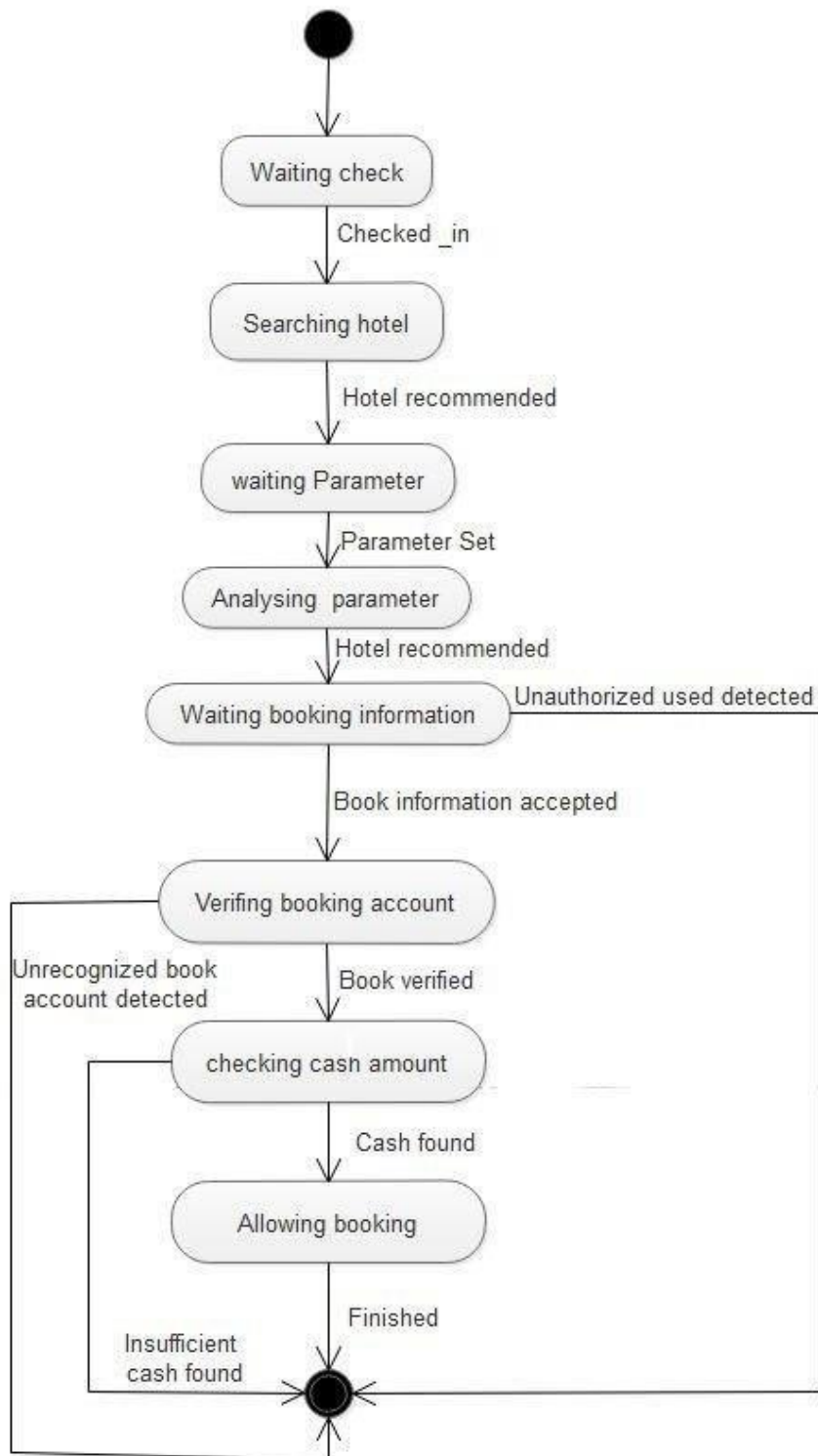
### Quiz:

- 1) Contrast Branches and Forks?
- 2) What are the initial and final states of an Activity diagram represented by?
- 3) What differentiates one state of a software component from another?
- 4) What is the source of a transition that begins at the border of a composite state?

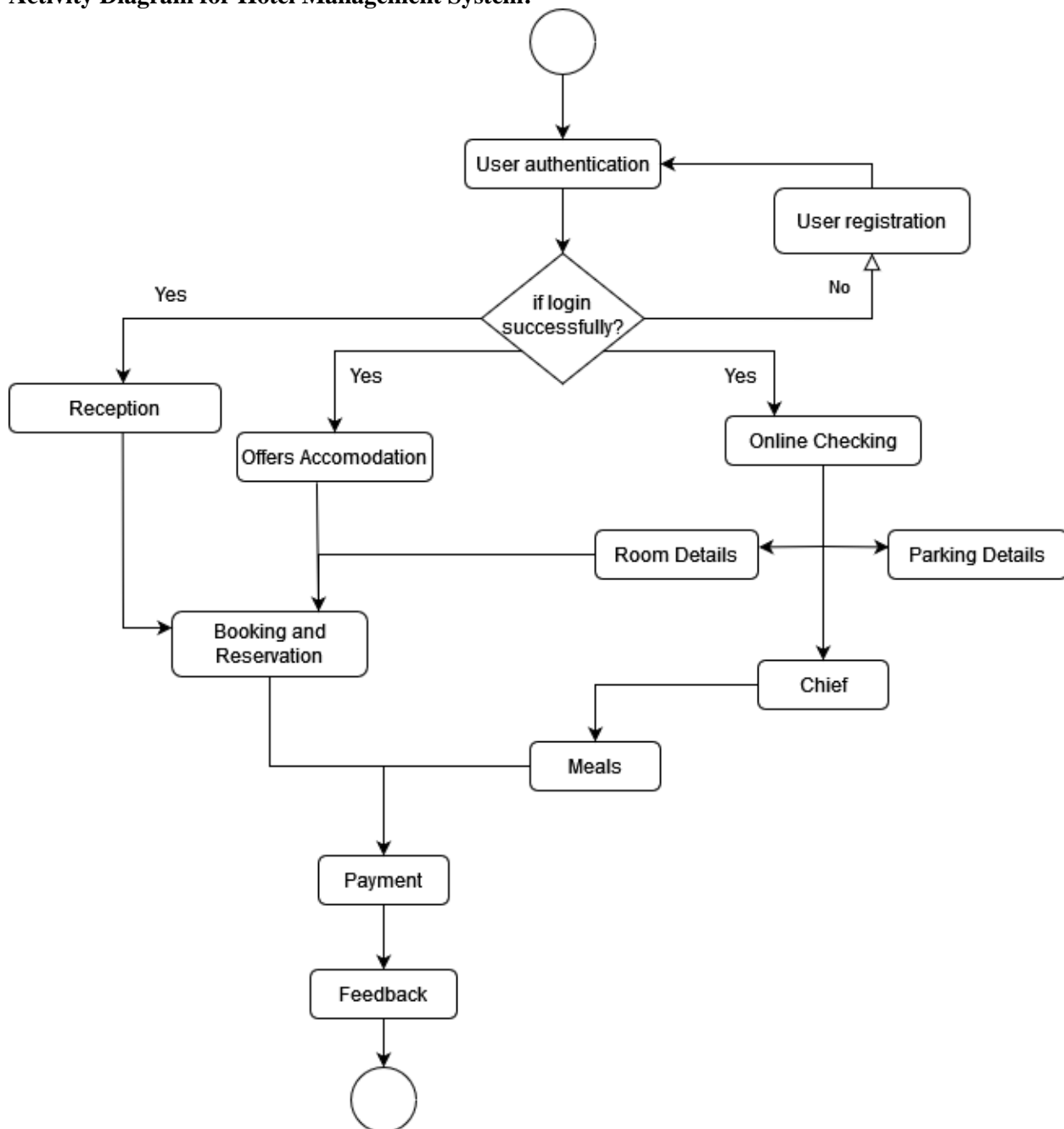
### Suggested Reference:

- 1) UML 2 State Machine Diagrams.
- 2) Modeling Behavior with UML Interactions and State charts
- 3) UML 2 State Machine Diagramming Guidelines
- 4) State chart Diagram
- 5) PlantUML (Open-Source tool in Java to draw UML Diagram)

**State chart diagram for Hotel Management System:**



**Activity Diagram for Hotel Management System:**



**Rubric wise marks obtained:**

<b>Rubrics</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>Total</b>
<b>Marks</b>	Complete implementation as asked	Complete implementation as asked  Problem analysis	Complete implementation as asked  Problem analysis  Development of the Solution	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding  Correct answer to all questions	

**Signature of Faculty:**

## Practical – 8

**AIM:** Prepare the implementation view: Draw Component diagram and Deployment diagram.

- **Objectives:** to explore and use various elements UML component diagram and deployment diagram.

- **Theory:**

- A component diagram is used to break down a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides within the node.
- It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behavior is explained by the provided and required interfaces.

- **Purpose of a Component Diagram**

Since it is a special kind of a UML diagram, it holds distinct purposes. It describes all the individual components that are used to make the functionalities, but not the functionalities of the system. It visualizes the physical components inside the system. The components can be a library, packages, files, etc.

