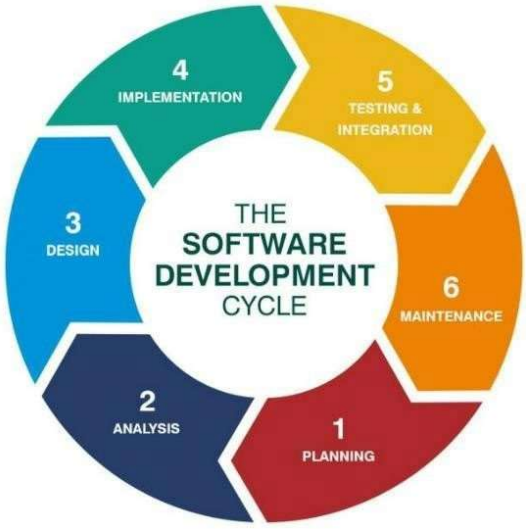
PRACTICAL 1

Aim: Study the complete Software Development Life Cycle (SDLC) and Waterfall Model. Analyze various activities conducted as a part of various phases. For each SDLC phase, identify the objectives and summaries outcomes.

Software Development Life Cycle (SDLC):

Software Development Life Cycle (SDLC) is a well-defined sequence of stages in software engineering to develop the intended software product.

SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC framework includes the following steps:



Planning

This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given studying the existing or obsolete system and software, conducting interviews of users and developers, referring to the database or collecting answers from the questionnaires.

After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyses if a software can be made to fulfil all requirements of the user and if there is any possibility of software being no more useful. It is found out, if the project is financially, practically and technologically feasible for the organization to take up.

Analysis:

At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyses the scope of the project and plans the schedule and resources accordingly.

Design

Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.

Implementation

This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing errorfree executable programs efficiently.

Integration and Testing

An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user's end. Early discovery of errors and their remedy is the key to reliable software.

Software may need to be integrated with the libraries, databases and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

Maintenance

This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by

updating the code according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

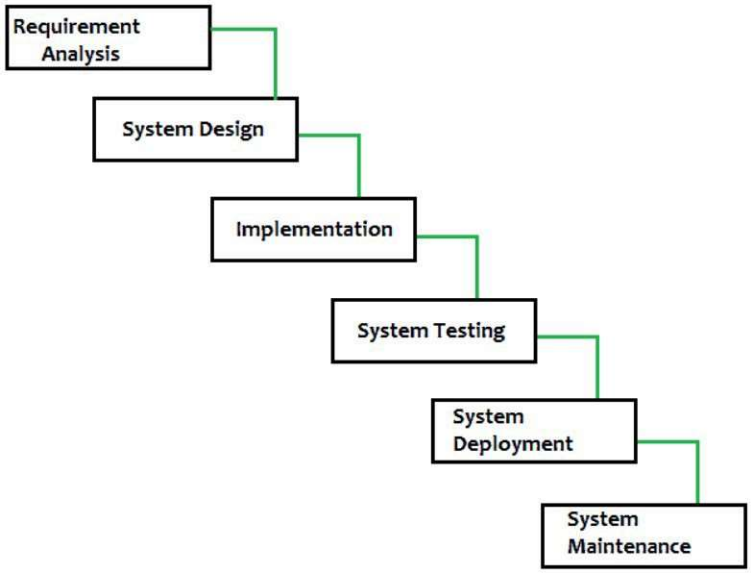
SDLC - Waterfall Model

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The Waterfall model is the earliest SDLC approach that was used for software development. The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially. The following illustration is a representation of the different phases of the Waterfall Model.



[Waterfall Model]

The sequential phases in Waterfall model are –

Requirement Gathering and analysis

All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

System Design

The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

Implementation

With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

Testing

All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Deployment of system

Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

Maintenance

There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model – Advantages:

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one. Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows −

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model – Disadvantages:

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The major disadvantages of the Waterfall Model are as follows –

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.

- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Objectives:

The objective of a courier management system is to automate and streamline the process of managing and tracking deliveries, including package pickup, transportation, and delivery. It can also include features such as tracking and tracing, delivery status updates, and electronic signature capture. The overall goal is to improve the efficiency and accuracy of the delivery process, and to provide customers with real-time information about the status of their deliveries.

Summary:

A courier management system is a software application or platform used to manage the process of delivering packages and mail from a sender to a recipient. This can include tasks such as tracking the location of packages, scheduling delivery routes, and generating shipping labels. The system can also be used to manage inventory and process payments for shipping services. The goal of a courier management system is to make the delivery process more efficient and streamlined for businesses and individuals.

PRACTICAL 2

Aim: Define Requirement Gathering and Technical Requirement specification for selected project.

Requirement Gathering

Requirement gathering for courier management system involves identifying and documenting the specific needs and requirements of a system that is responsible for scheduling and executing courier. This process typically involves a thorough analysis of the current courier management system, as well as consultation with customers, such as product issue and end-users, to determine their requirements. The following are some key elements that should be considered during the requirement gathering process for Courier Management System:

1. Identifying the needs and goals of the business and its stakeholders.

2. Identifying the key features and functionality that the system should have, such as tracking and monitoring of deliveries, scheduling of pickups and deliveries, and integration with existing systems and platforms.

3. Identifying the user groups that will be using the system, such as customers, couriers, and dispatchers, and understanding their specific needs and requirements.

4. Identifying any regulatory or compliance requirements that the system must meet.

5. Identifying the technical infrastructure and resources required to support the system, such as hardware, software, and data storage.

6. Identifying the project budget, timelines, and resource constraints.

7. Identifying the expected benefits of the system, such as cost savings, improved efficiency, and enhanced customer satisfaction.

8. Identifying the risks and contingencies that may impact the project, such as delays, technical issues, or changes in business requirements.

Overall, requirement gathering for a courier management system involves a thorough understanding of the business and its needs, as well as the technical and regulatory requirements that must be met.

Technical Requirement

Courier management systems typically require the following technical specifications:

1. **Hardware:** A computer or server with sufficient processing power and memory to handle the system's workload.

2. **Operating System:** The system should be compatible with popular operating systems such as Windows, Linux, or macOS.

3. **Database:** The system should be able to store and manage large amounts of data, often using a relational database management system (RDBMS) such as MySQL or SQL Server.

4. **Network Connectivity:** The system should be able to connect to the internet and other systems as needed, such as for real-time tracking and status updates.

5. **Web-based Interface:** A web-based interface allows users to access the system from any device with a web browser and internet connection.

6. **Mobile Application:** A mobile application for the couriers to update the package delivery status and for customers to track the package.

7. **Integration with other systems:** The system should be able to integrate with other systems such as inventory management, accounting, and customer relationship management (CRM) systems.

8. **Security:** The system should have proper security measures in place to protect sensitive data and prevent unauthorized access.

Functionality of Courier Management system

1. **Package tracking:** The ability to track the location and status of packages in real-time.

2. **Delivery scheduling:** The ability to schedule deliveries and assign them to specific couriers.

3. **Route optimization:** The use of algorithms to optimize delivery routes for maximum efficiency.

4. **Invoicing and payment:** The ability to generate invoices and process payments for deliveries.
5. **Customer management:** The ability to manage customer information and preferences, such as delivery addresses and contact details.

6. **Courier management:** The ability to manage courier information, including their schedules, availability, and performance.

7. **Reporting and analytics:** The ability to generate reports and perform data analysis on various aspects of the courier management system.

8. **Integration with external systems:** The ability to integrate the courier management system with other systems, such as e-commerce platforms and shipping carriers.

Non-Functionality of Courier Management system

1. **Performance:** The response time and overall performance of the system under various conditions, such as heavy traffic and high volumes of deliveries.
2. **Scalability:** The ability of the system to handle increased loads and support growth over time.
3. **Reliability:** The ability of the system to function consistently and without interruption, even in the event of failures or errors.
4. **Security:** The measures in place to protect sensitive information, such as customer and courier data, from unauthorized access and theft.
5. **Usability:** The ease of use and user experience of the system, including the user interface, navigation, and overall design.
6. **Compliance:** The ability of the system to comply with relevant laws and regulations, such as data privacy laws and security standards.
7. **Maintainability:** The ease of maintenance and updating the system, including the ability to make changes and implement new features.
8. **Support:** The level of support provided by the vendor or development team, including response times and availability of resources.

PRACTICAL 3

Aim: Development of DFD and E-R diagram for the software domain problem.

Data flow diagram (DFD)?

A Data Flow Diagram (DFD) is traditional visual representation of the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or combination of both.

It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

It is usually beginning with a context diagram as the level 0 of DFD diagram, a simple representation of the whole system. To elaborate further from that, we drill down to a level 1 diagram with lower level functions decomposed from the major functions of the system. This could continue to evolve to become a level 2 diagram when further analysis is required. Progression to level 3, 4 and so on is possible but anything beyond level 3 is not very common. Please bear in mind that the level of details for decomposing particular function really depending on the complexity that function.

DFD Diagram Notations

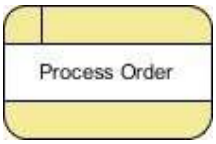
Now we'd like to briefly introduce to you a few diagram notations which you'll see in the tutorial below.

External Entity

An external entity can represent a human, system or subsystem. It is where certain data comes from or goes to. It is external to the system we study, in terms of the business process. For this reason, people used to draw external entities on the edge of a diagram.

Process

A process is a business activity or function where the manipulation and transformation of data takes place. A process can be decomposed to finer level of details, for representing how data is being processed within the process.

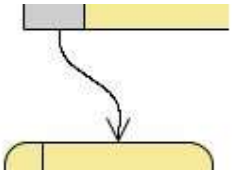


Data Store

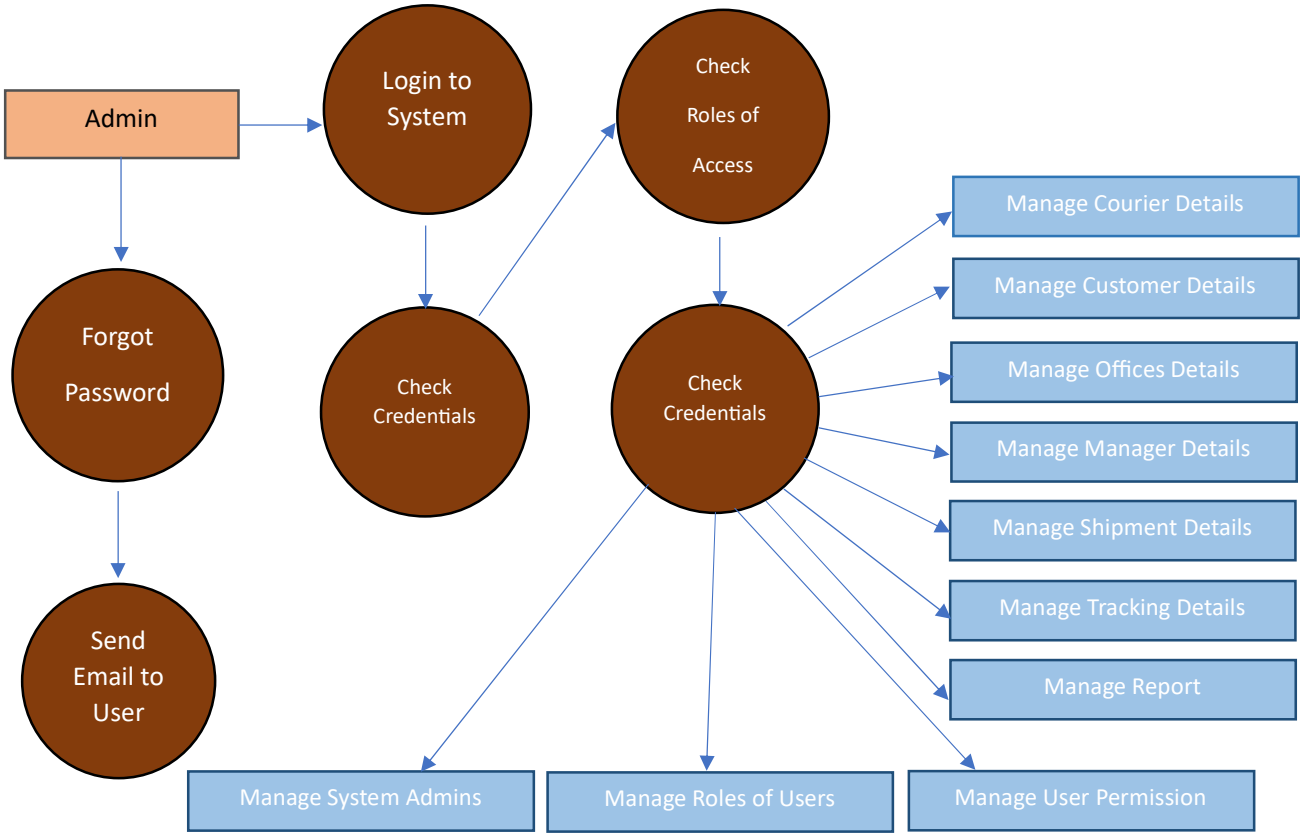
A data store represents the storage of persistent data required and/or produced by the process. Here are some examples of data stores: membership forms, database table, etc.

