

Experiment 1: Apply the knowledge of SRS and prepare Software Requirement Specification (SRS) document in IEEE format for the project

Learning Objective: Students will able to List various hardware and software requirements, Distinguish between functional and nonfunctional requirements, indicate the order of priority for various requirements, analyze the requirements for feasibility.

Tools: IEEE template and MS Word

Theory:

The srs should contain the following Table of Contents

Introduction.....	2
Purpose.....	2
Intended Audience and Reading Suggestions	2
Product Scope	2
References.....	2
Overall Description.....	2
Product Perspective.....	2
Product Functions	2
User Classes and Characteristics.....	2
Operating Environment.....	3
Design and Implementation Constraints	3
Assumptions and Dependencies.....	3
External Interface Requirements	3
User Interfaces	3
Hardware Interfaces	3
Software Interfaces	3
Communications Interfaces.....	3
System Features	3
System Feature 1	3
Feature 2 (attach further requirements)	3
Other Nonfunctional Requirements.....	4
Performance Requirements	4
Safety Requirements	4
Security Requirements	4
Software Quality Attributes	4
Business Rules	4
Other Requirements	4
Appendix A: Glossary.....	4
Prepared by,	17

1. Introduction

The following section provides an overview of the derived Software Requirements Specification (SRS) for the subject Food Ordering System (FOS). To begin with, the purpose of the document is presented and its intended audience outlined.

Subsequently, the scope of the project specified by the document is given with a particular focus on what the resultant software will do and the relevant benefits associated with it. The nomenclature used throughout the SRS is also offered. To conclude, a complete document overview is provided to facilitate increased reader comprehension and navigation.

1.1 Purpose

The main purpose of an online ordering system is to provide customers for a way to place an order at a restaurant over the internet. So why is this important? The main reason is that it benefits both the customer and the business. It also enables customers to enjoy quality food at their favourite store or restaurant without needing to leave the comfort of their home. Because of its added convenience to customers, food delivery is something you can venture into in the world of business. The project aimed is to developing an order system that can be used in the small medium enterprise food & beverages (F&B) industries which can help the restaurants to simplified their entire daily operational task as well as improve the dining experience of customers. The system will be in 2 platforms which are mobile and computer based. For the mobile based platform will developed to let user to view the menu card information of the restaurant and able to let user place an order via the system. In computer based platform, the system will be able to let staff to update and make changes to their food and beverage imenu information. Next, it also allows staffs to generate report that they wish to generate such as monthly sales report. The most important function is allow staffs to make billing statement for consumer to make their payment after dine-in. At the end of the project, it will improve the restaurants productivity, efficiency, effectiveness and as well as accurateness. Because of this system, it will minimize all the manual work by replacing the traditional order system into a computer system. It will eliminate the manual work such as workers physically deliver food order ticket into the kitchen, manually replace the price tag of the food and manually calculate billing price. These are some main functional module that will exist in the system

1.2 Scope

Its main aim is to simplify and improve the efficiency of the ordering process for both customer and restaurant, minimize manual data entry and ensure data accuracy and security

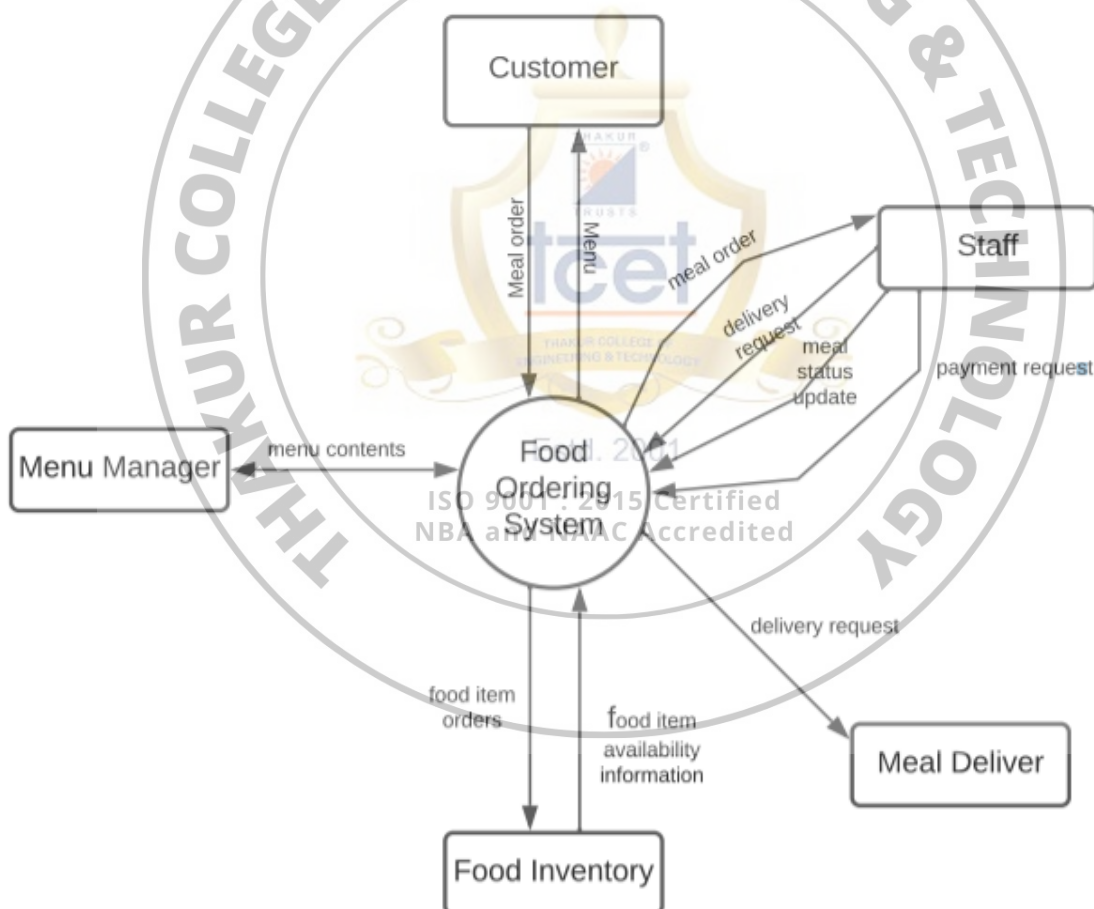
during order placement process. Customers will also be able view product menus and there ingredients and be able to have a visual confirmation that the order was place correctly.

2. Overall Description

2.1 Product Perspective

The Food ordering system is a new system that replaces the current manual processes for ordering and picking up lunches/dinners. The system is expected to evolve over several releases, ultimately connecting to the Internet ordering services for several local restaurants and to credit and debit card authorization services.

2.2 Product Functions



2.3 User Classes and Characteristics

- Customer The user who use our website to order food
- Staff It is restaurant staff who will receive order of customer, prepare meal, package them for delivery and request delivery
- Menu Manager The menu would be managed by menu manager and do changes in menu if some items are unavailable

2.4 Operating Environment

Mobile Environment

For mobile environment a basic **4 core CPU, 16GB storage and 4GB RAM** is at least required.

Software should be or above **Android 7** and **IOS 10.0**. Mobile should have basic networking sensor like **WIFI, GPS** etc. Internet connection should be at least 2g and above.

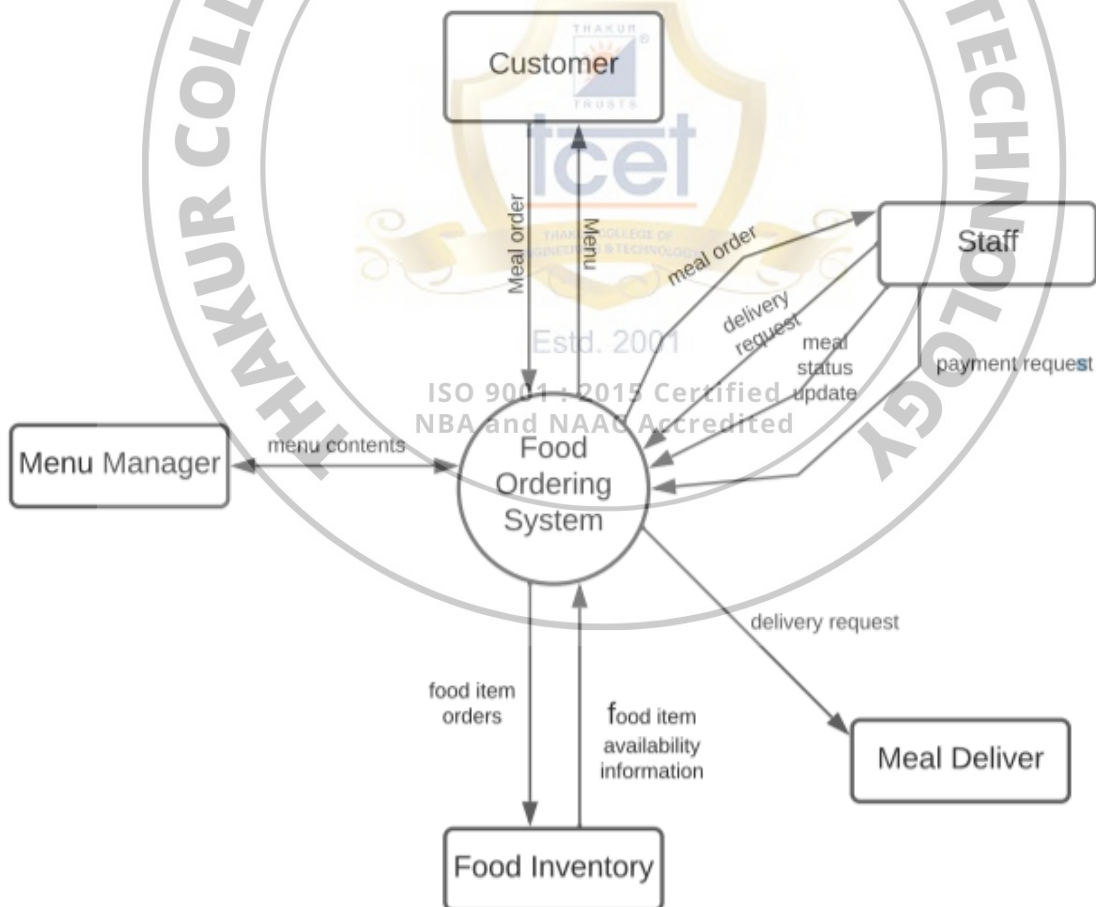
Computer Environment

For computer environment a basic **4 core CPU, 128GB storage and 4GB RAM** is at least required.

Software should be **Windows 7/10/11** and **MacOS Version 10.9: "Mavericks"**

Computer should have basic networking sensor like **WIFI, GPS** etc.

2.5 Design and Implementation Constraints



3. External Interface Requirements

3.1 User interfaces

There are two separate user interfaces used by the software, each related to an interfaced physical hardware device. These two user interfaces are the Computer UI and Mobile UI.

Computer UI

The computer UI is used by restaurants to monitor food orders and table reservation. When user makes an order or reserves a table the software notify the restaurants. The software helps the restaurant to interact with the user like cancelling orders etc. The software also shows all the data related to the restaurant like delivered items, pending orders etc.