The component diagram also describes the static view of a system, which includes the organization of components at a particular instant. The collection of component diagrams represents a whole system.

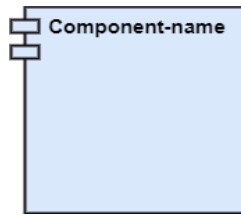
The main purpose of the component diagram are enlisted below:

1. It envisions each component of a system.
2. It constructs the executable by incorporating forward and reverse engineering.
3. It depicts the relationships and organization of components.

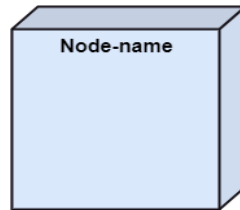
- **Background / Preparation:**

- **Notation of a Component Diagram**

- a) A component



b) A node



#### ○ When to use a Component Diagram?

It represents various physical components of a system at runtime. It is helpful in visualizing the structure and the organization of a system. It describes how individual components can together form a single system. Following are some reasons, which tells when to use component diagram:

1. To divide a single system into multiple components according to the functionality.
2. To represent the component organization of the system.

#### ● Tools / Material Needed:

- Hardware:
- Software:

#### ● Procedure / Steps:

##### ○ How to Draw a Component Diagram?

- The component diagram is helpful in representing the physical aspects of a system, which are files, executables, libraries, etc. The main purpose of a component diagram is different from that of other diagrams. It is utilized in the implementation phase of any application.
- Once the system is designed employing different UML diagrams, and the artifacts are prepared, the component diagram is used to get an idea of implementation. It plays an essential role in implementing applications efficiently.
- Following are some artifacts that are needed to be identified before drawing a component diagram:

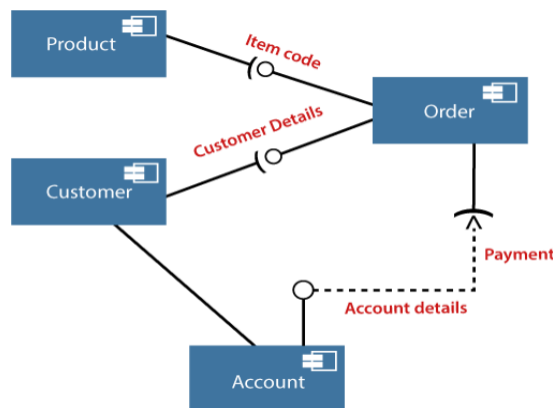
1. What files are used inside the system?
2. What is the application of relevant libraries and artifacts?
3. What is the relationship between the artifacts?

- Following are some points that are needed to be kept in mind after the artifacts are identified:

1. Using a meaningful name to ascertain the component for which the diagram is about to be drawn.
2. Before producing the required tools, a mental layout is to be made.
3. To clarify the important points, notes can be incorporated.

### ○ Example of a Component Diagram

- A component diagram for an online shopping system is given below:



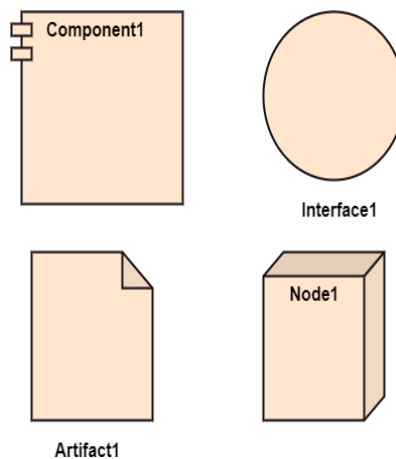
- The deployment diagram visualizes the physical hardware on which the software will be deployed. It portrays the static deployment view of a system. It involves the nodes and their relationships.
- It ascertains how software is deployed on the hardware. It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. Since it involves many nodes, the relationship is shown by utilizing communication paths.

### ○ Purpose of Deployment Diagram

- The main purpose of the deployment diagram is to represent how software is installed on the hardware component. It depicts in what manner a software interacts with hardware to perform its execution.
- Both the deployment diagram and the component diagram are closely interrelated to each other as they focus on software and hardware components. The component diagram represents the components of a system, whereas the deployment diagram describes how they are actually deployed on the hardware.
- The deployment diagram does not focus on the logical components of the system, but it



- put its attention on the hardware topology.
- Following are the purposes of deployment diagram enlisted below:
    - To envision the hardware topology of the system.
    - To represent the hardware components on which the software components are installed.
    - To describe the processing of nodes at the runtime.
  - **Symbol and notation of Deployment diagram**
    - The deployment diagram consist of the following notations:
      1. A component
      2. An artifact
      3. An interface
      4. A node



- **How to draw a Deployment Diagram?**
  - The deployment diagram portrays the deployment view of the system. It helps in visualizing the topological view of a system. It incorporates nodes, which are physical hardware. The nodes are used to execute the artifacts. The instances of artifacts can be deployed on the instances of nodes.
  - Since it plays a critical role during the administrative process, it involves the following parameters:
    1. High performance
    2. Scalability
    3. Maintainability
    4. Portability
    5. Easily understandable
  - One of the essential elements of the deployment diagram is the nodes and artifacts. So it is necessary to identify all of the nodes and the relationship between them. It becomes

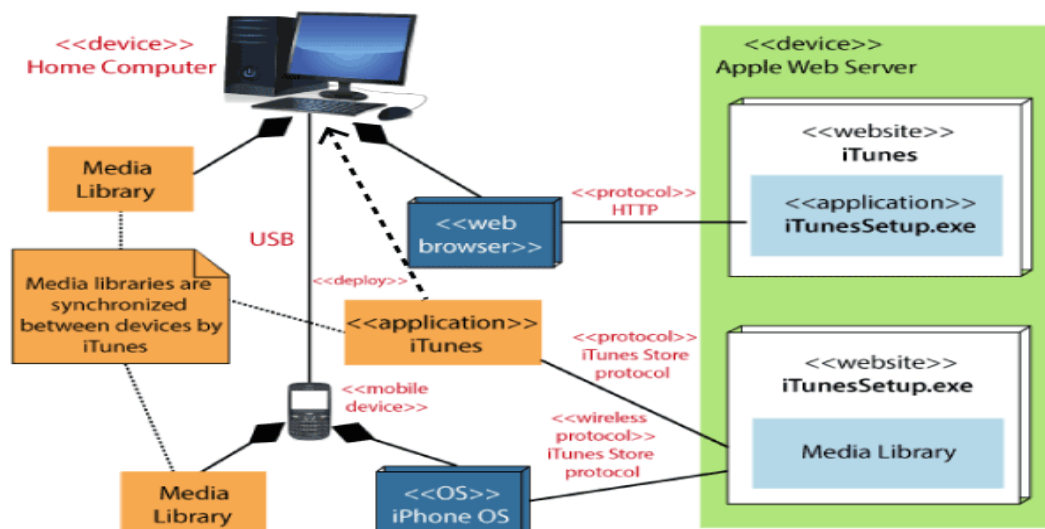
easier to develop a deployment diagram if all of the nodes, artifacts, and their relationship is already known.

- **Example of a Deployment diagram**

A deployment diagram for the Apple iTunes application is given below.

The iTunes setup can be downloaded from the iTunes website, and also it can be installed on the home computer. Once the installation and the registration are done, iTunes application can easily interconnect with the Apple iTunes store. Users can purchase and download music, video, TV serials, etc. and cache it in the media library.

Devices like Apple iPod Touch and Apple iPhone can update its own media library from the computer with iTunes with the help of USB or simply by downloading media directly from the Apple iTunes store using wireless protocols, for example; Wi-Fi, 3G, or EDGE.