Data Flow

A data flow represents the flow of information, with its direction represented by an arrow head that shows at the end(s) of flow connector.



Draw DFD Diagram of your project.
Courier Management System



E-R Diagram

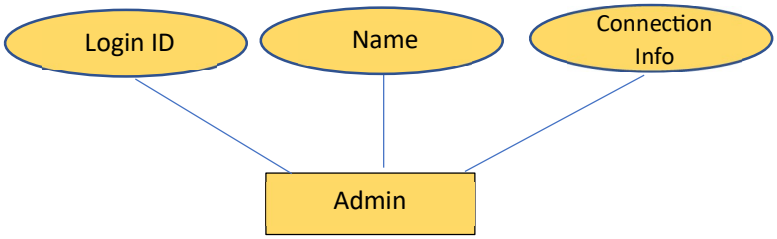
ER Model is represented by means of an ER diagram. Any object, for example, entities, attributes of an entity, relationship sets, and attributes of relationship sets, can be represented with the help of an ER diagram.

Entity

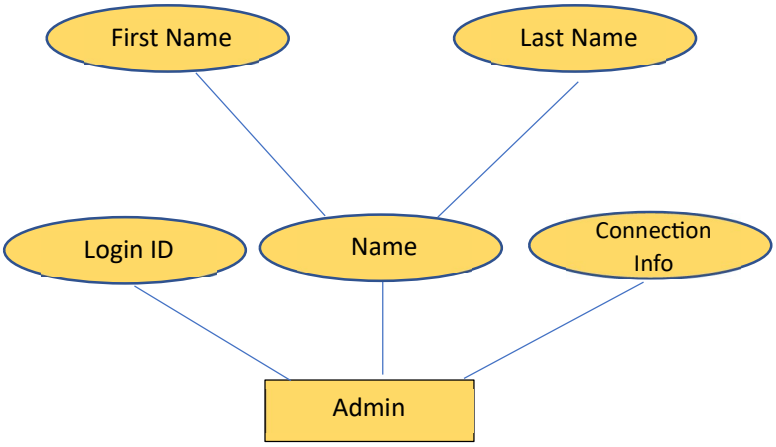
Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

Attributes

Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



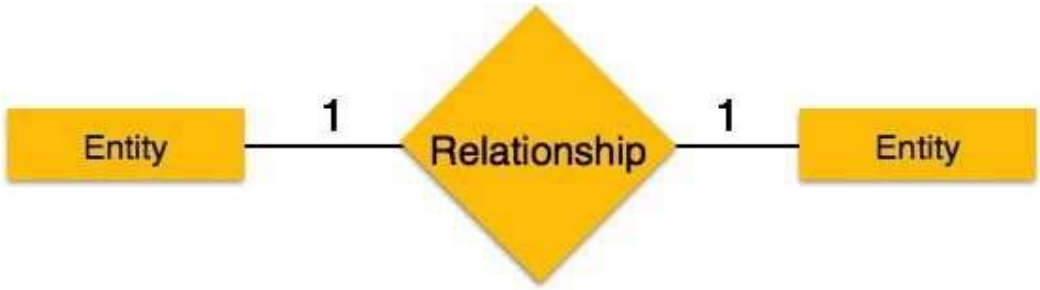
Relationship

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

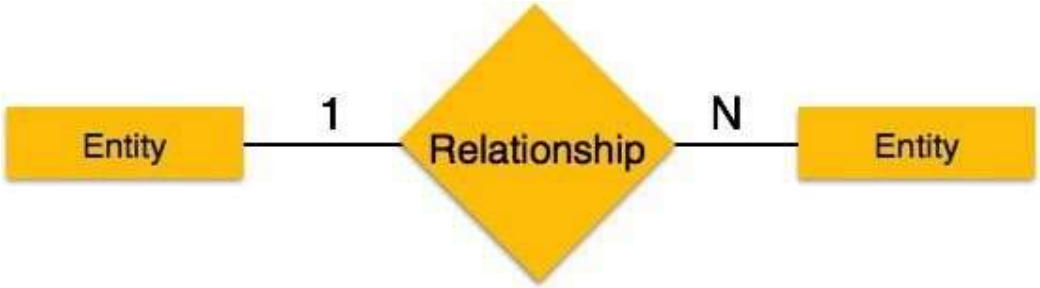
Binary Relationship and Cardinality

A relationship where two entities are participating is called a **binary relationship**. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

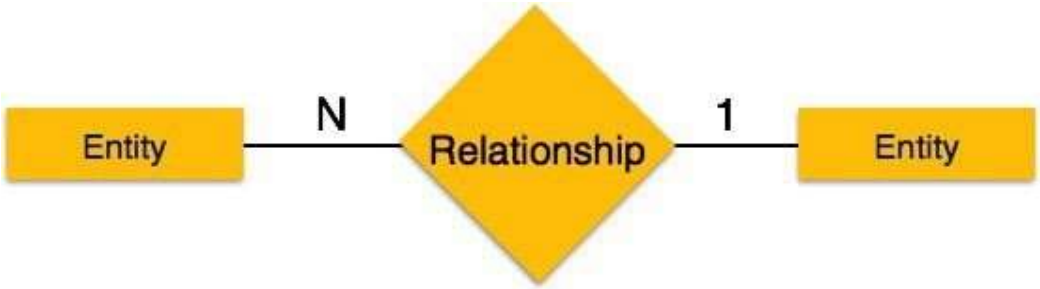
- **One-to-one** − When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



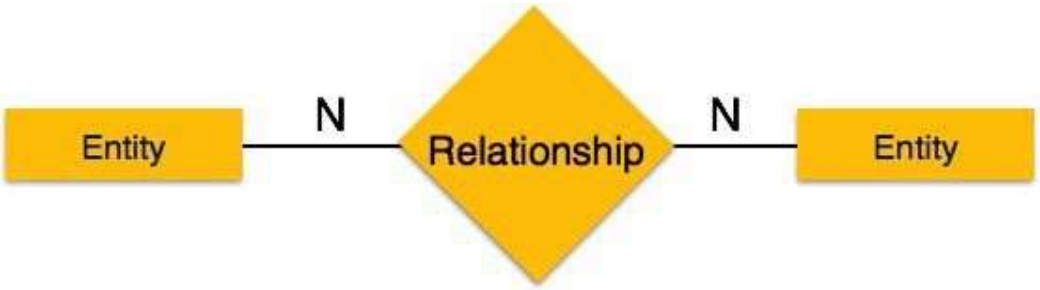
- **One-to-many** − When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.



- **Many-to-one** − When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.



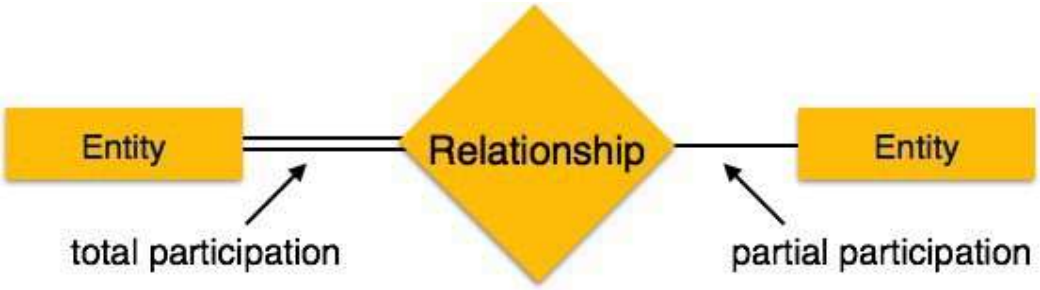
- **Many-to-many** − The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.



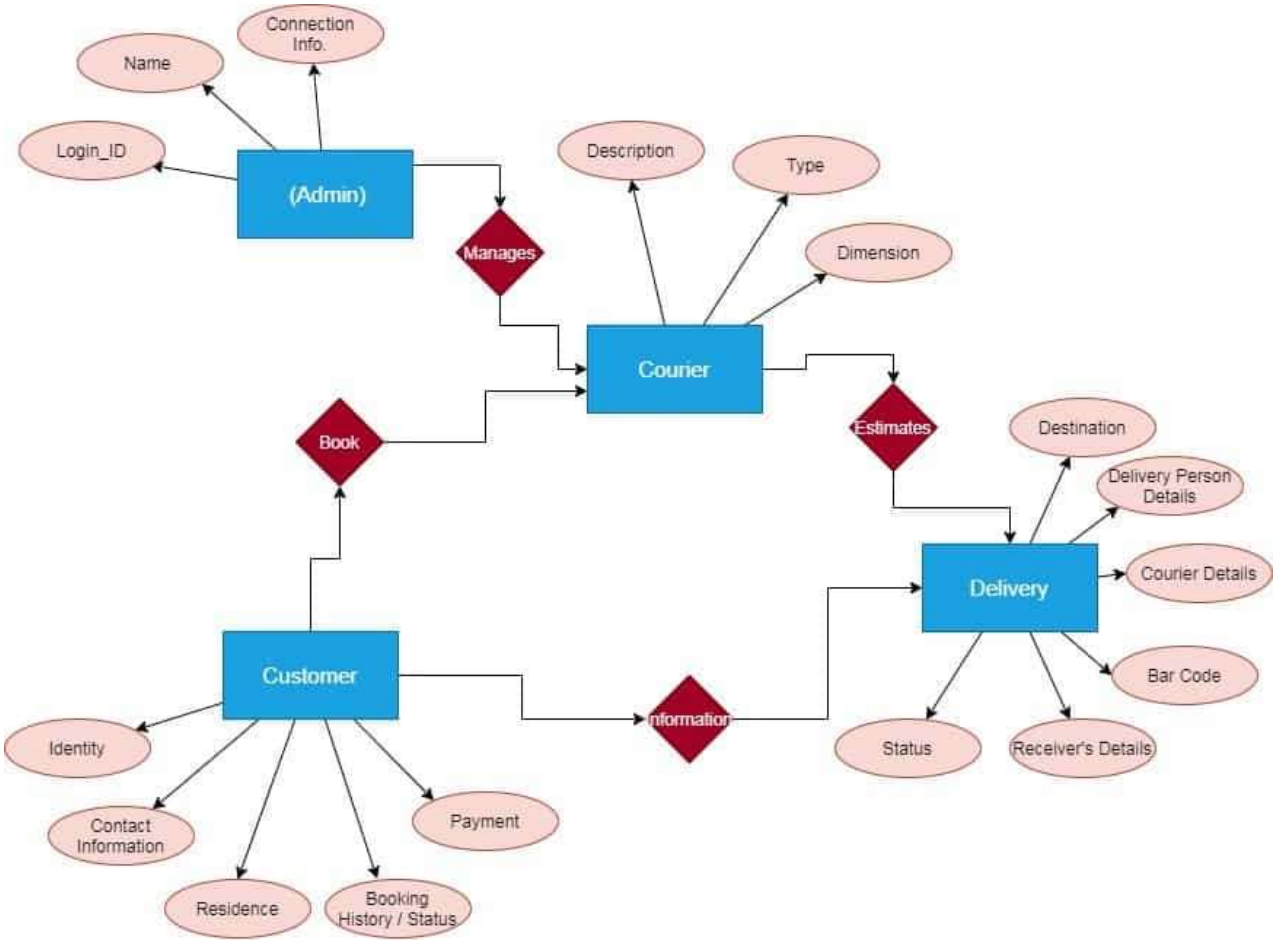
Participation Constraints

- **Total Participation** − Each entity is involved in the relationship. Total participation is represented by double lines.

- **Partial participation** − Not all entities are involved in the relationship. Partial participation is represented by single lines.



Draw ER Diagram of your project.
Courier Management System


Connection Info.
Name
Login_ID
(Admin)
Manages
Description
Type
Dimension
Courier
Book
Estimates
Destination
Delivery Person Details
Courier Details
Delivery
Bar Code
Customer
nformation
Identity
Contact Information
Residence
Booking History / Status
Payment
Status
Receiver's Details

PRACTICAL 4

Aim: Draw Use case and Activity Diagrams for the project definition.

Use Case Diagram

A use case diagram is a dynamic or behaviour diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system.

Basic Use Case Diagram Symbols and Notations

System

Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.

System

Use Case : Draw use cases using ovals. Label the ovals with verbs that represent the system's functions.

Use case

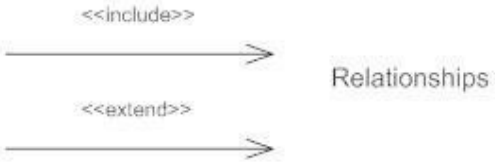
Actors

Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.