Mobile UI

The mobile UI is used by the user to order food or reserve a table of restaurant of their choice. The user interacts with the software to see restaurants nearby order meals reserve table and make payments. The UI will display the list of items in large, easy-to-read text, sorted by algorithms with additional information. Also has the ability to track order, notify user, find best deals etc.

3.2 Hardware interfaces

There are three external hardware devices used by the software its related to a user interface this devices are the computers used by the restaurants devices used by the user and computers used by the company's employees to provide service for user and restaurants which has its own user interface. These devices should have all the basic configuration to run a medium sized application. It should also support all the network connectivity tools like Wi-Fi etc.

3.3 Software interfaces

Mobile Software

The mobile version of the software will be used by the user to order food, reserve table, find restaurants nearby, track their deliveries etc. Mobile software is robust enough to work on low and as well as high end devices and can also run on Android and IOS. Once the user sign-up, the software detects its location and finds nearby restaurants it also finds good deals. User can order or reserve a table of restaurants of their choice within a certain km. The software sends the request to the server the server notifies the

restaurants about the order the restaurants have a separate software on the computer which notifies them about the current order they prepare the order and update the status of the user's meal. Once the order is delivered the user as well as the restaurants get modify all this activity is been stored in the database

Computer Software

The computer version of the software will be used by the restaurants to check their orders, communicate with the user, update status of the meals, create menu of their restaurants etc. The computer software should be robust enough to do a lot of different type of tasks like handling order, handling reservation of a table, providing data of the restaurants, should have option how to solve problem of the restaurants etc. Software also gives the ability of the restaurants to change its menu to update or cancel any order to track whether order is delivered customer reviews etc. All this data is send by API to the servers and all the stored in the database for further use.

3.4 Communications interfaces

The Software will interface with a Local Area Network (LAN) to maintain communication with all its devices. It should use a reliable-type IP protocol such as TCP/IP or reliable-UDP/IP for maximum compatibility and stability. All devices it will interface with should contain standard Ethernet compatible to maintain communication between the server and the the devices.

These devices should have all the basic configuration to run a medium sized application. It should also support all the network connectivity tools like Wi-Fi etc.

4. System Features

4.1 Finding Meals and Restaurants

4.1.1 Description and Priority

This feature helps user to find nearby restaurants and meals. Search is a very high priority feature without search user will have a terrible experience of ordering food or reserving table. Search is a complex feature if it doesn't works properly the user might not find its favourite restaurant or meals the penalty of having a bad results is may be losing customers.

4.1.2 Stimulus/Response Sequences

User click on search button and type it's query. User query is then sent back to the server and related results are matched and then send back to the user via API's.

4.1.3 Functional Requirements

These feature component is user friendly and responsive enough to work on any device and any display size. The component should also handle edge cases like if the user query doesn't match with the result or is there is any typing mistake. What if the server doesn't respond etc. Algorithms should be robust enough to match result with user query.

4.2 Ordering Meals

4.2.1 Description and Priority

This features helps user to order his favourite meals. Ordering features should work seamlessly so that user have a great experience. This feature is of high priority without this feature this software will be useless. If ordering features fail it may lead to many problems with bad user experience etc.

4.2.2 Stimulus/Response Sequences

When user scrolls the menu it shows the list of items available in the restaurants it has a add to cart button where quantity of the items is then push to the cart the user have many different payment option in the cart component. User choses the payment option and confirms the order.

4.2.3 Functional Requirements

These feature component is user friendly and responsive enough to work on any device and any display size the component should also handle edge cases like if the item is not available or is there any certain upper limit of ordering certain items etc. There should be an edge cases of what if payment faile or the server is down. If the payment failed how the user will get refund etc.

4.3 Tracking Meals

4.3.1 Description and Priority

This features help user to know the order's status and when the order will be delivered. It's a medium priority feature because it creates a good user experience with the users know about its order status and time taken to be delivered.

4.3.2 Stimulus/Response Sequences

When the order is made the API sends the data and the request to the server the server sends the request to the restaurant software. The restaurant updates the user order status that request is sent to the server and the server sends back the data and the response to the user. The delivery status is a bit more complex which uses Google maps API to show the accurate time it will take to deliver also the exact current location of the delivery partner.

4.3.3 Functional Requirements

This component should be used friendly and responsive enough to work on any device and any display size a component should also handle its cases like what if the restaurant cancel the order what if the server is down or what if the payment is failed or what if the delivery partner doesn't comes up etc.

4.4 Reserving Tables

4.4.1 Description and Priority

These features help user the book any table of any restaurant of their choice. It is a medium priority feature. Having option to book a table gives a good user experience. But it also bring more complexity to the software.

4.4.2 Stimulus/Response Sequences

Once the user finds restaurant it shows an option to book a table it shows the time slot and the available tables for reservation

The user then choose the number of reservation they want. The request is sent to the server the server notifies the restaurant software.

4.4.3 Functional Requirements

This component is medium complex. The component should be user friendly. The component should update the number of tables left for the reservations. The software in the restaurants also have the ability to update or cancel number of tables. It should also have the edge cases covered like failing of a server, editing reservation etc.

4.5 Updating meals status

4.5.1 Description and Priority

This features is available for the restaurant the restaurant has the ability to update meals ordered by the user. It is a medium priority feature. It has abilities like cancelling user order.

4.5.2 Stimulus/Response Sequences

The restaurant has a button to update the status of the order like preparing food or out for delivery etc.

4.5.3 Functional Requirements

This component should be user friendly and responsive enough on any device. It should handle edge cases.

4.6 Updating Restaurant Menu

4.6.1 Description and Priority

This feature helps restaurant to update their menu it gives them the ability to add or delete certain meals. It is a high priority feature.

4.6.2 Stimulus/Response Sequences

The restaurant has a dashboard in the software that dashboard has a menu option the menu option has update or delete feature which updates their menu and sends the data to the server which is stored.

4.6.3 Functional Requirements

These features give restaurant their ability to decide their menu it's a very important feature because this menu will be shown to the user it should also cover all the edge cases like validation, authentication etc.

4.7 Updating table status

4.7.1 Description and Priority

This feature gives restaurant the ability to update the table status like cancelling the reservation or resetting the reservation etc. It's a high priority feature

4.7.2 Stimulus/Response Sequences

The software has a updating table status component which help them to update or reset the current status of the remaining table to be reserved.

4.7.3 Functional Requirements

This feature gives restaurant the ability to update number of tables to be reserved.

5. Non-functional Requirements

5.1 Performance Requirements

- The system shall accommodate a high number of items and users without any fault.
- Responses to view information shall take no longer than 5 seconds to appear on the screen.

5.2 Safety Requirements

- System use shall not cause any harm to human users.

5.3 Security Requirements

- System will use secured database.
- Normal users can just read information but they cannot edit or modify anything except their personal and some other information.
- System will have different types of users and every user has access constraints.

6. Other Requirements

Not Applicable.

Learning Outcomes: Students should have the ability to

- LO1: List various hardware and software requirements
- LO2: Distinguish between functional and nonfunctional requirements
- LO3: Indicate the order of priority for various requirements
- LO4: Analyze the requirements for feasibility Course

Course Outcomes: Upon completion of the course students will be able to prepare SRS document

Conclusion:

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Estd. 2001

ISO 9001 : 2015 Certified
NBA and NAAC Accredited

Experiment 2: Sketch a DFD (up to 2 levels)

Learning Objective: Students will be able to identify the data flows, processes, source and destination for the project, Analyze and design the DFD upto 2 levels.

Tools: Dia, StarUML

Theory:

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





It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

The following observations about DFDs are essential:

1. All names should be unique. This makes it easier to refer to elements in the DFD.
2. Remember that DFD is not a flow chart. Arrows in a flow chart represent the order of events; arrows in DFD represent flowing data. A DFD does not involve any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represent decision points with multiple existing paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

Symbol	Name	Function
	Data flow	Used to Connect Processes to each other, to sources or Sinks; the arrow head indicates direction of data flow.
	Process	Performs Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Symbols for Data Flow Diagrams

Circle: A circle (bubble) shows a process that transforms data inputs into data outputs.

Data Flow: A curved line shows the flow of data into or out of a process or data store.

Data Store: A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

Levels in Data Flow Diagrams (DFD)

The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0-level DFD

It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Then the system is decomposed and described as a DFD with multiple bubbles. Parts of the system represented by each of these bubbles are then decomposed and documented as more and more detailed DFDs. This process may be repeated at as many levels as necessary until the program at hand is well understood. It is essential to preserve the number of inputs and outputs

between levels, this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs x_1 and x_2 and one output y , then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in fig:

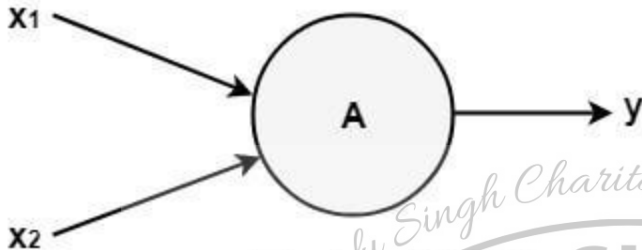


Fig: Level-0 DFD.

1-level DFD

In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into subprocesses.

2-Level DFD

2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.

Learning Outcomes: Students should have the ability to

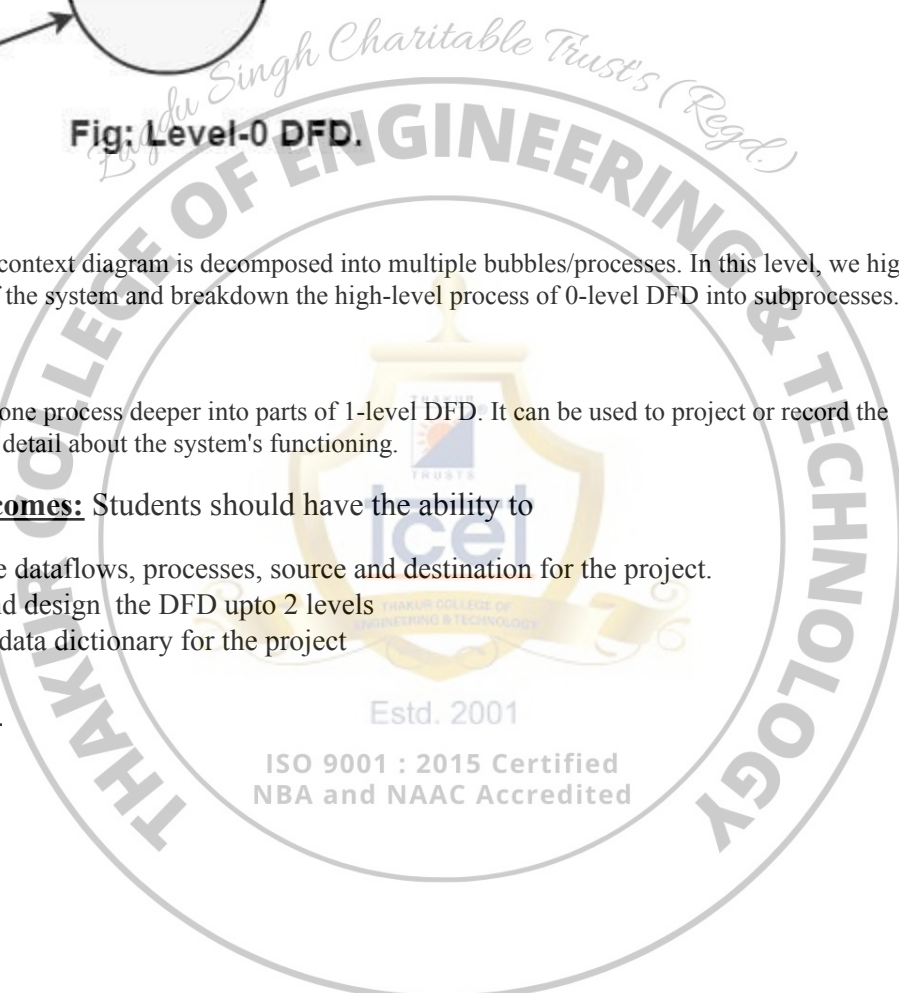
LO1: Identify the dataflows, processes, source and destination for the project.