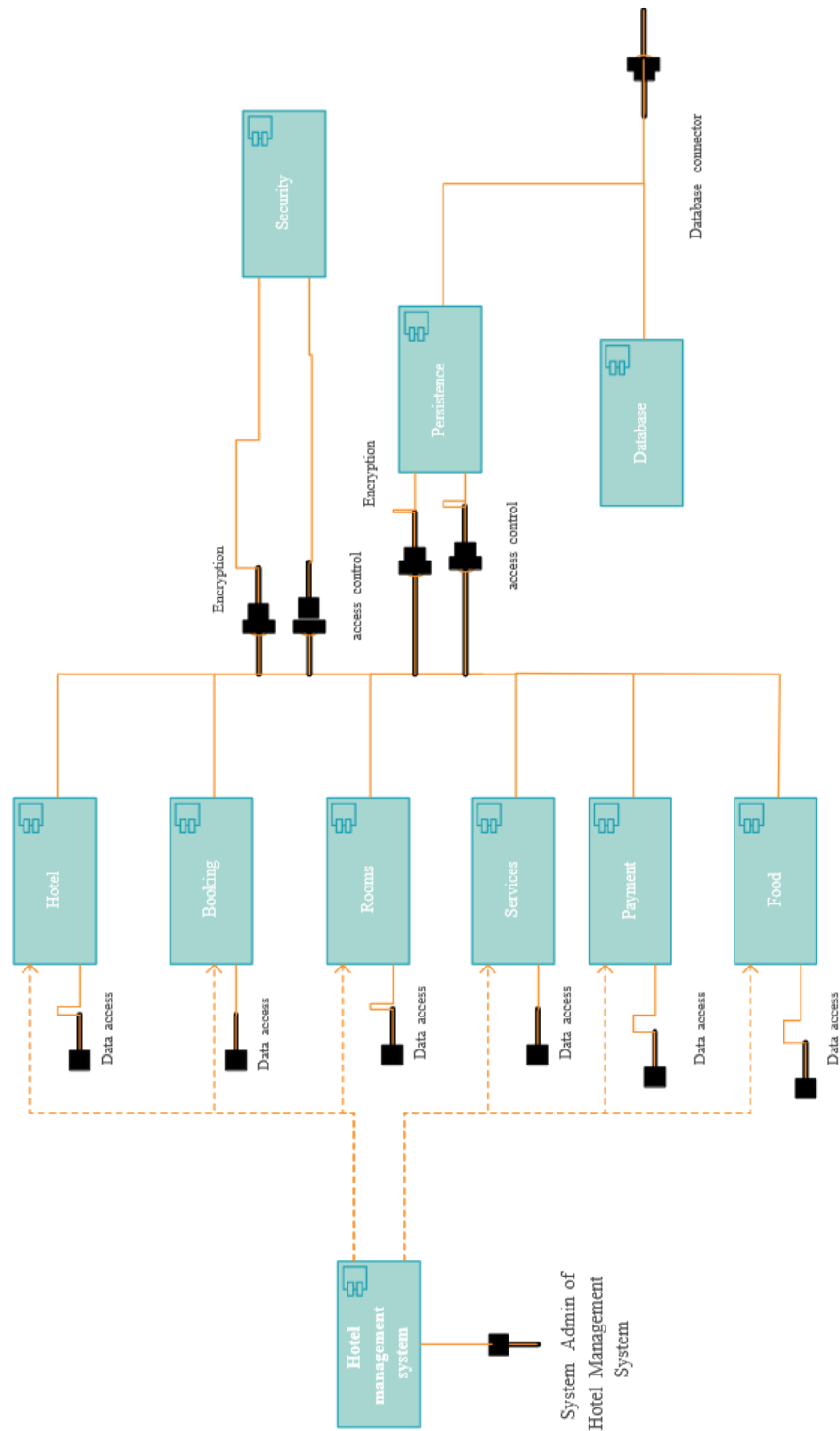
**Quiz:**

- 1) Component diagram and Deployment diagram.
- 2) Draw component diagram to show the run time dependency between a java class file , the java.exe runtime program and the java classes in .zip file.

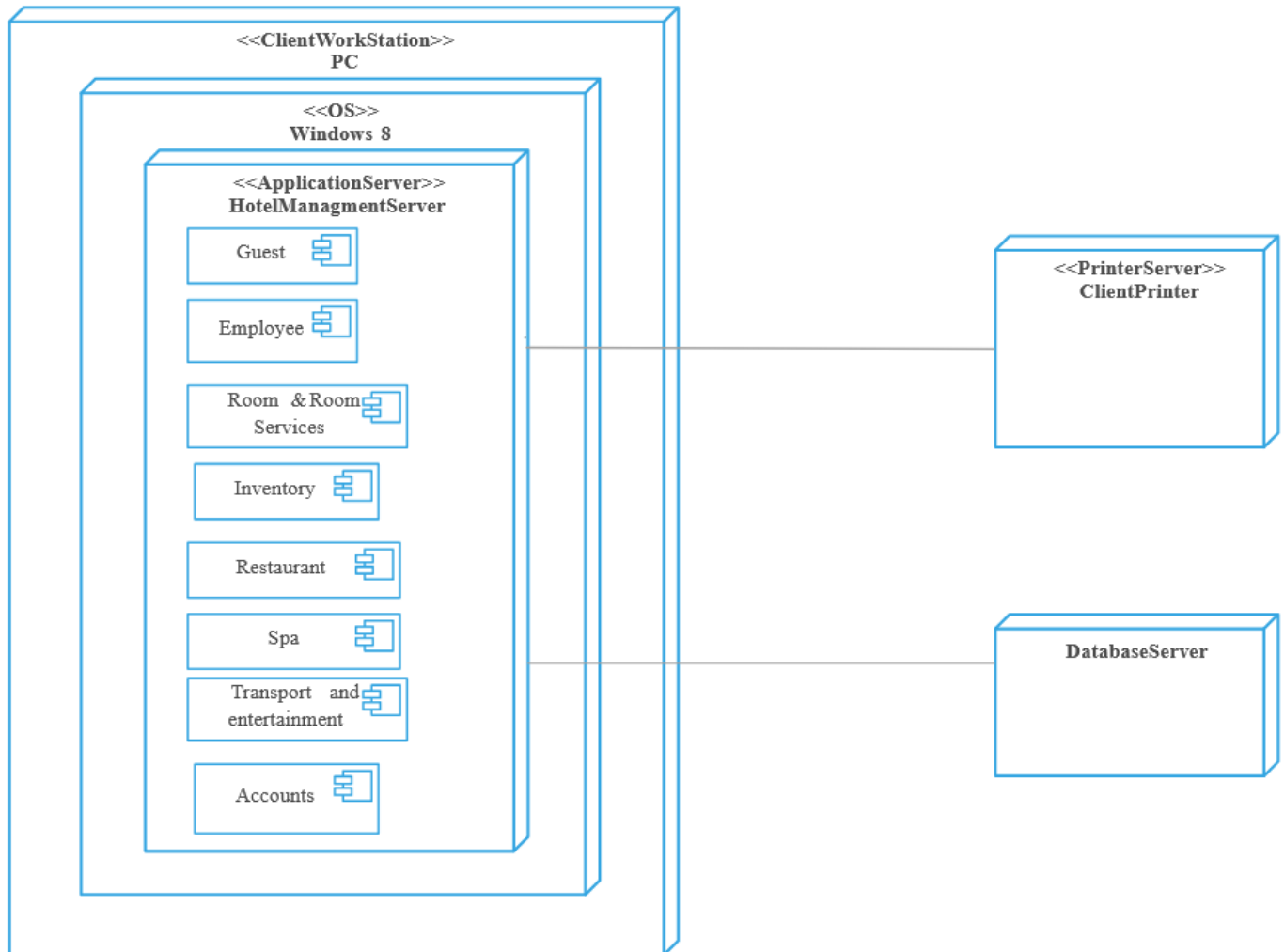
**Suggested Reference:**

1. Introduction to the Diagrams of UML 2.X

## Component Diagram of Hotel Management System:



## Deployment diagram for Hotel Management System:



**Rubric wise marks obtained:**

<b>Rubrics</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>Total</b>
<b>Marks</b>	Complete implementation as asked	Complete implementation as asked  Problem analysis	Complete implementation as asked  Problem analysis  Development of the Solution	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding  Correct answer to all questions	

**Signature of Faculty:**

## Practical – 9

**AIM:** Prepare the quality management plan.

- **Objectives:** To explore the contents which is required to create quality management plan
- **Theory:**

### Quality Management Plan Content

- Overview of Quality Management Plan
- Purpose
- Overview of Plan, Do, Check, Act
- Customer Quality Objectives
- Identify Customer Quality Objectives
- Identify Quality Threshold for each Quality Objective
- Quality Control Plans
- Address each major deliverable
- Identify Independent Technical Review Team(s)
- Quality Assurance
- Organizational Quality System Requirements (Organizational Quality Management Plan)
- Project-specific requirements
- Other Project Specific Information as required

## Sample Quality Management Plan

### Quality Management Relationships

	Quality Planning	Quality Control (QC)	Quality Assurance (QA)	Quality Improvement
	Plan	Do	Check	Act
<b>What Is Done</b>	Determine what will be quality on the project and how quality will be measured	Monitor specific project products to determine if they meet performance measurement thresholds defined in the quality management plan	Determine if measurement of quality is appropriate by evaluating overall performance on a regular basis to insure the project will satisfy customer quality expectations	Increase the effectiveness and efficiency of the project when corrective actions such as Change Requests are identified. Changes to the Quality Management Plan and the PMP may be required.
<b>When It Is Done</b>	Project Planning Phase  <b>Processes</b> <ul style="list-style-type: none"> <li>• PMP Development</li> <li>• Project Scope &amp; Customer Requirements Definition</li> <li>• Team Establishment</li> <li>• Activity/Schedule Development</li> <li>• Resource Estimate Development</li> <li>• Project Delivery Acquisition Strategy</li> </ul>	Project Execution, & Control Phase  <b>Processes</b> <ul style="list-style-type: none"> <li>• Project Execution &amp; Control</li> <li>• Lessons Learned</li> </ul>	Project Execution, & Control Phase  <b>Processes</b> <ul style="list-style-type: none"> <li>• Project Execution &amp; Control</li> </ul>	Project Execution, & Control Phase and Project Planning Phase  <b>Processes</b> <ul style="list-style-type: none"> <li>• Change Management</li> <li>• PMP Development</li> </ul>

### Plan

- Identify the customers Quality Objectives. Help customers express quality expectations in objective, quantitative terms.
- Identify professional standards including legal, environmental, economic, code, life safety and health.
- Balance needs and expectations of customers and stakeholders with cost, schedule, and professional standards. Evaluate the costs and benefits of selected quality objectives and the processes to be used to achieve objectives

- Develop an effective plan and processes, including quality assurance and quality control procedures, to achieve objectives. Consider risk/hazard factors and complexity of the project and adapt processes to provide the requisite level of quality. Document in the risk management plan any project variations from the local QMP requirements.
- Develop performance measure thresholds to ensure agreement on the definition of success relative to Quality Objectives.
- Ensure customer endorsement of all quality objectives included in the Quality Management Plan.