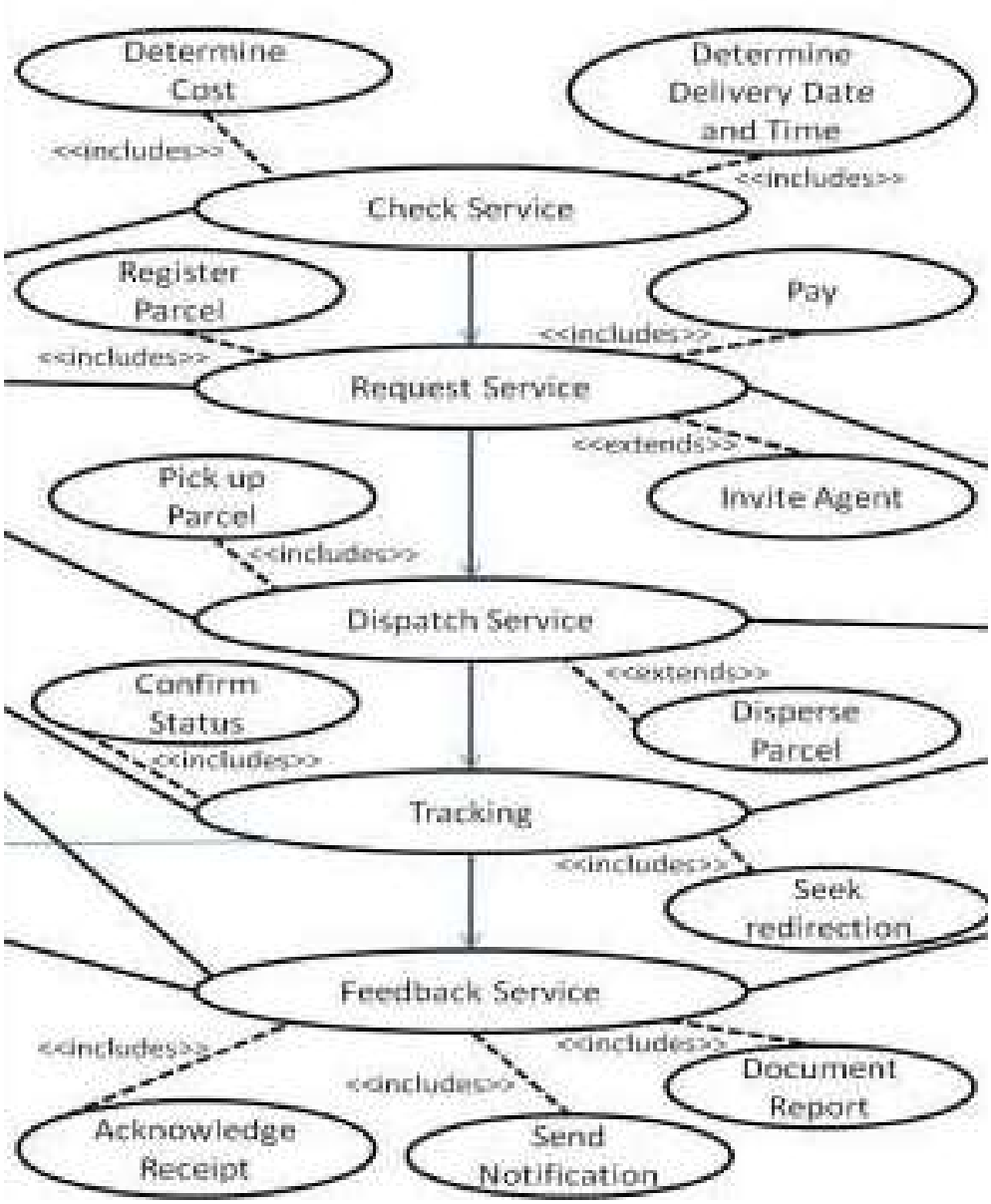
Actor

Relationships

Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.



Draw your USECASE Diagram for the Project



Activity Diagram

Activity diagrams are mainly used as a flowchart that consists of activities performed by the system. Activity diagrams are not exactly flowcharts as they have some

additional capabilities. These additional capabilities include branching, parallel flow, swim lane, etc.

Before drawing an activity diagram, we must have a clear understanding about the elements used in activity diagram. The main element of an activity diagram is the activity itself. An activity is a function performed by the system. After identifying the activities, we need to understand how they are associated with constraints and conditions.

Before drawing an activity diagram, we should identify the following elements −

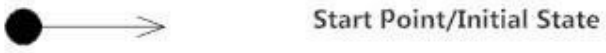
- Activities
- Association
- Conditions
- Constraints

Once the above-mentioned parameters are identified, we need to make a mental layout of the entire flow. This mental layout is then transformed into an activity diagram..

Attributes:

Initial State or Start Point

A small filled circle followed by an arrow represents the initial action state or the start point for any activity diagram. For activity diagram using swimlanes, make sure the start point is placed in the top left corner of the first column.



Activity or Action State

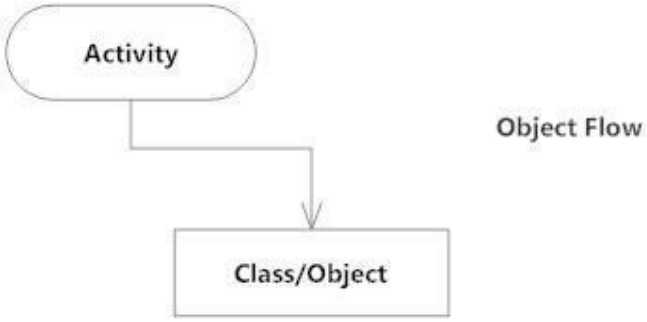
An action state represents the non-interruptible action of objects. You can draw an action state in SmartDraw using a rectangle with rounded corners.

Action Flow

Action flows, also called edges and paths, illustrate the transitions from one action state to another. They are usually drawn with an arrowed line.

Object Flow

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.



Decisions and Branching

A diamond represents a decision with alternate paths. When an activity requires a decision prior to moving on to the next activity, add a diamond between the two activities. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."

Guards

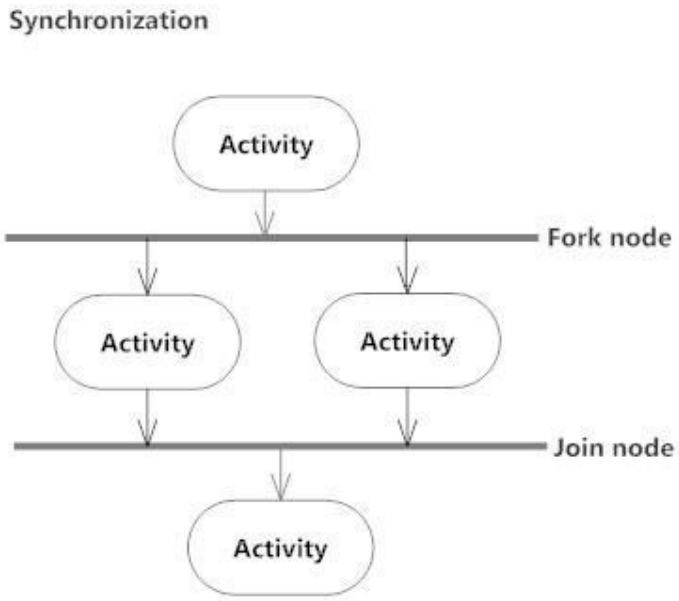
In UML, guards are a statement written next to a decision diamond that must be true before moving next to the next activity. These are not essential, but are useful when a specific answer, such as "Yes, three labels are printed," is needed before moving forward.

Synchronization

A fork node is used to split a single incoming flow into multiple concurrent flows. It is represented as a straight, slightly thicker line in an activity diagram.

A join node joins multiple concurrent flows back into a single outgoing flow.

A fork and join mode used together are often referred to as synchronization.

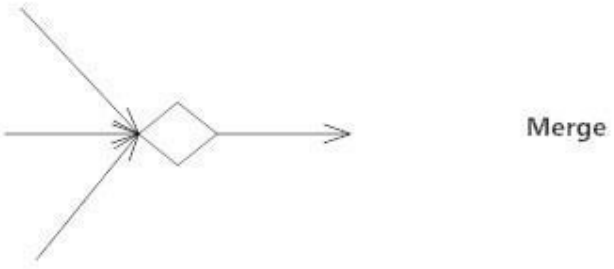


Time Event

This refers to an event that stops the flow for a time; an hourglass depicts it.

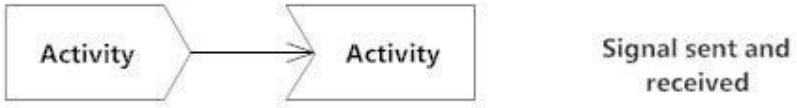
Merge Event

A merge event brings together multiple flows that are not concurrent.



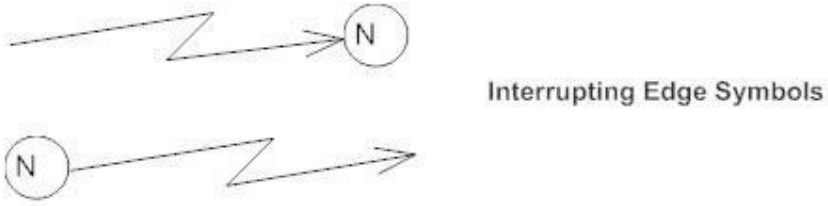
Sent and Received Signals

Signals represent how activities can be modified from outside the system. They usually appear in pairs of sent and received signals, because the state can't change until a response is received, much like synchronous messages in a sequence diagram. For example, an authorization of payment is needed before an order can be completed.



Interrupting Edge

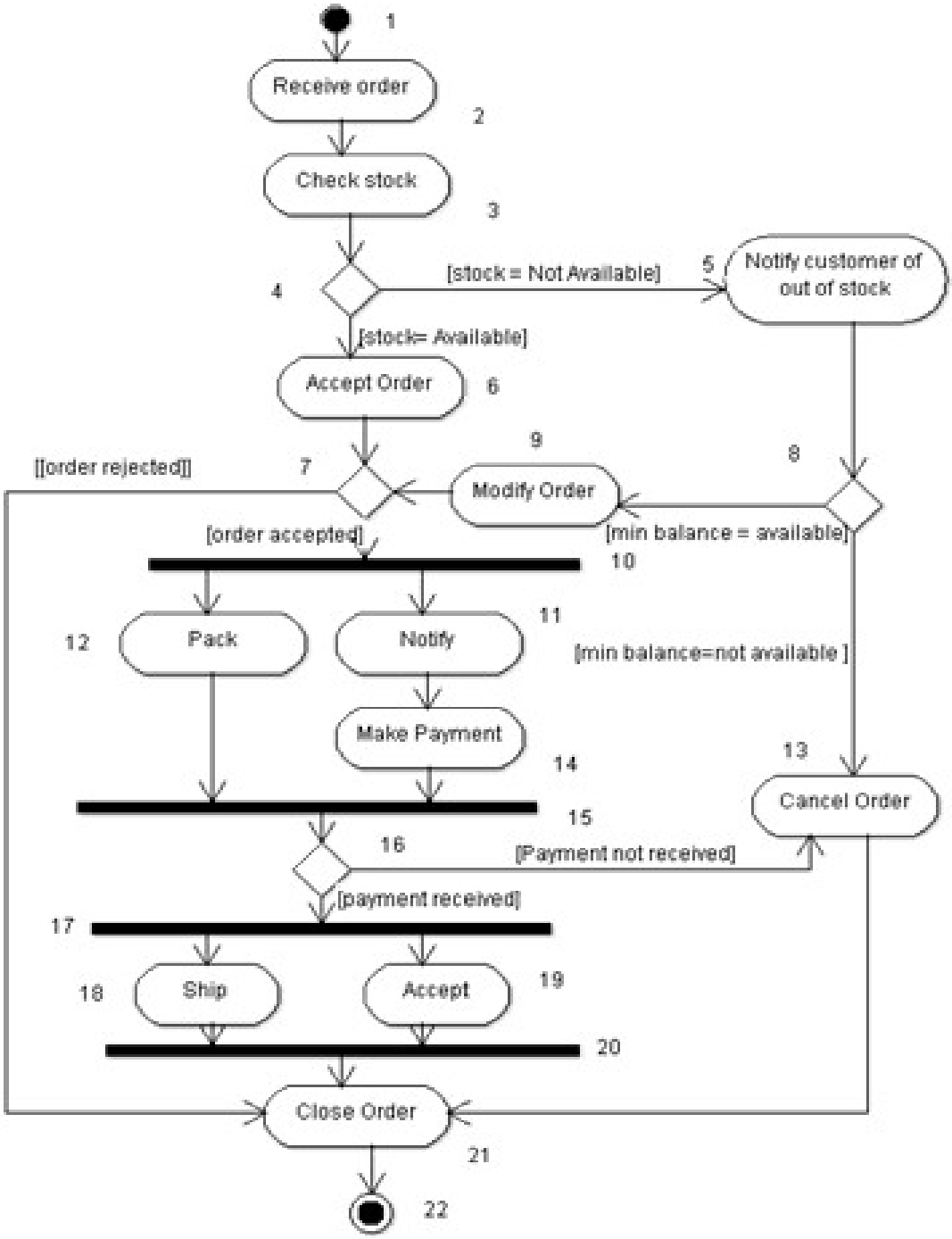
An event, such as a cancellation, that interrupts the flow denoted with a lightning bolt.



