CSE1005: Software Engineering

Module No. 2: Requirements Engineering and Design

Dr. K Srinivasa Reddy
SCOPE, VIT-AP University

Module No. 2: Requirements Engineering and Design

Requirements: Requirements Engineering, UML Model, Developing Use Cases, Building the Requirements Model, Negotiating Requirements, Validating Requirements.

Design: Design within the Context of Software Engineering, Design Process, Design Concepts, Design Model.

Text Book:

1. Roger Pressman, “Software Engineering: A Practitioner’s Approach”, McGraw-Hill, 7th Edition, 2016.

Expected Outcome: Model the structure and behavior a software system the UML class diagrams and state diagrams. (CO2)

- **Requirements analysis and specification** is one of the most important phase in the life of cycle of a software development work.
- It has many skills to learn.
- This lecture will focus at how to go about carrying out the requirements analysis and specifications.
- **Two aspects: Understanding** and **Specifying** the requirements.
  - For **small applications or projects** requirements specification is **easy.**
  - For **Industry standard problems**, requirements specification is typically the **hardest** and **most problematic and error prone among the development tasks** and it is important to note that any requirements error has a **huge cost overhead**, **unlike a coding error or something which can be quickly corrected.**

- For a **Specific Software Product (set of clients):** Software Requirement Specification (SRS) is must. **(Customized products)**

- For **Generic Software Product (no specific set of clients)?**
  - Example: Excel, Word, PowerPoint, Notepad, Outlook, a health embedded product (monitors various health parameters and give feedback to people).
  - **Who will give the requirements?**

    Sales personnel who understand that what will sell well?

Understanding the requirements

- Understanding the requirements of a given problem is among the **most difficult tasks that face a software engineer.**
- In general, Customer walks into your office,
  - Sits down, Looks you straight in the eye, and says,
  - “I know you think you understand what I said, but What you don’t understand is what I said is not what I meant.”
- Always, this happens **late in the project**, after deadline commitments have been made, reputations are on the line, and **serious money is at stake.**
- The software requirements are the **description of features and functionalities** of the **target system**.
- Requirements convey **the expectations of users from the software product.**
- The requirements can be **obvious or hidden**, **known or unknown**, **expected or unexpected** from the client’s point of view.

The Problems with our Requirements Practices

- **Trouble in understanding** the requirements that we do acquire from the customer
- Recording the requirements in a **disorganized manner**
- Spend far **too little time verifying** what we do record
- Allow change to control us, rather than establishing mechanisms to control change
- Most importantly, we **fail to establish a solid foundation** for the system or software that the user wants built.

→

The Problems with our Requirements Practices (continued)

Many software developers **argue that**

- **Building software is so compelling** that we want to **jump right in** (before having a clear understanding of what is needed)
- Things **will become clear** as we **build the software**
- Project stakeholders will be **able to better understand** what they need only **after examining early iterations of the software**
- Things change so rapidly that **requirements engineering is a waste of time**
- The **bottom line is producing a working program** and that all else is secondary

All of these arguments contain some truth, especially for small projects that take less than one month to complete.

However, as **software grows in size and complexity**, these arguments begin to break down and can lead to a failed software project.

A Solution: Requirements Engineering

The broad spectrum of tasks and techniques that lead to an understanding of requirements is called **requirements engineering**.

From a **software process perspective,**

- Requirements engineering is a major software engineering action, **begins during the communication activity and continues into the modeling activity**.
- **Builds a bridge** from the system requirements into software design and construction
- Allows the requirements engineer to examine
  - The context of the software work to be performed
  - The specific needs that design and construction must address
  - The priorities that guide the order in which work is to be completed
  - The information, function, and behavior that will have a profound impact on the resultant design

In general, requirements engineering provides the appropriate mechanism for
Understanding what the customer wants,
Analyzing need,
Assessing feasibility,
Negotiating a reasonable solution,
Specifying the solution unambiguously,
Validating the specification, and
Managing the requirements as they are transformed into an operational system.

Recap

- Importance of understanding the requirements
- The Problems with our Requirements Practices and consequences
- Definition of Requirements Engineering
    - Software engineering perspective
    - General perspective

Requirements Engineering Tasks

Seven distinct tasks

1. Inception
2. Elicitation
3. Elaboration
4. Negotiation
5. Specification
6. Validation
7. Requirements Management

- Some of these tasks may occur in parallel and all are adapted to the needs of the project
- All strive to define what the customer wants
- All serve to establish a solid foundation for the design and construction of the software

Requirements Engineering Tasks

Inception

Elicitation

Elaboration

Negotiation

Specification

Validation

Requirements Management

Requirements engineering begins with **Inception** - a task that defines the scope and nature of the problem to be solved, **Elicitation** - a task that helps stakeholders define what is required, **Elaboration** - where basic requirements are refined and modified,

As stakeholders define the problem, **negotiation** occurs—what are the priorities, what is essential, when is it required?

Finally, the problem is **specified** in some manner (SRS) and then reviewed or **validated** to ensure that your understanding of the problem and the stakeholders' understanding of the problem coincide.

How does a software project get started?

- In general, most projects begin **when a business need is identified** or a **potential new market** or **service is discovered.**

- **Stakeholders** from the business community (e.g., business managers, marketing people, product managers) define a business case for the idea, try to identify the breadth and depth of the market, do a rough feasibility analysis, and identify a working description of the project’s scope.

- All of this information is subject to change, but it is sufficient to precipitate discussions with the software engineering organization

1. Inception

- During inception, the requirements engineer asks a set of questions to establish…
  - A basic understanding of the problem
  - The people who want a solution
  - The nature of the solution that is desired
  - The effectiveness of preliminary communication and collaboration between the customer and the developer

Through these questions, the requirements engineer needs to…

  - Identify the stakeholders
  - Recognize multiple viewpoints (potential market, budget, easy to learn and use, maintainability, etc.)
  - Working toward collaboration
  - Break the ice and initiate the communication

- **Stakeholders** are the people who will find benefit in the project and the software being developed.

- Stakeholders are **the people who will find benefit** in the project and the software being developed.
- Examples:
  - Customers
  - End users
  - Advertising and marketing staff
  - Business operations managers
  - Product managers
  - Software engineers
  - Support engineers

An **end user** is someone who completes the final purchase of a product, while a **customer** can purchase a product and then resell it.
If someone is **both the purchaser and user of a product**, they're both the customer and the end-user.

1. Inception

- During inception, the requirements engineer **asks a set of questions to establish…**
  - A basic understanding of the problem
  - The people who want a solution
  - The nature of the solution that is desired
  - The effectiveness of preliminary communication and collaboration between the customer and the developer

Through these questions, the requirements engineer **needs to…**

  - Identify the stakeholders
  - Recognize multiple viewpoints (potential market, budget, easy to learn and use, maintainability, etc.)
  - Working toward collaboration
  - Break the ice and initiate the communication

1. Inception

The First Set of Questions

focuses on the customer, other stakeholders, the overall goals, and the benefits

- **Who is behind** the request for this work?
- **Who will use** the solution?
- **What will be** the **economic benefit** of a successful solution?
- **Is there another source** for the solution that you need?

The Next Set of Questions

enables to gain a better understanding of the problem and allow the customer to voice his or her perceptions about a solution

- How would you **characterize "good" output** that would be generated by a successful solution?
- What **problem(s) will this solution address**?
- Can you **show me (or describe) the business environment** in which the solution will be used?
- Will **special performance issues or constraints affect** the way the solution is approached?

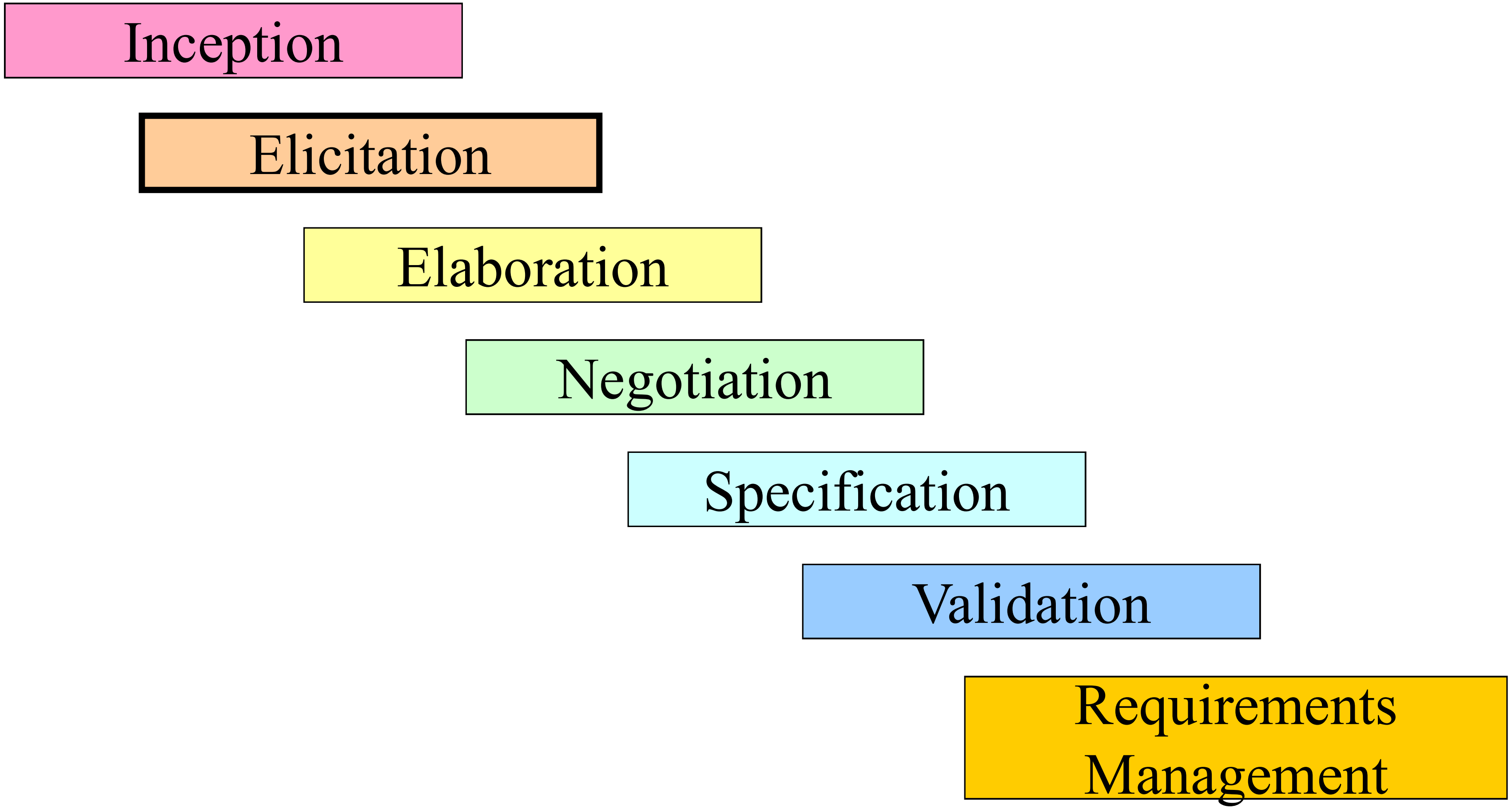
1. Inception

The Final Set of Questions

focuses on the effectiveness of the communication activity itself

- **Are you the right person** to answer these questions? Are your answers "official"?
- **Are my questions relevant** to the problem that you have?
- **Am I asking too** many questions?
- **Can anyone else provide** additional information?
- **Should I be asking** you anything else?

Requirements Engineering Tasks



Inception

Elicitation

Elaboration

Negotiation

Specification

Validation

Requirements Management

2. Elicitation

- **Requirements elicitation** (also called **requirements gathering**) combines elements of **problem solving**, **elaboration**, **negotiation**, and **specification**.
- **Seems simple enough**
  - **Ask the** customer, and other stakeholders
  - **What are the objectives** for the system or product,
  - **What is to** be accomplished,
  - **How the system or product fits** into the needs of the business, and finally,
  - **How the system or product** is to be **used** on a day-to-day basis.
- **But it isn’t simple - it’s very hard.**

Problems that are encountered as elicitation occurs:

Problems of scope: The boundary of the system is ill-defined or the Customers / users specify unnecessary technical detail that may confuse, **rather than clarify, overall system objectives**.

Problems of understanding The customers/users are **not completely sure of**

- What is needed,
- Have a poor understanding of the capabilities and limitations of their computing environment,
- Don’t have a full understanding of the problem domain,
- Have trouble communicating needs to the system engineer, omit information that is believed to be “obvious”
- Specify requirements that conflict with the needs of other customers/users, Specify requirements that are ambiguous or untestable.

Problems of volatility The requirements change over time

To overcome these problems, approach requirements gathering in an organized manner.

Requirements elicitation Methods:

Following are the few requirements elicitation methods.

1. Interviews
2. Brainstorming Sessions
3. Facilitated Application Specification Technique (FAST)
4. Quality Function Deployment (QFD)
5. Use Case Approach

The **success of an elicitation** technique used **depends on the maturity of the** analyst, developers, users, and the customer involved.

Requirements elicitation Methods:
Interviews

- Objective of conducting an interview is to understand the customer’s expectations from the software.
- It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.
- Interviews maybe be **open-ended or structured**.
- In **open-ended** interviews there is **no pre-set agenda**. Context free questions may be asked to understand the problem.
  - Are you my only client?
  - Who is the customer of the product?
  - Who are the other stakeholders?
  - how do we measure the success of this project?
  - What problems does this product solve?
- In **structured interview**, agenda of fairly open questions is prepared or a proper questionnaire is designed for the interview.

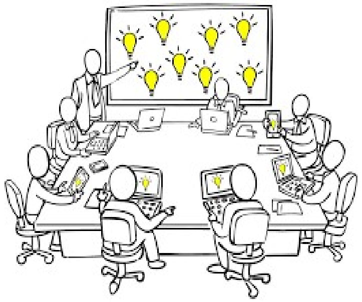
Requirements elicitation Methods:

Brainstorming Sessions: (Collaborative Requirements Gathering)

- It is a group technique, **meetings are conducted and attended** by both software engineers, customers, and other interested stakeholders
- Rules for preparation and participation are established
- **An agenda is suggested formally** to cover all important points but **informal** enough to encourage the free flow of ideas, hence providing a platform to share views
- A highly trained facilitator (can be customer, developer, or an outsider) is required to handle group bias and group conflicts.
- Every idea is documented (can be work sheets, flip charts, wall stickers, electronic bulletin board, chat room, or virtual forum) called as **"definition mechanism"** so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible **write a one - or two-page “product request.”**

Collaborative Requirements Gathering

- The goal of **collaborative, team-oriented approach** is
  - to identify the problem,
  - propose elements of the solution,
  - negotiate different approaches and
  - specify a preliminary set of solution requirements to achieve the gal.



Meeting between all of the stakeholders

This approach is sometimes called as ***Facilitated Application Specification Technique* (FAST) :** The **objective** of FAST approach is to **bridge the expectation gap**, a **difference** between **what developers think they are supposed to build** and **what customers think they are going to get.**

Example: Safe Home Project

- **Product request written by a Marketing person.**
- The home security function would
  - Protect against and / or recognize a variety of undesirable “situations” such as illegal entry, fire, flooding, carbon monoxide levels, and others.
  - It’ll use wireless sensors to detect each situation.
  - It can be programmed by the homeowner, and will automatically telephone a monitoring agency when a situation is detected.

Product Request Review Meeting: (Stake holders meet)

- **Attendees are asked days before the meeting a list of**
  - **Objects**
    - Objects that are part of the environment that surrounds the system,
    - Objects that are to be produced by the system, and
    - Objects that are used by the system to perform its functions.
  - **Services (processes or functions)**
    - manipulate or interact with the objects
  - **Constraints** (e.g., cost, size, business rules) and
  - **Performance criteria** (e.g., speed, accuracy)

The lists are not expected to be exhaustive but are expected to reflect each person's perception of the system.

Product Request Review Meeting: (Stake holders meet)

- A list of **Objects**
  - Control panel, smoke detectors, window and door sensors, motion detectors, an alarm, an event (a sensor has been activated), a display, a PC, telephone numbers, a telephone call, etc.,
- A list of **Services (processes or functions)**
  - Configuring the system, setting the alarm, monitoring the sensors, dialing the phone, programming the control panel, and reading the display (note that services act on objects).
- A list of **Constraints (e.g., cost, size, business rules)**
  - The system must recognize when sensors are not operating, must be user-friendly, must interface directly to a standard phone line
- **Performance criteria**
  - A sensor event should be recognized within one second, and an event priority scheme should be implemented

Product Request Review Meeting: (Stake holders meet)

- The lists of objects can be pinned to the walls of the room on a large sheets of paper, or written on a wall board, or posted on an electronic bulletin board, or at an internal website, or posed in a chat room environment for review prior to the meeting.
- After individual lists are presented in one topic area, the group creates a combined list by eliminating redundant entries, adding any new ideas that come up during the discussion, but not deleting anything.
- After combined lists are created for all topic areas, discussion - coordinated by the facilitator—ensues or terminates meeting.
- The combined list is shortened, lengthened, or reworded to properly reflect the product/system to be developed.
- **The objective is to develop a consensus list of objects, services, constraints, and performance for the system to be built.**

- **Product Request Review Meeting: (Stake holders meet)**
  - **Importance of Mini-Specifications:**
    - In many cases, an object or service described on a list will **require further explanation**.
    - To accomplish this, stakeholders develop mini-specifications for entries on the lists.
    - Each mini-specification is an elaboration of an object or service.
  - Example: **the mini-spec for the SafeHome object Control Panel :**
    - The control panel is a wall-mounted unit that is approximately 9X5 inches in size.
    - The control panel has wireless connectivity to sensors and a PC. User interaction occurs through a keypad containing 12 keys.
    - A 3X3 inch LCD color display provides user feedback.
    - Software provides interactive prompts, echo, and similar functions.
- The mini-specs are presented to all stakeholders for discussion. Additions, deletions, and further elaboration are made.
- In some cases, the development of mini-specs will **uncover new objects, services, constraints, or performance requirements that will be added to the original lists.**
- During all discussions, the team may raise an **issue** that cannot be resolved during the meeting. An **issues list** is maintained so that these ideas will be acted on later.

Rather than creating a mini-specification, many software teams elect to develop user scenarios called *use cases.*

The Use Case Approach: It simply describes and displays the relation or interaction between the users or customers and providers of application service or the system

Quality Function Deployment (QFD)

- **Translates the needs** of the customer into **technical requirements** for software.
- QFD **"concentrates on maximizing customer satisfaction from the software engineering process",** hence it emphasizes **on the requirements which are valuable to the customer.**

Quality Function Deployment (QFD): Three types of requirements

1.Normal requirements: The objective and goals of the proposed software are discussed with the customer. If these requirements are present, the customer is satisfied.

- **Example:** Graphical displays, specific system functions, and defined levels of performance. **Result management system:** entry of marks, calculation of results, etc.

2. Expected requirements: These requirements are so clear that the customer need not explicitly state them. These are **implicit to the product or system.** Their absence will be a cause for significant dissatisfaction.

- **Example:** Ease of human/machine interaction, overall operational correctness and reliability, and ease of software installation. **Result management system:** protection from unauthorized access.