#### **Do**

- Do the work according to the approved PMP and standard operating procedures.
- Project execution is a dynamic process. The PDT must communicate, meet on a regular basis, and adapt to changing conditions. The Quality Management Plan and PMP may require modification to ensure that project objectives are met.

#### **Check**

- Perform independent technical review, management oversight, and verification to ensure that quality objectives are met consistent with District Quality Management Plans.
- Check performance against the PMP and Customer Quality Objectives performance measure thresholds to verify that performance will accomplish Quality Objectives and to verify sufficiency of the plan. Share findings with all project stakeholders to facilitate continuous improvement.

#### **Act**

- If performance measures thresholds are exceeded, take specific corrective actions to fix the systemic cause of any non-conformance, deficiency, or other unwanted effect.
- Document quality improvements that could include appropriate revisions to the quality management plan, alteration of quality assurance and control procedures, and adjustments to resource allocations.

#### **Quiz:**

1. Compare Cohesion and coupling with suitable example.
2. Different between software quality assurance vs software quality control.

#### **Suggested Reference:**

- 1) Rajib Mall, Fundamentals of software Engineering, Prentice Hall of India.
- 2) Pankaj Jalote, Software Engineering – A Precise Approach Wiley



## **Overview of Quality Management Plan:**

- This Quality Management Plan outlines the approach for ensuring the quality of the Hotel Management System project. It defines the processes, roles, and activities related to quality management.

## **Purpose:**

- The primary purpose of this Quality Management Plan is to ensure that the Hotel Management System is developed and delivered to meet or exceed customer expectations, adhere to industry standards, and maintain a high level of quality throughout its lifecycle.

## **Overview of Plan, Do, Check, Act:**

- The Plan-Do-Check-Act (PDCA) cycle will be integral to our quality management approach. We will continuously plan, execute, monitor, and make necessary improvements to maintain and enhance the quality of the Hotel Management System.

## **Customer Quality Objectives:**

### **1. Reservation Process:**

- Ensure a seamless and efficient reservation process for guests.
- Minimize the time it takes for customers to make reservations.

### **2. System Availability:**

- Maintain 24/7 system availability to support booking and check-in/check-out.
- Ensure minimal downtime to prevent disruptions to guest services.

### **3. Data Accuracy:**

- Guarantee data accuracy for room availability and pricing information.
- Minimize errors or inconsistencies in data that could impact reservations.

### **4. Data Security and Compliance:**

- Uphold robust data security measures to protect guest information.

### **5. User Interface:**

- Provide a user-friendly and intuitive user interface.
- Make it easy for guests and staff to interact with the system.

### **6. Payment Processing:**

- Implement reliable and prompt payment processing.
- Ensure secure and efficient payment transactions.

### **7. Customer Support:**

- Offer excellent customer support and issue resolution.

- Address customer inquiries and problems promptly and effectively.

## **Identify Quality Threshold for each Quality Objective:**

### **1. Efficient booking process:**

- Average booking time should not exceed 3 minutes. Additionally, the system should handle a peak load of 1,000 bookings per hour without a significant increase in booking time.

### **2. Security:**

- No critical security vulnerabilities should exist (as per regular scans). Additionally, any identified medium or low-severity vulnerabilities should be patched within 48 hours of discovery to maintain a strong security posture.

### **3. System availability:**

- The system should maintain 99.9% uptime over a quarter. However, for essential services like room reservation and check-in/check-out, the uptime should be 99.95% to minimize disruption to guests and staff.

### **4. Data accuracy:**

- Less than 1% discrepancy in room availability data. Data should be synchronized across all channels (website, mobile app, reception) within a maximum delay of 30 seconds.

### **5. User-friendliness:**

- At least 90% positive feedback in user satisfaction surveys. Any negative feedback should trigger an action plan for usability improvements, and the user satisfaction score should be periodically reviewed to ensure it remains above the 90% threshold.

## **Quality Control Plans:**

### **1. Reservation Module:**

**Plan:** Plan detailed test cases, including functional, performance, security, and usability testing. Define acceptable performance thresholds under normal and peak load conditions.

**Do:** Execute the planned testing activities, ensuring that all functionalities are thoroughly tested. Monitor system performance during peak load to validate its behavior.

**Check:** Review test results to identify and address bugs or issues. Analyze system performance data to ensure it meets the defined response time and load requirements.  
**Act:** Implement necessary fixes or optimizations based on the testing results. Adjust testing procedures and performance thresholds for future iterations as needed.

### **2. Guest Information Database:**

**Plan:** Define data validation and security protocols. Plan for regular security audits and

compliance checks.

Do: Implement data validation and security measures. Conduct security audits and data validation processes.

Check: Review audit results and data accuracy assessments. Verify compliance with data protection regulations.

Act: Address vulnerabilities identified in security audits. Continuously improve data validation processes based on findings.

### **3. User Interfaces:**

Plan: Plan for usability testing and user satisfaction surveys. Define accessibility standards and testing procedures.

Do: Execute usability testing and collect user feedback. Ensure that the interfaces meet accessibility standards.

Check: Analyze usability test results and user feedback. Review interface accessibility compliance.

Act: Address usability issues identified in testing. Implement improvements to enhance accessibility and user satisfaction.

### **4. System Availability:**

Plan: Plan for continuous monitoring of system uptime and availability. Develop an incident management plan for addressing unplanned downtime.

Do: Implement real-time monitoring and incident management procedures.

Check: Monitor system uptime and analyze incident reports.

Act: Investigate and document the root causes of any unplanned downtime. Implement preventative measures based on incident analysis.

### **5. Compliance Checks:**

Plan: Plan for regular compliance audits and documentation review. Develop compliance training for staff members.

Do: Conduct regular compliance audits, document reviews, and training.

Check: Review audit results and compliance documentation. Assess staff members' compliance knowledge.

Act: Address any non-compliance issues identified during audits. Provide additional training or resources as needed.

### **Identify Independent Technical Review Team(s):**

- Identify any independent technical review teams responsible for assessing and ensuring quality. A third-party security audit firm will conduct independent security reviews periodically to identify vulnerabilities.

### **Quality Assurance:**

- Describe the quality assurance processes and activities that will be implemented. Quality assurance includes adherence to coding standards, documentation, and testing protocols defined in the Organizational Quality Management Plan.

### **Project-Specific Requirements:**

1. **Integration with Existing Systems:** If the hotel already uses certain software systems (e.g., accounting, inventory management, or point of sale systems), the Hotel Management System may need to integrate seamlessly with these systems. This ensures that data flows smoothly between different parts of the hotel's operations.
2. **Multi-Language and Multi-Currency Support:** Many hotels serve an international clientele, and it's crucial for the system to support multiple languages and currencies. This requirement enables the hotel to cater to a diverse customer base.
3. **Customization of Room Types and Pricing:** Different hotels have unique room types and pricing strategies. The system should allow for flexible configuration of room categories, rates, and packages to accommodate the hotel's specific offerings.
4. **Third-Party API Integration:** The Hotel Management System may need to integrate with third-party services like online booking platforms, payment gateways, or property management systems. This integration ensures that the hotel can effectively manage reservations and payments.
5. **Reporting and Analytics:** Hotels often rely on data analytics to make informed decisions. The system should offer robust reporting capabilities to provide insights into occupancy rates, revenue, and guest preferences.
6. **Mobile-Friendly Interface:** In an era of smartphones and tablets, having a mobile-friendly interface is critical. Guests and staff should be able to access and manage reservations and services through mobile devices.
7. **Scalability:** The system should be scalable to accommodate the growth of the hotel business. This includes the ability to add more rooms, services, or properties without significant disruption.
8. **User Training and Support:** Comprehensive training and ongoing support for hotel staff is essential to ensure that they can effectively use the system and address any issues that may arise during day-to-day operations.

## **Other Project Specific Information are required:**

1. **Data Migration:** If the hotel is transitioning from an existing system to the new Hotel Management System, specific requirements for data migration, such as transferring guest profiles, reservations, or historical data, should be outlined.
2. **Custom Reports:** If the hotel requires specific custom reports beyond standard analytics, these should be detailed in this section.
3. **User Roles and Permissions:** Specify any unique user roles and permissions needed for the project. For example, who can access financial data, who can manage room assignments, or who can modify rates.
4. **Local Partnerships:** If the hotel has partnerships with local businesses or service providers (e.g., nearby restaurants, transportation services), outline any integration or collaboration requirements.
5. **Guest Feedback Mechanisms:** Define how the system will collect and manage guest feedback, including surveys, ratings, or comments, and how this feedback will be used to enhance the guest experience.
6. **Emergency Protocols:** Specify any specific emergency response or communication procedures that should be integrated into the system, such as alerting staff in case of fire alarms or medical emergencies.
7. **Marketing and Promotions:** Detail any specific marketing or promotional features that the system should support, such as discounts, special offers, or loyalty program promotions.