Final State or End Point

An arrow pointing to a filled circle nested inside another circle represents the final action state.

Draw activity Diagram for your project.


1
Receive order
2
Check stock
3
4
[stock = Not Available]
5
Notify customer of out of stock
[stock= Available]
Accept Order
6
9
8
[[order rejected]]
7
Modify Order
[min balance = available]
[order accepted]
10
12
Pack
Notify
11
[min balance=not available]
Make Payment
13
14
Cancel Order
15
16
[Payment not received]
[payment received]
17
18
Ship
Accept
19
20
Close Order
21
22

PRACTICAL 5

Aim: Draw the Sequence Diagram for the project definition.

Sequence Diagram

The purpose of interaction diagrams is to capture the dynamic aspect of a system. So to capture the dynamic aspect, we need to understand what a dynamic aspect is and how it is visualized.

Dynamic aspect can be defined as the snapshot of the running system at a particular moment

We have two types of interaction diagrams in UML. One is the sequence diagram and the other is the collaboration diagram. The sequence diagram captures the time sequence of the message flow from one object to another and the collaboration diagram describes the organization of objects in a system taking part in the message flow.

Following things are to be identified clearly before drawing the interaction diagram

Objects taking part in the interaction.

- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

Following are two interaction diagrams modelling the order management system. The first diagram is a sequence diagram and the second is a collaboration diagram

Attributes:

Class Roles or Participants

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

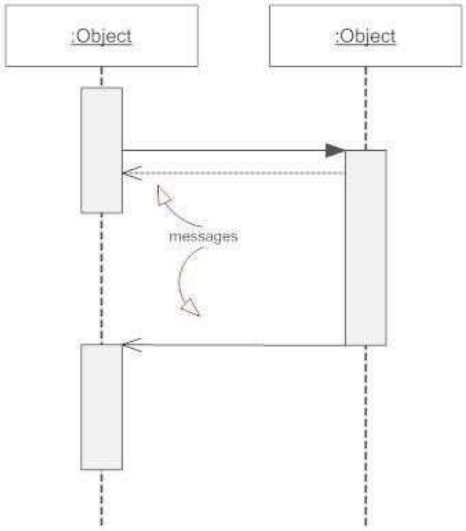
Activation or Execution Occurrence

Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.

Activation or Execution Occurrence

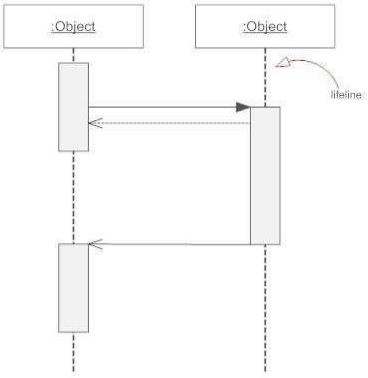
Messages

Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks. For message types, see below.



Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.



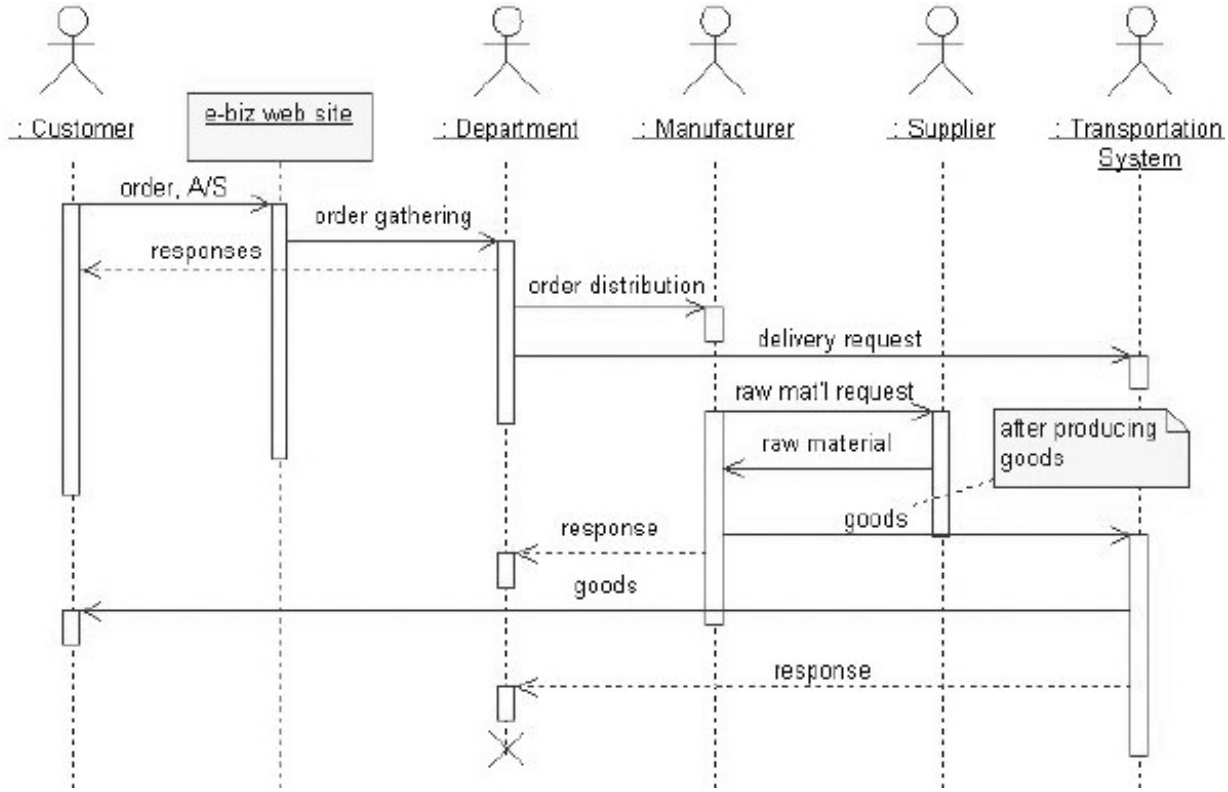
Destroying Objects

Objects can be terminated early using an arrow labelled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets.

Draw Sequence Diagram of your project.



PRACTICAL 6

Aim: Development of State Transition Diagram for Software project definition.

State Transition Diagram

State chart diagram is used to describe the states of different objects in its life cycle. Emphasis is placed on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.

State chart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

Before drawing a State chart diagram we should clarify the following points −

Identify the important objects to be analyzed.

- Identify the states.
- Identify the events.

Following is an example of a state chart diagram where the state of Order object is analyzed.

The first state is an idle state from where the process starts. The next states are arrived for events like send request, confirm request, and dispatch order. These events are responsible for the state changes of order object.

During the life cycle of an object (here order object) it goes through the following states and there may be some abnormal exits. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete, it is considered as a complete transaction as shown in the following figure. The initial and final state of an object is also shown in the following figure.

Attributes:

States

States represent situations during the life of an object. You can easily illustrate a state in SmartDraw by using a rectangle with rounded corners.

A simple state

A state with internal activities

Transition

A solid arrow represents the path between different states of an object. Label the transition with the event that triggered it and the action that results from it. A state can have a transition that points back to itself.

Transition

Initial State

A filled circle followed by an arrow represents the object's initial state.

Initial state

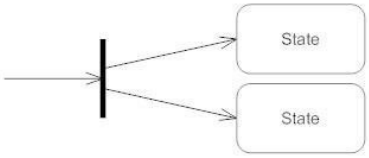
Final State

An arrow pointing to a filled circle nested inside another circle represents the object's final state.

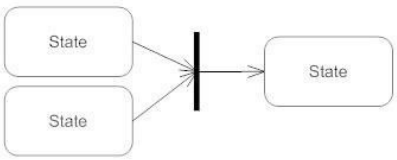
Final state

Synchronization and Splitting of Control

A short heavy bar with two transitions entering it represents a synchronization of control. The first bar is often called a fork where a single transition splits into concurrent multiple transitions. The second bar is called a join, where the concurrent transitions reduce back to one.



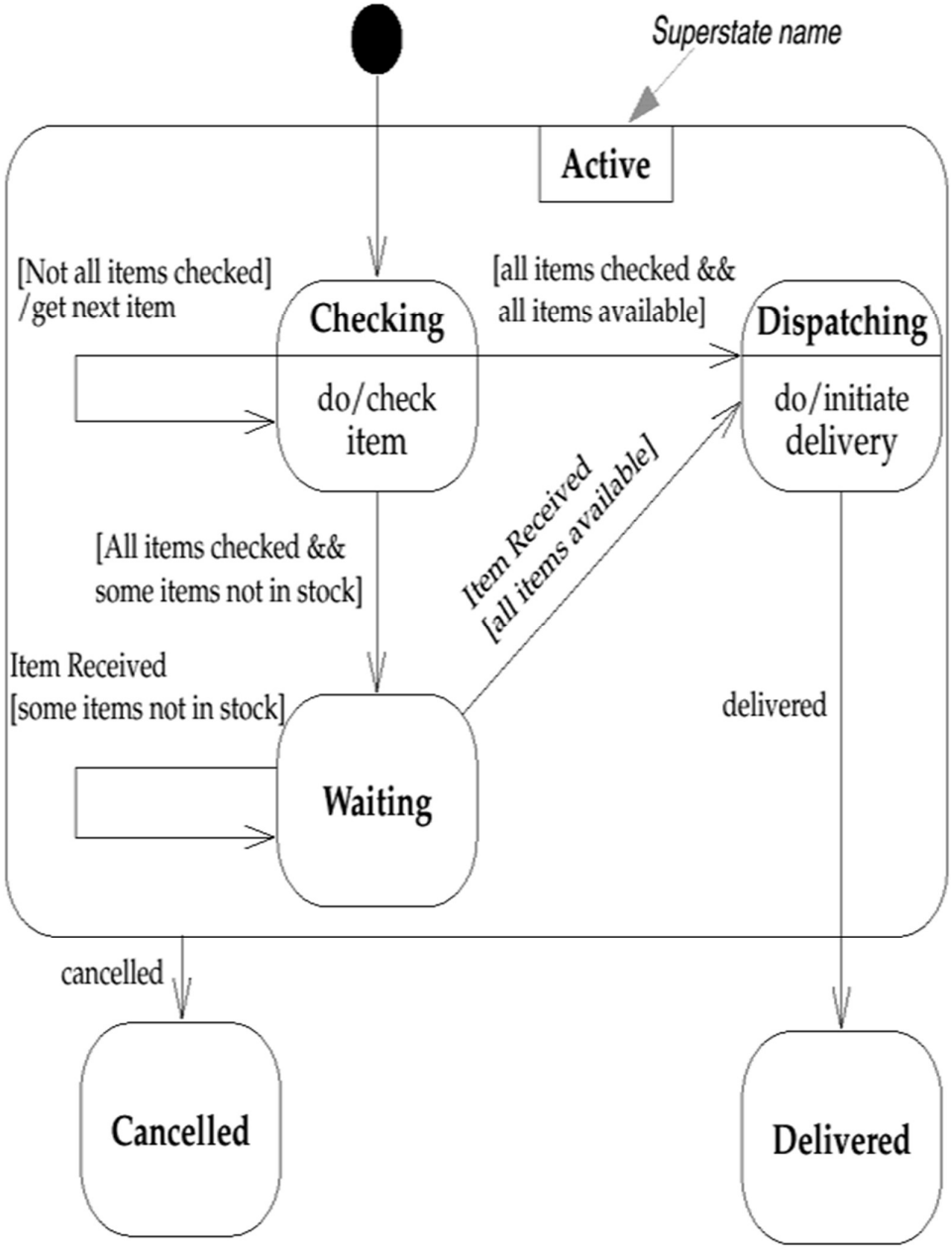
Fork



Join

Draw State Diagram of your project.

Courier Management System



PRACTICAL 7

Aim: Design Class & Object Diagrams for software domain problem.

Class Diagram

Class diagrams are the most popular UML diagrams used for construction of software applications.

It is very important to learn the drawing procedure of class diagram.

Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view.

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represent the whole system.

The following points should be remembered while drawing a class diagram −

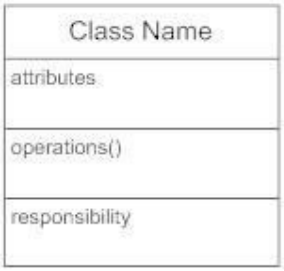
- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified
- For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.

- First of all, Order and Customer are identified as the two elements of the system. They have a one-to-many relationship because a customer can have multiple orders.
- Order class is an abstract class and it has two concrete classes (inheritance relationship) SpecialOrder and NormalOrder.
- The two inherited classes have all the properties as the Order class. In addition, they have additional functions like dispatch () and receive ().