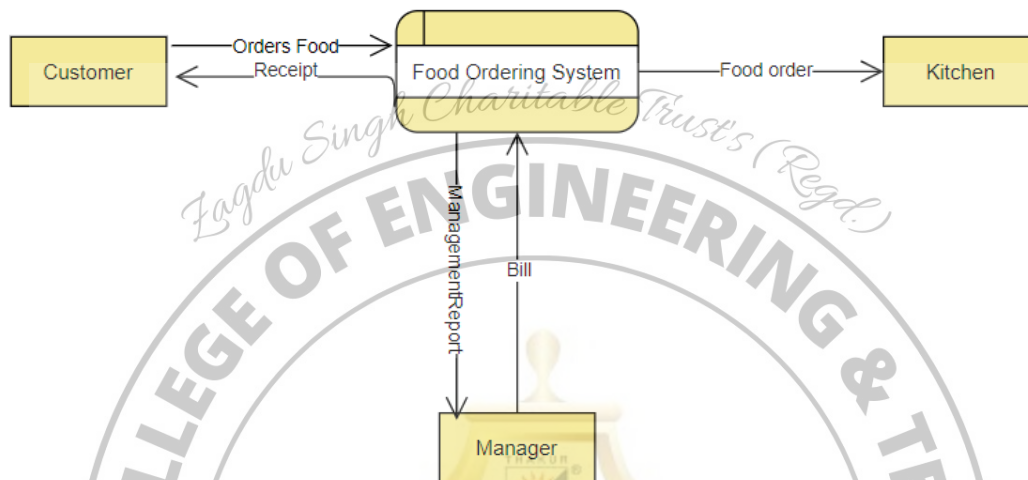
LO2: Analyze and design the DFD upto 2 levels

LO3: Develop a data dictionary for the project

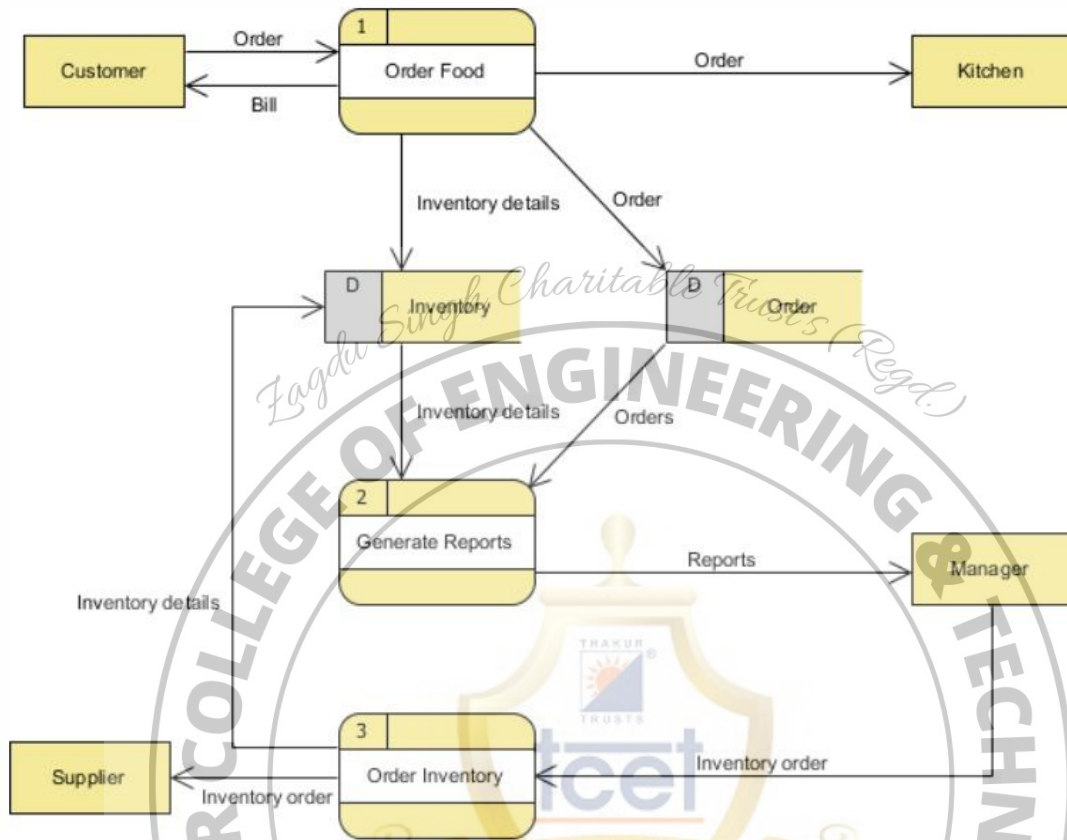
DFD Diagrams:

LEVEL 0:

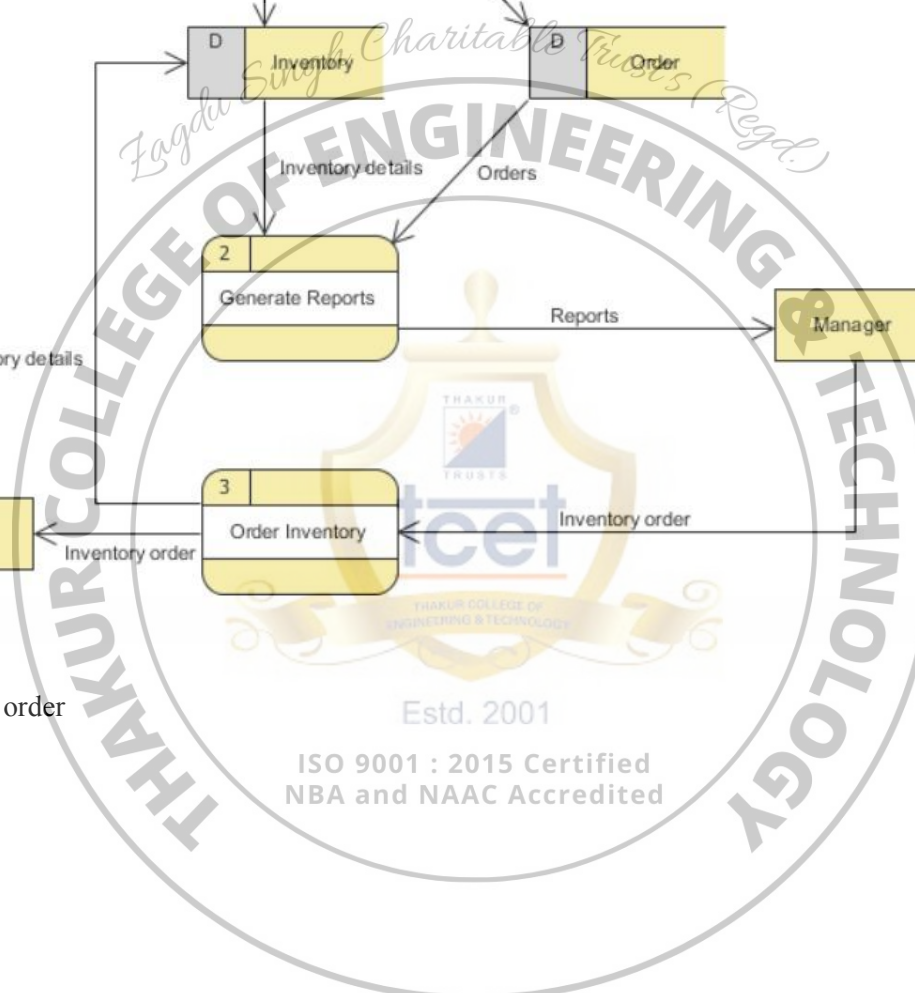


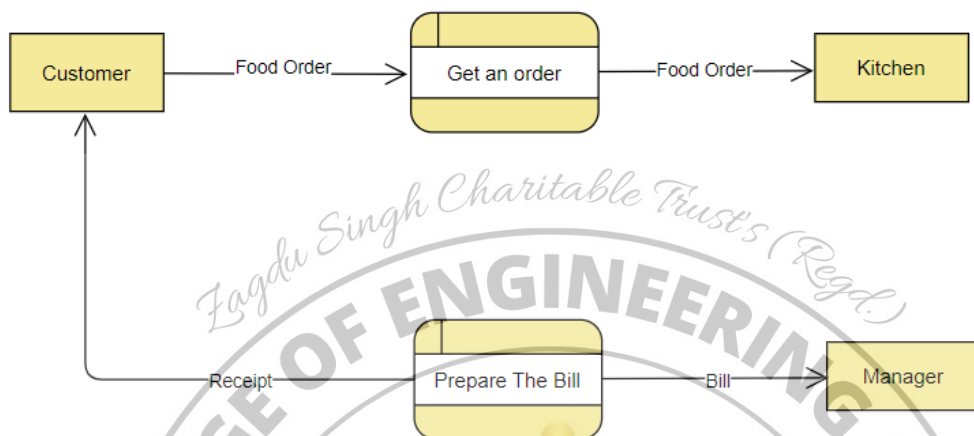


LEVEL 1:



LEVEL 2:
Processing an order





Outcomes: Upon completion of the course students will be able to prepare Draw DFD (upto 2 levels).

Conclusion: Hence, we successfully understood All about DFD and drew DFD diagram for food ordering system upto level 2.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment 3: Sketch UML Use case Diagram for the project.

Learning Objective: To implement UML use-case diagram for the project.

Tools: MS Word, draw.io

Theory:

Use case diagrams

Use case diagrams belong to the category of behavioral diagram of UML diagrams. Use case diagrams aim to present a graphical overview of the functionality provided by the system. It consists of a set of actions (referred to as use cases) that the concerned system can perform one or more actors, and dependencies among them.

Actor

An actor can be defined as an object or set of objects, external to the system, which interacts with the system to get some meaningful work done. Actors could be human, devices, or even other systems.

For example, consider the case where a customer *withdraws cash* from an ATM. Here, customer is a human actor.

Actors can be classified as below:

- **Primary actor:** They are principal users of the system, who fulfill their goal by availing some service from the system. For example, a customer uses an ATM to withdraw cash when he needs it. A customer is the primary actor here.
- **Supporting actor:** They render some kind of service to the system. "Bank representatives", who replenishes the stock of cash, is such an example. It may be noted that replenishing stock of cash in an ATM is not the prime functionality of an ATM.

In a use case diagram primary actors are usually drawn on the top left side of the diagram.

Use Case

A use case is simply a functionality provided by a system.