3. Exciting requirements: **Features that are beyond the customer's expectations** and prove to be very satisfying when present.

- **Example:** Software for a new mobile phone comes with standard features, but is coupled with a set of unexpected capabilities (e.g., multi touch screen, visual voice mail) that delight every user of the product. **Result management system:** when unauthorized access is detected, it should backup and shutdown all processes.

Quality Function Deployment (QFD)

QFD uses customer interviews and observation, surveys, and examination of historical data (e.g., problem reports) as raw data for the requirements gathering activity.

These data are then translated into a table of requirements—called the ***customer voice table*** - that is reviewed with the customer and other stakeholders.

A variety of diagrams, matrices, and evaluation methods are then used to extract expected requirements and to attempt to derive exciting requirements

Elicitation Work Products

The work products will vary depending on the system, but should include one or more of the following items

- A statement of **need and feasibility**
- A bounded statement of **scope for the system or product**
- A list of **customers, users, and other stakeholders who participated in requirements elicitation**
- A **description of the system's** technical environment
- A **list of requirements** (organized by function) and the **domain constraints** that apply to each
- A set of **usage_scenarios** (in the form of use cases) that provide insight into the use of the system or product **under different operating conditions**
- **Any prototypes developed** to better define requirements

Each of these work products is reviewed by all people who have participated in requirements elicitation

Requirements elicitation

Requirements elicitation (also called **requirements gathering**) combines elements of **problem solving**, **elaboration**, **negotiation**, and **specification**.

Requirements elicitation Methods:

1. Interviews
2. Brainstorming Sessions
3. Facilitated Application Specification Technique (FAST)
4. Quality Function Deployment (QFD)
5. **Use Case Approach**

The **success of an elicitation** technique used **depends on the maturity of the** analyst, developers, users, and the customer involved.

Developing Use Cases – Requirements capture with Unified Modeling Language (UML)

- A use case **describes the interaction** (tells a stylized story) **between the persons (playing one of a number of possible roles) and the system** for a **system function** to accomplish a goal **under a specific set of circumstances.**
- **The story may be**
  - Narrative text,
  - An outline of tasks or interactions,
  - A template-based description, or
  - A diagrammatic representation.
- A use case helps to **understand where errors could occur** in the process and design features to **resolve those errors**.

Developing Use Cases : Steps in writing a use case

Step One – **Define** the **set of actors** that will be involved in the story

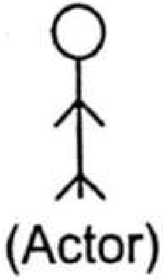
- Actors are **people, devices, or other systems that use the system or product** within the context of the function and behavior that is to be described.
- Communicate with the system or product and that are external to the system itself
- **Types of actors:**

1. Primary: Work **directly** and **frequently** with the software

Example: A customer uses an ATM to withdraw cash when he needs it.
Customer is the primary actor here.

2. Secondary: Render some kind of **service or support** to the system so that primary actors can do their work

Example: "Bank representatives", who reloads the stock of cash



Actor is represented as a stick figure.

Developing Use Cases : Steps in writing a use case

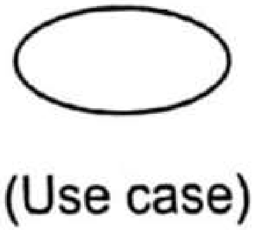
Step One – How to find actors?

- Who uses the main functionality of the system?
- Which hardware devices the system needs to handle?
- Which other systems does the system need to interact with?
- What **nouns / subjects** are used to describe the system?
  - A **student** must login in to do course registration ..
  - The **librarian** issues a book using the system, based on the...

Developing Use Cases : Steps in writing a use case

Step Two – Develop use cases

- A use case is simply a **functionality provided by a system**, where each one answers a set of questions.
- Describe the **sequence of interactions** between **actors** and **the system**.
- Capture **who(actors) do what(interaction) with the system**.
- **A complete set of use cases specifies** all possible ways to use the system.



- **An oval or ellipse is used to represent a use case.**
- **Label the ovals with verbs that represent the system's functions.**

- ***withdraw cash,* check balance**, **change PIN** are functionalities that the ATM provides.
- Use cases include both **successful and unsuccessful scenarios** of user interactions with the system.
- Example: Authentication of a customer by the ATM would fail if **wrong PIN** was entered. In such case, an **error message is displayed** on the screen of the ATM.

Developing Use Cases : Steps in writing a use case

Step Two – Develop use cases

How to identify use cases?

- What **functions** does the system perform?
- What **functions** do the “actors” require?
- What **input/output** does the system need?
- What **verbs** are used to describe the system?

Examples:

- The Librarian can **issue a book** using the system, based on the...
- The Student can **make an entry for course registration**. He can also **modify registration details**, provided....
- The faculty can **choose a course**, **design a course**, provided....
- The Reservation Clerk **makes a booking** using the system, based on the….
- The Airport Manager can **make an entry for a new flight**. He can also **modify flight details**, provided....

Developing Use Cases : Questions Commonly Answered by a Use Case

- Who is the **primary** actor(s), the **secondary** actor(s)?
- What are the actor's **goals**?
- What **preconditions** should exist before the scenario begins?
- What **main tasks** or **functions** are performed by the actor?
- What **exceptions** might be considered as the scenario is described?
- What **variations** in the actor's interaction are possible?
- What **system information** will the actor acquire, produce, or change?
- Will the actor have to inform the system about **changes in the external environment?**
- What **information** does the actor desire **from the system**?
- Does the actor wish to be informed about **unexpected changes**?

Case Study :- RAILWAY RESERVATION SYSTEM

Railway Reservation System is a system used for booking tickets over internet. Any Customer can book tickets for different trains. Customer can book a ticket only if the tickets are available. Customer searches for the availability of tickets then if the tickets are available he books the tickets by initially filling details in a form. Tickets can be booked in two ways by i-ticket or by e-ticket booking.

In case of i-ticket booking customer can book the tickets online and the tickets are couriered to Particular customer at their address. But in case of e-ticket booking and cancelling tickets are booked and cancelled online sitting at the home and customer himself has to take print of the ticket but in both the cases amount for tickets are deducted from customers account.

For cancellation of ticket the customer has to go at reservation office than fill cancellation form and ask the clerk to cancel the ticket than the refund is transferred to customer account. After booking ticket the customer has to checkout by paying fare amount to clerk.

Case Study :- RAILWAY RESERVATION SYSTEM

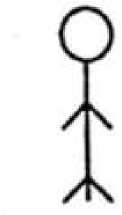
Railway Reservation System is a system used for **booking tickets** over internet. Any **Customer** can book tickets for different trains. Customer can book a ticket only if the tickets are available. Customer searches for the **availability of tickets** then if the tickets are available books the tickets by initially **filling details in a form**. Tickets can be booked in two ways by i-ticket or by e-ticket booking.

In case of i-ticket booking customer can book the tickets online and the tickets are couriered to Particular customer at their address. But in case of **e-ticket booking and cancelling tickets are booked and cancelled online sitting at the home** and customer himself has to take **print of the ticket** but in both the cases amount for tickets are deducted from customers account.

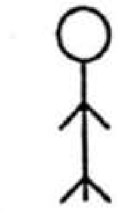
For **cancellation** of ticket the customer has to go at reservation office than **fill cancellation form** and ask the **clerk or admin** to cancel the ticket than the **refund is transferred** to customer account. After booking ticket the customer has to checkout by **paying fare amount** to clerk.

Case Study :- RAILWAY RESERVATION SYSTEM

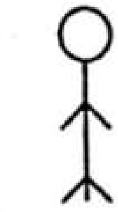
Who are all the actors?



Admin



Traveler

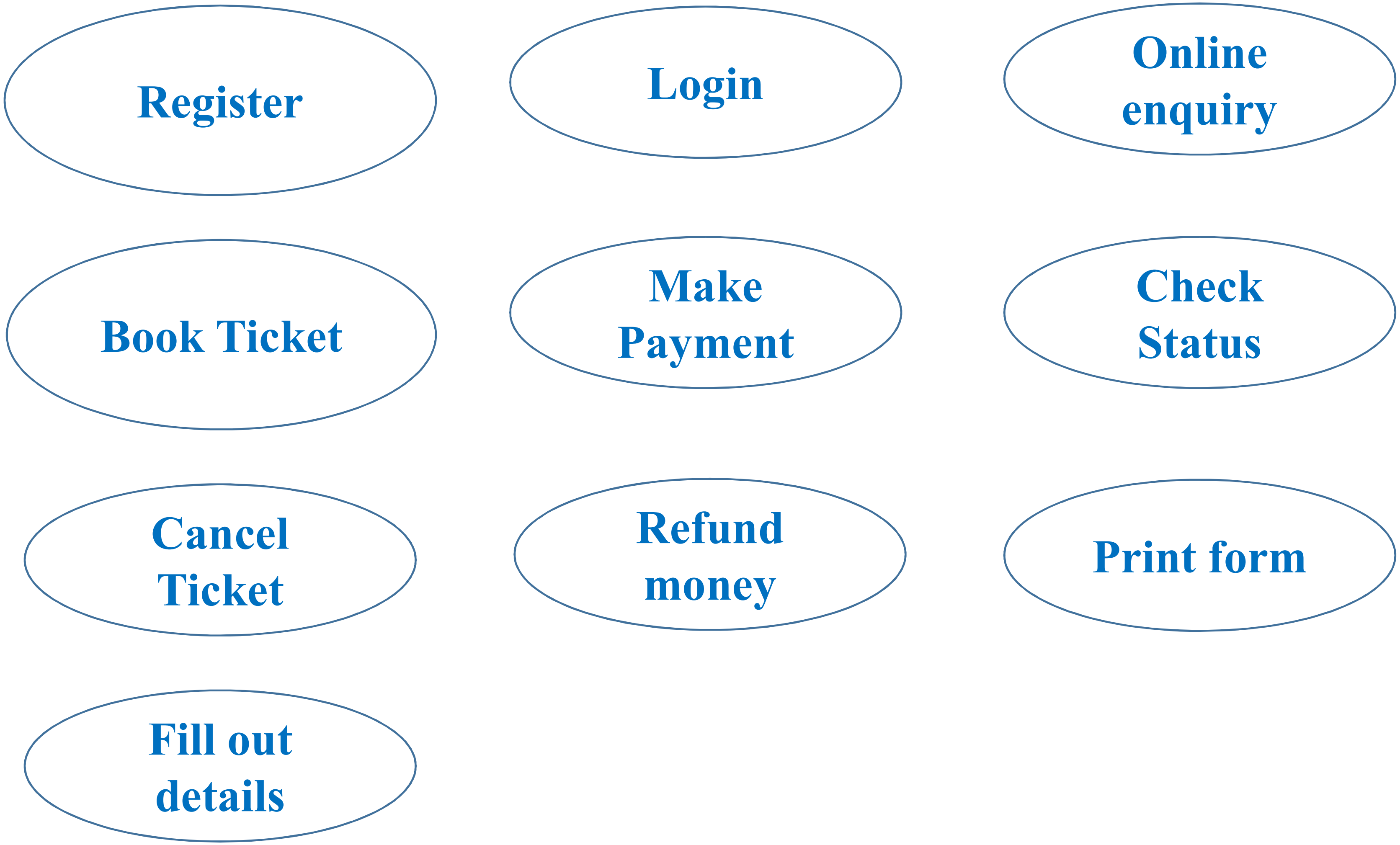


Railway Website

Actors are classifiers (not individual users)

Case Study :- RAILWAY RESERVATION SYSTEM

Name the identified use cases :



Register

Login

Online enquiry

Book Ticket

Make Payment

Check Status

Cancel Ticket

Refund money

Print form

Fill out details

Developing Use Cases : Steps in writing a use case

Step Three – Use case diagram

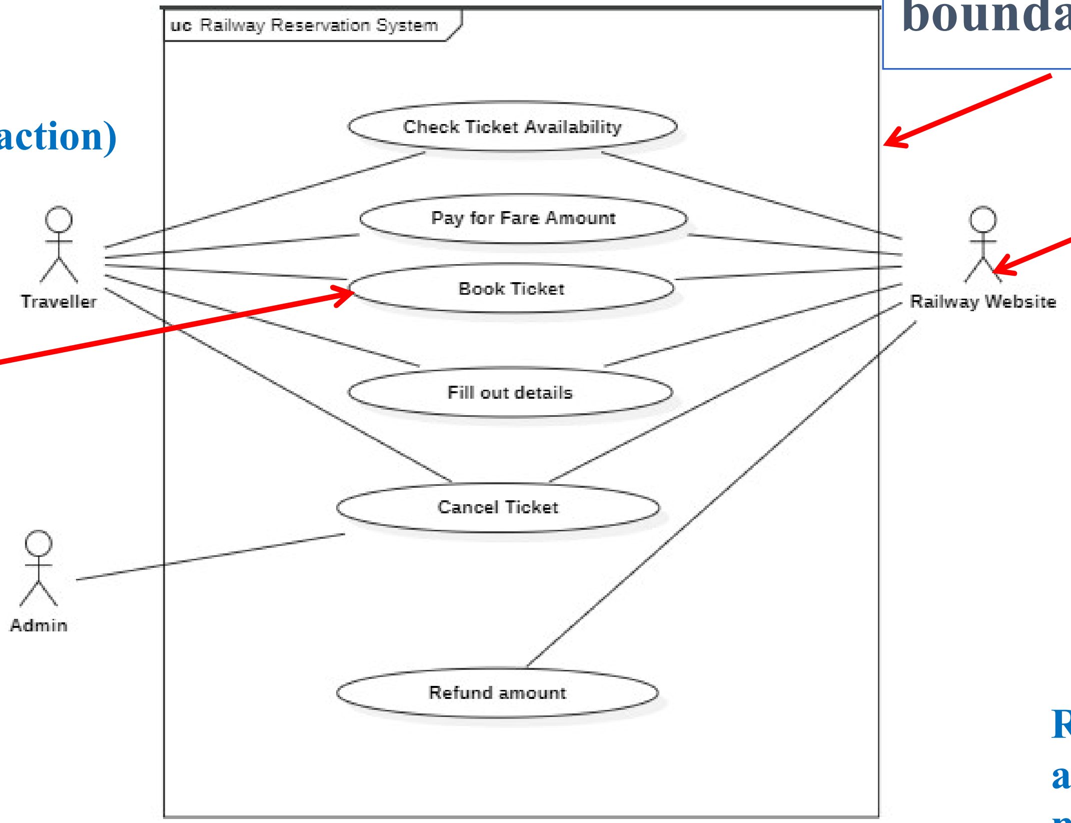
- A **use case diagram** graphically represents **what happens when an actor interacts with a system**.
- It captures the functional aspect of the system using actors and use cases.

Represents the relationship between actor and use case and / or between the use cases

line

Case Study :- RAILWAY RESERVATION SYSTEM

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



Show system boundary

Shows Actors outside boundary

Use case

who(actors) do what(interaction) with the system

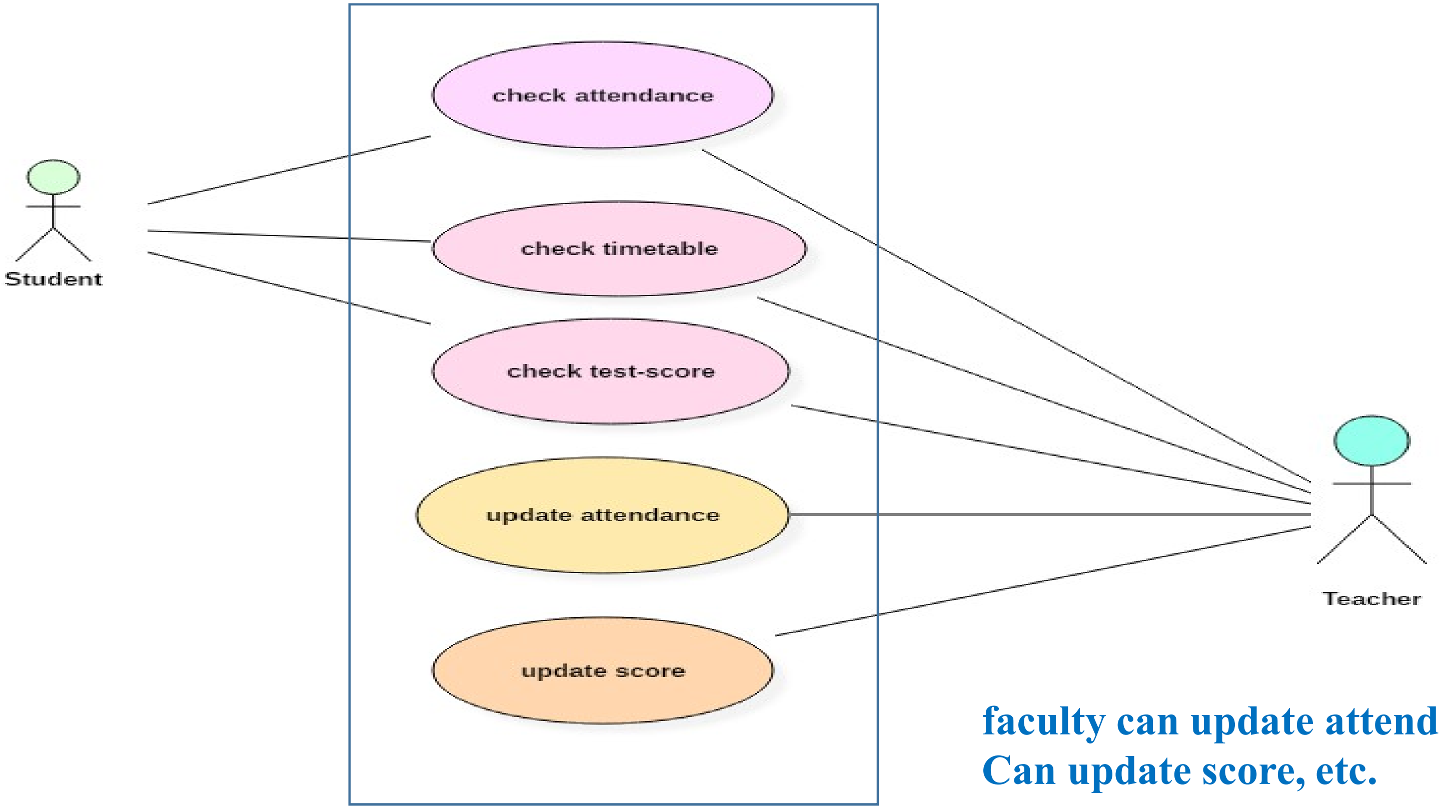
Traveller can check ticket availability, pay fare amount, Book ticket, etc.

Railway website can check ticket availability, book ticket, refund money etc.

The basic use case diagram presents a high-level story that describes the interaction between the actor and the system.

Simple Use Case Diagram showing working of the student management system:

Draw system's boundaries using a rectangle that contains use cases.



