Terna Engineering College

Computer Engineering Department

Program: Sem VI

Course: Software Engineering Lab

LAB Manual

PART A

(PART A: TO BE REFERRED BY STUDENTS)

Experiment No.04

A.1 Aim:

Identify scenarios & develop a UML Use case for the selected mini-project.

A.2 Prerequisite:

1. Preliminary requirements
2. Knowledge of different process models

A.3 Outcome:

After successful completion of this experiment, students will be able to:

- ✔ Refine requirements from a set of preliminary requirements and able to identify actors and possible use cases.
- ✔ Able to model requirements using UML.

A.4 Theory:

Use case diagrams

- ➔ Use case diagrams belong to the category of behavioural 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, the customer is a human actor.
- ➔ Actors can be classified as below:
    - **Primary actor**: They are principal users of the system, who fulfil 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 a 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 the 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 the wrong PIN. In such a case, an error message is displayed on the screen of the ATM.

Subject

- ➔ The 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

- ➔ A stick figure represents an actor, and the name of the actor is written below it. An ellipse depicts a use case, and the name of the use case is written inside it. The subject is shown by drawing a rectangle. Labels 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 fig.

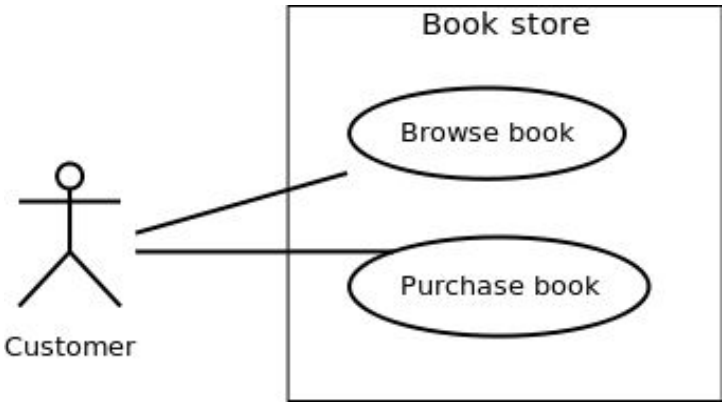


Figure - 01: A use case diagram for a bookstore

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 communicate 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:
    1. Include relationship
    2. Extend the relationship
    3. Use case generalization

Include Relationship

- ➔ Include relationships are used to depict common behaviour that is shared by multiple use cases. This could be considered analogous to writing functions in a program 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 *composing mail*, *reply*, and *forward email* use cases. The relationship is shown in figure - 02.

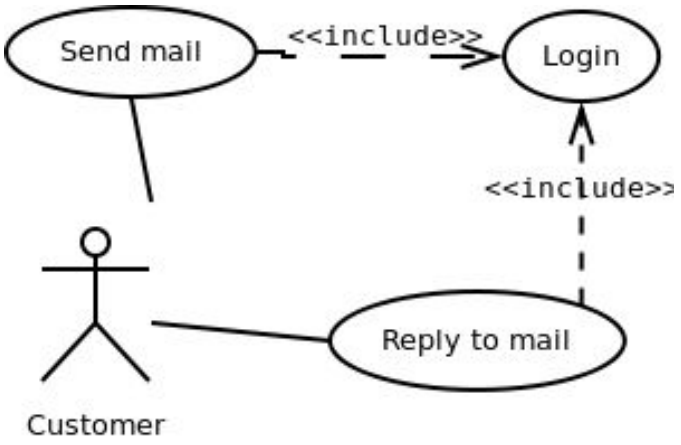


Figure - 02: Include the relationship between use cases

Notation

➔ Include relationship is depicted by a dashed arrow with an «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 the selected book(s). While the order is being placed, the system also allows you to specify 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 a relationship.