**Rubric wise marks obtained:**

<b>Rubrics</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>Total</b>	
<b>Marks</b>	Complete implementation as asked	Complete implementation as asked  Problem analysis	Complete implementation as asked  Problem analysis  Development of the Solution	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding  Correct answer to all questions		

**Signature of Faculty:**

## Practical – 10

**AIM:** To perform various testing using the testing tool unit testing, integration testing design test cases..

- **Objectives:** to explore and learn about different testing techniques and use them.
- **Theory:**
  - Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.
  - **Software Validation**
    - Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.
    - Validation ensures the product under development is as per the user requirements.
    - Validation answers the question – "Are we developing the product which attempts all that user needs from this software ?".
    - Validation emphasizes on user requirements.
  - **Software Verification**
    - Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.
    - Verification ensures the product being developed is according to design specifications.
    - Verification answers the question– "Are we developing this product by firmly following all design specifications ?"
    - Verifications concentrates on the design and system specifications.
  - Target of the test are -
    - **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
    - **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.

- **Failure** - failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

### ○ **Testing Levels**

- Testing itself may be defined at various levels of SDLC. The testing process runs parallel to software development. Before jumping on the next stage, a stage is tested, validated and verified.
- Testing separately is done just to make sure that there are no hidden bugs or issues left in the software. Software is tested on various levels -

### ○ *Unit Testing*

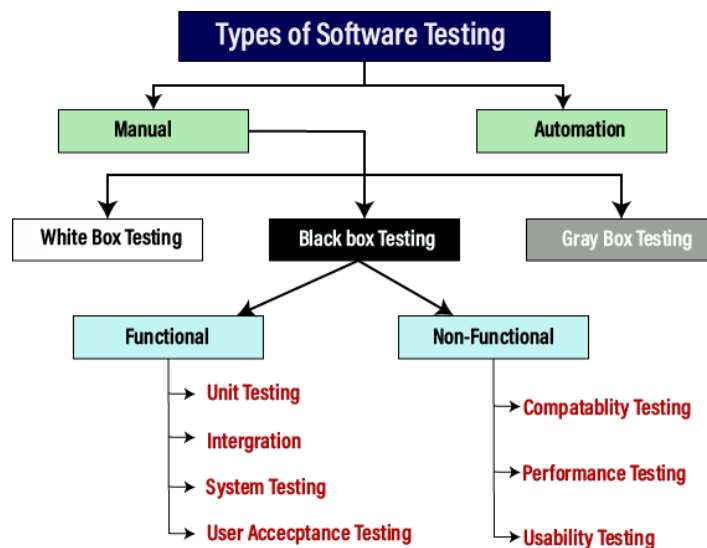
While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

### ○ *Integration Testing*

Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updation etc.

### ○ *System Testing*

The software is compiled as product and then it is tested as a whole.





- **Background / Preparation:**

- **Test management tool**

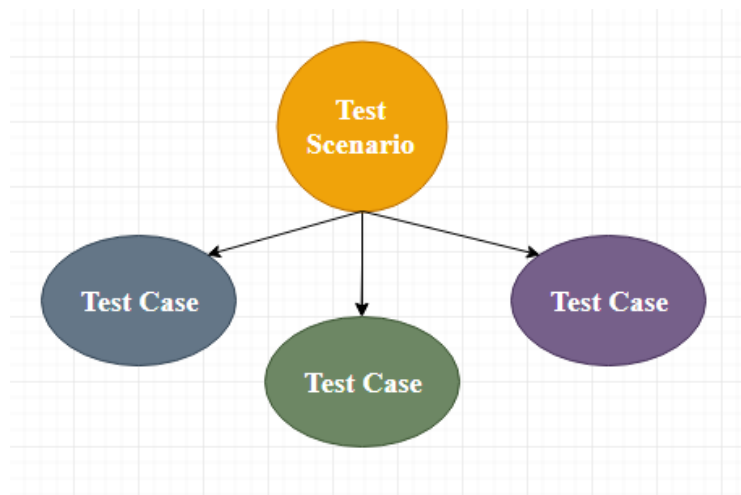
- Test management tools are used to keep track of all the testing activity, fast data analysis, manage manual and automation test cases, various environments, and plan and maintain manual testing as well.
    - Test management tools are used to keep track of all the testing activity, fast data analysis, manage manual and automation test cases, various environments, and plan and maintain manual testing as well.
    - The test management tool is connected with the automation software. These types of tools had various strategies for testing and multiple sets of features. Some of the test management tools had capabilities to design the test case with the help of requirements.
    - It is best for test managing, scheduling, defect logging, tracking, and analysis.
    - Some of the most commonly used test management tools are as follows:
      - Quality center
      - RTH
      - Testpad
      - Test Monitor
      - PractiTest

- **Tools / Material Needed:**

- **Hardware:**
  - **Software:**

- **Procedure / Steps:**

- RTH is another web-based open-source tool, and it stands for **the Requirement and testing hub**. It is used to manage the requirements, test results, and also have bug tracking facilities. This tool follows a structured approach to extend the visibility of the testing process with the help of a common repository for test cases, test result, requirements, and test plans.
  - Test Cases:
    - The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.



- **Test case template**

- The primary purpose of writing a test case is to achieve the efficiency of the application.

**Header**

Test Case Name/ID :- **Release - Version - Application Name - Module**

Test Case Type:- 

F.T.C	I.T.C	S.T.C
-------	-------	-------

Requirement Number:-

Module:-

Serverity:- **Critical/Major/Minor**

Status:-

Release:-

Version:-

Pre-condition:-

Test Data:-

Summary:-

**Body**

Step No.	Descri- ption	Inputs	Expected Result	Actual Reasult	Status	Comments
...	...	...	...	...	...	...
...	...	...	...	...	...	...

**Footer**

Author:-

Reviewd By:-

Date:-

Approved By:-

As we know, the **actual result** is written after the test case execution, and most of the time, it would be same as the **expected result**. But if the test step will fail, it will be different. So, the actual result field can be skipped, and in **the Comments** section, we can write about the bugs.

And also, the **Input field** can be removed, and this information can be added to the **Description field**.

The above template we discuss above is not the standard one because it can be different for each company and also with each application, which is based on the test engineer and the test lead. But, for testing one application, all the test engineers should follow a usual template, which is formulated.

The test case should be written in simple language so that a new test engineer can also understand and execute the same.

In the above sample template, the header contains the following:

**Step number**

It is also essential because if step number 20 is failing, we can document the bug report and hence prioritize working and also decide if it's a critical bug.

**Test case type**

It can be functional, integration or system test cases or positive or negative or positive and negative test cases.

**Release**

One release can contain many versions of the release.

**Pre-condition**

These are the necessary conditions that need to be satisfied by every test engineer before starting the test execution process. Or it is the data configuration or the data setup that needs to be created for the testing.

**For example:** In an application, we are writing test cases to add users, edit users, and delete users. The pre-condition will be seen if user A is added before editing it and removing it.

**Test data**

These are the values or the input we need to create as per the pre-condition.

**For example,** Username, Password, and account number of the users.

The test lead may be given the test data like username or password to test the application, or the test engineer may themselves generate the username and password.

- *Test Cases – Login Page*

- Following is the possible list of functional and non-functional test cases for a login page:

○ *Functional Test Cases:*

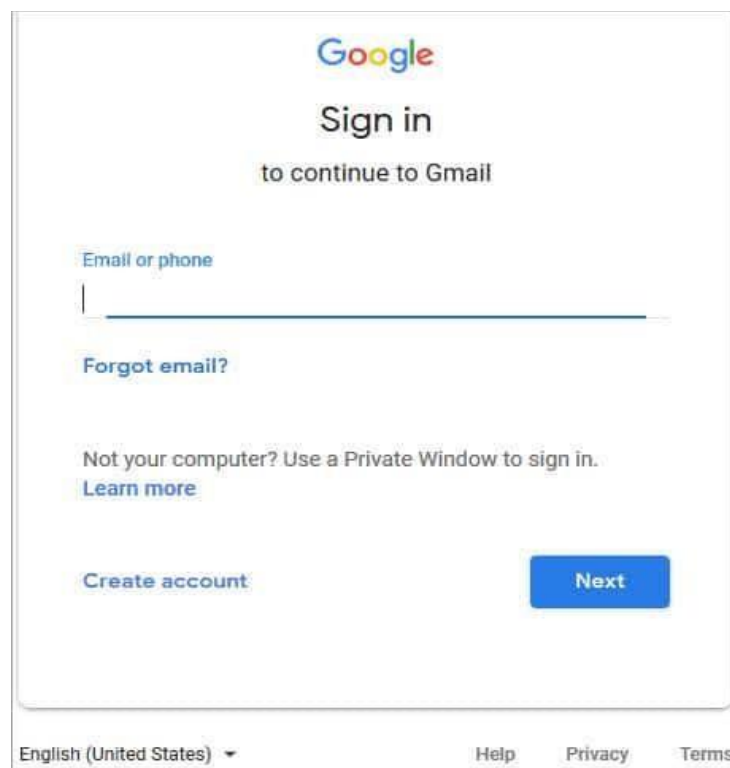
Sr. No.	Functional Test Cases	Type- Negative/ Positive Test Case
1	Verify if a user will be able to login with a valid username and valid password.	Positive
2	Verify if a user cannot login with a valid username and an invalid password.	Negative
3	Verify the login page for both, when the field is blank and Submit button is clicked.	Negative
4	Verify the 'Forgot Password' functionality.	Positive
5	Verify the messages for invalid login.	Positive
6	Verify the 'Remember Me' functionality.	Positive
7	Verify if the data in password field is either visible as asterisk or bullet signs.	Positive
8	Verify if a user is able to login with a new password only after he/she has changed the password.	Positive
9	Verify if the login page allows to log in simultaneously with different credentials in a different browser.	Positive
10	Verify if the 'Enter' key of the keyboard is working correctly on the login page.	Positive
	Other Test Cases	
11	Verify the time taken to log in with a valid username and password.	Performance & Positive Testing
12	Verify if the font, text color, and color coding of the Login page is as per the standard.	UI Testing & Positive Testing
13	Verify if there is a 'Cancel' button available to erase the entered text.	Usability Testing
14	Verify the login page and all its controls in different browsers	Browser Compatibility & Positive Testing