Attributes of Class Diagram: **Classes**

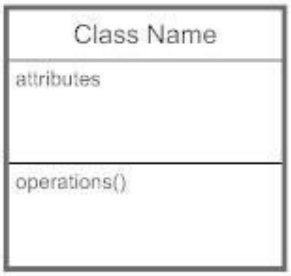
- Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.
- Illustrate classes with rectangles divided into compartments. Place the name of the class in the first partition (centered, bolded, and capitalized), list the attributes in the second partition (left-aligned, not bolded, and lowercase), and write operations into the third.



Class

Active Classes

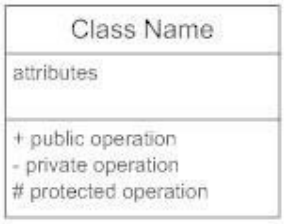
Active classes initiate and control the flow of activity, while passive classes store data and serve other classes. Illustrate active classes with a thicker border.



Active class

Visibility

- Use visibility markers to signify who can access the information contained within a class. Private visibility, denoted with a - sign, hides information from anything outside the class partition. Public visibility, denoted with a + sign, allows all other classes to view the marked information. Protected visibility, denoted with a # sign, allows child classes to access information they inherited from a parent class.

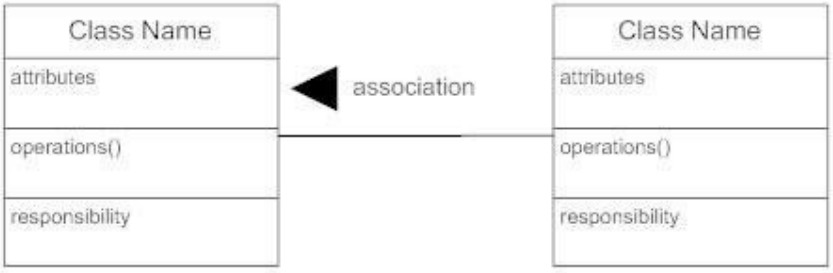


Visibility

Marker	Visibility
+	public
-	private
#	protected
~	package

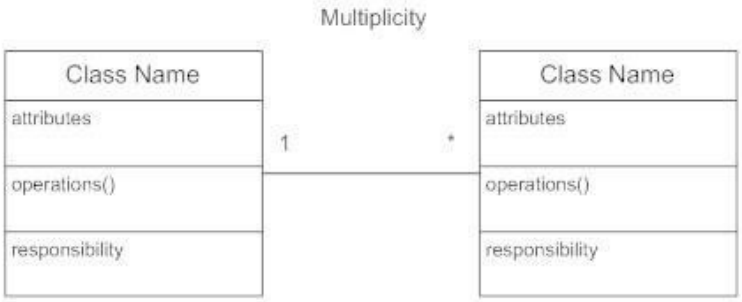
Associations

- Associations represent static relationships between classes. Place association names above, on, or below the association line. Use a filled arrow to indicate the direction of the relationship. Place roles near the end of an association. Roles represent the way the two classes see each other.



Multiplicity (Cardinality)

Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For example, one company will have one or more employees, but each employee works for just one

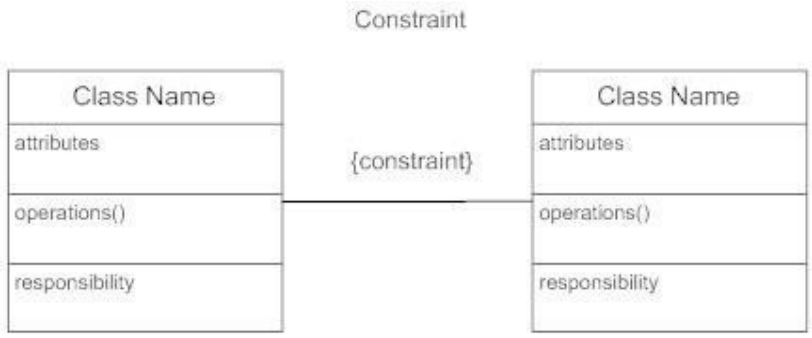


company

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	0 or more
1..* *	1 or more
n	Only n (where n > 1)
0..n	Zero to n (where n >1)
1..n	One to n (where n > 1)

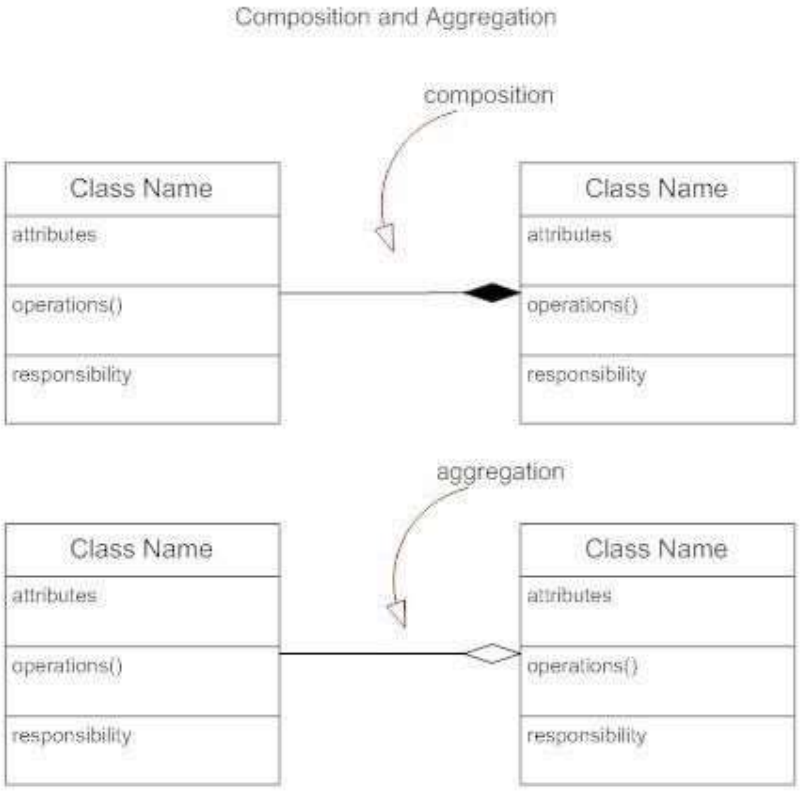
Constraint

Place constraints inside curly braces {}.



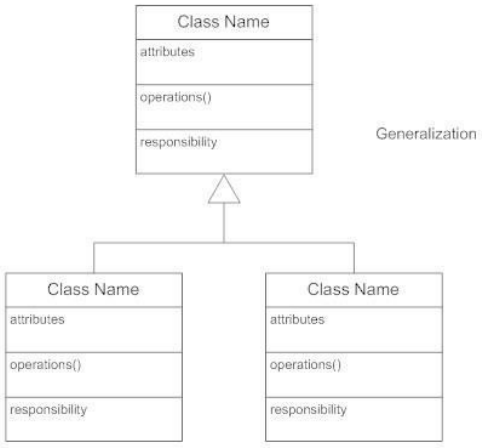
Composition and Aggregation

Composition is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part. Illustrate composition with a filled diamond. Use a hollow diamond to represent a simple aggregation relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other. The diamond ends in both composition and aggregation relationships point toward the "whole" class (i.e., the aggregation).



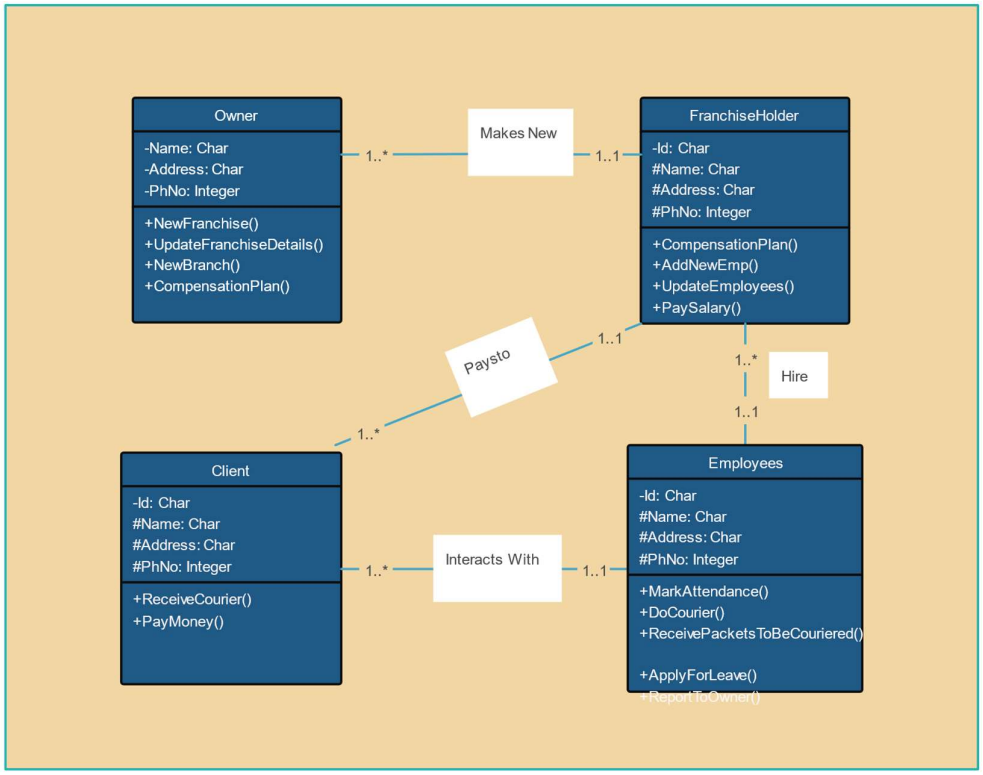
Generalization

- Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. For example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.



- In real life coding examples, the difference between inheritance and aggregation can be confusing. If you have an aggregation relationship, the aggregate (the whole) can access only the PUBLIC functions of the part class. On the other hand, inheritance allows the inheriting class to access both the PUBLIC and PROTECTED functions of the superclass.

Draw your Class Diagram & Object Diagram for your project.



PRACTICAL 8

Aim: Prepare a data dictionary for Software project definition.

Data Dictionary

Data Dictionary is the major component in the structured analysis model of the system. It lists all the data items appearing in DFD. A data dictionary in Software Engineering means a file or a set of files that includes a database's metadata (hold records about other objects in the database), like data ownership, relationships of the data to another object, and some other data.

Importance of Data Dictionary:

- It provides developers with standard terminology for all data.
- It provides developers to use different terms to refer to the same data.
- It provides definitions for different data
- Query handling is facilitated if a data dictionary is used in RDMS.

PRACTICAL 9

Aim: Draw the Deployment Diagram for the project definition.

Deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware used to deploy the application.

Deployment diagrams are useful for system engineers. An efficient deployment diagram is very important because it controls the following parameters

- Performance
- Scalability
- Maintainability
- Portability

So before drawing a deployment diagram the following artifacts should be identified:

- Nodes
- Relationships among nodes

The following deployment diagram is a sample to give an idea of the deployment view of order management system. Here we have shown nodes as:

- Monitor
- Modem
- Caching server
- Server

The application is assumed to be a web based application which is deployed in a clustered environment using server 1, server 2 and server 3. The user is connecting to the application using internet. The control is flowing from the caching server to the clustered environment.

So the following deployment diagram has been drawn considering all the points mentioned above:

Where to use Deployment Diagrams?

Deployment diagrams are mainly used by system engineers. These diagrams are used to describe the physical components (hardware), their distribution and association.

To clarify it in details we can visualize deployment diagrams as the hardware components/nodes on which software components reside.

Software applications are developed to model complex business processes. Only efficient software applications are not sufficient to meet business requirements. Business requirements can be described as to support increasing number of users, quick response time etc.