Continuing with the example of the ATM, *withdraw cash* is a functionality that the ATM provides. Therefore, this is a use case. Other possible use cases include, *check balance*, *change PIN*, and so on.

Use cases include both successful and unsuccessful scenarios of user interactions with the system. For example, authentication of a customer by the ATM would fail if he enters wrong PIN. In such case, an error message is displayed on the screen of the ATM.

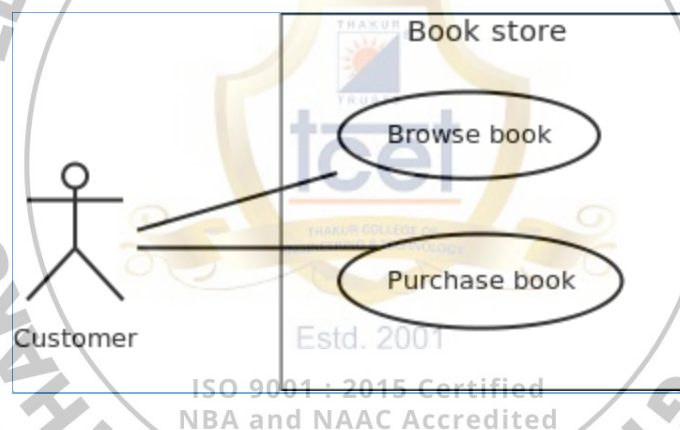
Subject

Subject is simply the system under consideration. Use cases apply to a subject. For example, an ATM is a subject, having multiple use cases, and multiple actors interact with it. However, one should be careful of external systems interacting with the subject as actors.

Graphical Representation

An actor is represented by a stick figure and name of the actor is written below it. A use case is depicted by an ellipse and name of the use case is written inside it. The subject is shown by drawing a rectangle. Label for the system could be put inside it. Use cases are drawn inside the rectangle, and actors are drawn outside the rectangle, as shown in figure - 01.

Figure - 01: A use case diagram for a book store



Association between Actors and Use Cases

A use case is triggered by an actor. Actors and use cases are connected through binary associations indicating that the two communicates through message passing.

An actor must be associated with at least one use case. Similarly, a given use case must be associated with at least one actor. Association among the actors is usually not shown. However, one can depict the class hierarchy among actors.

Use Case Relationships

Three types of relationships exist among use cases:

- Include relationship
- Extend relationship
- Use case generalization

Include Relationship

Include relationships are used to depict common behavior that are shared by multiple use cases. This could be considered analogous to writing functions in a program in order to avoid repetition of writing the same code. Such a function would be called from different points within the program.

Example

For example, consider an email application. A user can send a new mail, reply to an email he has received, or forward an email. However, in each of these three cases, the user must be logged in to perform those actions. Thus, we could have a *login* use case, which is included by *compose mail*, *reply*, and *forward email* use cases. The relationship is shown in figure - 02.



Figure - 02: Include relationship between use cases

Notation

Include relationship is depicted by a dashed arrow with a «include» stereotype from the including use case to the included use case.

Extend Relationship

Use case extensions are used to depict any variation to an existing use case. They are used to specify the changes required when any assumption made by the existing use case becomes false.

Example

Let's consider an online bookstore. The system allows an authenticated user to buy selected book(s). While the order is being placed, the system also allows specifying any special shipping instructions, for example, call the customer before delivery. This *Shipping Instructions* step is optional, and not a part of the main *Place Order* use case. Figure - 03 depicts such relationship.

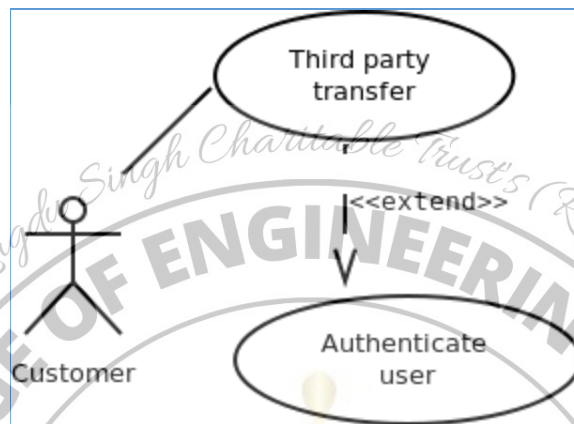


Figure - 03: Extend relationship between use cases

Notation

Extend relationship is depicted by a dashed arrow with a «extend» stereotype from the extending use case to the extended use case.

Generalization Relationship

Generalization relationship is used to represent the inheritance between use cases. A derived use case specializes some functionality it has already inherited from the base use case.

Example

To illustrate this, consider a graphical application that allows users to draw polygons. We could have a use case *draw polygon*. Now, rectangle is a particular instance of polygon having four sides at right angles to each other. So, the use case *draw rectangle* inherits the properties of the use case *draw polygon* and overrides its drawing method. This is an example of generalization relationship. Similarly, a generalization relationship exists between *draw rectangle* and *draw square* use cases. The relationship has been illustrated in figure - 04.

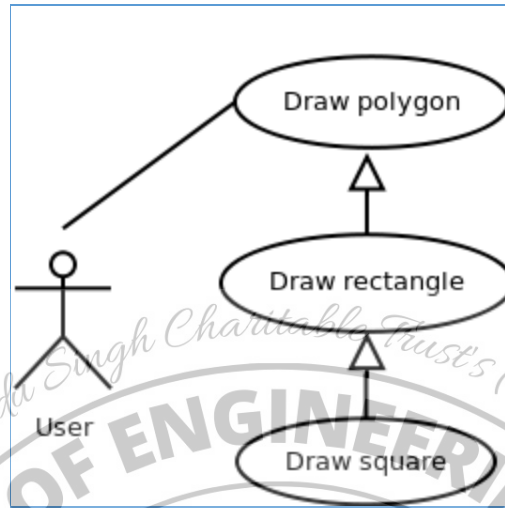


Figure - 04: Generalization relationship among use cases

Notation

Generalization relationship is depicted by a solid arrow from the specialized (derived) use case to the more generalized (base) use case.

Identifying Actors

Given a problem statement, the actors could be identified by asking the following questions :

- Who gets most of the benefits from the system? (The answer would lead to the identification of the primary actor)
- Who keeps the system working? (This will help to identify a list of potential users)
- What other software / hardware does the system interact with?
- Any interface (interaction) between the concerned system and any other system?

Identifying Use cases

Once the primary and secondary actors have been identified, we have to find out their goals i.e. what the functionality they can obtain from the system is. Any use case name should start with a verb like, "Check balance".

Guidelines for drawing Use Case diagrams

Following general guidelines could be kept in mind while trying to draw a use case diagram :

- Determine the system boundary
- Ensure that individual actors have well-defined purpose
- Use cases identified should let some meaningful work done by the actors
- Associate the actors and use cases -- there shouldn't be any actor or use case floating without any connection
- Use include relationship to encapsulate common behavior among use cases , if any

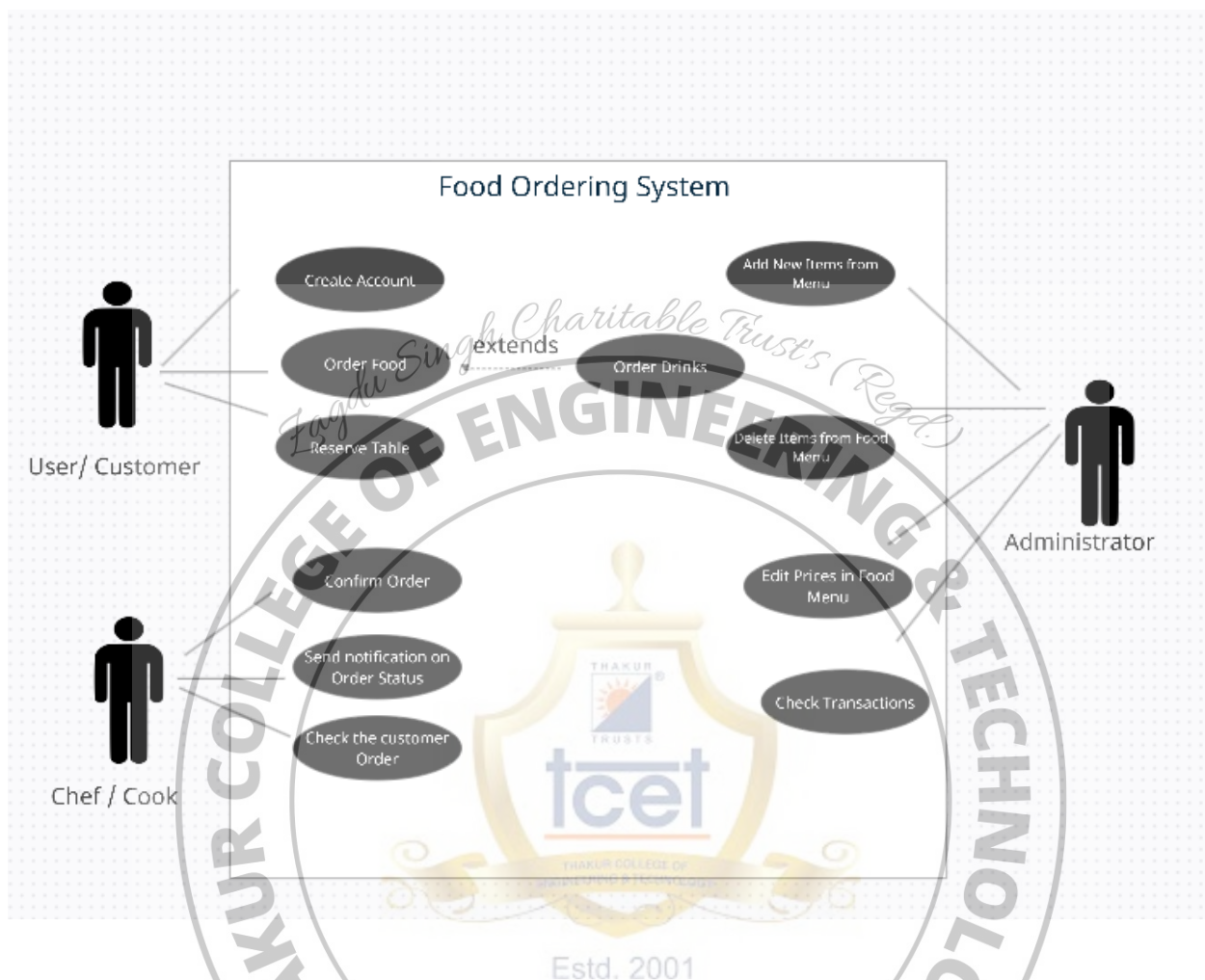
Procedure:

A Use Case model can be developed by following the steps below.

1. Identify the Actors (role of users) of the system.
2. For each category of users, identify all roles played by the users relevant to the system.
3. Identify what are the users required the system to be performed to achieve these goals.
4. Create use cases for every goal.
5. Structure the use cases.
6. Prioritize, review, estimate and validate the users.

Result and Discussion:





Learning Outcomes: The student should have the ability to:

LO 1: Identify the importance of use-case diagrams.

LO 2: Draw use-case diagrams for a given scenario.

Course Outcomes: Upon completion of the course students will be able to understand and demonstrate use-case diagrams.

Conclusion: Thus, students have understood and successfully drawn use-case diagrams.