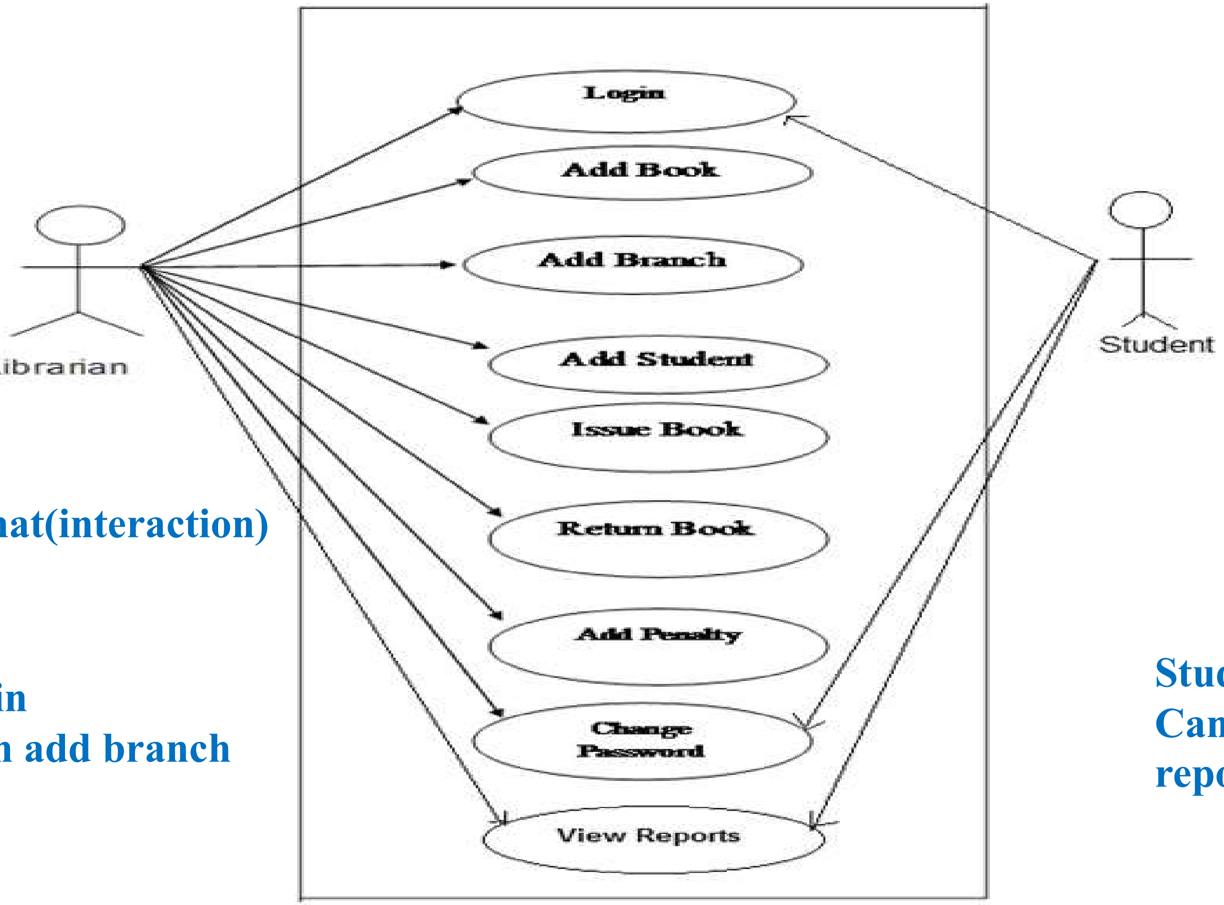
who(actors) do what(interact with the system

Student can check attendanc Can check timetable, etc.

faculty can update attendance, Can update score, etc.

The basic use case diagram presents a high-level story that describes the interaction between the actor and the system.

Simple Use Case Diagram showing Library Management System:



who(actors) do what(interaction) with the system

Librarian can login
Can add book, can add branch
Etc.

Student can login
Can change password, can view reports.

The basic use case presents a high-level story that describes the interaction between the actor and the system.

Developing Use Cases : Steps in writing a use case

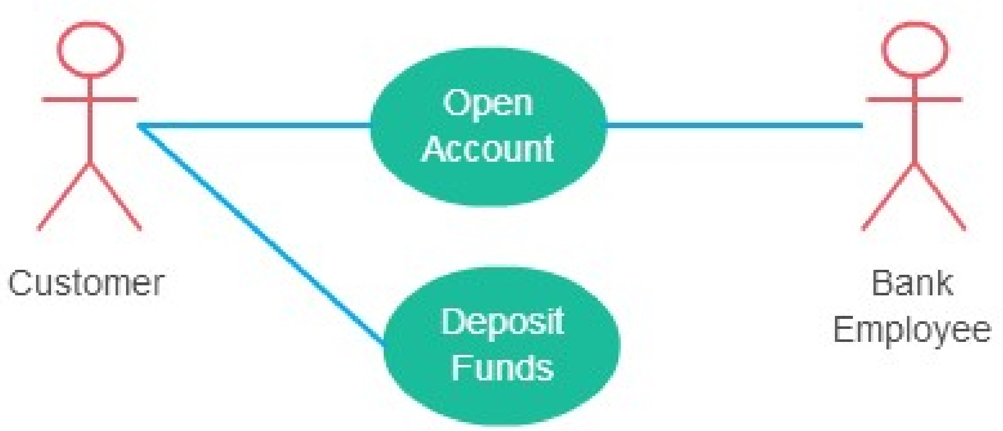
05 relationship types in a use case diagram.

- **Association between actor and use case**
- **Generalization of an actor**
- **Extend between two use cases**
- **Include between two use cases**
- **Generalization of a use case**

Developing Use Cases : relationship types in a use case diagram

Association between actor and use case

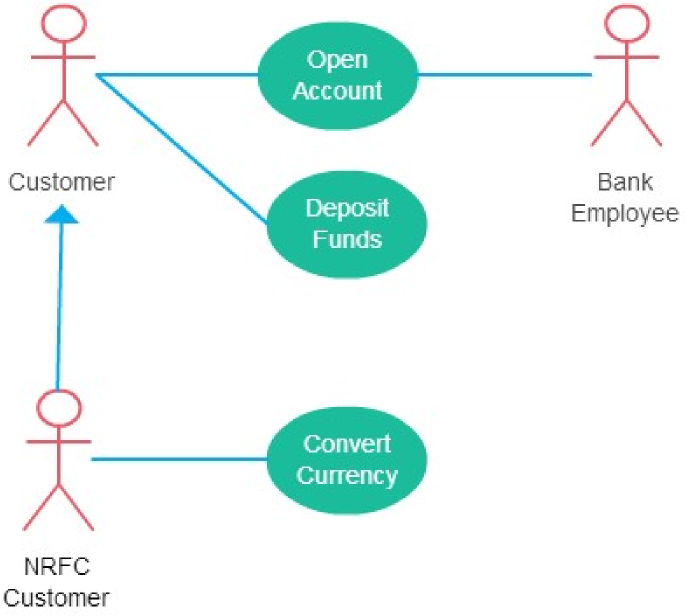
- An actor must be associated with **at least one use case.**
- An actor can be **associated with multiple use cases.**
- **Multiple actors** can be **associated** with a **single use case**.



Developing Use Cases : relationship types in a use case diagram

Generalization of an Actor ——▷

- One actor can **inherit** the role of the other actor.
- The descendant **inherits all the use cases** of the ancestor.
- The descendant has **one or more use cases** that are **specific to that role**.



Developing Use Cases : relationship types in a use case diagram

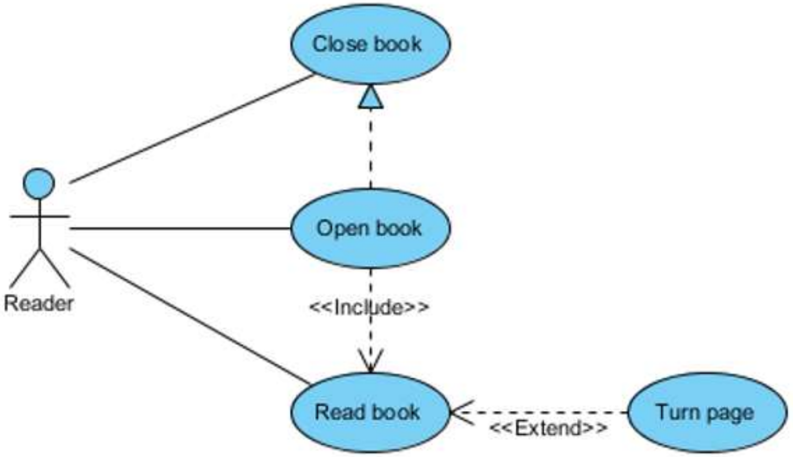
Extend Relationship Between Two Use Cases

- As the name implies it **extends the base use case** and **adds more functionality** to the system.
- The use case is **optional** and **comes after the** base use case. It is represented by a dashed arrow **in the direction of the base use case** with the notation **<<extend>>**
- Usually for handling an abnormal situation

Include Relationship Between Two Use Cases

- The use case is **mandatory** and **part of the base use case.**
- It is represented by a dashed arrow **in the direction of the included use case** with the notation **<<include>>**
- **Includes Dependency:** Defines how one use case can invoke behavior defined by another use case

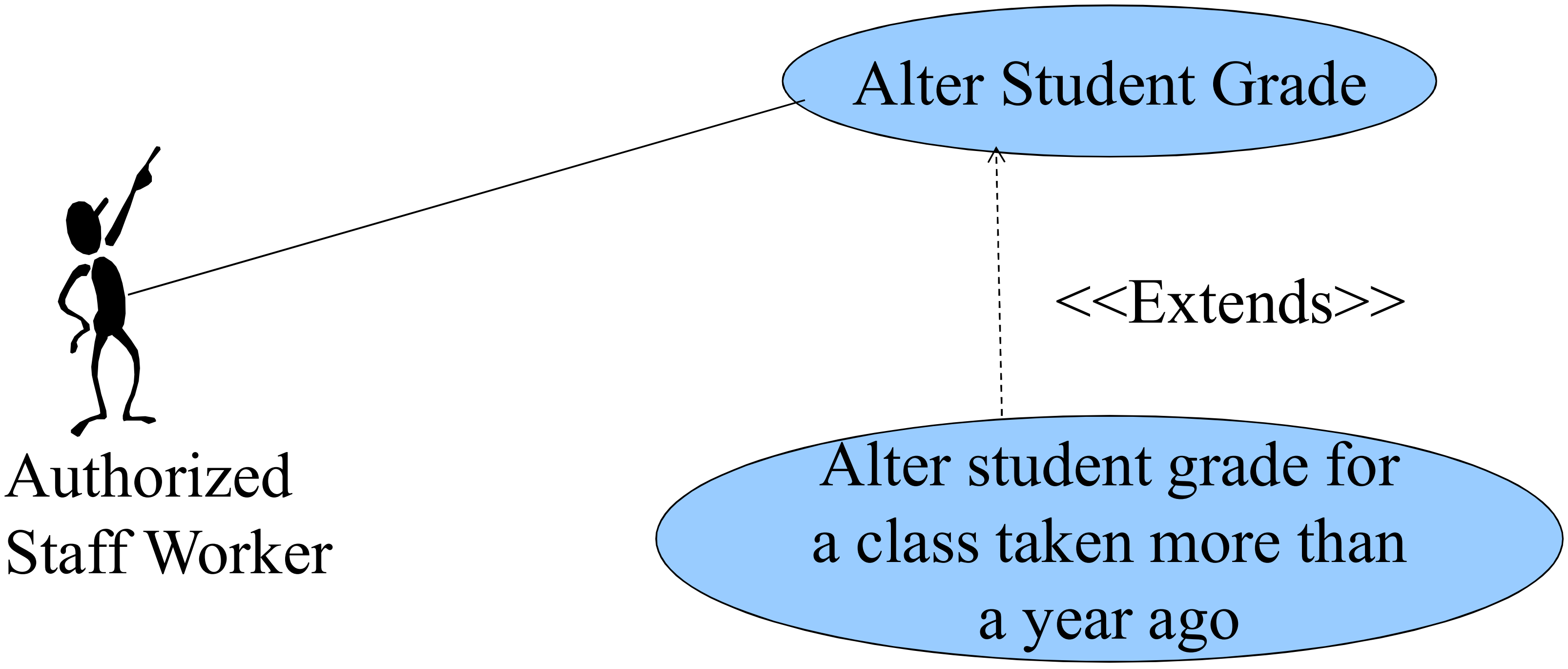
Developing Use Cases : relationship types in a use case diagram



- The use case **"Open book"** **includes** the use case **"Read book".**
  - If a reader opens the book, must read it too, as it is **mandatory** for the **base use case (open book).**
- The use case **"read book"** **extends** to the use case **"turn page",** which means that turning the page while reading the book is **optional.**
  - The "Turn Page" use case doesn't make much sense without the "Read book" use case.
  - The base use case in this scenario (read book) is complete without the extended use case.

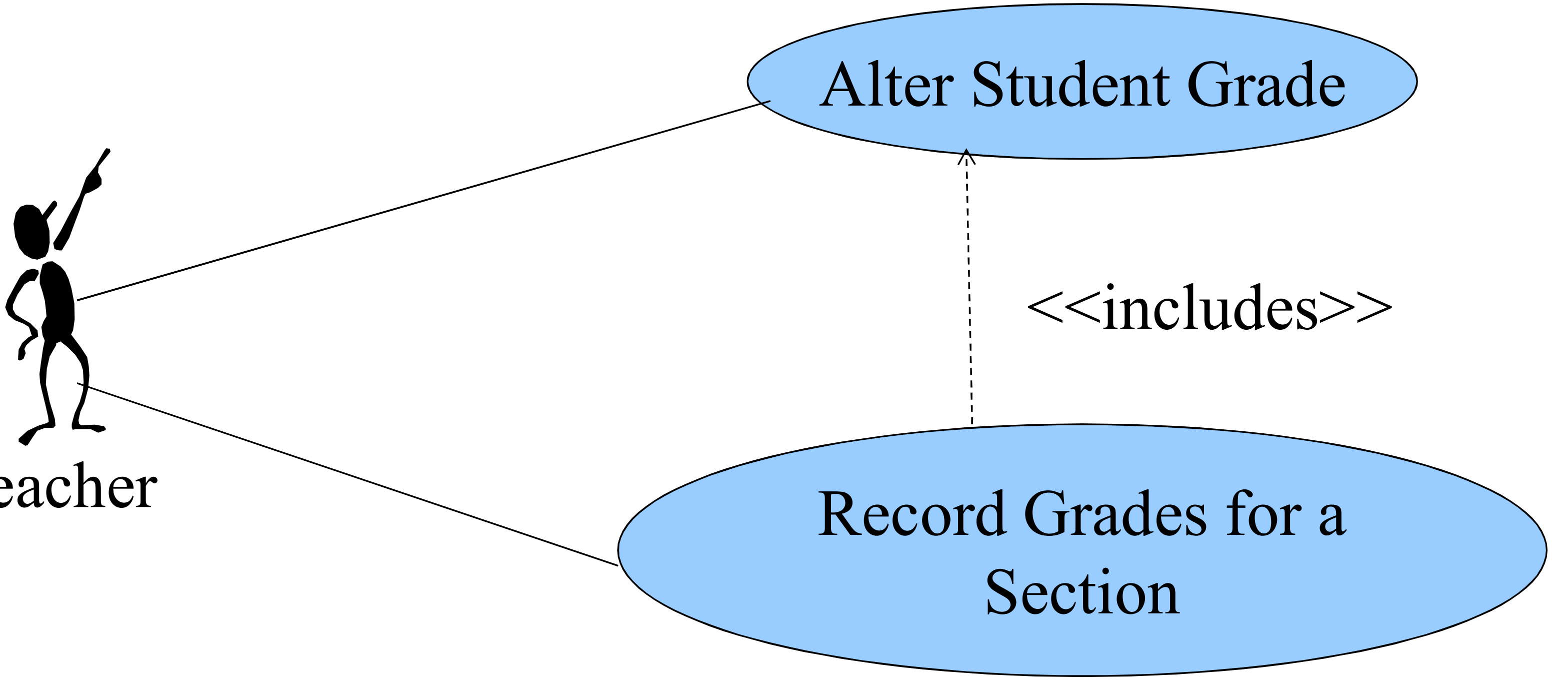
Developing Use Cases : relationship types in a use case diagram

Extends Relationship Between Two Use Cases



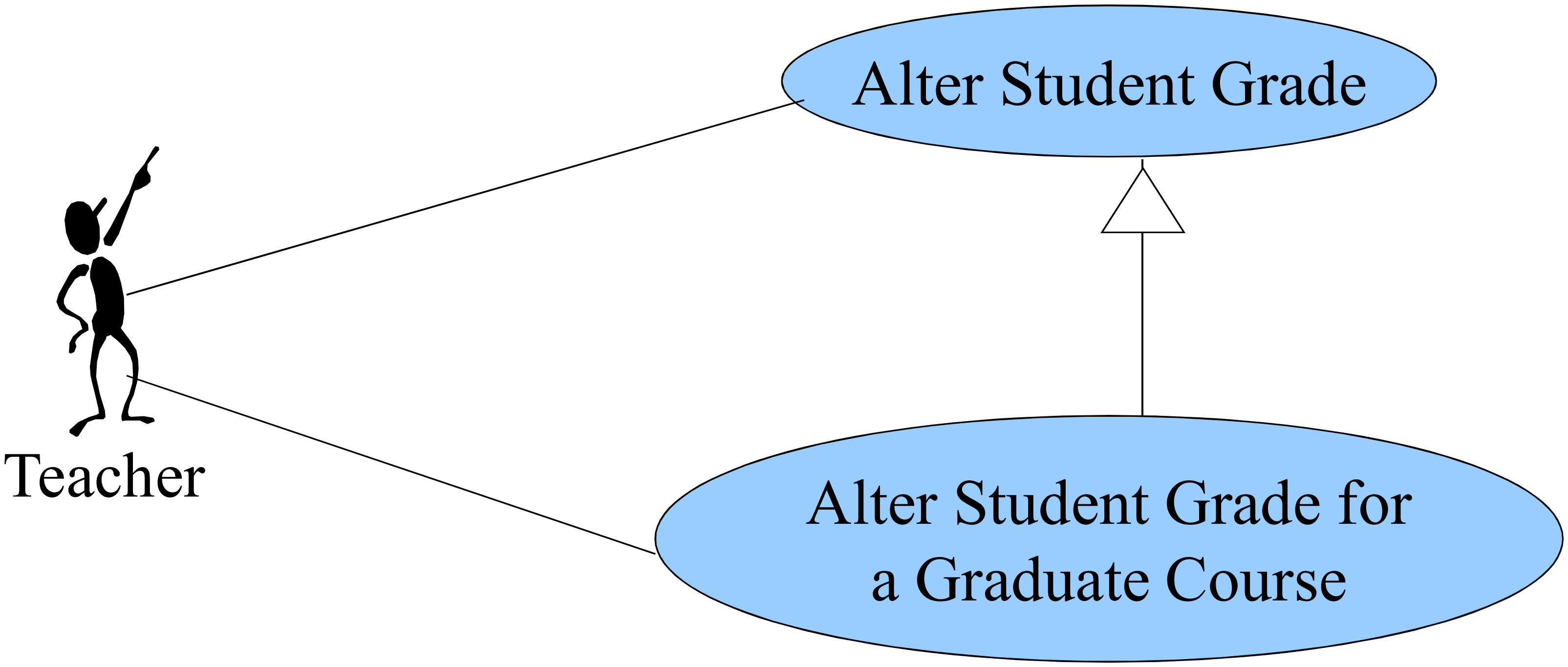
Developing Use Cases : relationship types in a use case diagram