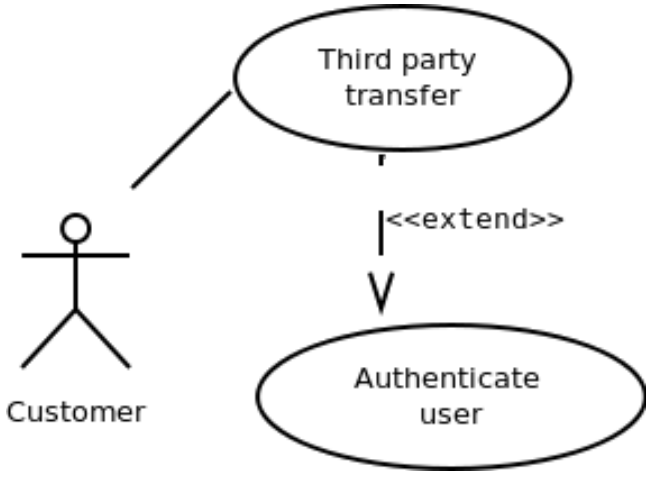


Figure - 03: Extend relationship between use cases

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

➔ The generalization relationship is used to represent the inheritance between use cases. A derived use case specializes in 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, the rectangle is a particular instance of a polygon having four sides at right angles to each other. So, the use case *draw rectangle* inherits the properties of the use case *draw a polygon* and overrides it's drawing method. This is an example of a generalization relationship. Similarly, a generalization relationship exists between *draw rectangle* and *draw square* use cases. The relationship has been illustrated in figure - 04.

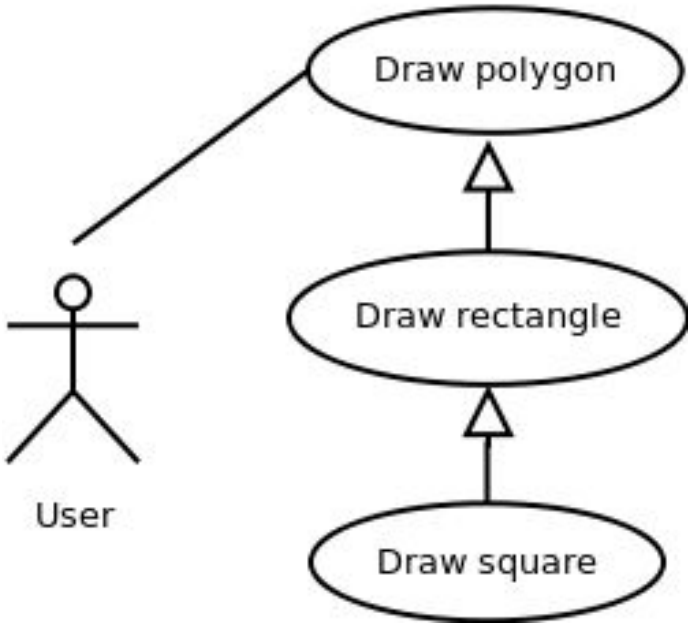


Figure - 04: Generalization relationship among use cases

Notation

➔ The 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?