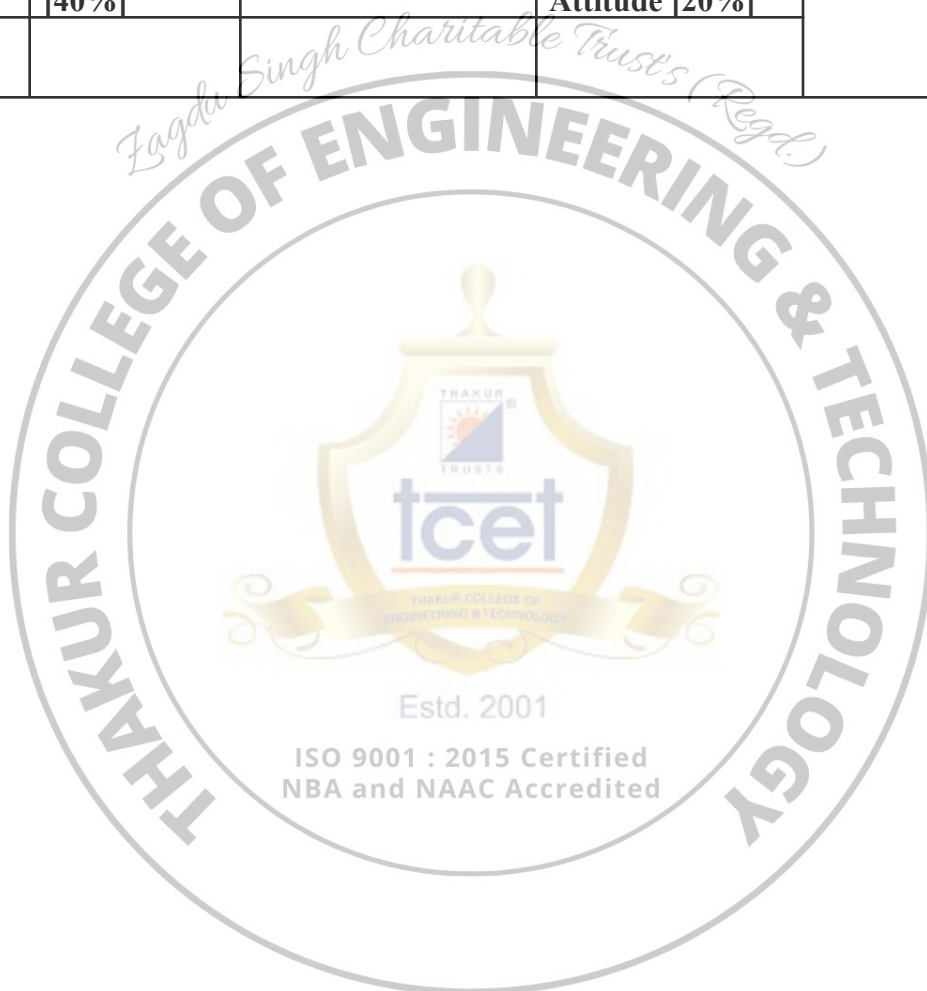
Viva Questions:

1. What is use-case diagrams used for?
2. Enumerate on the type of relationships that exists for use-case diagram.



For Faculty Use:

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				



Experiment 04- Sketch a Class Diagram for the project

Learning Objective: To implement UML class diagram for the project.

Tools: MS Word, draw.io

Theory:

Class Diagrams:

Classes are the structural units in object oriented system design approach, so it is essential to know all the relationships that exist between the classes, in a system. All objects in a system are also interacting to each other by means of passing messages from one object to another.

Elements in class diagram

Class diagram contains the system classes with its data members, operations and relationships between classes.

Class

A set of objects containing similar data members and member functions is described by a class. In UML syntax, class is identified by solid outline rectangle with three compartments which contain

- **Class name**

A class is uniquely identified in a system by its name. A textual string [2] is taken as class name. It lies in the first compartment in class rectangle.

- **Attributes**

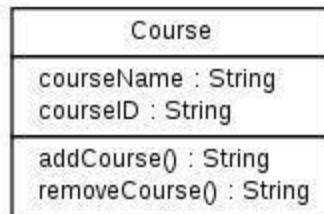
Property shared by all instances of a class. It lies in the second compartment in class rectangle.

- **Operations**

An execution of an action can be performed for any object of a class. It lies in the last compartment in class rectangle.

Example

To build a structural model for an Educational Organization, 'Course' can be treated as a class which contains attributes 'courseName' & 'courseID' with the operations 'addCourse()' & 'removeCourse()' allowed to be performed for any object to that class.



- **Generalization/Specialization**

It describes how one class is derived from another class. Derived class inherits the properties of its parent class.

Example

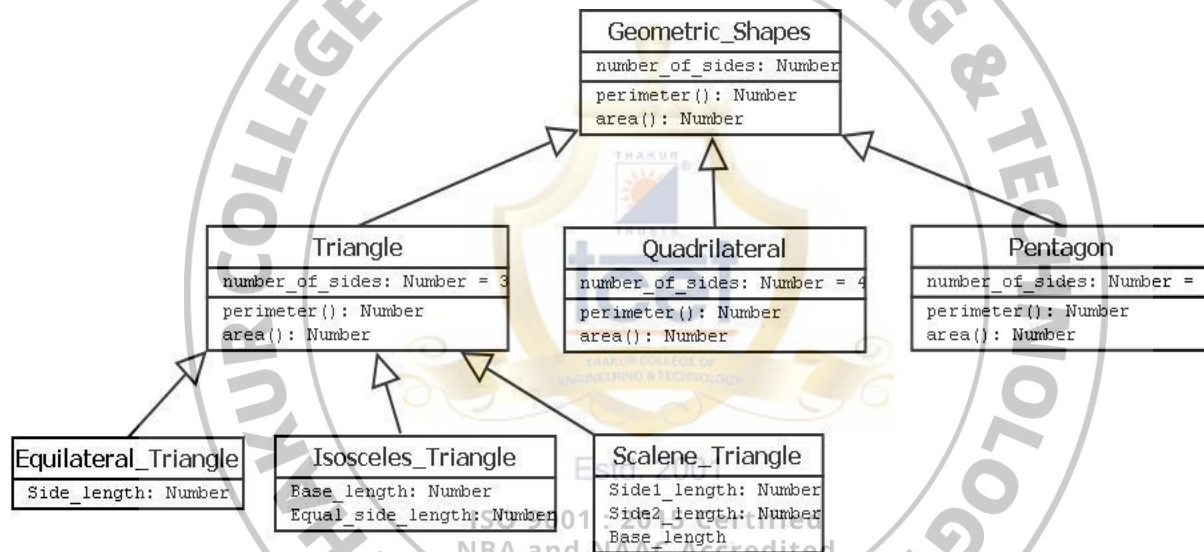


Figure-02:

Geometric_Shapes is the class that describes how many sides a particular shape has. **Triangle**, **Quadrilateral** and **Pentagon** are the classes that inherit the property of the **Geometric_Shapes** class. So the relations among these classes are generalization. Now **Equilateral_Triangle**, **Isosceles_Triangle** and **Scalene_Triangle**, all these three classes inherit the properties of **Triangle** class as each one of them has three sides. So, these are specialization of **Triangle** class.

Relationships

Existing relationships in a system describe legitimate connections between the classes in that system.

- **Association**

It is an instance level relationship that allows exchanging messages among the objects of both ends of association. A simple straight line connecting two class boxes represent an association. We can give a name to association and also at the both end we may indicate role names and multiplicity of the adjacent classes. Association may be uni-directional.

Example

In structure model for a system of an organization an employee (instance of 'Employee' class) is always assigned to a particular department (instance of 'Department' class) and the association can be shown by a line connecting the respective classes.



Figure-03:

- **Aggregation**

It is a special form of association which describes a part-whole relationship between a pair of classes. It means, in a relationship, when a class holds some instances of related class, then that relationship can be designed as an aggregation.

Example

For a supermarket in a city, each branch runs some of the departments they have. So, the relation among the classes 'Branch' and 'Department' can be designed as aggregation. In UML, it can be shown as in the fig. below.



Figure-04:

- **Composition**

It is a strong form of aggregation which describes that whole completely owns its part. Life cycle of the part depends on the whole.

Example

Let consider a shopping mall has several branches in different locations in a city. The existence of branches completely depends on the shopping mall as if it is not exist any branch of it will no longer exists in the city. This relation can be described as composition and can be shown as below



Figure-05:

- Multiplicity**

It describes how many numbers of instances of one class is related to the number of instances of another class in an association.

Notation for different types of multiplicity:

Single instance	1
Zero or one instance	0..1
Zero or more instance	0..*
One or more instance	1..*
Particular range(two to six)	2..6

Figure-06:

Example

One vehicle may have two or more wheels

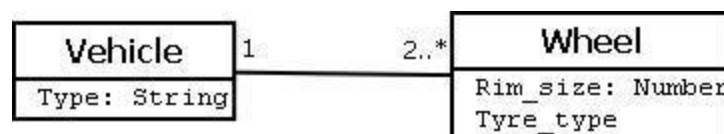


Figure-07:

Procedure:

When required to describe the static view of a system or its functionalities, we would be required to draw a class diagram. Here are the steps you need to follow to create a class diagram.

Step 1: Identify the class names

The first step is to identify the primary objects of the system.

Step 2: Distinguish relationships

Next step is to determine how each of the classes or objects are related to one another. Look out for commonalities and abstractions among them; this will help you when grouping them when drawing the class diagram.

Step 3: Create the Structure

First, add the class names and link them with the appropriate connectors. You can add attributes and functions/ methods/ operations later.