○ *Non-functional Security Test Cases:*

Sr. No.	Security test cases	Type- Negative/ Positive Test Case
1	Verify if a user cannot enter the characters more than the specified range in each field (Username and Password).	Negative
2	Verify if a user cannot enter the characters more than the specified range in each field (Username and Password).	Positive
3	Verify the login page by pressing 'Back button' of the browser. It should not allow you to enter into the system once you log out.	Negative
4	Verify the timeout functionality of the login session.	Positive
5	Verify if a user should not be allowed to log in with different credentials from the same browser at the same time.	Negative
6	Verify if a user should be able to login with the same credentials in different browsers at the same time.	Positive
7	Verify the Login page against SQL injection attack.	Negative
8	Verify the implementation of SSL certificate.	Positive

Showing 1 to 8 of 8 entries

We can take an **Example** of Gmail Login page. Here is the image of it.



## Test Cases for Gmail Login page

Google

### Create your Google Account

to continue to Gmail

First name  Last name

Username  @gmail.com

You can use letters, numbers & periods

Password  Confirm password

Use 8 or more characters with a mix of letters, numbers & symbols

[Sign in instead](#) [Next](#)

English (United States) ▼

Help Privacy Terms

One account. All of Google working for you.

Sr. No.	Test Scenarios
1	Enter the valid email address & click next. Verify if the user gets an option to enter the password.
2	Don't enter an email address or phone number & just click the Next button. Verify if the user will get the correct message or if the blank field will get highlighted.
3	Enter the invalid email address & click the Next button. Verify if the user will get the correct message.
4	Enter an invalid phone number & click the Next button. Verify if the user will get the correct message.
5	Verify if a user can log in with a valid email address and password.
6	Verify if a user can log in with a valid phone number and password.
7	Verify if a user cannot log in with a valid phone number and an invalid password.
8	Verify if a user cannot log in with a valid email address and a wrong password.
9	Verify the 'Forgot email' functionality.
10	Verify the 'Forgot password' functionality.

### *Test Scenarios for the Sign-up page*

- 1) Verify the messages for each mandatory field.
- 2) Verify if the user cannot proceed without filling all the mandatory fields.
- 3) Verify the age of the user when the DOB is selected.
- 4) Verify if the numbers and special characters are not allowed in the First and Last name.
- 5) Verify if a user can sign-up successfully with all the mandatory details.
- 6) Verify if a user can log in with the valid details.
- 7) Verify if the Password and Confirm Password fields are accepting similar strings only.
- 8) Verify if the Password field will prompt you for the weak passwords.
- 9) Verify if duplicate email address will not get assigned.
- 10) Verify that hints are provided for each field on the form, for the ease of use.

### **Quiz:**

- 1 What elements of the WebApp can be “unit tested”? What types of tests must be conducted only after the WebApp elements are integrated?
- 2 What is white box testing? What are the different coverage based testing strategies.
- 3 What is black box testing? What are the different black box testing techniques?

### **Suggested Reference:**

- 1 Software Testing: A Craftsman's Approach, by Paul C. Jorgensen, Third Edition
- 2 Software Engineering by Rajib Mall, PHI 2014

**Test Case for User Management:**

Test Case	Functional Test cases	Expected Result
1	User Registration with valid information	P
2	User Registration with an invalid email format	N
3	User Registration with an invalid password format	N
4	Registration with an existing email	N
5	User Login with correct credentials	P
6	User Login with incorrect email and password	N
7	"Remember Me" functionality test	P
8	Resetting a forgotten password with a valid email	P
9	Resetting a forgotten password with an invalid email	N
10	Changing a user's password	P

**Test Case for Room Booking:**

Test Case	Functional Test cases	Expected Result
1	Search for available rooms for a date range	P
2	Booking a room for a specific date range	P
3	Booking a room that is already booked	N
4	Attempt to book a room for a past date	N
5	Booking multiple rooms for the same date range	P
6	Canceling a room reservation	P

**Test Case for Check-In and Check-Out:**

Test Case	Functional Test cases	Expected Result
1	Guest check-in to a booked room	P
2	Attempt to check into an unbooked room	N
3	Guest check-out of a room	P
4	Attempt to check out of an already checked-out room	N
5	Attempt to check into a room that is already checked-in	N
6	Attempt to check out a room that is already checked-out	N



**Test Case for Payment Processing:**

Test Case	Functional Test cases	Expected Result
1	Calculation of the correct total cost for a room	P
2	Payment processing with different methods	P
3	Payment processing failure (e.g., insufficient funds)	N
4	Refund processing for canceled room reservations	P

**Test Case for Admin Functionality:**

Test Case	Description	Expected Result
1	Admin user adds new rooms to the system	P
2	Admin user edits room details	P
3	Admin user removes rooms from the system	P
4	Attempting to edit room details with invalid data	N
5	Deleting a room and verifying its removal from the system	P

**Test Case for Security and Performance:**

Test Case	Description	Expected Result
1	Unauthorized access to admin functionality	N
2	Security testing	N
3	High load simulation and system response time	P
4	Load testing with a higher number of concurrent users	P
5	Testing the system's response to a sudden increase in traffic	P
6	Testing data backup and recovery procedures	P

**Rubric wise marks obtained:**

<b>Rubrics</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>Total</b>
<b>Marks</b>	Complete implementation as asked	Complete implementation as asked  Problem analysis	Complete implementation as asked  Problem analysis  Development of the Solution	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding  Correct answer to all questions	

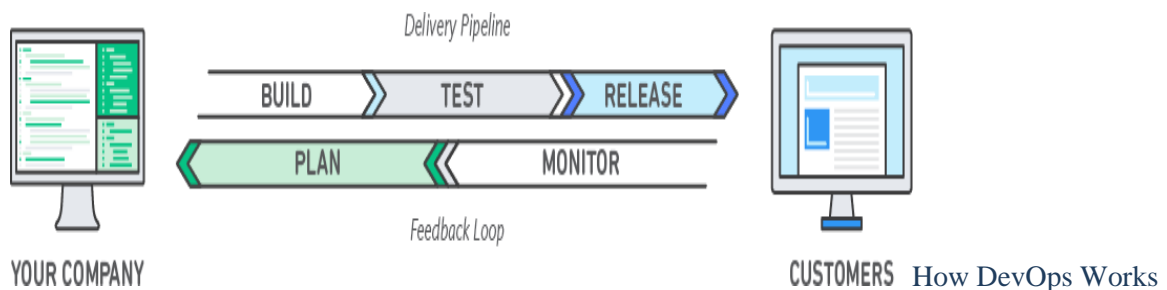
**Signature of Faculty:**

## Practical – 11

**AIM:** Case Study: Study of Open-source tools in DevOps for Infrastructure Automation, Configuration Management, Deployment Automation, Performance Management, Log Management. Monitoring

- **Objectives:** to explore various case studies where DevOps is applied.
- **Theory:**

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.



Under a DevOps model, development and operations teams are no longer “siloed.” Sometimes, these two teams are merged into a single team where the engineers work across the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

In some DevOps models, quality assurance and security teams may also become more tightly integrated with development and operations and throughout the application lifecycle. When security is the focus of everyone on a DevOps team, this is sometimes referred to as DevSecOps.

These teams use practices to automate processes that historically have been manual and slow. They use a technology stack and tooling which help them operate and evolve applications quickly and reliably. These tools also help engineers independently accomplish tasks (for example, deploying code or provisioning infrastructure) that normally would have required help from other teams, and this further increases a team's velocity.

Why DevOps Matters

Software and the Internet have transformed the world and its industries, from shopping to entertainment to banking. Software no longer merely supports a business; rather it becomes an integral component of every part of a business. Companies interact with their customers through software delivered as online services or applications and on all sorts of devices. They also use software to increase operational efficiencies by transforming every part of the value chain, such as logistics, communications, and operations. In a similar way that physical goods companies transformed how they design, build, and deliver products using industrial automation throughout the 20th century, companies in today's world must transform how they build and deliver software.

## DevOps Practices

### *Continuous Integration*

Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates. [Continuous Delivery](#)

Continuous delivery is a software development practice where code changes are automatically built, tested, and prepared for a release to production. It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When continuous delivery is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

### *Microservices*

The microservices architecture is a design approach to build a single application as a set of small services. Each service runs in its own process and communicates with other services through a well-defined interface using a lightweight mechanism, typically an HTTP-based application programming interface (API). Microservices are built around business capabilities; each service is scoped to a single purpose. You can use different frameworks or programming languages to write microservices and deploy them independently, as a single service, or as a group of services.