Includes Relationship Between Two Use Cases

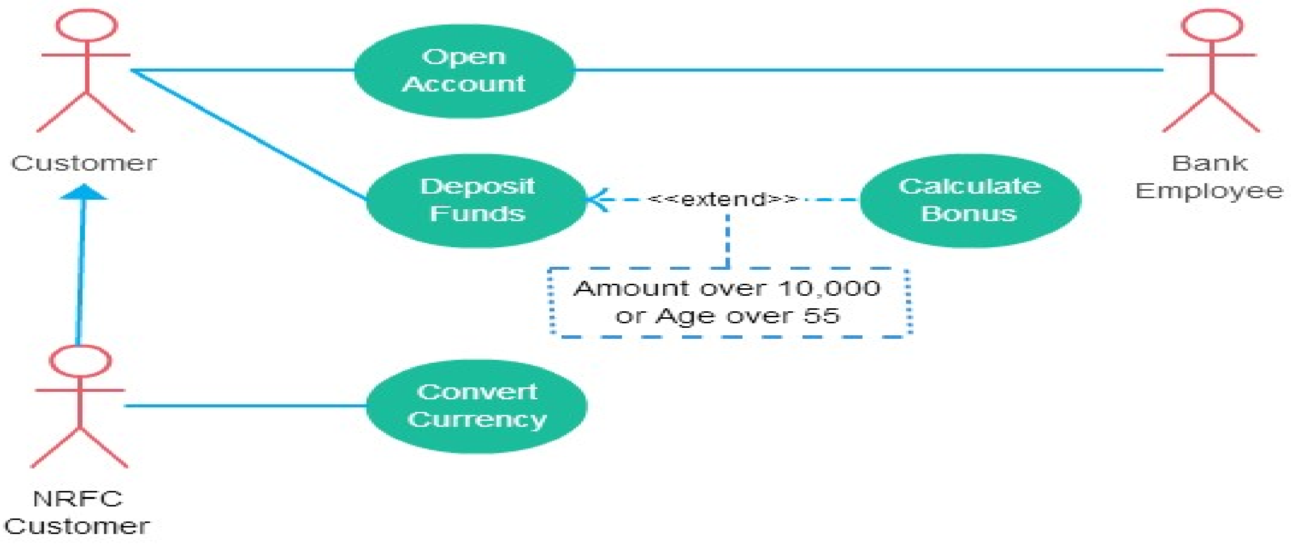


Developing Use Cases : relationship types in a use case diagram

Generalization: Defines one use case as a generalization of another. Replaces generic functionality with alternate implementation



Developing Use Cases : relationship types in a use case diagram



- The **extending use case** is dependent on the extended (base) use case:
    - The “Calculate Bonus” use case doesn’t make much sense without the “Deposit Funds” use case.
- The **extending use case** is usually optional and can be triggered conditionally.
    - The extending use case is triggered only for deposits over 10,000 or when the age is over 55.

Developing Use Cases

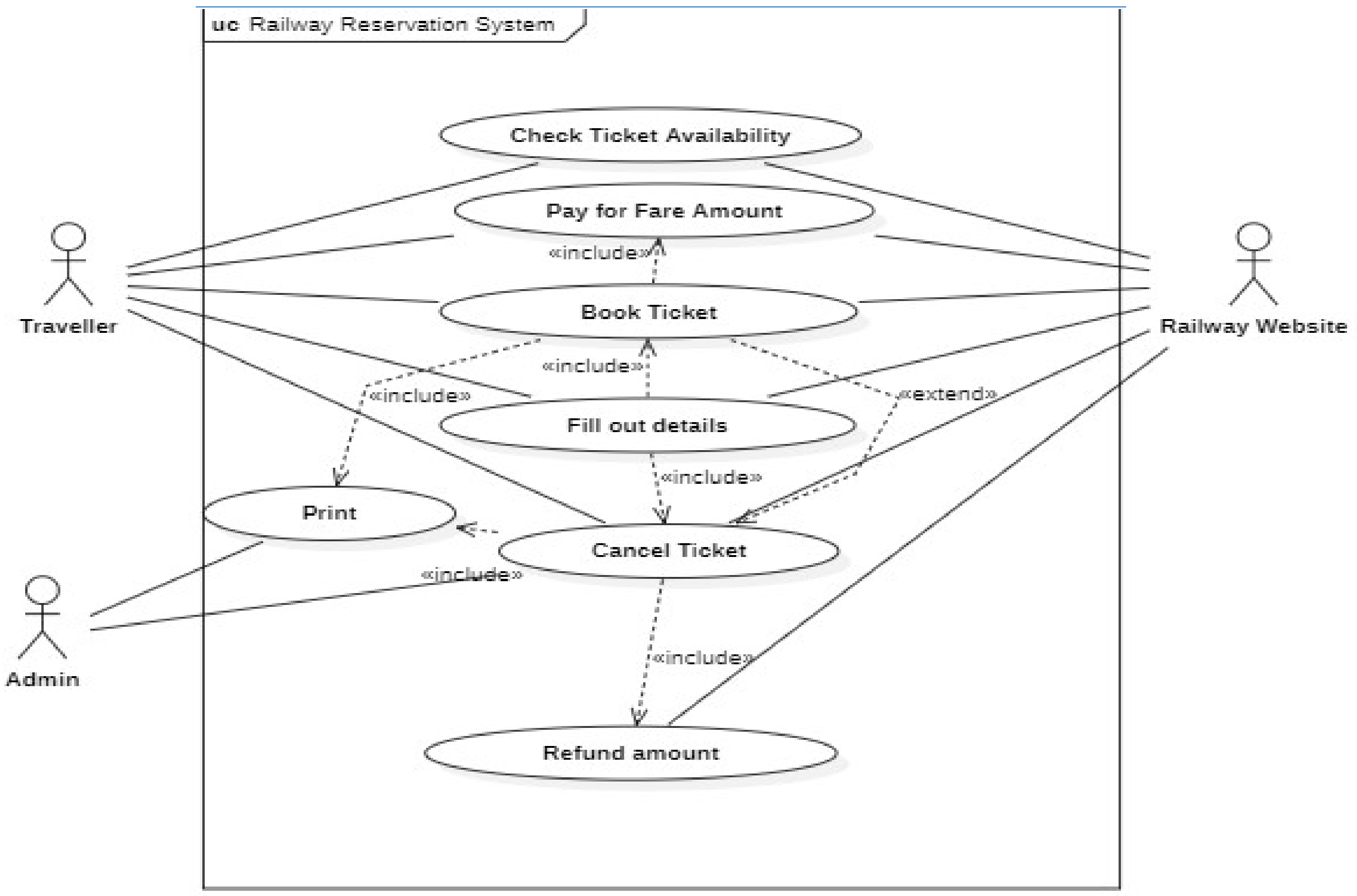
Additional elements that are included in a complex use case:

Stakeholders, which are those who have an interest in how the **system turns out**, even if they aren't direct users.

Preconditions, which are things that must be true before a use case is run.

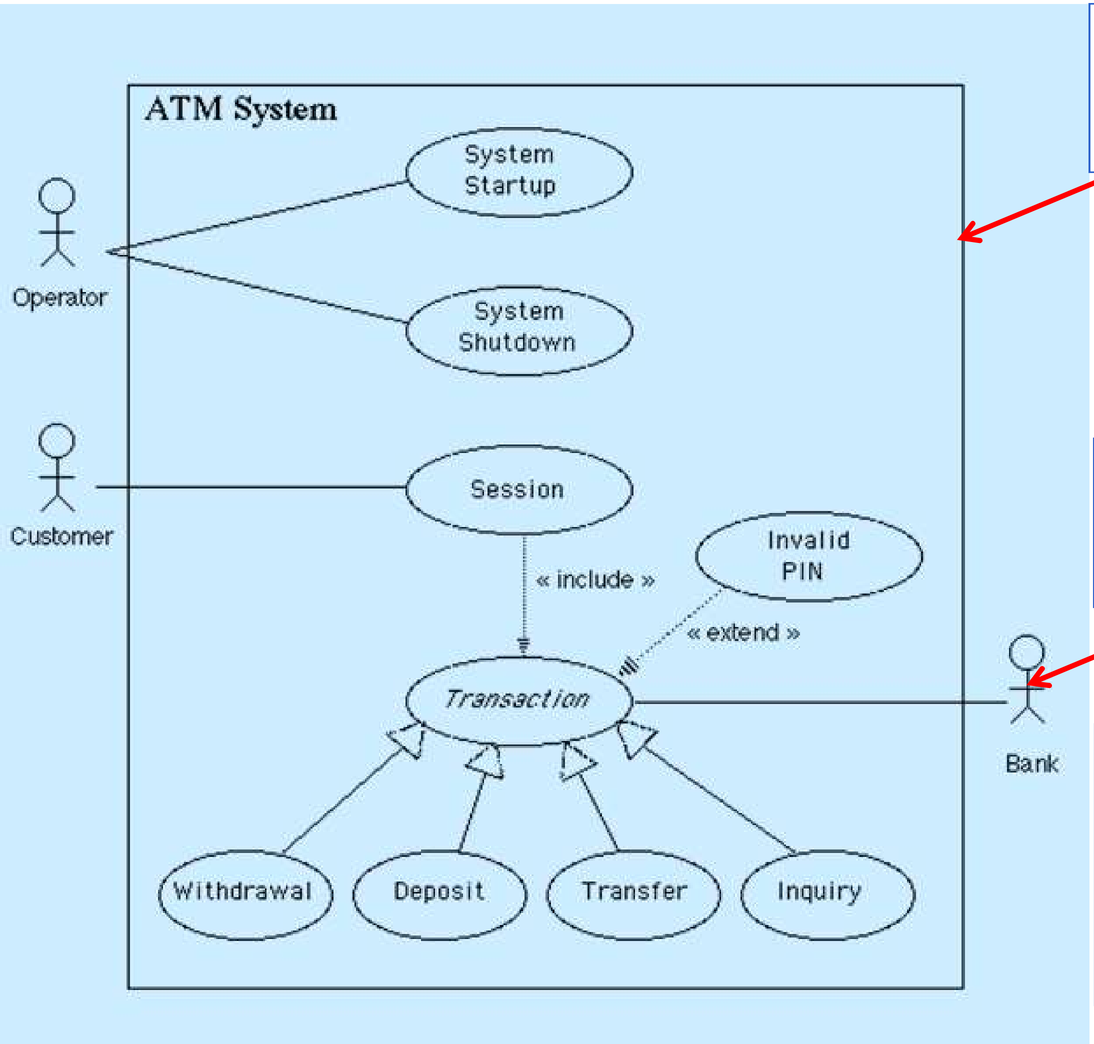
Triggers, which are events that occur for a use case to begin.

Case Study :- RAILWAY RESERVATION SYSTEM



The basic use case diagram presents a high-level story that describes the interaction between the actor and the system.

Simple Use Case Diagram showing ATM System:



Show system boundary

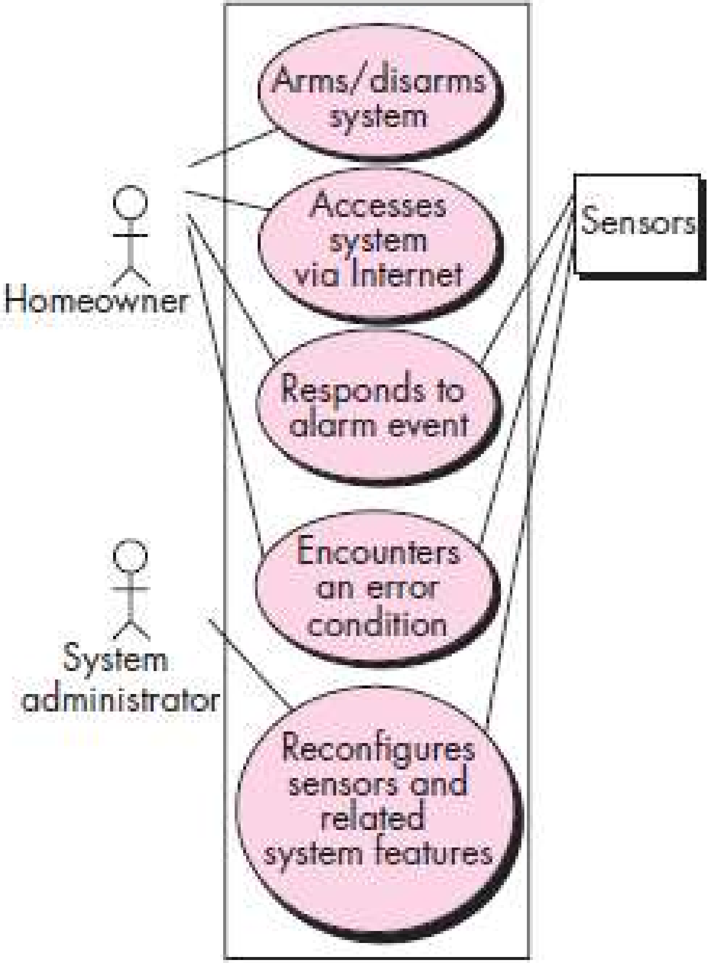
Shows Actors outside boundary

Typically one diagram for your project is sufficient

Use extend, include, generalization / specialization where appropriate

The basic use case presents a high-level story that describes the interaction between the actor and the system.

Example: SafeHome Project Developing use cases


Arms/disarms system
Accesses system via Internet
Responds to alarm event
Encounters an error condition
Reconfigures sensors and related system features
Homeowner
System administrator
Sensors

UML use case diagram for *SafeHome* home security function

Purpose of Use Cases

- To **capture and analyze the functional requirements** of a system.
- To **communicate with end users** and domain experts.
- To **design test cases** for **validating system functionality**.
- To **provide traceability** from requirements into actual classes and operations.
- High-level visual software designing
- To **drive the** development process.
- To **plan iterations and releases**
- Forward and reverse engineering of a system using various test cases.

Use cases are intended to convey desired functionality so the exact scope of a use case may vary according to the system and the purpose of creating UML model.

UML Model

- What is UML and why we use UML?
- How to use UML diagrams to design software system?
- What UML Modeling tools we use today?

UML Model

- UML → “Unified Modeling Language”
- Language: express idea, not a methodology
- Modeling: Describing a software system at a high level of abstraction
- Unified: UML has become a world standard
  - Object Management Group (OMG): www.omg.org
- It is a industry-standard graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems
- The UML uses mostly graphical notations to express the Object Oriented analysis and design of software projects.
- Simplifies the complex process of software design

UML Model

- **Why we use UML?**
  - Use graphical notation: more clearly than natural language (imprecise) and code (too detailed).
  - Help acquire an overall view of a system.
  - UML is not dependent on any one language or technology.
  - UML moves us from fragmentation to standardization.

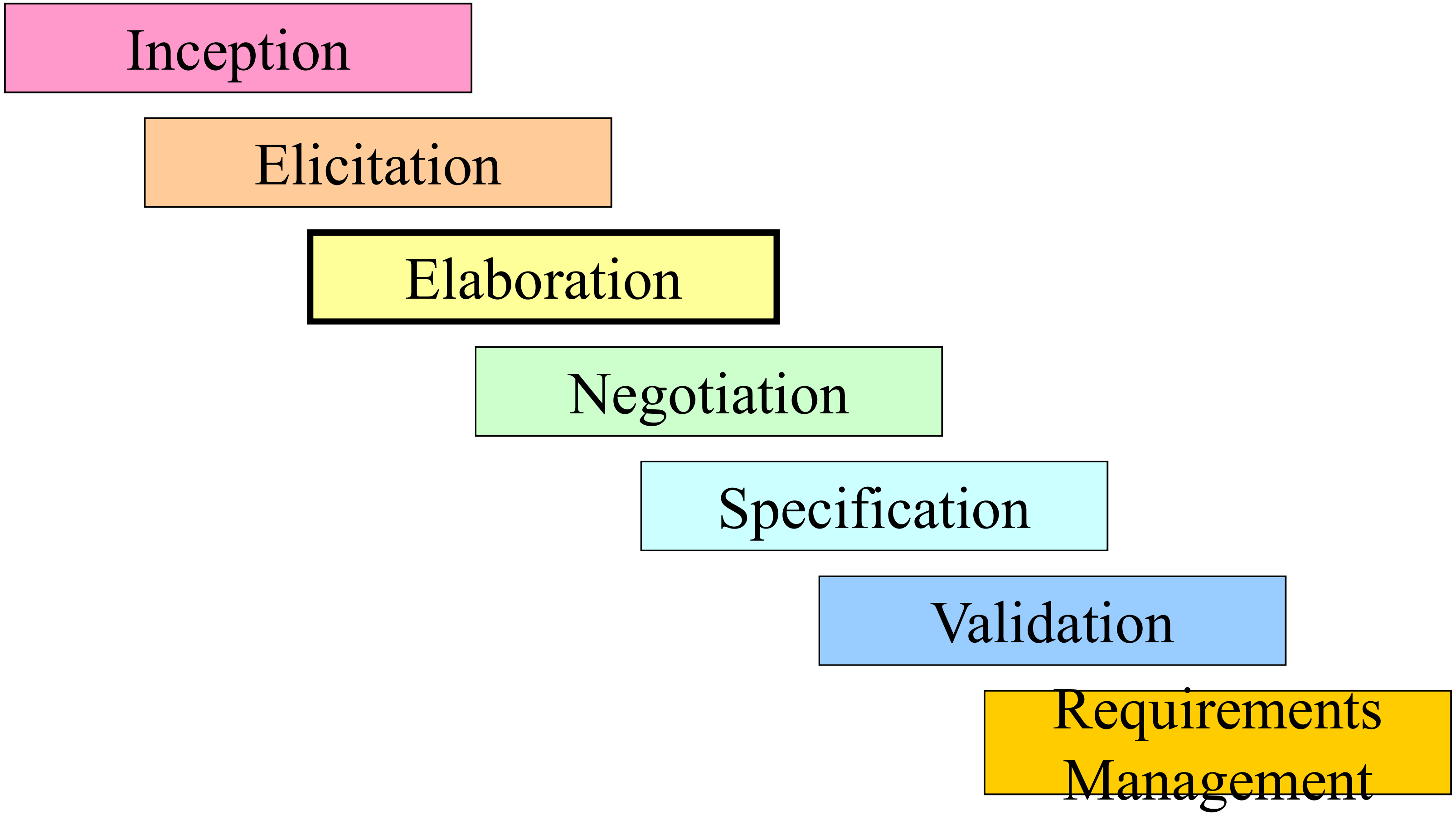
UML Model

Types of UML Diagrams:

- Use Case Diagram
- Class Diagram
- Sequence Diagram
- Collaboration Diagram
- State Diagram

This is only a subset of diagrams … but are most widely used

Requirements Engineering Tasks



Inception

Elicitation

Elaboration

Negotiation

Specification

Validation

Requirements Management

3. Elaboration: Building the Analysis Model

What is it?