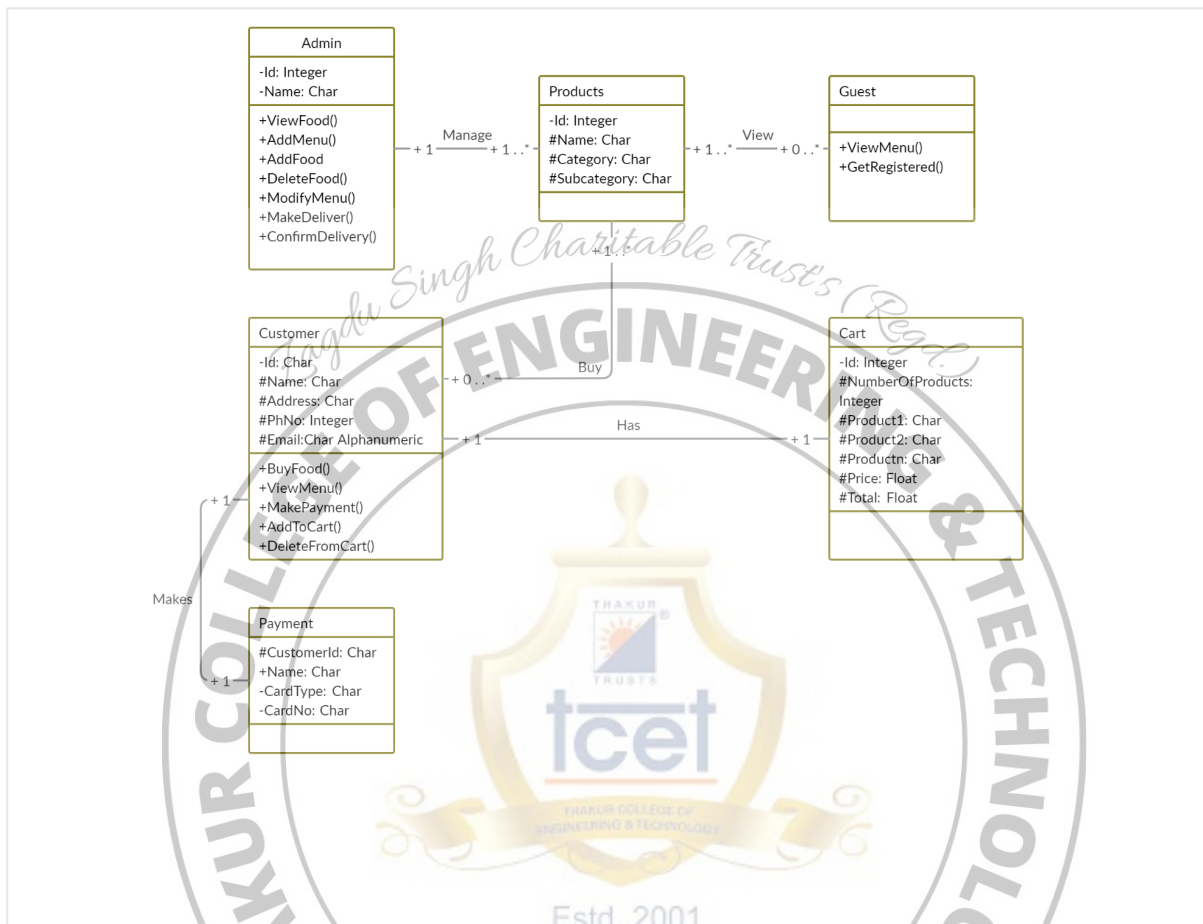
Result and Discussion:

Q.1) What is a class diagram? Draw at least two class diagrams for your projects.

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Purpose of Class Diagrams:

1. Shows static structure of classifiers in a system
2. Diagram provides a basic notation for other structure diagrams prescribed by UML
3. Helpful for developers and other team members too
4. Business Analysts can use class diagrams to model systems from a business perspective



Learning Outcomes: The student should have the ability to:

LO 1: Identify the importance of class diagrams.

LO 2: Draw class diagrams for a given scenario.

Course Outcomes: Upon completion of the course students will be able to understand and demonstrate class diagrams.

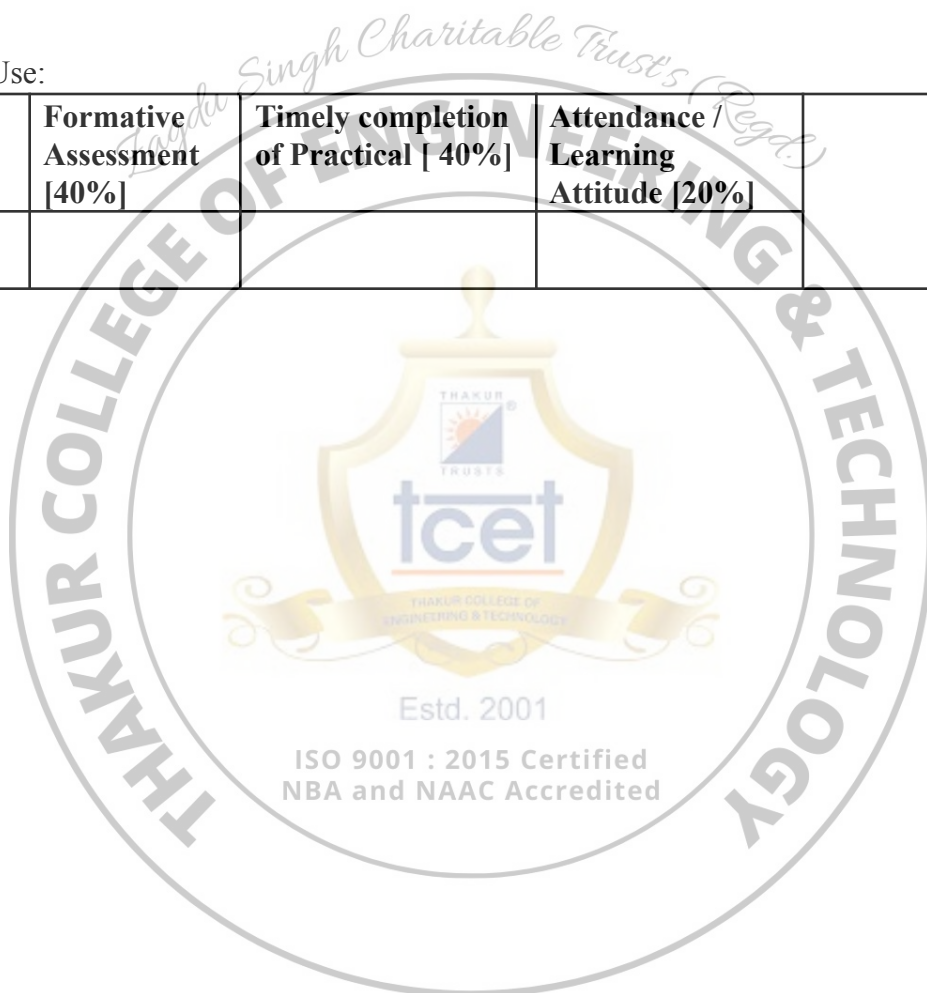
Conclusion: Thus, students have understood and successfully drawn class diagrams.

Viva Questions:

1. What are class diagrams used for?
2. Enumerate various relationships in a class diagram.

For Faculty Use:

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				



Experiment 05- Sketch Activity diagram for the project.

Learning Objective: To implement a dynamic view of a system using Activity diagrams.

Tools: MS Word, draw.io

Theory:

Activity Diagrams

Activity diagrams fall under the category of behavioral diagrams in Unified Modeling Language. It is a high level diagram used to visually represent the flow of control in a system. It has similarities with traditional flow charts. However, it is more powerful than a simple flow chart since it can represent various other concepts like concurrent activities, their joining, and so on.

Activity diagrams, however, cannot depict the message passing among related objects. As such, it can't be directly translated into code. These kinds of diagrams are suitable for confirming the logic to be implemented with the business users. These diagrams are typically used when the business logic is complex. In simple scenarios it can be avoided entirely.

Components of an Activity Diagram

Below we describe the building blocks of an activity diagram.

Activity

An activity denotes a particular action taken in the logical flow of control. This could simply be invocation of a mathematical function, alter an object's properties and so on. An activity is represented with a rounded rectangle, as shown in table-01. A label inside the rectangle identifies the corresponding activity.

There are two special types of activity nodes: initial and final. They are represented with a filled circle, and a filled in circle with a border respectively (table-01). Initial node represents the starting point of a flow in an activity diagram. There could be multiple initial nodes, which mean that invoking that particular activity diagram would initiate multiple flows.

A final node represents the end point of all activities. Like an initial node, there could be multiple final nodes. Any transition reaching a final node would stop all activities.

Flow

A flow (also termed as edge or transition) is represented with a directed arrow. This is used to depict transfer of control from one activity to another, or to other types of components, as we will see below. A

flow is often accompanied with a label, called the guard condition, indicating the necessary condition for the transition to happen. The syntax to depict it is [guard condition].

Decision

A decision node, represented with a diamond, is a point where a single flow enters and two or more flows leave. The control flow can follow only one of the outgoing paths. The outgoing edges often have guard conditions indicating true-false or if-then-else conditions. However, they can be omitted in obvious cases. The input edge could also have guard conditions. Alternately, a note can be attached to the decision node indicating the condition to be tested.

Merge

This is represented with a diamond shape, with two or more flows entering, and a single flow leaving out. A merge node represents the point where at least a single control should reach before further processing could continue.

Fork

Fork is a point where parallel activities begin. For example, when a student has been registered with a college, he can in parallel apply for student ID card and library card. A fork is graphically depicted with a black bar, with a single flow entering and multiple flows leaving out.

Join

A join is depicted with a black bar, with multiple input flows, but a single output flow. Physically it represents the synchronization of all concurrent activities. Unlike a merge, in case of a join all of the incoming controls **must be completed** before any further progress could be made. For example, a sales order is closed only when the customer has received the product, **and** the sales company has received its payment.

Note

UML allows attaching a note to different components of a diagram to present some textual information. The information could simply be a comment or may be some constraint. A note can be attached to a decision point, for example, to indicate the branching criteria.

Partition

Different components of an activity diagram can be logically grouped into different areas, called partitions or swim lanes. They often correspond to different units of an organization or different actors. The drawing area can be partitioned into multiple compartments using vertical (or horizontal) parallel

lines. Partitions in an activity diagram are not mandatory. The following table shows commonly used components with a typical activity diagram.





Component	Graphical Notation
Activity	An Activity
Flow	[A Flow]
Decision	
Merge	
Fork	
Join	
Note	A simple note

Table-01: Typical components used in an activity diagram
ISO 9001 : 2015 Certified
NBA and NAAC Accredited

A Simple Example

Figure-04 shows a simple activity diagram with two activities. The figure depicts two stages of a form submission. At first a form is filled up with relevant and correct information. Once it is verified that there is no error in the form, it is then submitted. The two other symbols shown in the figure are the initial node (dark filled circle), and final node (outer hollow circle with inner filled circle). It may be noted that there could be zero or more final node(s) in an activity diagram.

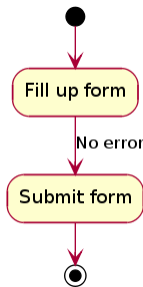


Figure-04: A simple activity diagram.

Procedure:

Guidelines for drawing State chart Diagrams

Following steps could be followed, to draw a state chart diagram:

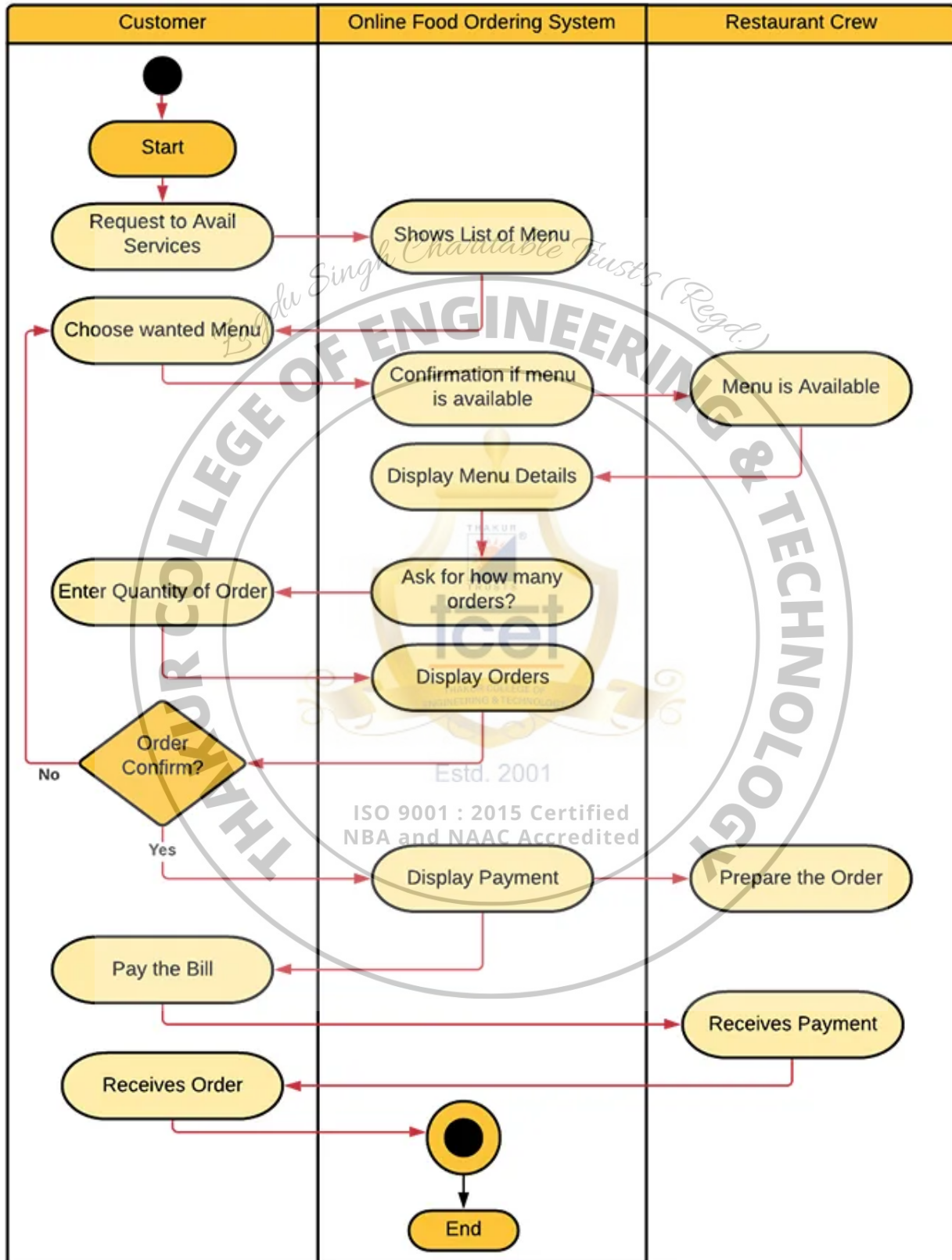
- For the system to developed, identify the distinct states that it passes through
- Identify the events (and any precondition) that cause the state transitions. Often these would be the methods of a class as identified in a class diagram.
- Identify what activities are performed while the system remains in a given state

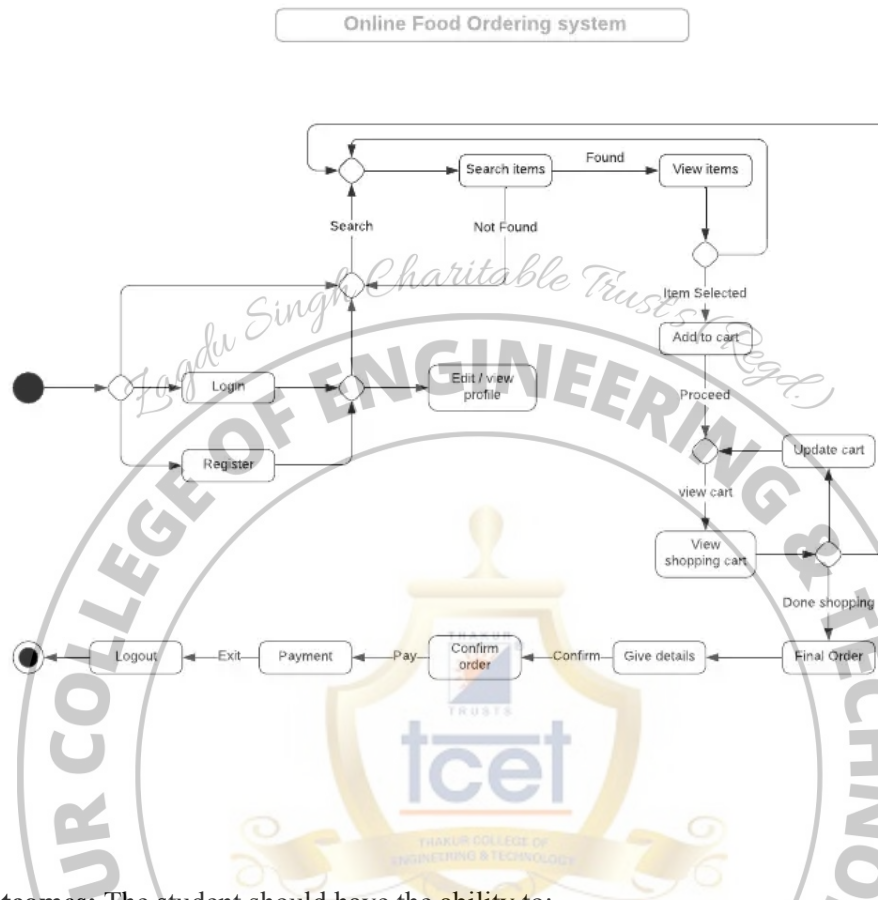
Result and Discussion:

Q.1) What is a dynamic view of a system? Draw at least one state diagram and one activity diagram for your mini project.

Dynamic UML diagrams describe the behavior of systems. A dynamic diagram describes the operations, actions, and changes that occur in a system over time. A static diagram, on the other hand, describes the characteristics of a system or part of a system.

It emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams, and state machine diagrams





Learning Outcomes: The student should have the ability to:

LO 1: Identify the importance of a state diagram.

LO 2: Draw activity diagrams for a given scenario.

Course Outcomes: Upon completion of the course students will be able to understand and demonstrate state and activity diagrams.

Conclusion: Thus, students have understood and successfully drawn state and activity diagrams.

Viva Questions:

1. What is a state diagram used for?
2. Enumerate the steps to draw an activity diagram.



TCET

DEPARTMENT OF COMPUTER ENGINEERING (COMP)

(Accredited by NBA for 3 years, 3rd Cycle Accreditation w.e.f. 1st July 2019)

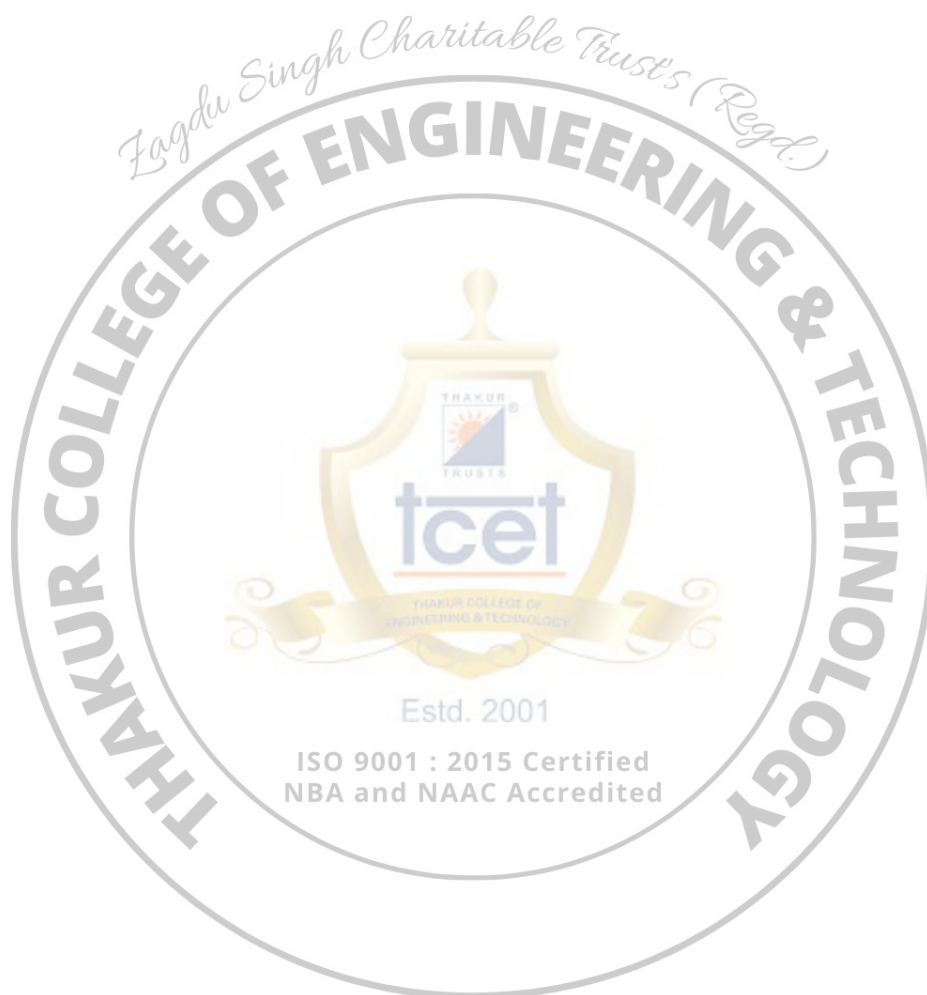
Choice Based Credit Grading Scheme (CBCGS)

Under TCET Autonomy



For Faculty Use:

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				



Experiment 7: Sketch Sequence and Collaboration diagram for the project

Learning Objective: Students will able to draw Sequence and Collaboration diagram for the project

Tools: Dia, StarUML

Theory:

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

Sequence Diagram representation

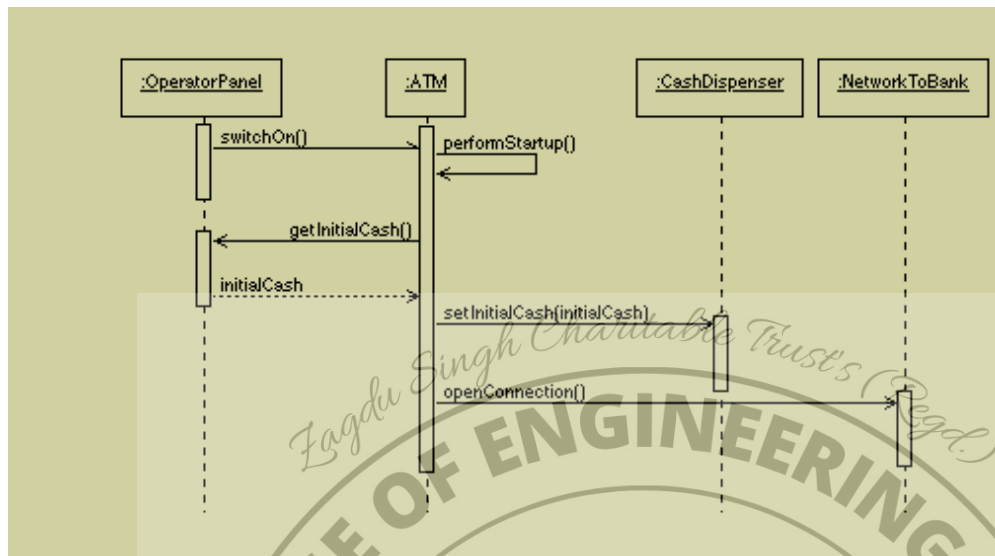
Call Message: A message defines a particular communication between Lifelines of an Interaction.

Destroy Message: Destroy message is a kind of message that represents the request of destroying the lifecycle of the target lifeline.

LifeLine: A lifeline represents an individual participant in the Interaction.