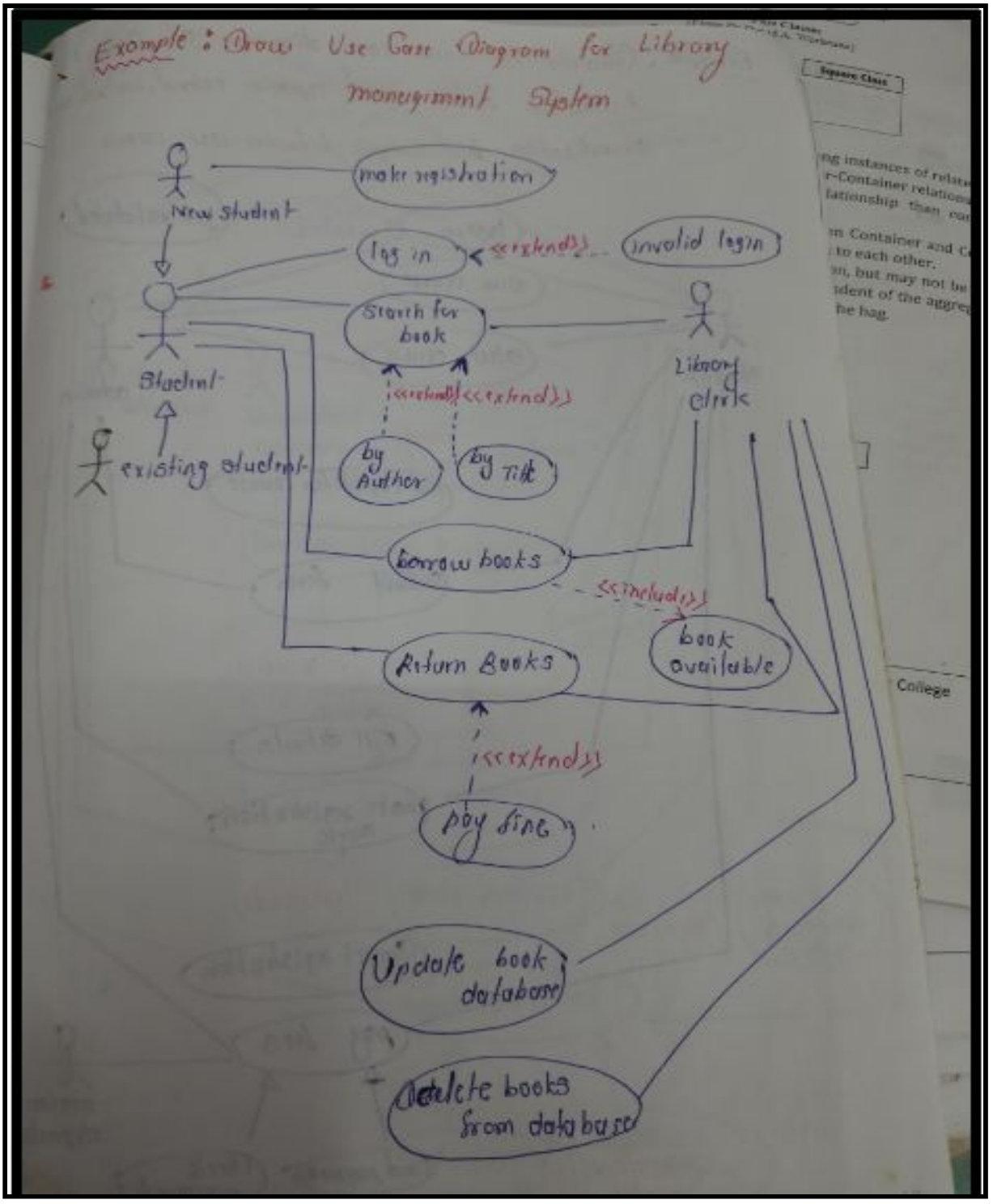
Identifying Use cases

➔ Once the primary and secondary actors have been identified, we have to find out their goals i.e. what is the functionality they can obtain from the system. 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 a 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 behaviour among use cases if any

Example: **Library Management system**



PART B

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case there is no Blackboard access available)

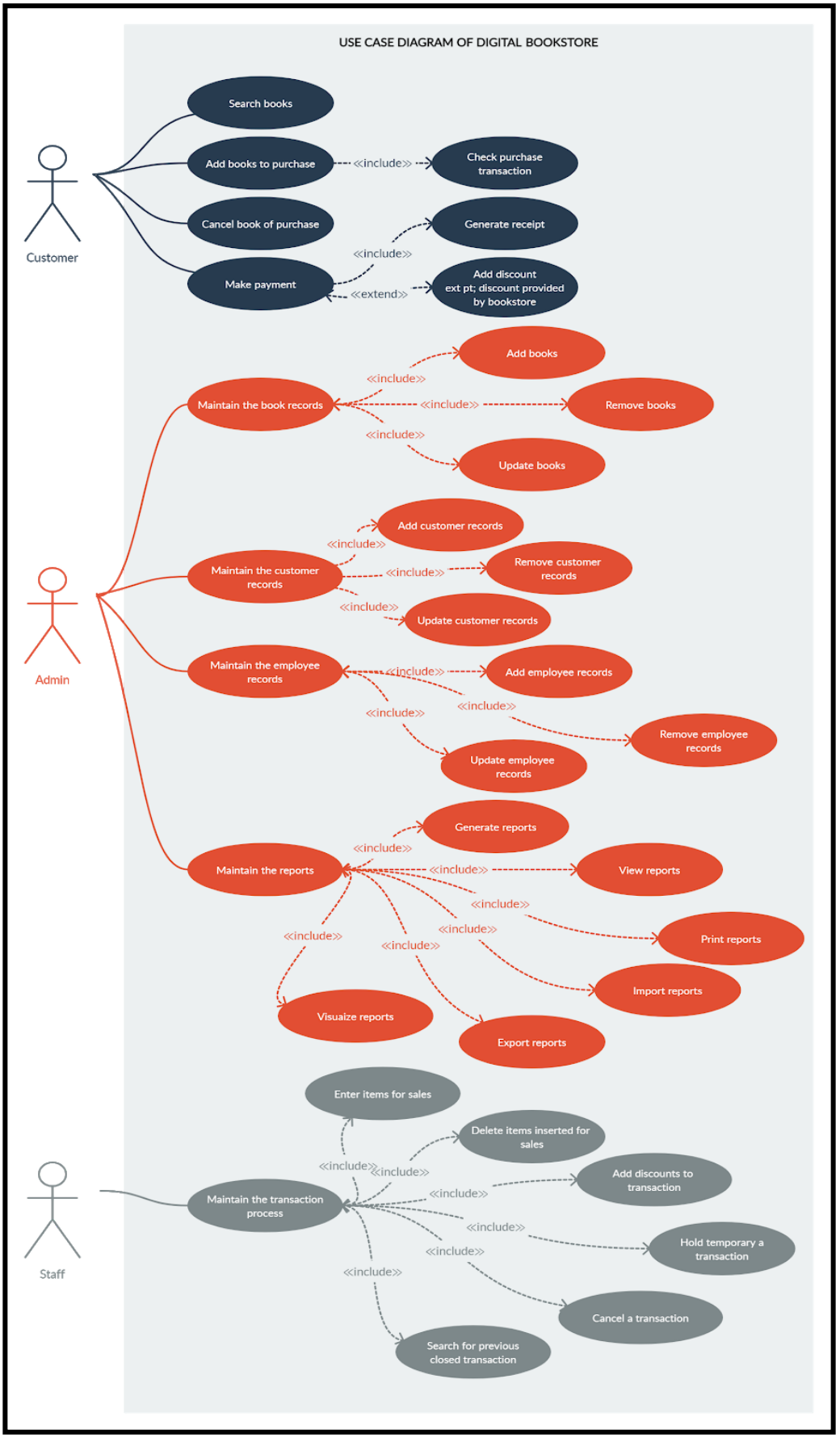
Roll No. 50	**Name:** AMEY THAKUR
Class: Comps TE B	**Batch:** B3
Date of Experiment: 17/02/2021	**Date of Submission:** 17/02/2021
Grade:	