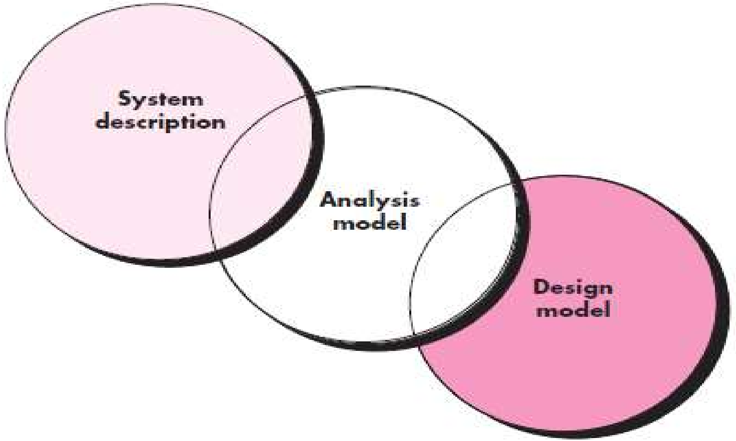
- The process by which customer **needs are understood and documented.**
- Expresses **“what”** is to be built and NOT **“how”** it is to be built.
- Example 1:
  - The system shall allow users to withdraw cash. **[What?]**
- Example 2:
  - A sale item’s name and other attributes will be stored and updated each time any attribute changes. **[what?]**
  - A sale item’s name and other attributes will be stored **in a hash table** and updated each time any attribute changes. **[How?]**

3. Elaboration: Building the Analysis Model

- The requirements engineer **takes** the information obtained during inception and elicitation and **begins to expand and refine it.**
- Elaboration focuses on **developing a refined requirements model which identifies various aspects of software function, behavior, and information.**
  - Elaboration is **driven by** the **creation and refinement of user scenarios** that describe how the end user (and other actors) will interact with the system.
  - Each user scenario is parsed to **extract analysis classes** - business domain entities that are visible to the end user.
  - The **attributes** of each **analysis class** are defined, and the **services** that are required by each class are identified.
  - The **relationships and collaboration** between classes are identified, and a variety of supplementary diagrams are produced.
- The **end result** is an **analysis model** that **defines** the **functional, informational, and behavioral domains of the problem .**
- **Analysis is the action that occurs as requirements are derived.**
- **Both (Analysis and Requirements) represent the modeling activity that defines various aspects of the problem to be solved.**

Elements of the Analysis Model

The requirements model as a bridge between the system description and the design model



The **requirements model** must achieve **three primary objectives**:

(1) To **describe** what the customer **requires,**

(2) To **establish** a basis for the **creation of a software design**, and

(3) To **define a set of requirements** that can be **validated once the software is built**.

The analysis model **bridges the gap between a system-level description (**which describes overall system or business functionality as it is achieved by applying software, hardware, data, human, and other system elements) and **a software design** (which describes the software's application architecture, user interface, and component-level structure).

Analysis Rules of Thumb

- The model should **focus** on requirements that are visible within the **problem or business domain**. The **level of abstraction** should be relatively **high**, i.e. no need to give details.
- Each element of the analysis model should **add to an overall understanding of software requirements** and **provide insight into the information domain, function and behavior of the system.**
- **Delay consideration** of infrastructure and other non-functional models until design.
- **Minimize coupling** throughout the system i.e., represent relationships between classes and functions. However, if the level of **"interconnectedness"** is extremely high, **effort should be made to reduce it.**
- **Be certain** that the analysis model provides value to all stakeholders.
- Keep the model as simple as it can be.

- The job of the toolsmith is to design and build tools that may be used by many people doing similar but not necessarily the same jobs.
- The role of the **domain analyst** is **to discover and define** analysis patterns, analysis classes, and related information that may be used by many people working on similar but not necessarily the same applications.
- Every member of a software team should have some understanding of the domain in which the software is to be placed.

Domain Analysis

- Example: The analysis of requirements for a new application indicates that 100 classes are needed.
  - Two teams are assigned to build the application, each will design and construct a final product.
  - Each team is populated by people with the same skill levels and experience.
  - Team A does not have access to a class library, and therefore, it must develop all 100 classes from scratch.
  - Team B uses a robust class library and finds that 55 classes already exist.
  - Which results in

1. Team B **will finish the project much sooner than** Team A.

2. **The cost** of Team B’s product will be **significantly lower than the cost** of Team A’s product.

3. The product produced by Team B will have **fewer delivered defects** than Team A’s product.

Domain Analysis

- The margin by which Team B's work would exceed Team A's accomplishments is open to debate, **few would argue that reuse provides Team B with a substantial advantage.**
- But **where did the "robust class library" come from?**
- **How were the entries in the library** determined to be appropriate for use in new applications?
- To answer these questions, **the organization that created and maintained the library had to apply domain analysis.**

Domain Analysis

Case Study - 1: Outline in one paragraph the information you would need to gather in order to perform domain analysis for the airline reservation system.

- You would attempt to learn as much as possible about
- How airline flights are scheduled;
- How fares are set and structured, and
- How ticketing and booking works in the customer’s airline and other airlines.
- You would study how the various people in the airline reservation business, including travel agents and airline employees, do their jobs;
- What existing reservation systems are capable of doing and how they work; and
- What laws, regulations and other rules govern the industry.
- You would study the functionality of competing reservation systems.
- You would also study the transition that has been happening from booking of flights by travel agents, to on-line booking of flights by the passengers.

Domain Analysis

Case Study - 2: Imagine you are performing a domain analysis in order to develop a new and better telephone response and dispatch system for medical emergencies. The system will be used by operators and paramedics who respond to calls to the emergency number 108. Summarize the information you would expect to learn.

Domain Analysis

- Software domain analysis **can be designed as a process of recognizing, analyzing and specifying of common requirements from a specific application domain.** **(In simple terms the process by which a software engineer learns background information.)**
- Software engineer has to **learn sufficient information** so as to be able **to understand the problem and make good decisions during requirements analysis and other stages of the software engineering process.**
- Identify the common requirements in the project like objects, classes, frameworks and can be reused, which **improves time-to market and reduces development costs.**
- Example:

The specific application domain may be 'bus reservation system' can be used for 'railway reservation system'.

Domain Analysis

Technical Domain:

- **Technical domain** of the software is related to the **common technical requirements which can be shared by many products.**
- Example: Most of the mobile applications use **common facilities called calling, sending messages, access to the Internet etc.**
- Many applications can be developed where we **do not write above requirements again and again.**
- They can be used by any applications once installed on the mobile phone.
- These activities use specific technical requirements that combine hardware with software.

Domain Analysis

Application Domain :

- The application domain is the **common library that contains the classes that can be used by other products to minimize their work.**
- **Specific application domain : Domain analysis helps in finding out common requirements of the software and its domain is created.**
- Example: In finance and banking, different financial products are offered to the customers such as different types of accounts (savings, current, joint etc.), fixed deposits, mutual funds, insurance, loans, etc., comes under specific application domain.
- Once it is created, many other software products can use it.

Domain Analysis

Goals / Benefits of Domain Analysis:

- Find out common requirement specification.
- To save the time – Faster development
- Better system
- Reduce the repeated or duplicate work.
- Reduction in the complications of the project.
- To make library of classes available.
- Anticipation of extensions - obtain insights into emerging trends and you will notice opportunities for future development.

Input and Output of domain analysis

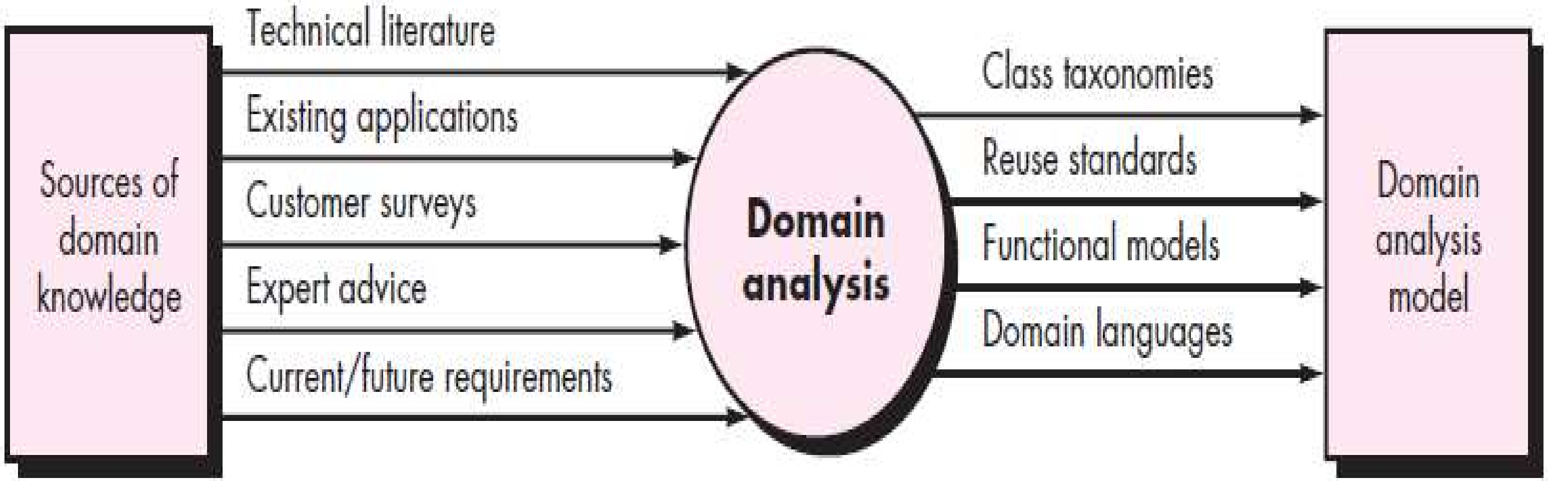


Figure shows the flow of the input and the output data in the domain analysis module.

- The main goal is to create the analysis classes and common functions.
- The input consists of the knowledge domain.
- The input is based on the technical survey, customer survey and expert advice.
- This data is then analyses, meaningful information comes out from this.
- The output domain consists of reusable classes, standards, functional models and domain language.

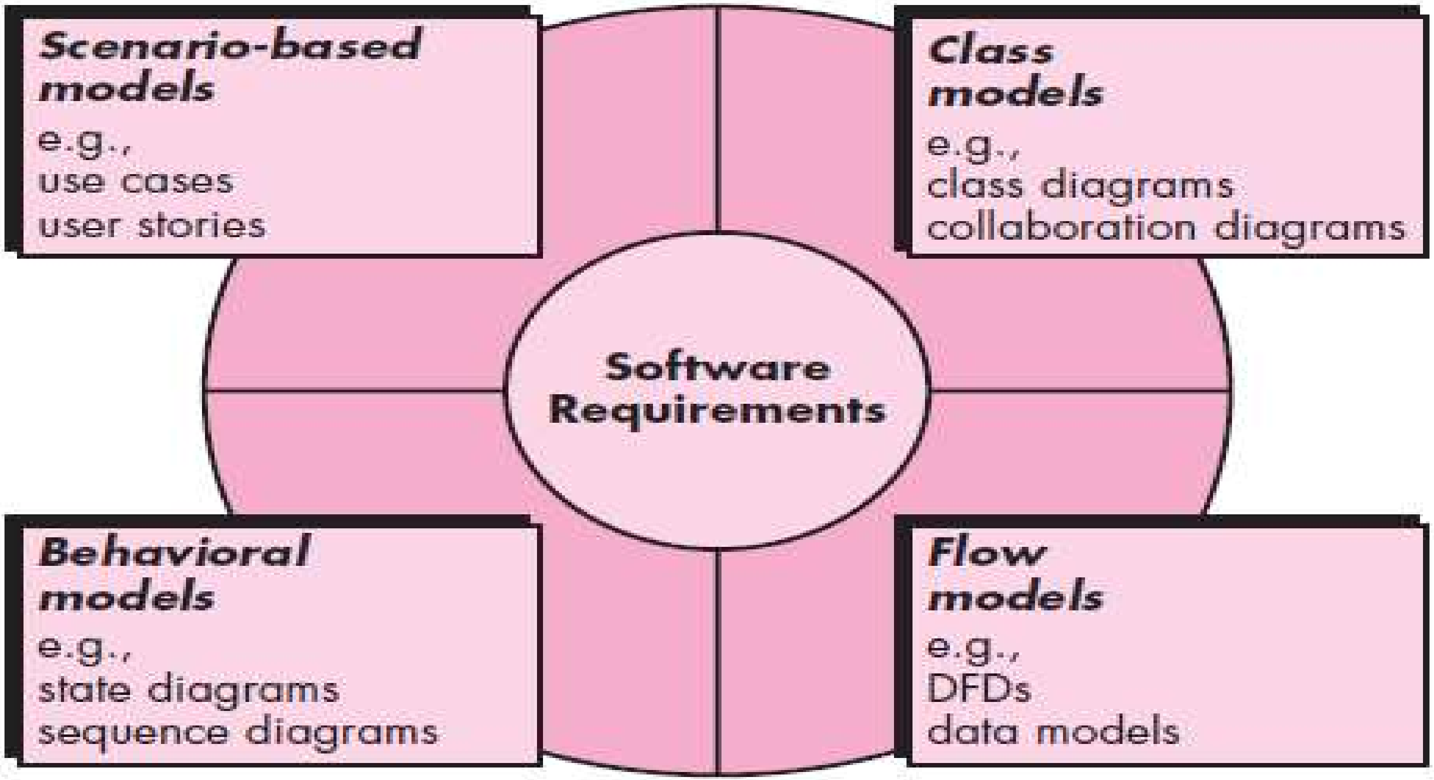
Note: Domain analysis should simply include a brief summary of the information that was found, along with references that will enable others to find that information. Not to write an excessive amount of detailed information. It is a waste of effort to duplicate the original source material;

Requirements Modeling Approaches

- **Structured Analysis and Object-Oriented Analysis**

Structured Analysis	Object-Oriented Analysis
focus is on the **process and procedures of the system.**	**focus** is on **data structure and real-world objects that are important.**
It uses System Development Life Cycle (SDLC) methodology for different purposes like planning, analyzing, designing, implementing, and supporting an information system.	It uses Incremental or Iterative methodology to refine and extend our design.
It is suitable for well-defined projects with stable user requirements.	It is suitable for large projects with changing user requirements.
Risk while using this analysis technique is high and reusability is also low.	Risk while using this analysis technique is low and reusability is also high.
Structuring requirements include DFDs (Data Flow Diagram), Structured Analysis, ER (Entity Relationship) diagram, CFD (Control Flow Diagram), Data Dictionary, Decision table/tree, and the State transition diagram.	Requirement engineering includes the Use case model (find Use cases, Flow of events, Activity Diagram), the Object model (find Classes and class relations, Object interaction, Object to ER mapping), State chart Diagram, and deployment diagram.
This technique is old and is not preferred usually.	This technique is new and is mostly preferred.

Elements of the Analysis Model / Requirements model



Each element of the requirements model presents the problem from a different point of view.

Elements of the Analysis Model

Scenario-based elements represent how the user interacts with the system and the specific sequence of activities that occur as the software is used. **Use Case diagrams, activity diagrams**

Class-based elements model the objects that the system will manipulate, the operations that will be applied to the objects to effect the manipulation, relationships (some hierarchical) between the objects, and the collaborations that occur between the classes that are defined. **Class diagrams, collaboration diagrams**

Behavioral elements represents how external events change the state of the system or the classes that reside within it. **State diagrams, Sequence diagrams**

Flow-oriented elements represent the system as an information transform, depicting how data objects are transformed as they flow through various system functions. **Data flow diagrams**

Scenario-Based Modeling

- Requirements modeling, it typically **begins** with scenario-based modeling because it identifies the possible use cases for the system and produces the use case diagram, to which all the other stages of requirements modeling refer.
- A scenario-based approach is used to describe the system from the perspective of the user.
- Scenario-based requirements model elements are frequently the first parts of the model to be produced.

Scenario-Based Modeling

Questions that must be answered if use cases are to provide value as a requirements modeling tool

- What are the main tasks or functions that are performed by the actor?
- What system information will the actor acquire, produce or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

Scenario-Based Modeling

Creating a Preliminary Use Case - SafeHome surveillance function (subsystem) identifies the following functions that are performed by the homeowner actor

- Select camera to view.
- Request thumbnails from all cameras.
- Display camera views in a PC window.
- Control pan and zoom for a specific camera.
- Selectively record camera output.
- Replay camera output.
- Access camera surveillance via the Internet.

In general, use cases are written first in an informal narrative fashion.

If more formality is required, the same use case is rewritten using a structured format.

Scenario-Based Modeling

Use case: Access Camera Surveillance via the Internet - Display Camera Views (ACS-DCV)

The stakeholder who takes on the role of the homeowner actor might write the following narrative.

Actor: homeowner

If I'm at a remote location, I can use any PC with appropriate browser software to log on to the *SafeHome Products* website. I enter my user ID and two levels of passwords and once I'm validated, I have access to all functionality for my installed *SafeHome* system. To access a specific camera view, I select "surveillance" from the major function buttons displayed. I then select "pick a camera" and the floor plan of the house is displayed. I then select the camera that I'm interested in. ***Alternatively***, I can look at thumbnail snapshots from all cameras simultaneously by selecting "all cameras" as my viewing choice. Once I choose a camera, I select "view" and a one-frame-per-second view appears in a viewing window that is identified by the camera ID. If I want to switch cameras, I select "pick a camera" and the original viewing window disappears and the floor plan of the house is displayed again. I then select the camera that I'm interested in. A new viewing window appears.

Scenario-Based Modeling

Use case: Access Camera Surveillance via the Internet - Display Camera Views (ACS-DCV)

A variation of a narrative use case presents the interaction as **an ordered sequence of user actions**.

Actor: homeowner

1. The homeowner logs onto the *SafeHome Products* website.
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects the “surveillance” from the major function buttons.
6. The homeowner selects “pick a camera.”
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the “view” button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

Sequential presentation does not consider any alternative interactions

Scenario-Based Modeling

Refining a Preliminary Use Case:

Each step in the primary scenario is evaluated by asking the following questions

- Can the actor take **some other action** at this point?
- Is it possible that the actor will encounter some **error condition at this point**? If so, what might it be?
- Is it possible that the actor will **encounter some other behavior at this point** (e.g., behavior that is invoked by some event outside the actor's control)? If so, what might it be?

- **Answers to these questions result in the creation of a set of secondary scenarios that are part of the original use case but represent alternative behavior.**
- **Example: consider steps 6 and 7 in the primary scenario presented in the previous slide:**

6. The homeowner selects "pick a camera."
7. The system displays the floor plan of the house.

Scenario-Based Modeling

6. The homeowner selects “pick a camera.”
7. The system displays the floor plan of the house.

Refining a Preliminary Use Case:

Can the actor take some other action at this point?

Yes, the actor may **choose to view thumbnail snapshots** of all cameras simultaneously **secondary scenario** might be “View thumbnail snapshots for all cameras.”