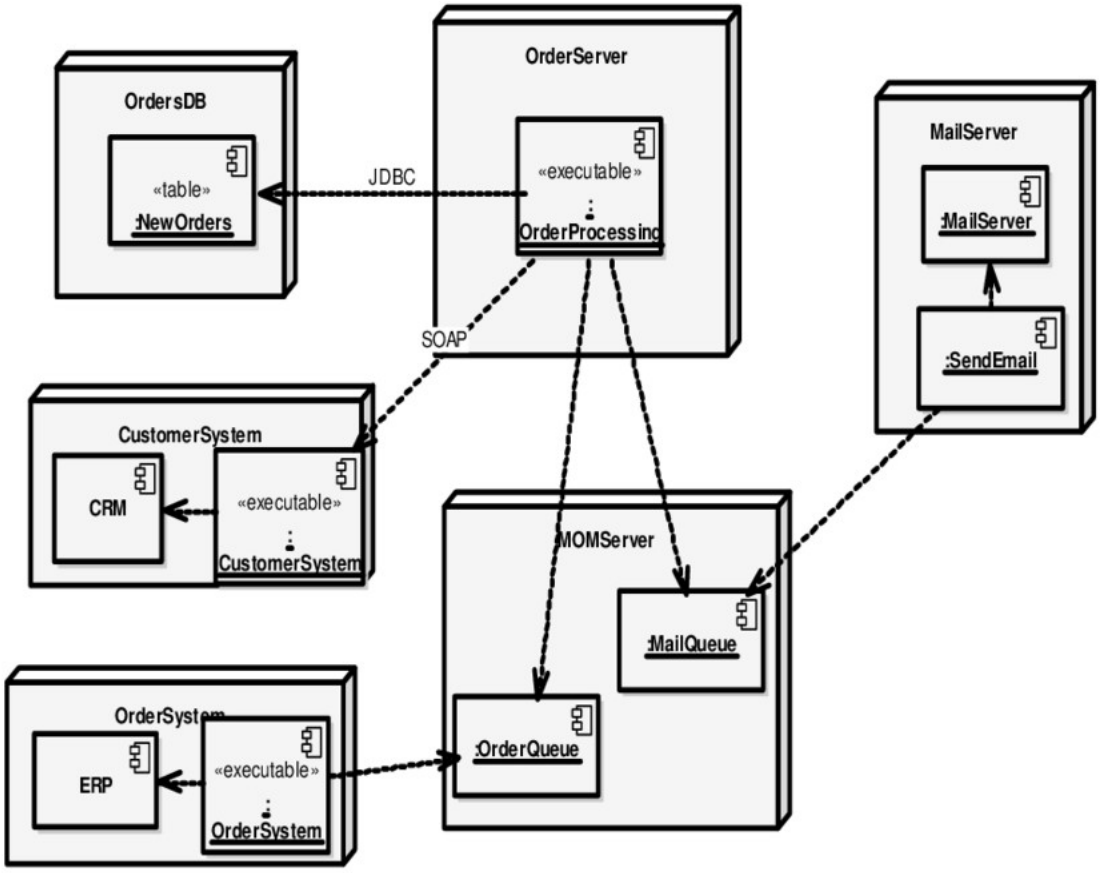
To meet these types of requirements hardware components should be designed efficiently and in a cost effective way.

Now a day's software applications are very complex in nature. Software applications can be stand alone, web based, distributed, mainframe based and many more. So it is very important to design the hardware components efficiently.

So the usage of deployment diagrams can be described as follows:

- To model the hardware topology of a system.
- To model embedded system.
- To model hardware details for a client/server system.
- To model hardware details of a distributed application.
- Forward and reverse engineering.

Deployment Diagram of Courier Management System


OrderServer
OrdersDB
«table»
:NewOrders
JDBC
«executable»
:OrderProcessing
MailServer
:MailServer
:SendEmail
SOAP
CustomerSystem
CRM
«executable»
:CustomerSystem
MOMServer
:MailQueue
OrderSystem
ERP
«executable»
:OrderSystem
:OrderQueue

PRACTICAL 10

Aim: Design the Test Cases for software domain problem.

Srno	Test case id	Test case name	Test case desc	Step	Expected result	Actual Result	Test case status pass/fail
1	Login admin	Validate login	To verify that login name on login page	Enter the login name and password and click submit button	Login successful or an error message "In valid login or password" must be displayed	Login successful	Pass
2	Login Staff	Validate login	To verify that login name on login page	Enter the login name and password and click submit button	Login successful or an error message "In valid login or password" must be displayed	Login successful	Pass
3	Password	Validate password	To verify that password on login page	Enter password and login name click submit button	An error message "password invalid" must be displayed	An error message "password invalid" must be displayed	fail

Sr. No	Test Case Id	Test Case Name	Test Case desc	Step	Expected Result	Actual Result	Test case status pass/fail
1	Create customer details	Validate allocation form	To allocate separate name address, mobile no. for the customers	Nothing entered and click submit button	An error message customer details not equal to null must be displayed	Inserted successful	Pass
2	Create seller details	Validate allocation form	To allocate separate seller id, product details for the sellers	Nothing entered and click submit button	An error message seller details not equal to null must be displayed	Inserted successful	Pass
3	Create order	Validate allocation form	To verify that data stored on database	Nothing entered and click submit button	An error message not click not order details not equal to null must be displayed	Inserted successful	Pass
4	View	Check details of all data	To verify that data stored on database	Generated	An error message return null will be displayed	An error message return null will be displayed	Fail

PRACTICAL 11

Aim: Generate SRS Document as per given format.

Software Requirements Specification

For

Courier management system

Prepared By:-

(12002040601070) Vikram Mali

ADIT

March13, 2023

Table of Contents

SYSTEM REQUIREMENT SPECIFICATON

1 Introduction

A System Requirements Specifications (SRS) is a documentation that describes the features and behavior of a system of software application. This chapter shall provide a detailed description of the requirements for the online Courier Management System (EC).

1.1 Purpose

The purpose of this document is to present a description of an On-Line Courier Management System(). The project is aimed at developing a courier management system for bus company. This structured document sets out the system functionalities in details and it enables the system developer to have a general view of the client's/user's requirements and will enable him to extract the main functionalities requirements of the system so as to understand general view of the system requirements. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli.

The purpose of the System Requirement Specification(SRS) is:

- To identify Roles and Responsibilities of those involved in this process.
- To identify the scope of what is to be done including the interaction styles and interaction devices to be used to come up with a standard system which suits al the user needs.
- To develop a logical business-related model of the proposed system and to make recommendations.
- To determine fully how the system that shall be developed will work better to support the business.
- To identify the tools that shall be used during the requirement elicitation process.
- To facilitate the budget estimations for the system's undertakings i.e. the actual costs and the time that it will take to come up with the system.
- To define all the functional and non-functional requirements.

1.2 Project Scope

The system to be developed is an On-Line Courier Management System for Bus company to automate the courier operations of Bus company. The first subsystem is the parcel booking. The second subsystem is Billing for the parcel. The third subsystem is a General Management Service and Automated Tasks system that shall generate reports for the bookings done during the day. The fourth subsystem shall be for allocating the parcels to the various destinations. The fifth subsystem shall be for Tracking the parcel once the vehicles have been dispatched to their various destinations.

There are four accounts for the On-Line Courier management System as below:

a. **End-users:** these are the customer service representatives who shall use the system when booking and dispatching the parcels to the customers. These end-users work in the courier section for the bus company.

b. **Administrators:** these are the Logistics Officers who shall log in into the system, download and print the reports received from the customer service staff who are the end-users.

c. **Customers:** who can login in the On-line system and track their parcels using their Parcel Number or Mpesa transaction number.

d. **Drivers:** These are also end-users who shall use the system to check destinations and vehicles assigned to them.

The objective of this system is to automate . The system will have modules that will allow all details of the buses and parcels to be entered and stored in a database. It will also be able to update various fields in the database and save the changes made to the system when prompted. The system will accept specific changes that shall be made from time to time and will be able to track the dates when the changes are made to the system data.

2 Overall Description

2.1 Product Perspective

The Courier Management System(EC) will be designed and implemented as an on-line system for the Bus company with the scalability capability. It will be implemented on a central database for authentication and it will be compatible with most of the available operating systems.

2.2 Definition of The Acronyms and Abbreviations

- SRS-System Requirements Specification.
- EC- Courier Management System.
- Subjective satisfaction-The overall satisfaction of the system.
- End-users-The people who will be actually using the system.

2.3 User Characteristics

2.3.1 Parcel Booking/Issuing Officers

They are the ones who receive parcels and register them into the system for dispatch and issue the parcels to the customers when the customers come to collect them. They are the ones who enter the customer data and perform other less technical tasks when necessary. Its therefore necessary to give them basic understanding of the system to be able to support the function of the system from time to time. After the day's bookings, they forward the reports to the Logistics Officer who schedules the parcels for dispatch.

They will require basic ICT skills to be able to understand their responsibilities and what the system will be able to do them. He/she must have basic skills in searching through MYSQL and Php web language.

2.3.2 Logistics Officer

The Logistics Officer is in-charge of Courier Section. He/she is the employee who receives the parcel bookings from the Booking clerks, carry out vehicle allocation to the various route and assigns the drivers to those vehicles together with the parcels on a day to day basis.

Due to the nature of this position, the Logistics Officer Must be a well-educated leader with an in-depth of understanding the Logistics operations and computers. The developer of the new system will be required to take the Logistics Officer on how to use the system after which the developer will delegate the rest of the administration of the system to him/her. He/she must have basic skills in searching through MYSQL and Php web language which shall be used to store the data captured in the system.

2.3.3 Driver

The drivers play an important role in the delivery of the customer parcels by ensuring that they reach their respective destinations. They arrive the vehicles that transport the parcels. To know their scheduled routes and times, the drivers need to login into system. They then confirm the vehicles assigned to them and print out their log sheets sent by the Logistics Officer. The drivers will be required to have basic ICT skills to facilitate them interact with the system.

2.3.4 Fleet Management Officer

The fleet management officers play an important role of managing the fleet. They receive the new vehicles drivers and enter the details in the system. They also ensure that the vehicles are fitted with the GPS devices that shall facilitate the live tracking of the vehicles. They manage the database of the fleet and generate reports of the same.

2.4 Operating Environment

The server(s) hosting the CMS should have a supported and compatible operating system, such as Windows, Linux, or macOS, depending on the chosen technology stack.

Web Server: A web server, such as Apache or Nginx, is typically required to host the web-based components of the CMS, including the user interface and APIs.

Database Management System: A database management system, such as MySQL, PostgreSQL, or MongoDB, is typically required to store and manage the data for the CMS, including orders, shipments, customer information, and other relevant data.

Development Frameworks and Libraries: The CMS may be developed using specific programming languages, frameworks, and libraries, depending on the technology stack chosen for the system.

3 System Features

• Data Input

Data will be entered directly by the parcel booking clerk or any other designated user of the system. The Logistics Officer will be responsible managing the accessibility of the system and so he/she will delegate and assign duties to the parcel clerks, drivers and any other designated system user.

Considering that the staff work in shifts, system security will also be enforced by use of profile creation, authentication and authorization and other measures like access control.

• Users

The users of the system will be the parcel booking clerks, the Logistics Officers drivers and the Parcel Issuing clerks. The roles of the users are as below:

 Booking Clerk: This is a system user who receives the parcels and registers the information of the parcel and sender in the system.

 The Logistics Officer: This is a Super user system user. He/She will manage the users, manage the courier and manage the customer. He/She shall access the system to check and download the booking reports of the day to assign the drivers parcels for dispatch to respective destinations.

 The drivers: These are system users who will access the system to check the job allocations.

- **The Issuing Clerk:** The clerk shall retrieve parcel information from the system up collection by the customer.
- **Customers:** These are external customers who can request for login credentials so that they can be able to track their parcels.

  All the transactions carried out on the system will be logged with the names of the users accessing the system, the date of access and the particular transaction carried out.

• Data Processing:

Before any information of the parcel is entered into the system, it will have to be validated first by the user. Once the data has been keyed into the system, the

system will prompt the user to verify the data before is accepted into the database. It will not be possible to delete an entry after the data has been stored but the data can be updated, archived or retrieved.

- **Data output:**

The processed data will be displayed on the screen first to view if there are any errors that need correction before it is submitted for storage in the database. The authorized users can retrieve the data and send it to a printer in case a hard copy is required or in this case for the customers to sign as required by the business policy.

- **Control**:

The system will be able to provide some useful controls such as navigating from one record to another or the previous one. Any user who is to access the system will be required to log onto the system using a valid username and password so as to be able to book a parcel, assign jobs, print jobs allocated to them and retrieve parcel information.

Any unauthorized user will **NOT** have access to the system **NEITHER** will users be allowed to share passwords as per system security audit requirements.