### *Infrastructure as Code*

Infrastructure as code is a practice in which infrastructure is provisioned and managed using code and software development techniques, such as version control and continuous integration. The cloud's API-driven model enables developers and system administrators to interact with infrastructure programmatically, and at scale, instead of needing to manually set up and configure resources. Thus, engineers can interface with infrastructure using code-based tools and treat infrastructure in a manner similar to how they treat application code. Because they are defined by code, infrastructure and servers can quickly be deployed using standardized patterns, updated with the latest patches and versions, or duplicated in repeatable ways.

### ***Configuration Management***

Developers and system administrators use code to automate operating system and host configuration, operational tasks, and more. The use of code makes configuration changes repeatable and standardized. It frees developers and systems administrators from manually configuring operating systems, system applications, or server software.

### ***Policy as Code***

With infrastructure and its configuration codified with the cloud, organizations can monitor and enforce compliance dynamically and at scale. Infrastructure that is described by code can thus be tracked, validated, and reconfigured in an automated way. This makes it easier for organizations to govern changes over resources and ensure that security measures are properly enforced in a distributed manner (e.g. information security or compliance with PCI-DSS or HIPAA). This allows teams within an organization to move at higher velocity since non-compliant resources can be automatically flagged for further investigation or even automatically brought back into compliance.

### ***Monitoring and Logging***

Organizations monitor metrics and logs to see how application and infrastructure performance impacts the experience of their product's end user. By capturing, categorizing, and then analyzing data and logs generated by applications and infrastructure, organizations understand how changes or updates impact users, shedding insights into the root causes of problems or unexpected changes. Active monitoring becomes increasingly important as services must be available 24/7 and as application and infrastructure update frequency increases. Creating alerts or performing real-time analysis of this data also helps organizations more proactively monitor their services.























### ***Communication and Collaboration***

Increased communication and collaboration in an organization is one of the key cultural aspects of DevOps. The use of DevOps tooling and automation of the software delivery process establishes collaboration by physically bringing together the workflows and responsibilities of development and operations. Building on top of that, these teams set strong cultural norms around information sharing and facilitating communication through the use of chat applications, issue or project tracking systems, and wikis. This helps speed up communication across developers, operations, and even other teams like marketing or sales, allowing all parts of the organization to align more closely on goals and projects.

### ***DevOps Tools***

The DevOps model relies on effective tooling to help teams rapidly and reliably deploy and innovate for their customers. These tools automate manual tasks, help teams manage complex environments at scale, and keep engineers in control of the high velocity that is enabled by DevOps. AWS provides services that are designed for DevOps and that are built first for use with the AWS cloud. These services help you use the DevOps practices described above.

#### **❖ Open-source tools in DevOps:**

Configuration Management	Continuous integration	Microservices	Collaboration	Monitoring	Development
 CHEF™	 Jenkins	 docker	 JIRA Software	 MONIT	 Visual Studio
 SALTSTACK	 TeamCity	 MESOS	 slack	 Ganglia	 APACHE ACTIVE MQ
 puppet	 CodeShip	 TRITON Compute	 HipChat	 SNORT	 Vagrant
 ANSIBLE	 circleci	 ElasticBox	 Trello Organize anything, together.	 cacti	 Microsoft Azure

## ❖ Open-source tools in DevOps for Infrastructure Automation:

Infrastructure automation is a critical component of modern DevOps practices. Open-source tools play a significant role in this space, offering flexibility and cost-effectiveness. Here are some open-source tools specifically designed for infrastructure automation:

### 1. Terraform:

**Description:** Terraform is an Infrastructure as Code (IaC) tool developed by HashiCorp. It allows you to define, provision, and manage infrastructure using a declarative configuration language.



### Key Features:

- Multi-cloud support (AWS, Azure, GCP, etc.).
- State management for tracking infrastructure changes.
- Modular and reusable code with Terraform

modules.

- Ecosystem of providers for various services and resources.

## 2. Ansible:

**Description:** Ansible is a popular configuration management and automation tool that uses YAML to define infrastructure as code. It can be used for provisioning, configuration management, and application deployment.



### Key Features:

- Agentless, making it easy to set up and use.
- Supports a wide range of modules for various tasks.
- Playbooks for defining automation tasks.
- Strong community support and a vast collection of roles.

## 3. Packer:

**Description:** Packer is a tool for creating machine images from a single source configuration. It allows you to build machine images for different platforms in an automated and repeatable manner.



### Key Features:

- Supports multiple builders (AWS, VirtualBox, Docker, etc.).
- Infrastructure as code for creating custom machine images.

- Integration with other provisioning tools like Ansible and Chef.

#### 4. Vagrant:

**Description:** Vagrant is a tool for creating and managing virtualized development environments. It helps developers set up and share consistent development environments easily.



##### Key Features:

- Uses a simple configuration file (Vagrantfile).
- Supports various providers (VirtualBox, VMware, etc.).
- Enables team collaboration with portable development environments.

#### 5. SaltStack:

**Description:** SaltStack is a configuration management and automation platform designed for speed and scalability. It can be used for remote execution, configuration management, and orchestration.



##### Key Features:

- Highly scalable and fast.
- Supports event-driven automation with the Reactor.
- Powerful remote execution capabilities.

#### 6. CloudInit:

**Description:** CloudInit is a standard for customizing cloud instances during their initial boot process. While not a standalone tool, it's a critical component for automating cloud infrastructure.





**Key Features:**

- Configures instance settings during the first boot.
- Supported by major cloud providers, including AWS, Azure, and Google Cloud.

**7. Rundeck:**

**Description:** Rundeck is an open-source automation platform that provides job scheduling and runbook automation. It's useful for automating operational tasks and workflows.



**Key Features:**

- Web-based interface for creating and managing automation workflows.
- Access control and auditing features.
- Integration with various tools and scripts.

**8. OpenStack:**

**Description:** OpenStack is an open-source cloud computing platform that can be used to build private and public clouds. It provides infrastructure automation capabilities, including compute, storage, and networking services.



**Key Features:**

- Modular architecture with services like Nova (compute) and Neutron (networking).
- Scalable and customizable cloud infrastructure.
- These open-source tools can be used individually or in combination to automate the provisioning, configuration, and management of infrastructure in a flexible and cost-effective manner. Depending on your specific needs and infrastructure, you can choose the tool or combination of tools that best suits your requirements.

**❖ Open-source tools in DevOps for Configuration Management:**

**Configuration management is a critical aspect of DevOps, ensuring that infrastructure and application configurations remain consistent, are version-controlled, and can be easily replicated across different environments. Here are some open-source tools for configuration management:**

**1. Ansible:**

**Description:** Ansible is an open-source configuration management and automation tool that uses simple, human-readable YAML scripts (playbooks) to define infrastructure and application configurations. It is agentless, which means it doesn't require agents to be installed on target systems.

**Key Features:**

- Declarative syntax for defining configurations.
- Supports various modules for tasks like package management, file manipulation, and more.
- Ansible Galaxy provides a library of pre-built roles.
- Strong community support and integrations with cloud platforms.

**2. Chef:**

**Description:** Chef is a powerful open-source configuration management tool that uses Ruby-based scripts (cookbooks) to automate infrastructure and application provisioning. It follows an "infrastructure as code" approach.



**Key Features:**

- Infrastructure automation with code-based cookbooks.
- Supports test-driven development (TDD) for infrastructure.
- Chef Supermarket offers a repository of pre-built cookbooks.
- Integrates with cloud providers and platforms.

**3. Puppet:**

**Description:** Puppet is an open-source configuration management tool that uses its own declarative language to define configurations. It is designed for managing and automating infrastructure at scale.



**Key Features:**

- Puppet's DSL for configuration definitions.
- Puppet Forge provides a repository of pre-built modules.
- Supports role-based management of configurations.
- Integrates with cloud services and container platforms.

**4. SaltStack:**

**Description:** SaltStack, or Salt, is an open-source, event-driven automation and configuration management tool. It is known for its speed and scalability, making it suitable for managing large infrastructures.



#### **Key Features:**

- Highly scalable and fast remote execution.
- Event-driven automation with the Salt Reactor.
- Supports infrastructure automation through states and pillars.
- Strong security features, including ZeroMQ encryption.

#### **5. Terraform:**

**Description:** Terraform is an open-source Infrastructure as Code (IaC) tool developed by HashiCorp. While it's primarily used for provisioning infrastructure, it also includes configuration management capabilities for defining infrastructure resources.



#### **Key Features:**

- Multi-cloud support (AWS, Azure, GCP, etc.).
- Infrastructure provisioning and management through Terraform code.
- Integration with configuration management tools like Ansible and Puppet.
- Modular and reusable code with Terraform modules.

#### **6. Rundeck:**

**Description:** Rundeck is an open-source automation platform with a focus on job scheduling and runbook automation. While not a traditional configuration management tool, it helps manage operational

tasks and automate workflows.