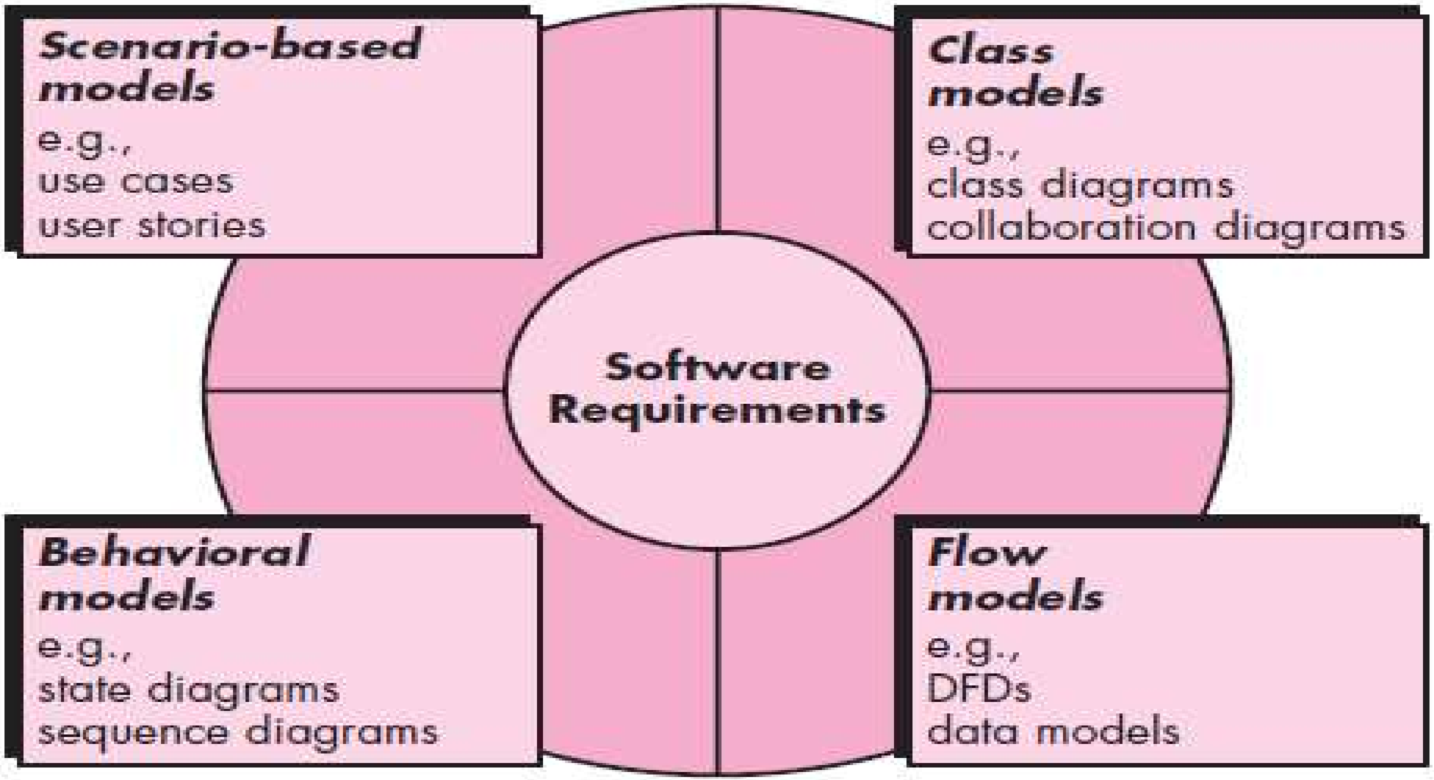
Is it possible that the actor will encounter some error condition at this point?

A floor plan with camera icons may have never been configured. Hence, **selecting “pick a camera” results in an error condition:** “No floor plan configured for this house.” This error condition becomes a **secondary scenario**.

Is it possible that the actor will encounter some other behavior at this point?

“yes.” the system **may encounter an alarm condition** which results in the system displaying a special alarm notification (type, location, system action) and providing the actor with a number of options relevant to the nature of the alarm. A **separate use case - Alarm condition encountered** - would be developed and referenced from other use cases as required.

Elements of the Analysis Model / Requirements model



Each element of the requirements model presents the problem from a different point of view.

Scenario-Based Modeling

Refining a Preliminary Use Case:

Questions to be asked in each step of the primary scenario for evaluation:

- Can the actor take **some other action** at this point?
- Is it possible that the actor will encounter some **error condition at this point**? If so, what might it be?
- Is it possible that the actor will **encounter some other behavior at this point** (e.g., behavior that is invoked by some event outside the actor's control)? If so, what might it be?

- **Answers to these questions result in the creation of a set of secondary scenarios that are part of the original use case but represent alternative behavior.**

Scenario-Based Modeling

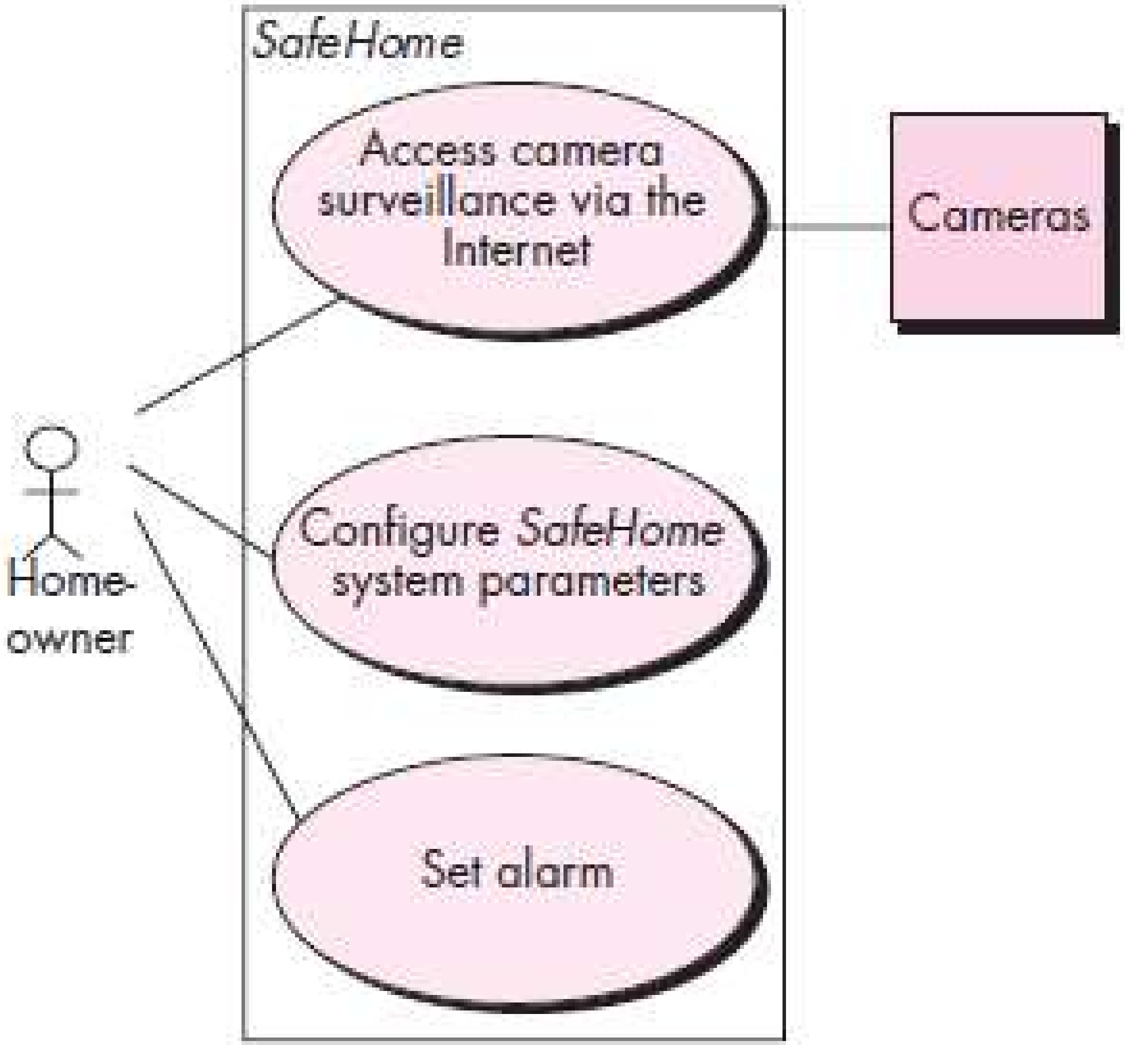
Exceptions

Describe situations (either a **failure condition** or an **alternative chosen** by the actor) that **cause the system to exhibit unusual behavior.**

An Exception is anything that leads to NOT achieving the use case's goal.

- **Brainstorming** should be **used to derive a reasonably complete set of exceptions for each use case**
  - Are there cases **where a supporting function (actor) fails to respond appropriately?**
  - Are there cases **where a validation function occurs for the use case?**
  - **Can poor system performance result** in unexpected or improper use actions?
  - **Handling exceptions** may require the **creation of additional use cases**

Scenario-Based Modeling

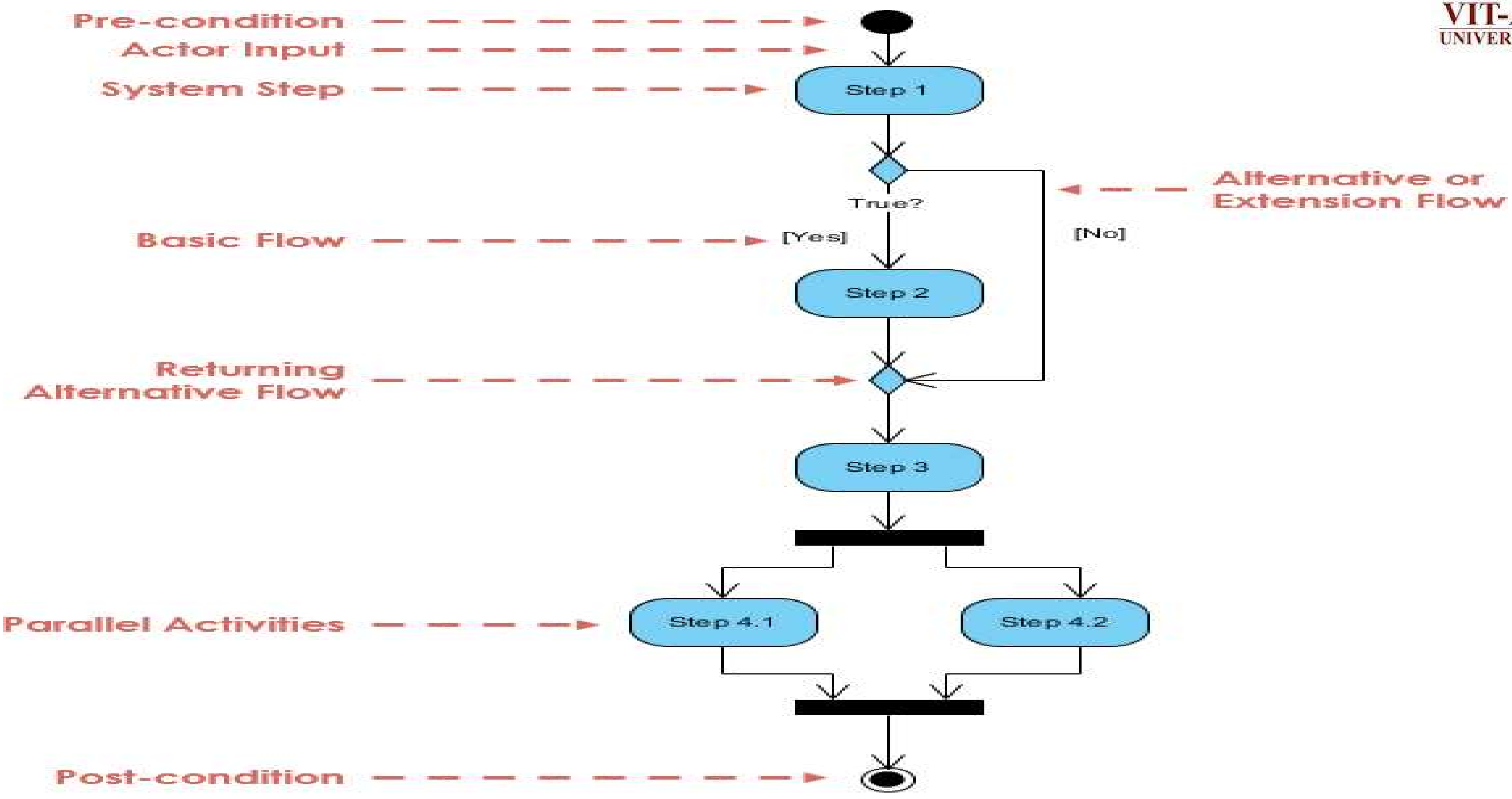

SafeHome
Access camera surveillance via the Internet
Cameras
Homeowner
Configure SafeHome system parameters
Set alarm

Preliminary use-case diagram for the *SafeHome* system

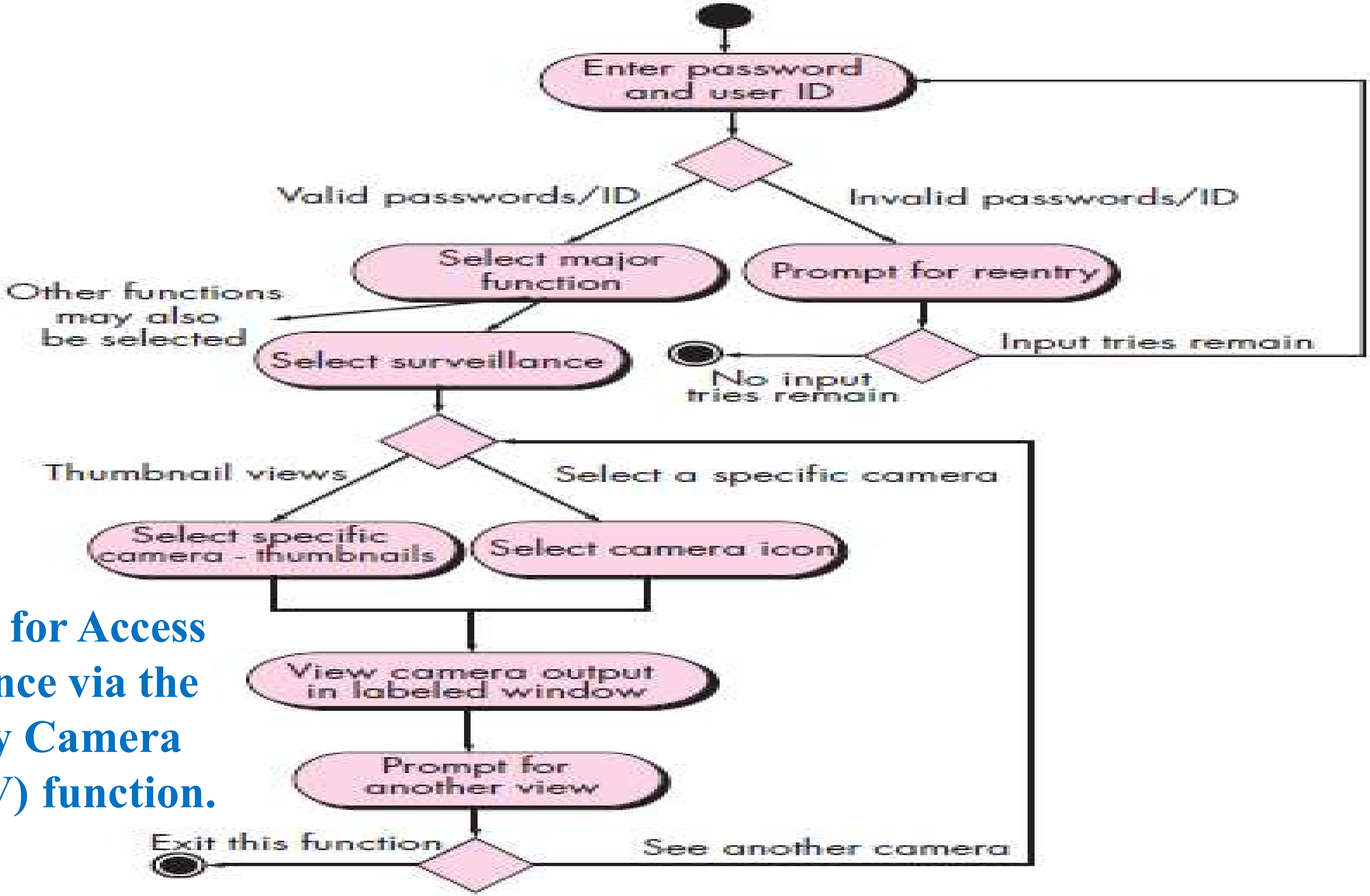
Scenario-Based Modeling - Activity Diagram

- A UML **activity diagram** **represents the** **actions and decisions** **that occur as some function is performed.**
- It **supplements the use case** by providing a **graphical representation** of the **flow of interaction within a specific scenario.**
- It is **essentially an advanced version of a flow chart** that modeling **the flow from one activity to another activity.**

Scenario-Based Modeling - Activity Diagram



Scenario-Based Modeling - Activity Diagram



Activity diagram for Access camera surveillance via the Internet - Display Camera Views (ACS-DCV) function.

When to Use Activity Diagram

- Activity Diagrams **describe how activities are coordinated to provide a service which can be at different levels of abstraction**.

1.Identify candidate use cases, through the examination of business workflows.

2.Identify pre - and post - conditions (the context) for use cases.

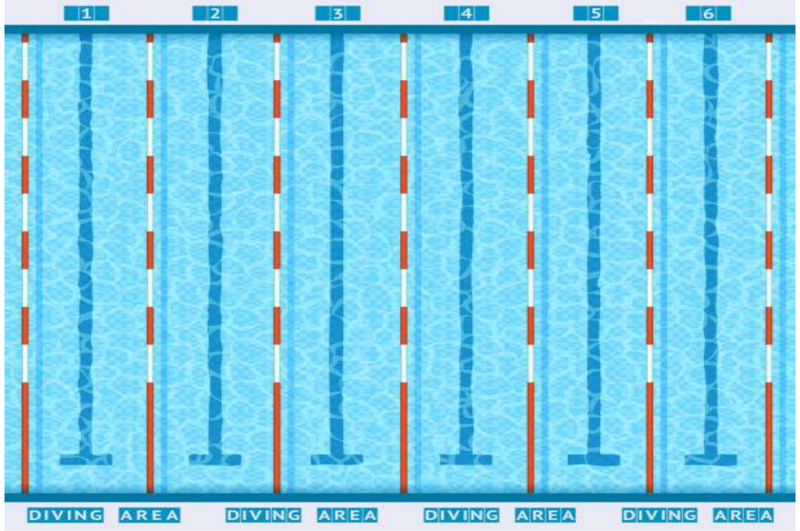
3.Model **workflows between / within** use cases.

4.Model complex **workflows in operations** on objects.