B.1 Draw Use case Diagram for selected mini project

What is a use case diagram?

- ➔ A UML use case diagram is the primary form of system/software requirements for a new software program underdeveloped. Use cases specify the expected behaviour (what), and not the exact method of making it happen (how). Use cases once specified can be denoted both textual and visual representation (i.e. use case diagram). A key concept of use case modelling is that it helps us design a system from the end user's perspective. It is an effective technique for communicating system behaviour in the user's terms by specifying all externally visible system behaviour.
- ➔ A use case diagram is usually simple. It does not show the detail of the use cases:
    - It only summarizes some of the relationships between use cases, actors, and systems.
    - It does not show the order in which steps are performed to achieve the goals of each use case.

Use Case Diagram (UML):



B.2 Conclusion:

(Students must write the conclusion)

Use Case Model(UML) Conclusion: User Requirements use cases

In this experiment, we discussed the purpose and scope, elements of, and notation used with the use case model.

1. The purpose and scope of the use case model: to model the user expectations for the system.
2. The elements of a use case model: the use case diagram and use case narratives.
3. The notation of the use case diagram: the system, actors, use cases, and associations.
4. How to construct a use case diagram from a problem statement.
5. How to document a use case using a narrative of the dialogue between an actor and a use case.
6. How to refine the use case diagram using UML extensions called stereotypes
7. How to use the Includes stereotype (<<includes>>).
8. How to use the Extends stereotype and generalization (<<extends>>).

B.3 Question of Curiosity

1. What do you mean by functional requirements? How to model functional requirements?

Ans:

Functional Requirements:

➔ A Functional Requirement (FR) is a description of the service that the software must offer. It describes a software system or its component. A function is nothing but inputs to the software system, its behaviour, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. Functional Requirements are also called Functional Specification.

➔ In software engineering and systems engineering, a Functional Requirement can range from the high-level abstract statement of the sender's necessity to detailed mathematical functional requirement specifications. Functional software requirements help you to capture the intended behaviour of the system.

How to model Functional Requirements?