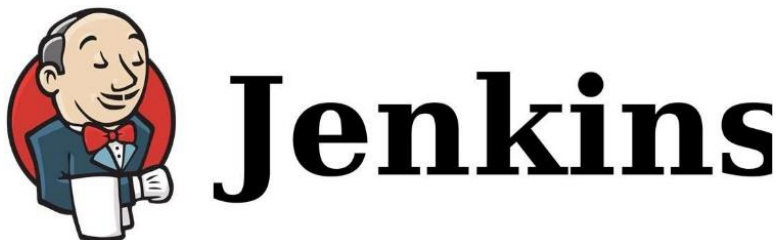
**Key Features:**

- Web-based interface for creating and managing automation workflows.
- Access control and auditing features.
- Integration with various tools and scripts.
- Supports scheduling and orchestrating complex tasks.

## ❖ Open-source tools in DevOps for Deployment Automation:

### 2. Jenkins for Continuous Integration and Continuous Deployment (CI/CD):

**Description:** Jenkins is an open-source automation server that can be used for various tasks in the CI/CD pipeline. It is highly extensible and widely used in the DevOps community.



**Key Features:**

- Automation: Jenkins enables automation of the entire software development lifecycle, from code integration and testing to deployment and monitoring.
- Pipeline as Code: It allows you to define CI/CD pipelines as code, which can be versioned, shared, and easily replicated.
- Plugins: Jenkins has a vast ecosystem of plugins, allowing you to integrate it with various tools and services.
- Scalability: Jenkins can be scaled horizontally to handle increasing workloads.

### 3. Docker for Containerization:

**Description:** Docker is an open-source platform that automates the deployment of applications inside lightweight, portable containers. Containers package applications and their dependencies in a consistent, isolated environment.



#### **Key Features:**

- Isolation: Containers ensure that applications run consistently regardless of the environment.
- Portability: Containers can be moved between environments with minimal changes.
- Version Control: Container images can be versioned and stored in container registries.
- Efficiency: Containers are lightweight and start quickly.

#### **4. Kubernetes for Container Orchestration:**

**Description:** Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.



# kubernetes

#### **Key Features:**

- Orchestration: Kubernetes schedules and manages containerized applications across a cluster of machines.
- Scalability: It can scale applications up or down automatically based on demand.
- Self-Healing: Kubernetes monitors the health of applications and restarts containers if needed.
- Load Balancing: It provides built-in load balancing for containerized services.

#### **5. Helm for Kubernetes Package Management:**

**Description:** Helm is an open-source package manager for Kubernetes. It simplifies defining, installing, and upgrading even complex Kubernetes applications through Helm charts.



#### **Key Features:**

- Charts: Helm charts are packages of pre-configured Kubernetes resources.
- Versioning: Helm charts can be versioned and shared via chart repositories.
- Templating: Helm allows parameterized templates for different environments or configurations.

### **6. Prometheus and Grafana for Monitoring:**

**Description:** Prometheus is an open-source monitoring and alerting toolkit. Grafana is an open-source platform for observability and visualization.



#### **Key Features (Prometheus):**

- Data Collection: Prometheus scrapes metrics from various targets.
- Flexible Query Language: It provides PromQL for querying and analyzing metrics.
- Alerting: Prometheus can trigger alerts based on defined rules.

#### **Key Features (Grafana):**

- Visualization: Grafana offers flexible and interactive data visualization.
- Dashboards: It supports the creation of custom dashboards for monitoring.
- Alerting: Grafana can integrate with Prometheus for alerting.

## **❖ Open-source tools in DevOps for Performance Management:**

### **1. JMeter for Load Testing:**

**Description:** Apache JMeter is an open-source tool for load testing, performance testing, and stress testing of web applications. It simulates a heavy load on a server, network, or object to test its strength or analyze overall performance under different load types.



### **Key Features:**

- HTTP/HTTPS Support: JMeter can simulate various web protocols, including HTTP and HTTPS.
- Scripting: It provides a graphical interface for building test plans and scripting tests.
- Scalability Testing: JMeter can simulate a large number of concurrent users, helping identify performance bottlenecks.

## **2. Prometheus for Monitoring:**

**Description:** Prometheus is an open-source monitoring and alerting toolkit that is designed for reliability and scalability. It collects and stores metrics from various sources, allowing real-time monitoring and alerting.



### **Key Features:**

- Data Collection: Prometheus scrapes metrics from targets such as applications, servers, and services.
- Flexible Query Language: PromQL enables querying and analyzing metrics.
- Alerting: Prometheus can trigger alerts based on defined rules.

## **3. Grafana for Visualization:**



**Description:** Grafana is an open-source platform for observability and data visualization. It can be used to create dashboards and graphs for monitoring and analyzing data.



**Key Features:**

- Visualization: Grafana offers flexible and interactive data visualization.
- Dashboards: It supports the creation of custom dashboards for monitoring and analysis.
- Alerting: Grafana can integrate with various monitoring systems, including Prometheus.

**4. Apache JMeter for Real User Monitoring:**

**Description:** In addition to load testing, Apache JMeter can be used for real user monitoring (RUM) to capture performance data from actual users and browsers.



**Key Features:**

- Browser Simulation: JMeter can simulate browser requests and interactions to collect real user data.
- Script Recording: It allows recording and playback of user interactions for performance testing and monitoring.

**❖ Open-source tools in DevOps for log management:**

**1. Elasticsearch for Log Storage:**

**Description:** Elasticsearch is an open-source, distributed search and analytics engine designed for handling

large volumes of data. It's often used for log storage and indexing.



### **Key Features:**

- Scalability: Elasticsearch can handle large volumes of data and scale horizontally.
- Full-Text Search: It offers powerful full-text search capabilities for log data.
- Real-Time Data: Elasticsearch provides real-time indexing and search for log data.

### **2. Logstash for Log Ingestion:**

Description: Logstash is an open-source server-side data processing pipeline that ingests, transforms, and enriches log data before sending it to a data store.



### **Key Features:**

- Data Transformation: Logstash allows data transformation using various filters and plugins.
- Data Enrichment: You can enrich log data with additional information using lookup tables and external sources.
- Support for Various Inputs and Outputs: Logstash supports a wide range of input and output sources, making it versatile for log ingestion.

### **3. Kibana for Log Visualization:**

**Description:** Kibana is an open-source data visualization and exploration tool that is often used with Elasticsearch for log data analysis and visualization.



# kibana

## Key Features:

- Data Visualization: Kibana offers interactive and customizable data visualization.
- Dashboards: You can create custom dashboards to display key log data and metrics.
- Elasticsearch Integration: Kibana seamlessly integrates with Elasticsearch for log data retrieval.

## 4. Filebeat for Log Shipper:

**Description:** Filebeat is an open-source lightweight log shipper that collects and forwards log data to a centralized location, often to Elasticsearch or Logstash.



# Beats

## Key Features:

- Lightweight: Filebeat has a small footprint and is resource-efficient.
- Log Data Collection: It can collect log data from various sources, including log files, system logs, and Docker containers.
- Real-Time Data Shipper: Filebeat ships log data in real-time to the central log management system.

## ❖ Open-source tools in DevOps for Monitoring:

### 1. Prometheus for Metrics Collection:

**Description:** Prometheus is an open-source monitoring and alerting toolkit. It is designed for reliability and scalability, collecting metrics from different sources and providing real-time monitoring and alerting.



### **Key Features:**

- **Data Collection:** Prometheus scrapes metrics from various targets, including applications, servers, and services.
- **Flexible Query Language:** PromQL allows for querying and analyzing metrics.
- **Alerting:** Prometheus can trigger alerts based on predefined rules.

## **2. Grafana for Data Visualization:**

**Description:** Grafana is an open-source platform for observability and data visualization. It is commonly used to create dashboards and visualizations for monitoring and data analysis.



### **Key Features:**

- **Visualization:** Grafana provides flexible and interactive data visualization.
- **Dashboards:** It supports the creation of custom dashboards for monitoring and analysis.
- **Alerting:** Grafana can integrate with various monitoring systems, including Prometheus.

## **3. ELK Stack for Log Management:**

**Description:** The ELK Stack consists of three open-source tools: Elasticsearch, Logstash, and Kibana. It is used for log management, log analysis, and log visualization.

**Key Features:**

- Elasticsearch: Indexes and stores log data for efficient searching and analysis.
- Logstash: Collects, processes, and enriches log data before sending it to Elasticsearch.
- Kibana: Provides visualization and dashboards for log analysis.

**4. Zabbix for Infrastructure Monitoring:**

**Description:** Zabbix is an open-source monitoring solution that can be used for network monitoring, server monitoring, and application monitoring.

**Key Features:**

- Auto Discovery: Zabbix can automatically discover and monitor devices and services.
- Custom Monitoring Items: It allows you to create custom monitoring items for specific metrics.
- Alerting: Zabbix supports alerting and notification when predefined thresholds are breached.

**Suggested Reference:**

- Deepak Gaikwad, Viral Thakkar, DevOps Tools from Practitioner's Viewpoint, Wiley.
- The DevOps Handbook - Gene Kim et. al.

**Rubric wise marks obtained:**

<b>Rubrics</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>Total</b>
<b>Marks</b>	Complete implementation as asked	Complete implementation as asked  Problem analysis	Complete implementation as asked  Problem analysis  Development of the Solution	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding	Complete implementation as asked  Problem analysis  Development of the Solution  Concept Clarity & understanding  Correct answer to all questions	

**Signature of Faculty:**