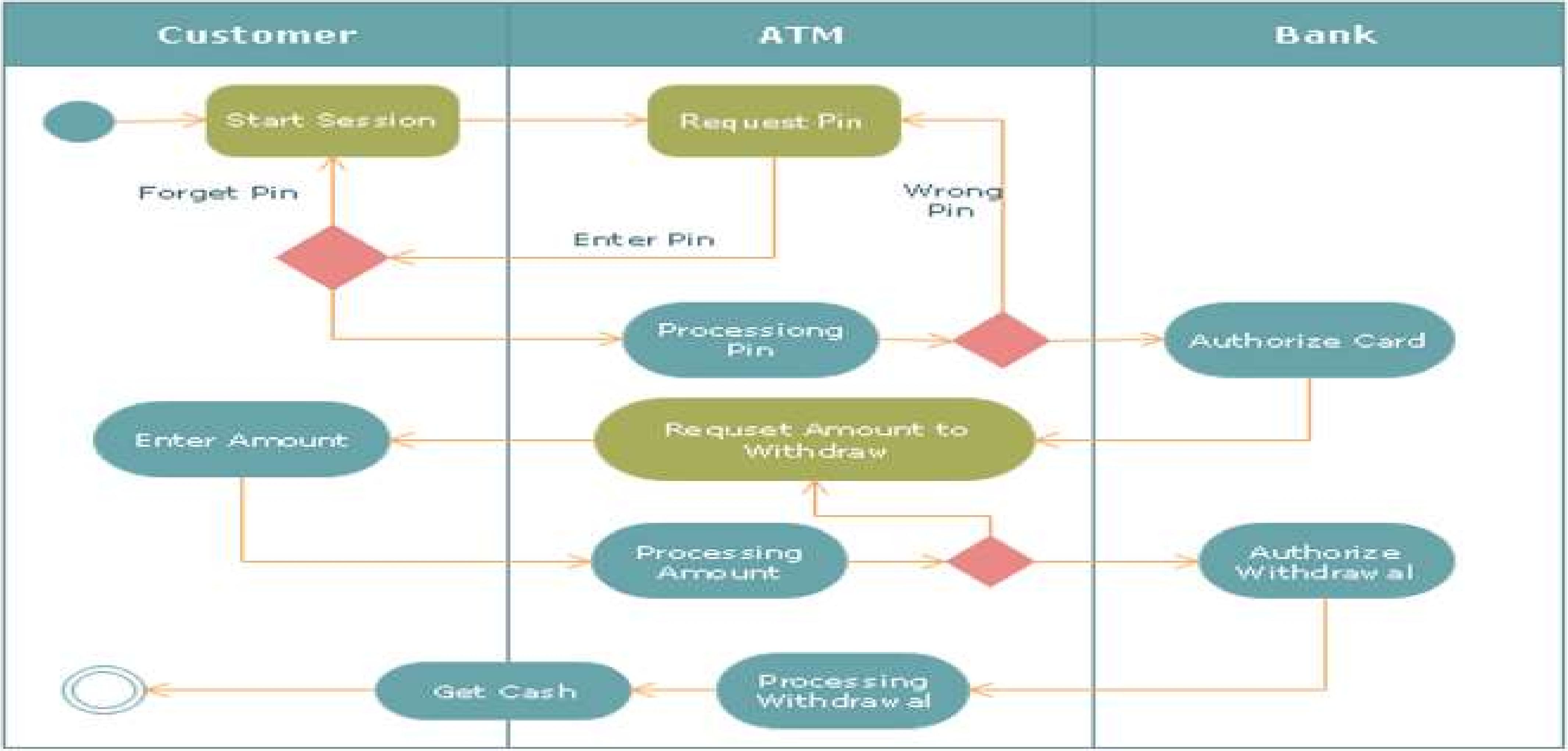
5.Model **in detail complex activities** in a **high level activity Diagram**.

Scenario-Based Modeling - Swimlane Diagrams

- The UML ***swimlane diagram*** is a useful **variation of the activity diagram** and **allows to represent the flow of activities** described by the use case and at the same time **indicate which actor** (if there are multiple actors involved in a specific use case) **or analysis class** has **responsibility** for the action described by an activity **rectangle**.
- **Responsibilities** are **represented as parallel segments** that divide the diagram **vertically**, like the **lanes in a swimming pool.**

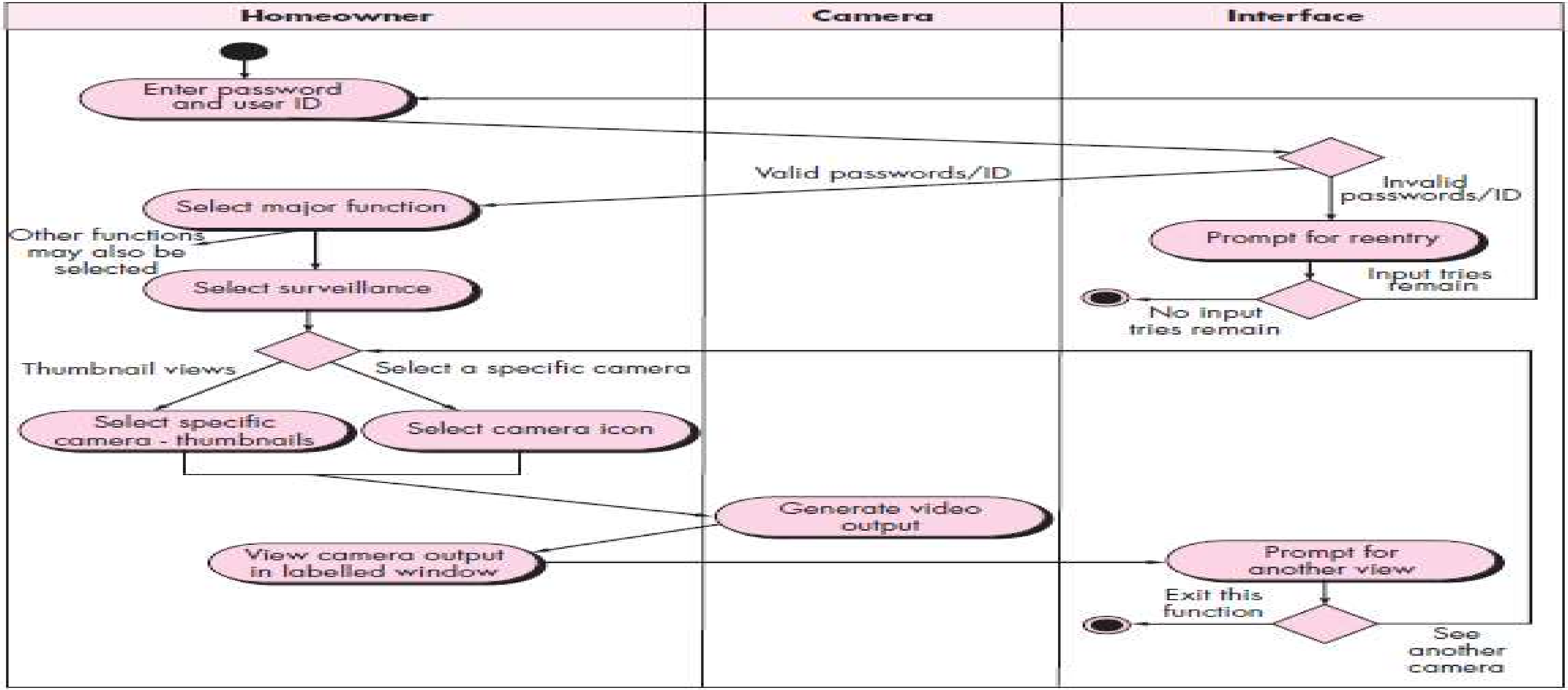


Scenario-Based Modeling - Swimlane Diagrams

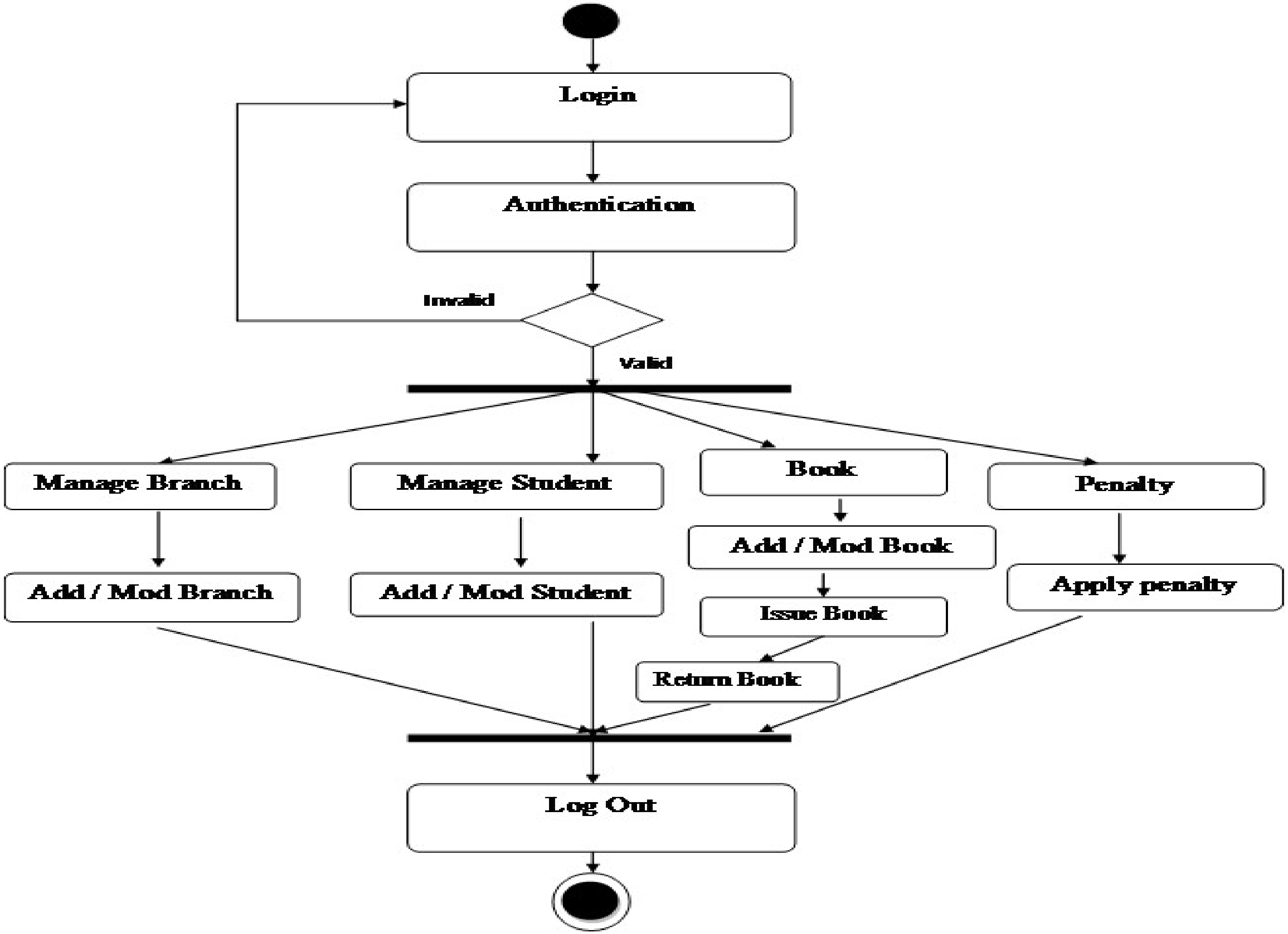


Scenario-Based Modeling - Swimlane Diagrams

Three analysis classes - **Homeowner, Camera,** and **Interface** - have direct or indirect responsibilities in the context of the activity diagram



Scenario-Based Modeling – Activity Diagram for Library Management System



- **Two analysis** classes - **Librarian, Library Management System.**
- **Draw the Swimlane Diagram**

Data Modelling Concepts:

- Data Modeling in software engineering is the **process of simplifying the diagram** or **data model of a software system** by applying certain formal techniques.
- It involves **expressing data with diagrams**, **symbols, or text to visualize the interrelation.**
- The data model **provides** the **blueprint for building a new database or reengineering legacy applications.**
- Data Modeling thus **helps** to increase consistency in **naming, rules, semantics, and security.**

Data Modelling Concepts:

- A software engineer or analyst defines all **data objects** that are processed within the system, the **relationships** between the data objects, and **other information** that is relevant to the relationships.

  ER (Entity-Relationship) Model

  - Based on the notion of real-world entities and relationships among them.
  - It creates an entity set, relationship set, general attributes, and constraints.
  - **The data model consists of three interrelated pieces of information:**
  - An entity is a **real-world object;An employee is an entity in an employee database.**
  - An attribute is a **property with value,** and **entity sets** share **attributes of identical value.**
  - Finally, there is the relationship between **entities.**

Data Modelling Concepts:

Data Objects

- A data object **defines** a composite data item processed by software, i.e., it incorporates a collection of individual data items (attributes) and gives the collection of items a name (the name of the data object).

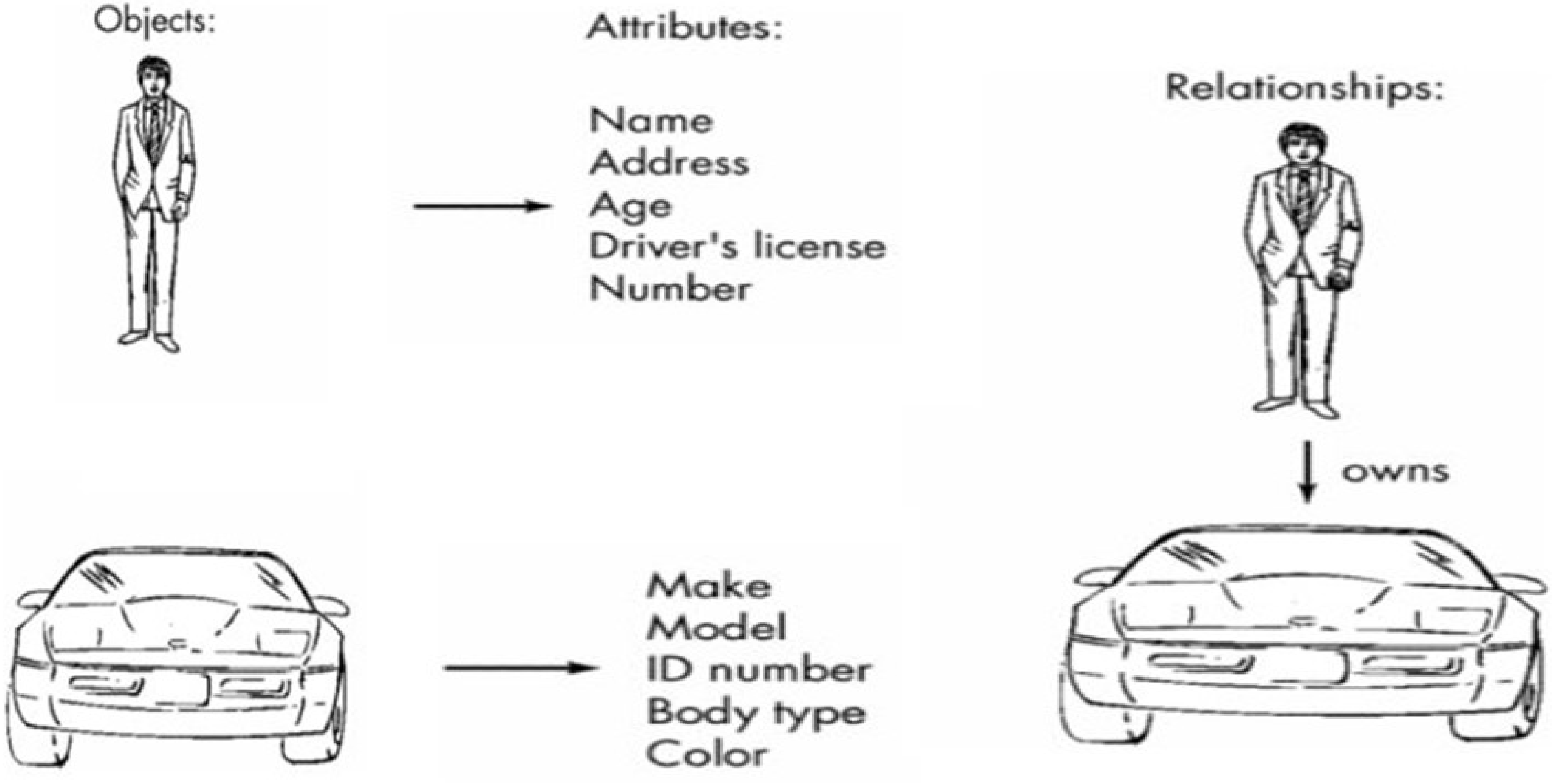
- A **data object** can be
  - An external entity (e.g., anything that produces or consumes information),
  - A thing (e.g., a report or a display),
  - An occurrence (e.g., a telephone call) or an event (e.g., an alarm),
  - A role (e.g., salesperson),
  - An organizational unit (e.g., accounting department, monitoring agency),
  - A place (e.g., a warehouse), or
  - A structure (e.g., a file)

Data Modelling Concepts:

Data Objects Attributes:

- Attributes define the **properties of a data object.**
- The attribute is a **quality or characteristic** that defines a person, group, or data objects, which can have a single or multiple or range of values.
- **Three types of attributes:**
  - **Naming attributes** – Make and model are naming attributes in a vehicle data object.
  - **Descriptive (Relationship) attributes** – used to describe the characteristics or features or the relationship of the data object.
    - In a vehicle, the color of a data object is a descriptive attribute that describes the features of the object.
  - **Referential attribute** – used to formalize **binary and associative relationships** and in making **reference to another instance in another table.**

Data Modelling Concepts:



Objects:

Attributes:

Name
Address
Age
Driver's license
Number

Make
Model
ID number
Body type
Color

Relationships:

owns

Data Modelling Concepts:

- **The relationship** represents **the connection or relation between different data objects** and **describes association among entities.**
- **Relationships are of three types:**
  - **One-to-many,**
  - **Many-to-many, and**
  - **Many-to-one.**
- **Cardinality** is the specification of the number of occurrences of one [object] that can be related to the number of occurrences of another [object].
- **Cardinality** is usually expressed as simply **'one' or 'many.'**
- Example:
  - A husband can have only one wife (in most cultures), while a parent can have many children.

Data Modelling Concepts:

- **One-to-one (l:l) -** An occurrence of [object] 'A' can relate to one and only one occurrence of [object] 'B,' and an occurrence of 'B' can relate to only one occurrence of 'A'
- **One-to-many (l:N) -** One occurrence of [object] 'A' can relate to one or many occurrences of [object] 'B,' but an occurrence of 'B' can relate to only one occurrence of 'A'

Example: A mother can have many children, but a child can have only one mother.

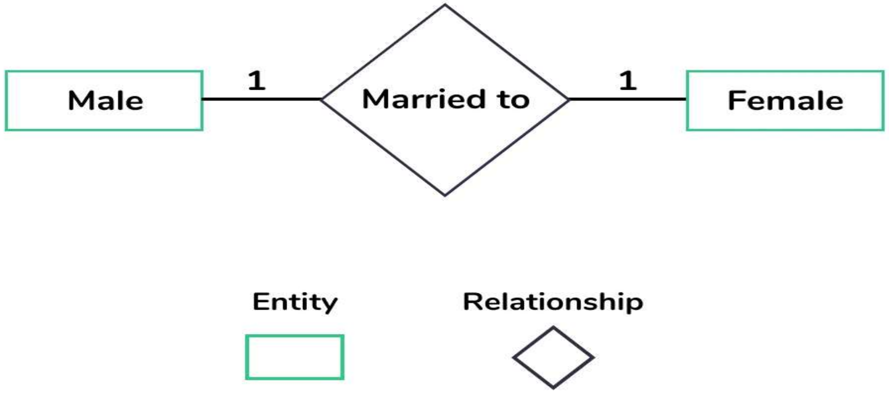
- **Many-to-many (M:N)** - An occurrence of [object] 'A' can relate to one or more occurrences of 'B,' while an occurrence of 'B' can relate to one or more occurrences of 'A‘

The ***cardinality*** of an object-relationship pair specifies “the number of occurrences of one [object] that can be related to the number of occurrences of another [object]”.