➔ Functional Requirements should include the following things:

1. Details of operations conducted on every screen

2. Data handling logic should be entered into the system
3. It should have descriptions of system reports or other outputs
4. Complete information about the workflows performed by the system
5. It should clearly define who will be allowed to create/modify/delete the data in the system
6. How the system will fulfil applicable regulatory and compliance needs should be captured in the functional document

2. List of various elements of use case diagram and specify their graphical notation.

Ans:

Elements of a Use Case Diagram:

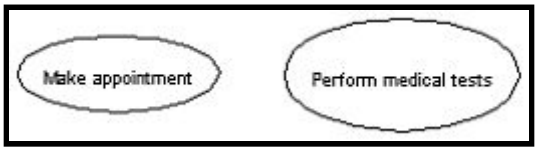
A use case diagram captures the business processes carried out in the system. Normally, domain experts and business analysts should be involved in writing use cases. Use cases are created when the requirements of a system need to be captured. A use case diagram is quite simple and depicts two types of elements: one representing the business roles and the other representing the business processes. Let us take a closer look at use at what elements constitute a use case diagram.

- **Actors:** An actor portrays any entity (or entities) that perform certain roles in a given system. The different roles the actor represents are the actual business roles of users in a given system. An actor in a use case diagram interacts with a use case. For example, for modelling a banking application, a customer entity represents an actor in the application. Similarly, the person who provides service at the counter is also an actor. But it is up to you to consider what actors make an impact on the functionality that you want to model. If an entity does not affect a certain piece of functionality that you are modelling, it makes no sense to represent it as an actor. An actor is shown as a stick figure in a use case diagram depicted "outside" the system boundary, as shown in the below figure.

An actor in a use case diagram

To identify an actor, search in the problem statement for business terms that portray roles in the system. For example, in the statement "patients visit the doctor in the clinic for medical tests," "doctor" and "patients" are the business roles and can be easily identified as actors in the system.

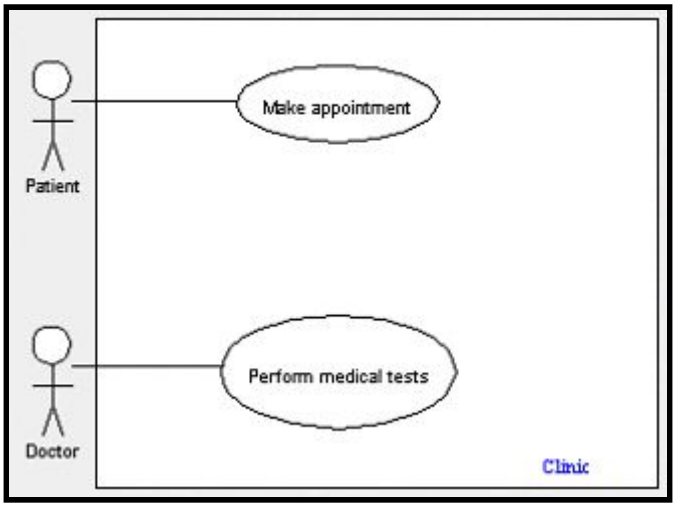
- **Use case:** A use case in a use case diagram is a visual representation of distinct business functionality in a system. The key term here is "distinct business functionality." To choose a business process as a likely candidate for modelling as a use case, you need to ensure that the business process is discrete. As the first step in identifying use cases, you should list the discrete business functions in your problem statement. Each of these business functions can be classified as a potential use case. Remember that identifying use cases is a discovery rather than a creation. As business functionality becomes clearer, the underlying use cases become more easily evident. A use case is shown as an ellipse in a use case diagram.



Use cases in a use case diagram

The above figure shows two use cases: "Make an appointment" and "Perform medical tests" in the use case diagram of a clinic system. As another example, consider that a business process such as "manage patient records" can in turn have sub-processes like "manage patient's personal information" and "manage patient's medical information." Discovering such implicit use cases is possible only with a thorough understanding of all the business processes of the system through discussions with potential users of the system and relevant domain knowledge.

- **System boundary:** A system boundary defines the scope of what a system will be. A system cannot have infinite functionality. So, it follows that use cases also need to have definitive limits defined. A system boundary of a use case diagram defines the limits of the system. The system boundary is shown as a rectangle spanning all the use cases in the system.



A use case diagram depicting the system boundary of a clinic application

The figure shows the system boundary of the clinic application. The use cases of this system are enclosed in a rectangle. Note that the actors in the system are outside the system boundary.

The system boundary is potentially the entire system as defined in the problem statement. But this is not always the case. For large and complex systems, each of the modules may be the system boundary. For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting, and so forth, can form the system boundary for use cases specific to each of these business functions. The entire system can span all of these modules depicting the overall system boundary.