3.1 Use Cases

Use Case 1: Parcel Booking Use case

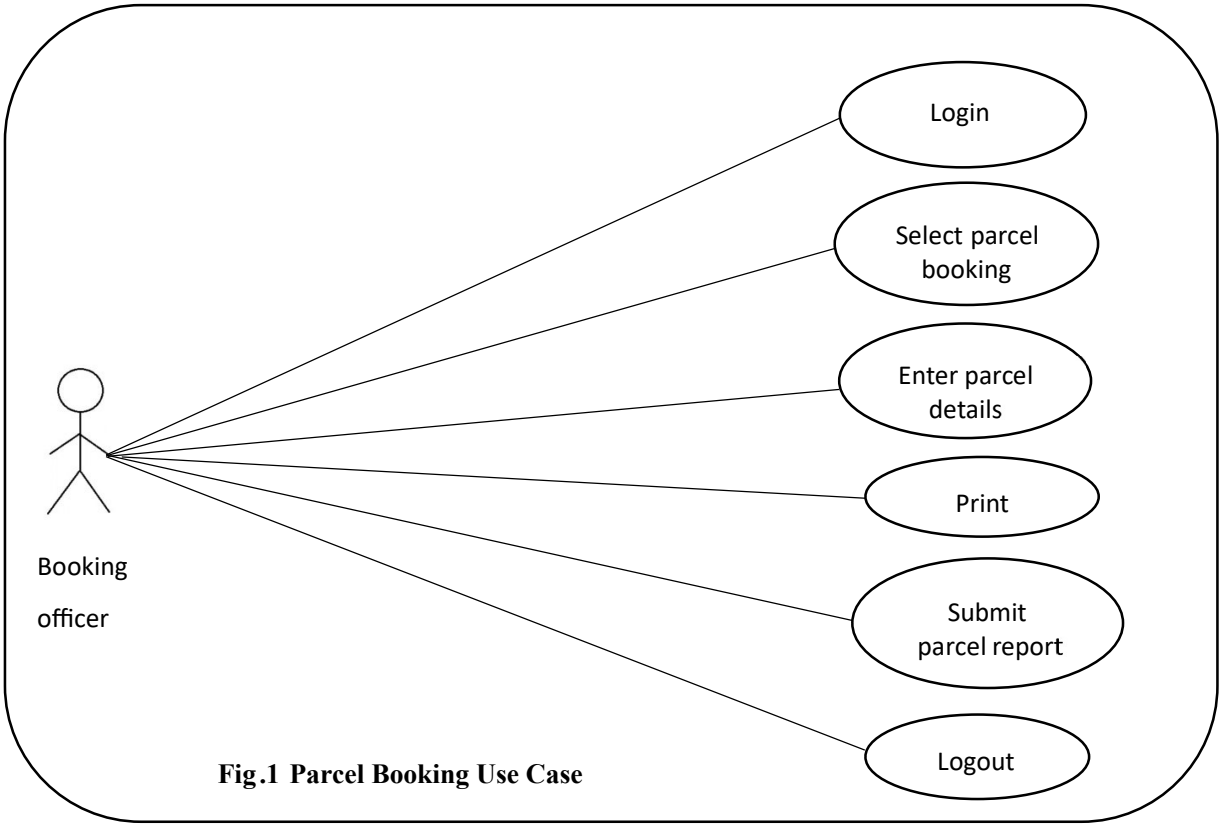


Fig.1 Parcel Booking Use Case

Stimuli/response sequence

i. The booking officer logs into the system.

ii. The officer then selects the parcel booking section of the system.

iii. The office selects the new booking section.

iv. The officer then enters all the details of the parcel.

v. The officer prints the receipt for the customer to sign.

vi. The officer then submits the details of the parcel.

vii. The officer logs out.

Use case 2: Allocation of Vehicle and Driver

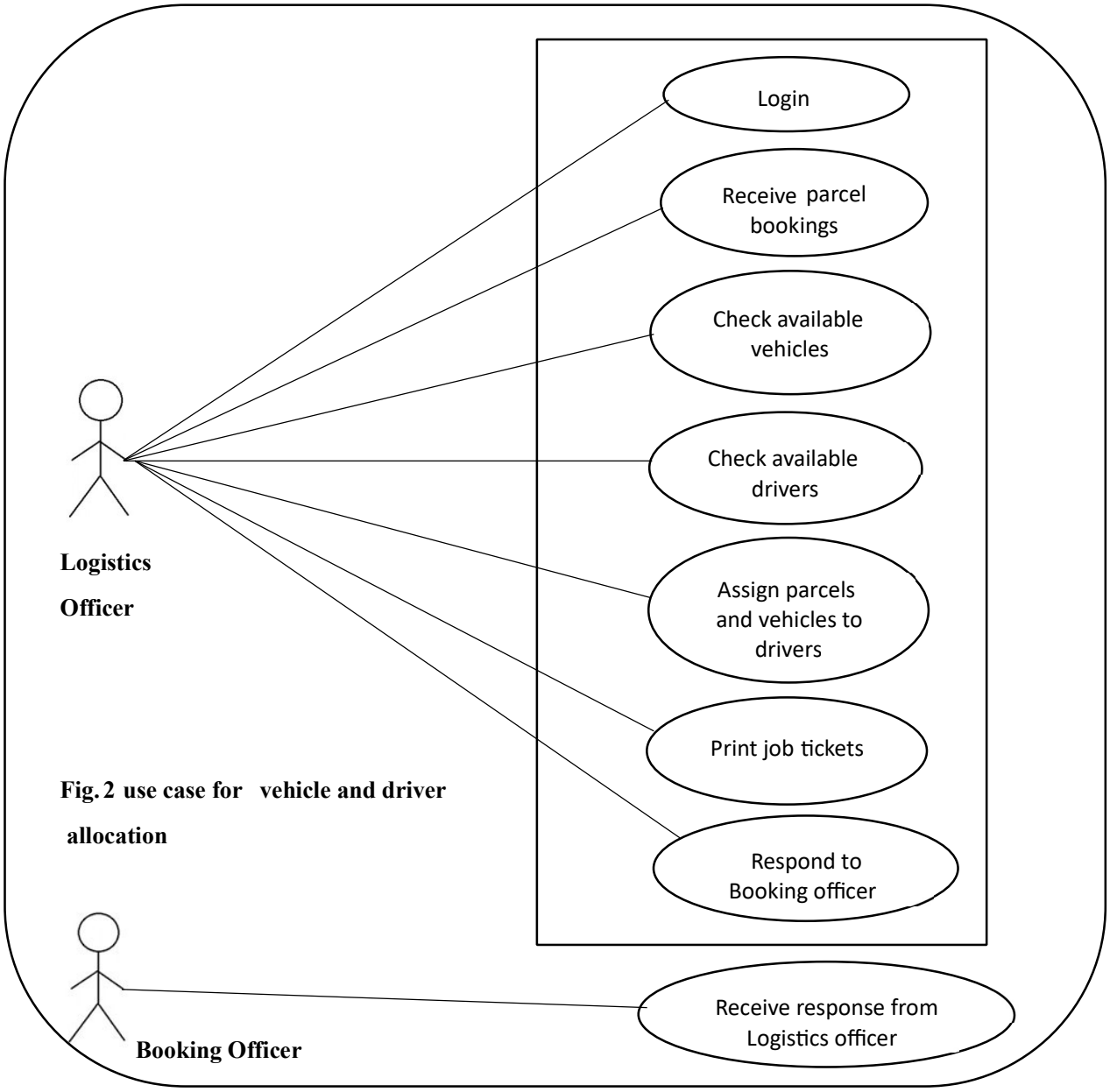


Fig. 2 use case for vehicle and driver allocation

Stimuli/ Response sequence

i. The Logistics officer logs into the system.
ii. The Logistics Officer receives the parcel bookings
iii. The Logistics officer checks for the vehicles that are available.

Use case 3: Check Allocated jobs.

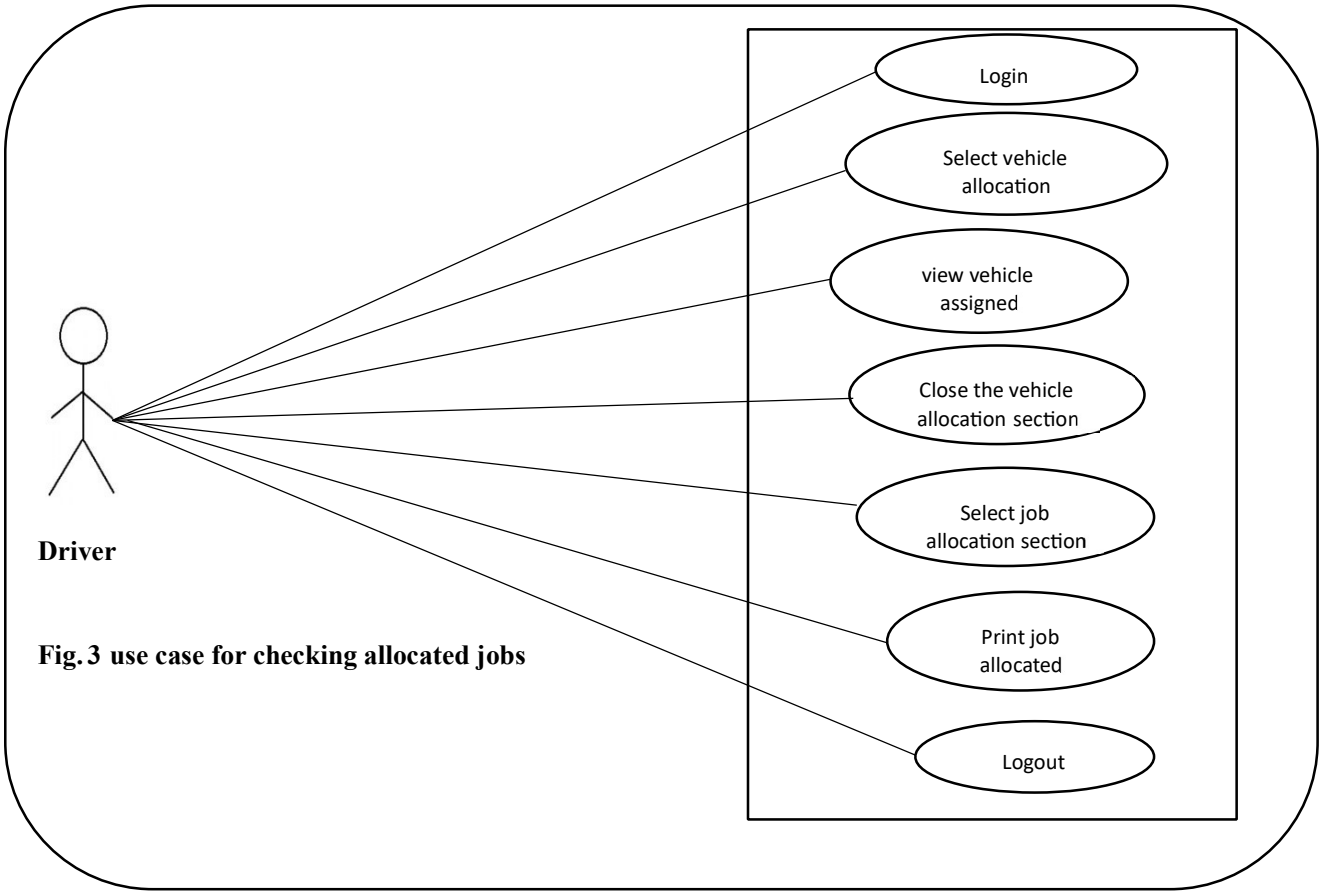


Fig. 3 use case for checking allocated jobs

a. The officer finds that the vehicles are unavailable.

b. The officer waits until a vehicle returns from a job.

iv. The Logistics officer checks for the drivers that are available.

a. The officer finds that the drivers are unavailable.

b. The officer waits until the drivers return from a job.

v. The Logistics Office assigns the parcels and a vehicle to the driver.

vi. The Logistics Officer prints a job ticket.

vii. The Logistics officer then responds to the booking officer that a vehicle and a driver have been found.

viii. The Logistics Officer sends a response to the booking officer for update purposes.

Stimuli/Response sequence

i. The driver logs into the system.

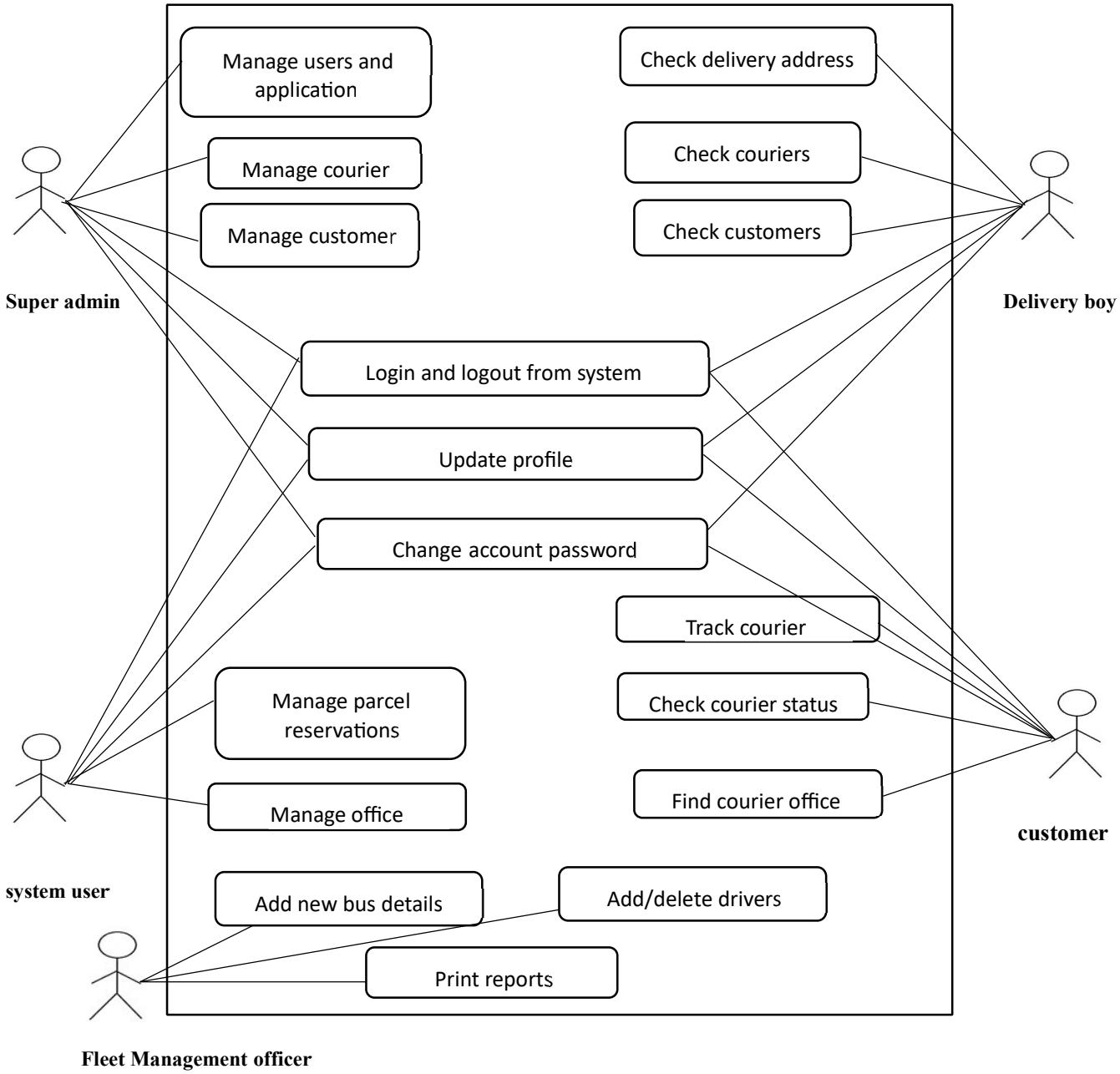
ii. The driver selects the job allocation section.

iii. The driver views the allocated jobs.

iv. The driver prints the allocated jobs.

v. The drover logs out of the system.