Recursive Message: Recursive message is a kind of message that represents the invocation of a message of the same lifeline. Its target points to an activation on top of the activation where the message was invoked from.

Sequence Diagram: Example for ATM System startup



Collaboration diagram for ATM System startup



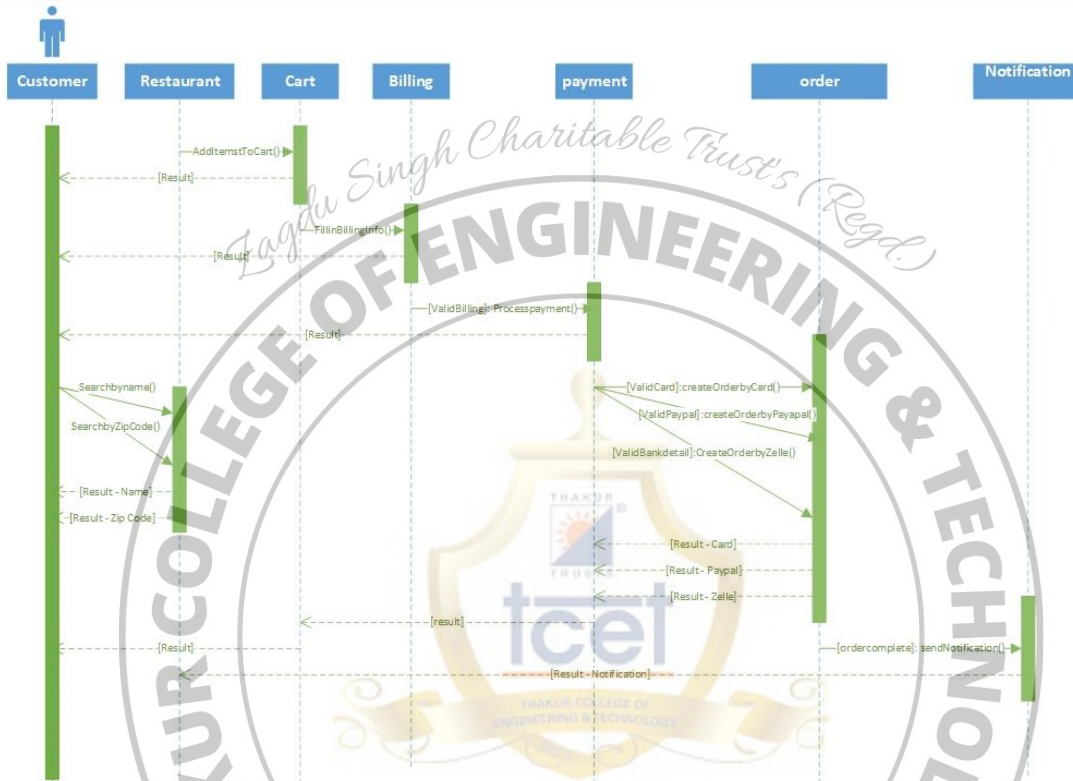
It is clear that sequence charts have a number of very powerful advantages. They clearly depict the sequence of events, show when objects are created and destroyed, are excellent at depicting concurrent operations, and are invaluable for hunting down race conditions. However, with all their advantages, they are not perfect tools. They take up a lot of space, and do not present the interrelationships between the collaborating objects very well.

A collaboration diagram, also known as a communication diagram, depicts the relationships and interactions among software objects in the UML diagrams.

Collaboration diagrams are best suited to the portrayal of simple interactions among relatively small numbers of objects.

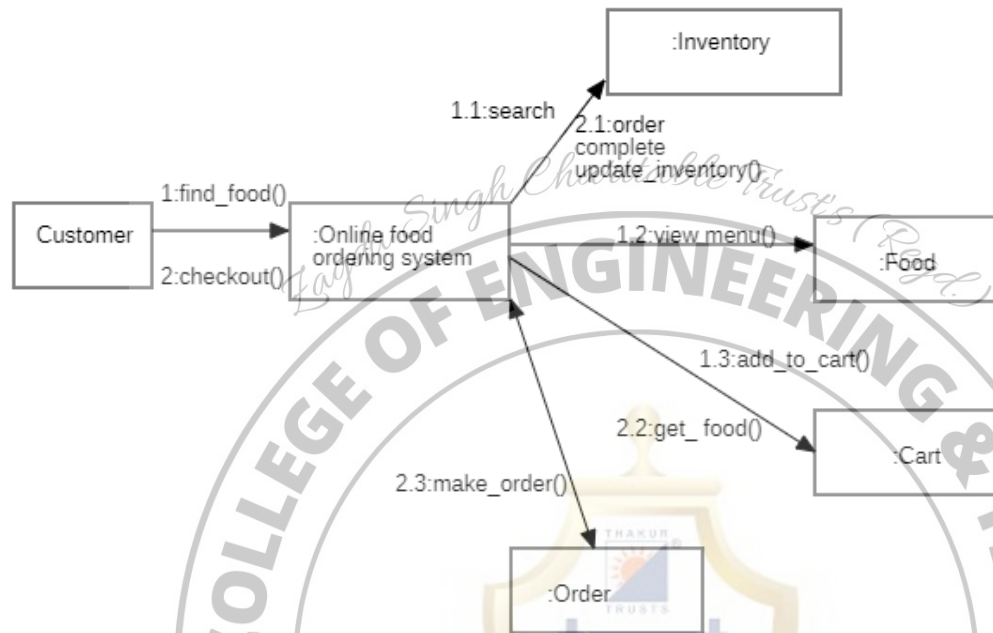
Collaboration diagrams are used to visualize the structural organization of objects and their interactions. Sequence diagrams, focus on the order of messages that flow between objects.

Sequence diagram:



Collaboration diagram:

ISO 9001 : 2015 Certified
NBA and NAAC Accredited



Learning Outcomes: Students should have the ability to

- LO1: Identify the classes and objects.
- LO2: Identify the interactions between the objects
- LO3: Develop a sequence diagram for different scenarios
- LO4: generate the collaboration diagram

Outcomes: Upon completion of the course students will be able to draw the sequence and collaboration diagram for the project.

Conclusion:

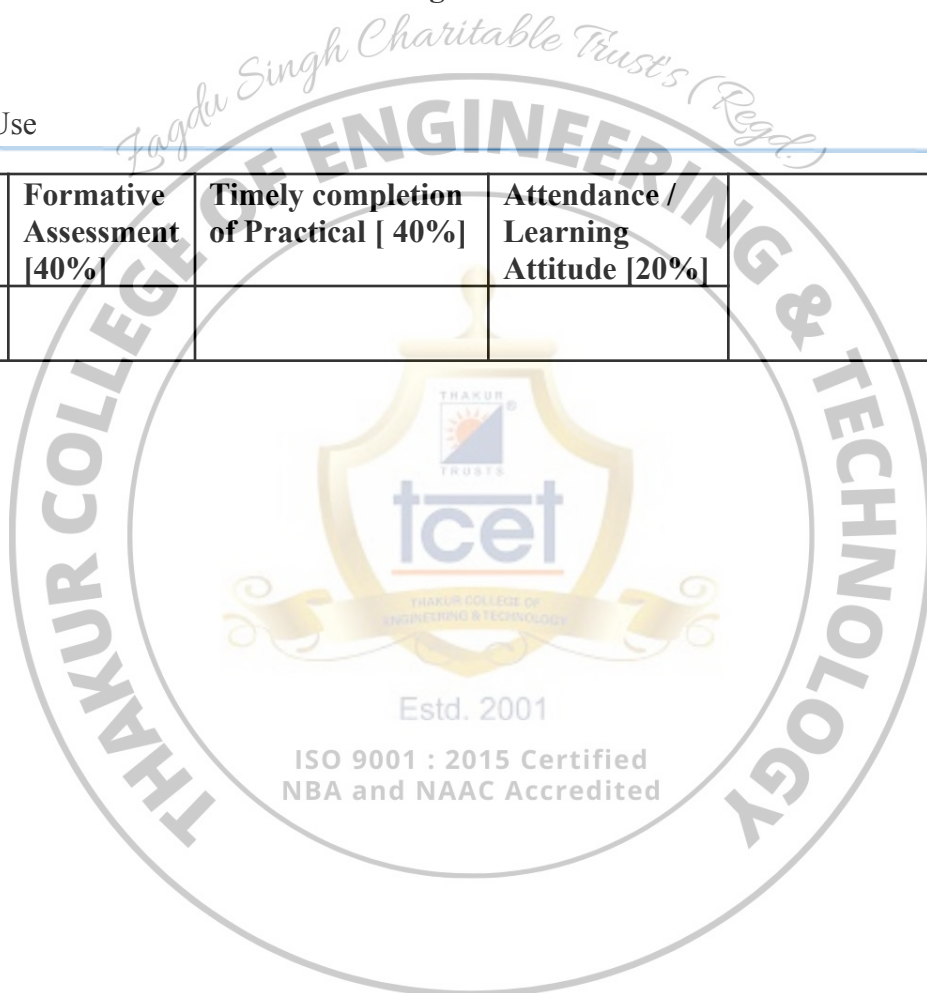
Thus, students have understood and successfully made a sequence and collaboration diagram for the project.

Viva Questions:

1. What is a sequence diagram
2. Difference between sequence and collaboration diagram?
3. What are entities in a sequence diagram?
4. Explain its relation with the class diagram?

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				



Experiment 7: Use project management tool to prepare schedule for the project.

Learning Objective: Students will be able to List the various activities in the project, analyze the various activities for schedule, estimate the time for each activity and develop a Gantt Chart for the activities.

Tools: Gantt Chart using any Project Management tool

Theory:

The main aim of PROJECT SCHEDULING AND TRACKING is to get the project completed on time. Program evaluation and review technique (PERT) and Gantt chart are two project scheduling methods that can be applied to software development

Split the project into tasks and estimate time and resources required to complete each task. Organize tasks concurrently to make optimal use of the workforce. Minimize task dependencies to avoid delays caused by one task waiting for another to complete

Gantt chart:

A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities (tasks or events) displayed against time. On the left of the chart is a list of the activities and along the top is a suitable time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity. This allows you to see at a glance:

- What the various activities are
- When each activity begins and ends
- How long each activity is scheduled to last
- Where activities overlap with other activities, and by how much
- The start and end date of the whole project

Task Name	Q1 2009				Q2 2009			Q3 2009	
	Dec '08	Jan '09	Feb '09	Mar '09	Apr '09	May '09	Jun '09	Jul '09	Aug
Planning									
Research									
Design									
Implementation									
Follow up									

Output:



Learning Outcomes: Students should have the ability to

- LO1: List the various activities in the project.
- LO2: Analyze the various activities for schedule.
- LO3: Estimate the time for each activity.
- LO4: Develop a Gantt Chart for the activities.

Outcomes: Upon completion of the course students will be able to use project management tools to prepare schedules for the project.

Conclusion: Thus, students have understood and successfully prepared schedule for the project.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
------------------------------	-----------------------------------	--	---	--



TCET

DEPARTMENT OF COMPUTER ENGINEERING (COMP)

(Accredited by NBA for 3 years, 3rd Cycle Accreditation w.e.f. 1st July 2019)

Choice Based Credit Grading Scheme (CBCGS)

Under TCET Autonomy



Marks Obtained				
-------------------	--	--	--	--