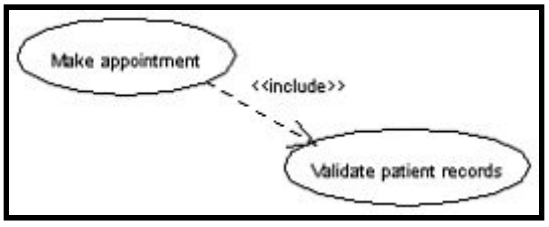
3. What are various types of relationships used in the UML use case diagram? Explain.

Ans:

Relationships in Use Cases:

Use cases share different kinds of relationships. A relationship between two use cases is a dependency between the two use cases. Defining a relationship between two use cases is the decision of the modeller of the use case diagram. This reuse of an existing use case using different types of relationships reduces the overall effort required in defining use cases in a system. Similar reuse established using relationships will be apparent in the other UML diagrams as well. Use case relationships can be one of the following:

- **Include:** When a use case is depicted as using the functionality of another use case in a diagram, this relationship between the use cases is named as an include relationship. Speaking, in an include relationship, a use case includes the functionality described in another use case as a part of its business process flow. An include relationship is depicted with a directed arrow having a dotted shaft. The tip of the arrowhead points to the child use case and the parent use case is connected at the base of the arrow. A key here is that the included use case cannot stand alone, i.e., one would not validate the patient record without making an appointment. The stereotype "<<include>>" identifies the relationship as an include relationship.



An example of an include relationship

For example, you can see that the functionality defined by the "Validate patient records" use case is contained within the "Make

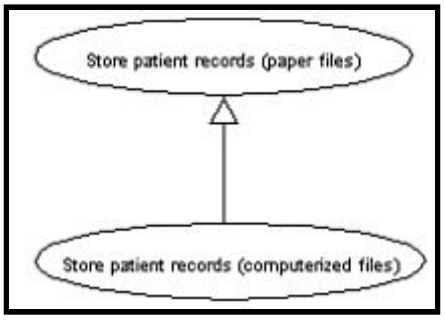
appointment" use case. Hence, whenever the "Make an appointment" use case executes, the business steps defined in the "Validate patient records" use case are also executed.

- **Extend:** In an extend relationship between two use cases, the child use case adds to the existing functionality and characteristics of the parent use case. An extend relationship is depicted with a directed arrow having a dotted shaft, similar to the include relationship. The tip of the arrowhead points to the parent use case and the child use case is connected at the base of the arrow. The stereotype "<<extend>>" identifies the relationship as an extend relationship, as shown:

An example of an extend relationship

The figure shows an example of an extend relationship between the "Perform medical tests" (parent) and "Perform Pathological Tests" (child) use cases. The "Perform Pathological Tests" use case enhances the functionality of the "Perform medical tests" use case. Essentially, the "Perform Pathological Tests" use case is a specialized version of the generic "Perform medical tests" use case.

- **Generalizations:** A generalization relationship is also a parent-child relationship between use cases. The child use case in the generalization relationship has the underlying business process meaning but is an enhancement of the parent use case. In a use case diagram, generalization is shown as a directed arrow with a triangle arrowhead. The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case.



An example of a generalization relationship

On the face of it, both generalizations and extends appear to be more or less similar. But there is a subtle difference between a generalization relationship and an extend relationship. When you establish a generalization relationship between use cases, this implies that the parent use case can be replaced by the child use case without breaking the business flow. On the other hand, an extend relationship between use cases implies that the child use case enhances the functionality of the parent use case into a specialized functionality. The parent use case in an extend relationship cannot be replaced by the child use case.

Let us see if we understand things better with an example. From the diagram of a generalization relationship, you can see that the "Store patient records (paper file)" (parent) use case is depicted as a generalized version of the "Store patient records (computerized file)" (child) use case. Defining a generalization relationship between the two implies that you can replace any occurrence of the "Store patient records (paper file)" use case in the business flow of your system with the "Store patient records (computerized file)" use case without impacting any business flow. This would mean that in future you might choose to store patient records in a computerized file instead of as paper documents without impacting other business actions.

Now, if we had defined this as an extend relationship between the two use cases, this would imply that the "Store patient records (computerized file)" use case is a specialized version of the "Store patient records (paper file)" use case. Hence, you would not be able to seamlessly replace the occurrence of the "Store patient records (paper file)" use case with the "Store patient records (computerized file)" use case.

****