Use Case 4: Diagram for the Online Management Tracking system



4 External Interface Requirements

External interface requirements specify the hardware, software or database elements with which a system or component must interface.

4.1 Hardware Interface

- The Desktop is Pentium 4.
- Operating System is Windows 7 Professional
- Hard disk capacity is 40Gb
- RAM is 512MB
- Keyboard is the standard QWERTY keyboard for interface.
- Mouse is the standard mouse with two buttons.
- The monitor is a 17' Display Monitor.
- Printer

4.2 Software Interface

- The Front End is CSS, JavaScript and HTML
- Back end is Php and Java

4.3 Communication interface

- Windows
- Wamp Server

5 Functional Requirements

A functional requirement is a function that the system must perform.

5.1 Parcel Booking

The following are the Functional requirements for Parcel Booking performed by the Booking officer:

- **Login**: The officer logs into the system.
- **Select Add Shipment:** The officer then selects the add shipment section of the system.
- **Enter Parcel details:** The officer enters all the details of the parcel.
- **Print receipt:** the officer prints the receipt for the customer to sign.

- **Submit Parcel report**: the officer then submits the parcel reports to the Logistics Officer.
- **Logout:** The officer logs out.

5.1.1 Allocation of Vehicle and Driver:

The following are the Functional requirements for allocation of vehicle and driver performed by Logistics Officer:

- **Login:** The Logistics officer (who is also the admin) logs into the system.
- **Receive Parcel bookings**: The Logistics Officer checks the parcel bookings.
- **Check available vehicles**: The Logistics officer checks the vehicles available.
- **Check available drivers**: The Logistics officer checks for available drivers.
- **Assign parcels and vehicles to drivers**: The Logistics officer assigns parcels to respective vehicles and drivers.
- **Print job tickets**: The Logistics officer prints out allocated jobs for filing.
- **Respond to Booking officer**: The Logistics officer then responds to the booking officer sharing the jobs allocated.
- **Receive response from Logistics Officer:** The booking officer receives the update from Logistics Officer.

5.2 Check allocated Jobs

The following a functional requirement for checking allocated jobs performed by the Driver:

- **Login**: the driver logs into the system to check the allocated jobs.
- **Select vehicle allocation**: the driver goes to select vehicle section to see what vehicle has been assigned to him/her. **View vehicle assigned**: The driver views the vehicle assigned to him/her.
- **Close the vehicle allocation section**: The driver closes the vehicle allocation section.
- **Select job allocation section**: The driver navigates to select job allocation section to view assigned jobs.
- **Print job allocated**: the driver prints out assigned jobs.
- **Logout**: the driver logs out of the system.

6 Non-functional Requirements

6.1 Security

Security is a very important aspect of the system. The server that shall host the Courier Management System database will have its own security measures put in place to prevent unauthorized write or delete access. The shall be no restriction on read access. Each user will have a security clearance which will ensure that they do not access restricted data.

Each user shall have a profile created that shall authenticate and authorize them to enable him/her to be able to log in to the PC and the Courier Management System.

6.2 Efficiency

The Courier Management system shall use possible minimum computing resources required to perform a function.

5.3 Reliability

The new system shall be reliable and dependable and the users shall be able to use when they need to with minimal failures possible. It will perform with the required precision.

54 Usability

The new system shall be user friendly to ensure that the end-users operate with ease and match their level of computer expertise. The interface and functionality will be made as suitable as possible for the users so that they can operate with ease.

5.5 Portability

The Courier Management System will use windows operating system. The programming language type to be used is PHP. The system shall not be limited to particular computer hardware either desktop or Laptop.

PRACTICAL 12

Aim: Why Agile process models with DevOps are recommended in big companies like Infosys. Justify with one of the case studies (You can explore it with any tool like JIRA). Write a report of 3-5 pages on case study.

- Agile process models with DevOps are recommended in big companies like Infosys and other organizations for several reasons. One of the key reasons is that Agile and DevOps methodologies promote faster, more efficient, and collaborative software development practices that align well with the dynamic and rapidly changing nature of the modern business environment.

- One case study that exemplifies the benefits of Agile with DevOps implementation is Infosys' transformation journey with a large financial services client, as reported in a published case study. The client was seeking to modernize their legacy applications to improve customer experience and reduce time-to-market for new features and enhancements. Infosys adopted an Agile methodology combined with DevOps practices to address the client's requirements, and the results were significant:

Faster Time-to-Market: The Agile-DevOps approach enabled Infosys to accelerate the development and deployment of new features and enhancements, reducing the time-to-market for the client's applications. This helped the client to quickly respond to changing market demands and stay ahead of the competition.

Improved Collaboration and Communication: Agile practices, such as daily stand-ups, sprint planning, and backlog grooming, fostered better collaboration and communication among the development teams, client stakeholders, and other relevant parties. This helped in reducing misunderstandings, resolving issues quickly, and ensuring alignment of the development efforts with the client's business goals.

Enhanced Flexibility and Adaptability: Agile methodologies provide the flexibility to adapt to changing requirements, priorities, and feedback from stakeholders. This allowed Infosys to continuously refine and prioritize the features and enhancements based on evolving business needs, resulting in a more customer-centric approach to software development.

Higher Quality and Reduced Defects: The integration of DevOps practices, such as continuous integration, automated testing, and continuous deployment, helped in identifying and fixing defects early in the development cycle, resulting in higher quality software with fewer defects. This reduced rework, improved overall application stability, and enhanced end-user satisfaction.

Increased Efficiency and Productivity: The Agile-DevOps approach optimized the software development processes, eliminating unnecessary delays and bottlenecks. This resulted in

increased efficiency and productivity, with shorter development cycles, faster feedback loops, and reduced idle time for development teams.

Cost Optimization: Agile methodologies with DevOps practices can lead to cost optimization by reducing waste, eliminating rework, and enhancing resource utilization. This can result in improved return on investment (ROI) for the client's software development efforts.

Overall, the adoption of Agile methodologies with DevOps practices helped Infosys in achieving faster time-to-market, improved collaboration, flexibility, higher quality software, increased efficiency, and cost optimization in their engagement with the financial services client, showcasing the benefits of Agile with DevOps in a large organization setting.

Case Study On Infosys

Here is a case study on how Infosys implemented Agile and DevOps practices to achieve better software delivery:

Infosys is a global consulting and IT services company that provides end-to-end software development services for clients in various industries. As part of their commitment to delivering high-quality software products, Infosys implemented Agile and DevOps practices across their software development teams.

Firstly, Infosys implemented Agile practices such as Scrum and Kanban to improve collaboration and transparency among teams. They also used continuous integration and continuous delivery (CI/CD) pipelines to automate the software delivery process, which helped to increase speed and efficiency.

Additionally, Infosys leveraged DevOps tools such as Jenkins, Git, and Docker to automate their software delivery process. They also adopted a culture of continuous improvement, encouraging their teams to continuously learn and improve their processes and tools.

As a result of these initiatives, Infosys was able to achieve significant improvements in software delivery. For example:

• The company was able to reduce their software delivery cycle time from months to weeks, resulting in faster time-to-market and increased customer satisfaction.

• They also achieved higher quality software products, with fewer defects and higher reliability.

• The adoption of Agile and DevOps practices also resulted in increased collaboration and communication among teams, which improved overall efficiency and productivity.

Overall, Infosys' implementation of Agile and DevOps practices helped them to achieve better software delivery, resulting in faster time-to-market, higher quality products, and increased customer satisfaction.

• Implementation across multiple teams: Infosys implemented Agile and DevOps practices across multiple teams and projects, which helped to standardize their software development process and improve overall efficiency.

• Training and support: Infosys provided training and support to their teams to help them learn and adopt Agile and DevOps practices. This included training on Scrum and Kanban methodologies, as well as on the DevOps tools and processes they were using.

• Metrics and measurement: Infosys established metrics and measurement practices to track the effectiveness of their Agile and DevOps initiatives. This helped them to identify areas for improvement and make data-driven decisions about their software development process.

• Continuous improvement: Infosys adopted a culture of continuous improvement, which encouraged their teams to continuously learn and improve their processes and tools. This helped to ensure that they were always using the most effective practices and tools for their software development projects.

• Customer focus: Infosys' adoption of Agile and DevOps practices was driven by a focus on delivering high-quality software products that met customer needs and requirements. This customer-focused approach helped to ensure that their software products were successful in the market and met the needs of their clients.

Overall, Infosys' implementation of Agile and DevOps was a comprehensive effort that involved training, support, metrics, and a focus on continuous improvement. This approach helped them to achieve better software delivery and meet the needs of their clients.

• Collaboration and communication: Infosys' adoption of Agile and DevOps practices helped to improve collaboration and communication among teams, both within Infosys and with their clients. This improved overall efficiency and reduced the risk of misunderstandings or miscommunications.

• Testing automation: Infosys used automation tools for testing, such as Selenium and Appium, to ensure that software products were thoroughly tested and validated before release. This helped to improve software quality and reliability.

• Cloud adoption: Infosys also adopted cloud computing technologies to enable faster and more efficient software development and delivery. This included using cloud infrastructure for testing and deployment, as well as leveraging cloud-based DevOps tools such as AWS CodePipeline.

• Agile maturity model: Infosys developed an Agile maturity model to help measure their progress and identify areas for improvement. This model helped to ensure that their Agile adoption was sustainable and effective over the long term.

• Agile and DevOps training for clients: Infosys also provided Agile and DevOps training to their clients, which helped to improve collaboration and communication between Infosys and their clients. This also helped to ensure that clients were able to understand and participate in the software development process.

Overall, Infosys' adoption of Agile and DevOps practices was a comprehensive effort that involved collaboration, communication, testing automation, cloud adoption, Agile maturity modeling, and training for both their internal teams and their clients. This approach helped to ensure that they were able to consistently deliver high-quality software products that met the needs of their clients.

• **DevSecOps:** Infosys incorporated security into their DevOps processes, adopting a DevSecOps approach. This involved implementing security testing and vulnerability scanning throughout the development lifecycle, ensuring that security considerations were taken into account at every stage of the process.

• **DevOps Center of Excellence (CoE):** Infosys established a DevOps Center of Excellence to promote collaboration and sharing of best practices across their teams. This CoE helped to ensure that DevOps practices were standardized across the organization and that teams had access to the tools and resources they needed to be successful.

• **Lean principles:** Infosys adopted Lean principles as part of their Agile and DevOps initiatives, which helped to reduce waste and improve efficiency. This involved identifying and eliminating bottlenecks in the software development process, reducing cycle time, and improving overall quality.

- **Continuous feedback:** Infosys implemented a continuous feedback mechanism to ensure that they were always aware of customer needs and expectations. This feedback was incorporated into their development process, helping to ensure that their software products were always aligned with customer requirements.

- **Agile and DevOps governance:** Infosys established governance processes for Agile and DevOps to ensure that they were adhering to best practices and standards. This helped to ensure that their software development process was consistent and that they were meeting customer expectations and requirements.

Overall, Infosys' adoption of Agile and DevOps was a holistic effort that incorporated security, Lean principles, continuous feedback, and governance processes. This helped to ensure that they were consistently delivering high-quality software products that met customer needs and expectations