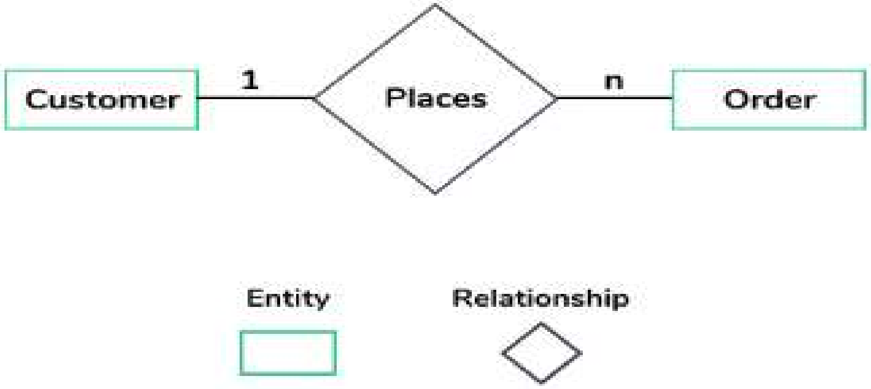
The ***modality*** of a relationship is **0 if there is no explicit need for the relationship to occur or the relationship is optional.**

The modality is **1 if an occurrence of the relationship is mandatory.**

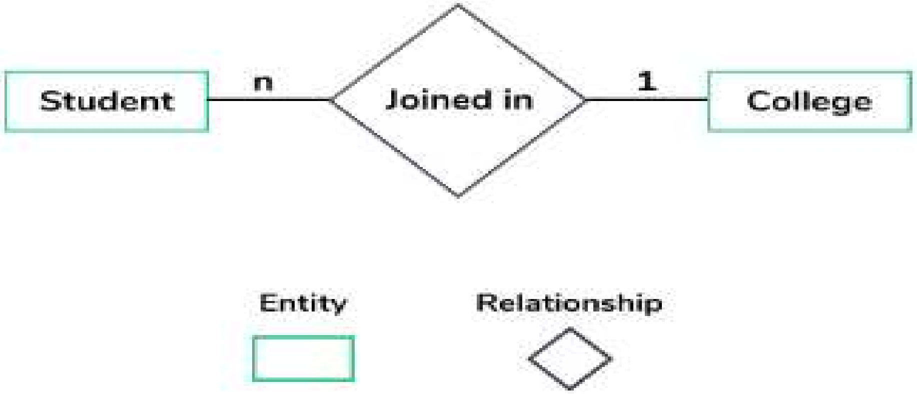
One to One Cardinality



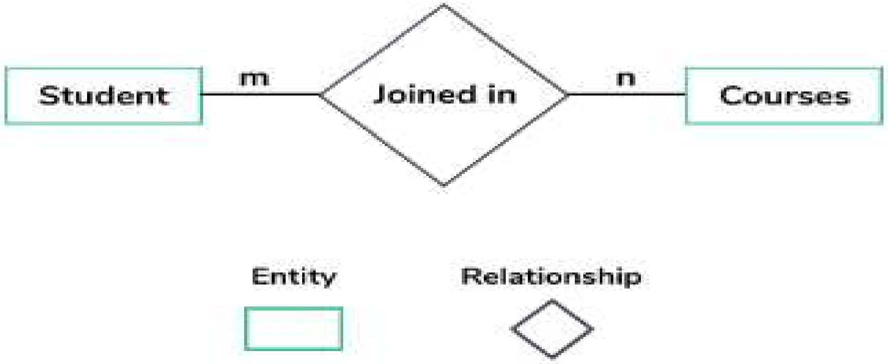
One to Many Cardinality

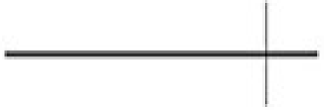
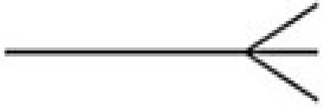
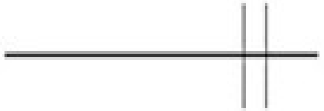
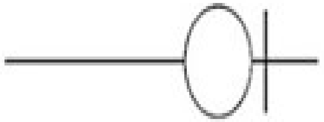
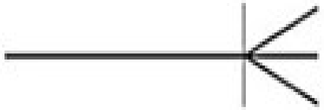
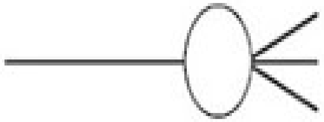


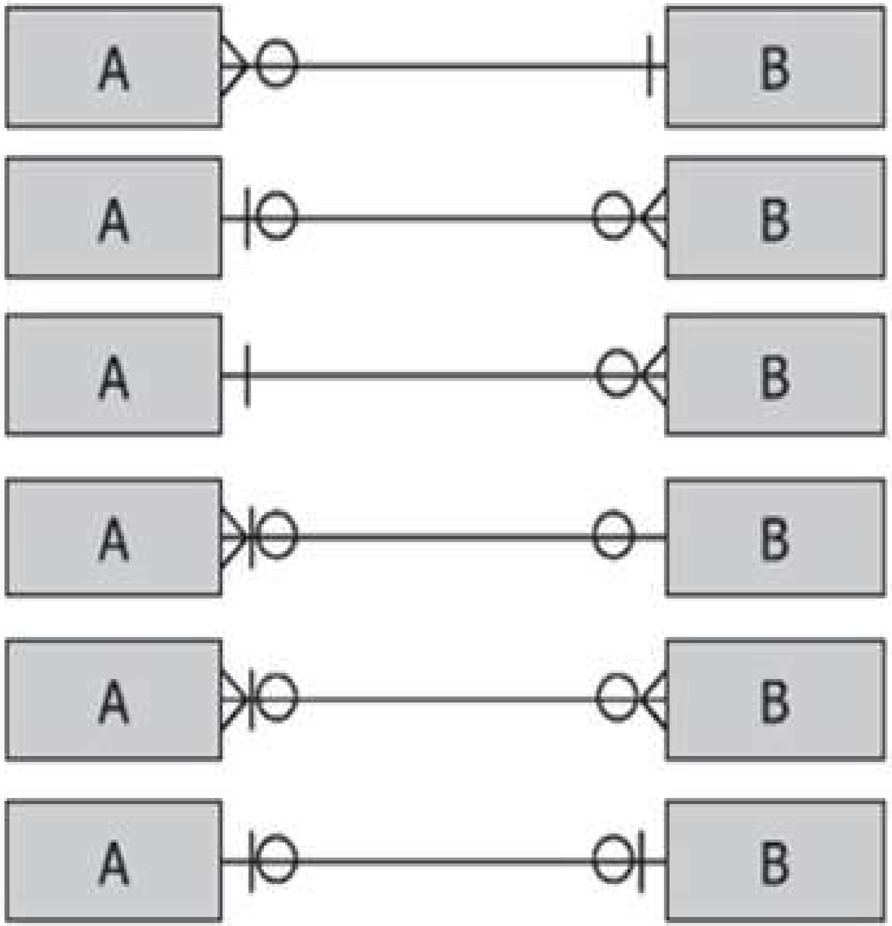
Many to One Cardinality

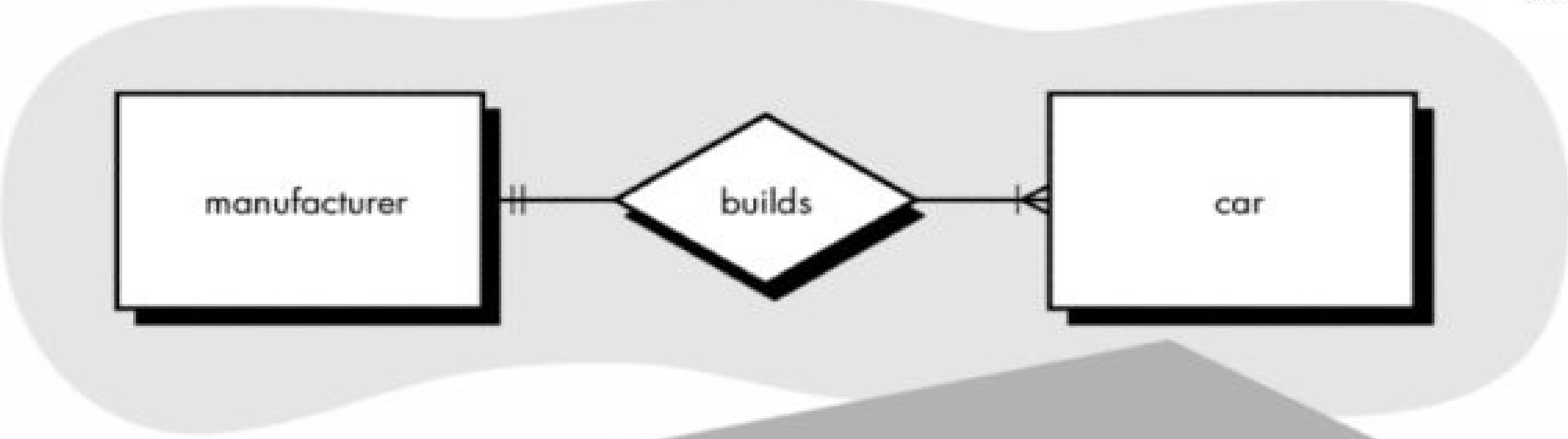


Many to Many Cardinality



Symbol	Meaning	Number
	One	N/A
	Many	N/A
	Mandatory-One	Exactly one
	Optional-One	Zero or one
	Mandatory-Many	One or More
	Optional-Many	Zero or more





Data object table

ID#	Model	Body type	Engine	Transmission	• • •

Bank ER diagram: Entities and their **Attributes**:

- **Bank Entity :**
  - Attributes of Bank Entity are Bank Name, Code and Address. Code is Primary Key for Bank Entity.
- **Customer Entity :**
  - Attributes of Customer Entity are Customer_id, Name, Phone Number and Address. Customer_id is Primary Key for Customer Entity.
- **Branch Entity :**
  - Attributes of Branch Entity are Branch_id, IFSC, Name and Address. Branch_id is Primary Key for Branch Entity.
- **Account Entity :**
  - Attributes of Account Entity are Account_number, Account_Type and Balance. Account_number is Primary Key for Account Entity.
- **Loan Entity :**
  - Attributes of Loan Entity are Loan_id, Loan_Type and Amount. Loan_id is Primary Key for Loan Entity.

Relationships are :

- **Bank has Branches**
  - 1:N One Bank can have many Branches but one Branch can not belong to many Banks, so the relationship between Bank and Branch is one to many relationship.

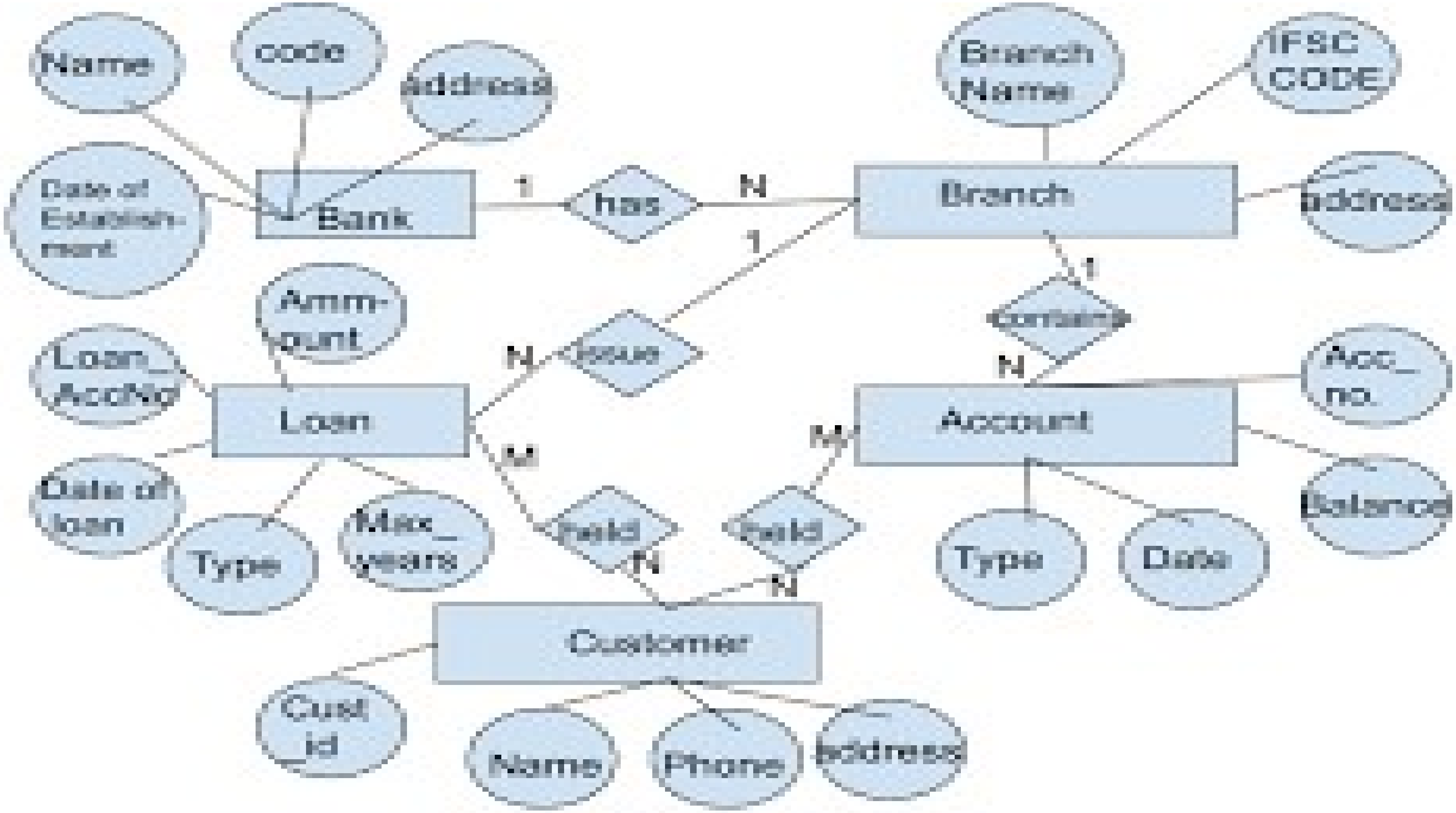
- **Branch maintain Accounts**
  - 1:N One Branch can have many Accounts but one Account can not belong to many Branches, so the relationship between Branch and Account is one to many relationship.

- **Branch offer Loans**
  - 1:N One Branch can have many Loans but one Loan can not belong to many Branches, so the relationship between Branch and Loan is one to many relationship.

Relationships are : Contd….

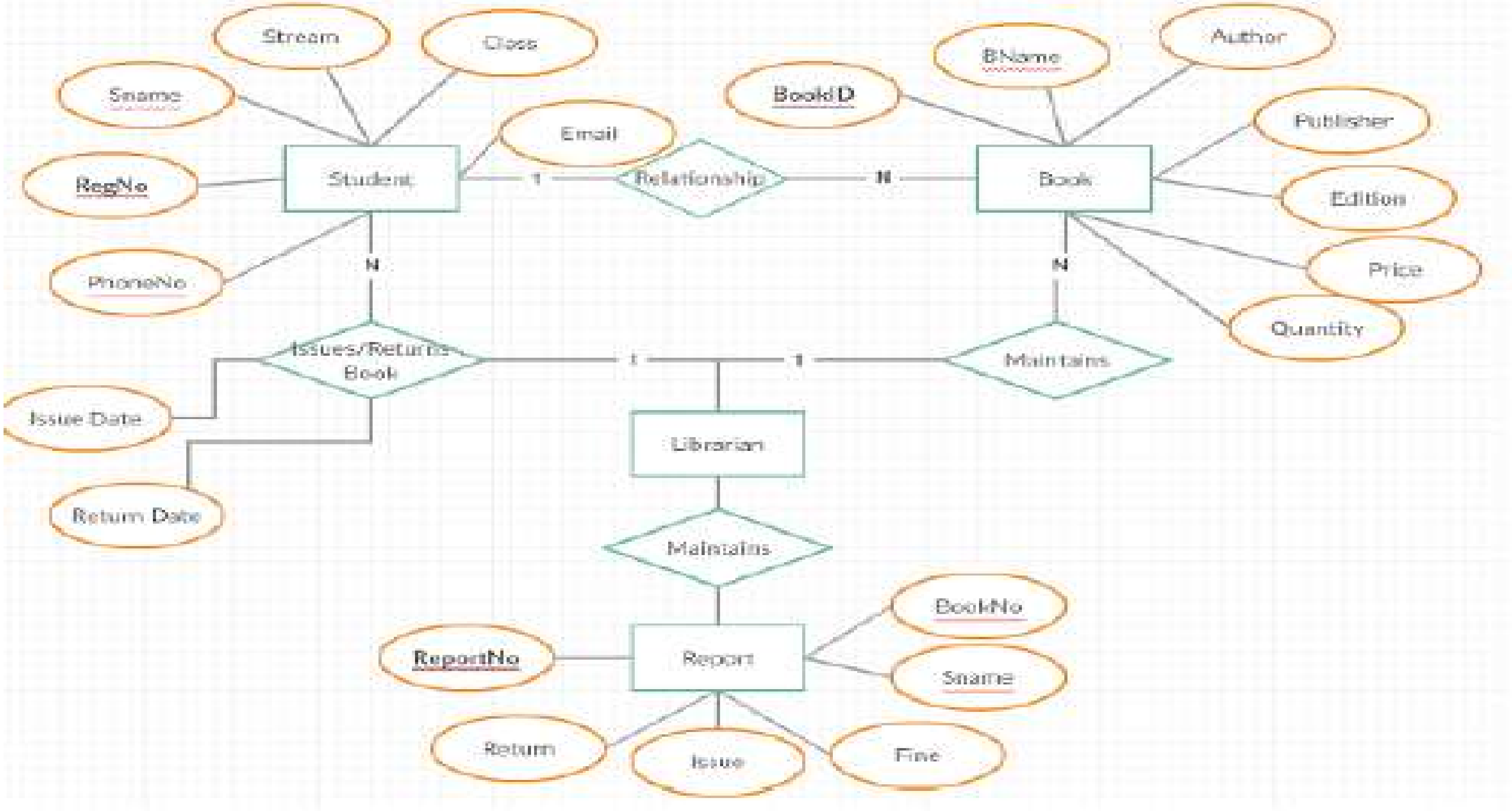
- **Account held by Customers**
- **M : N** One Customer can have more than one Accounts and also One Account can be held by one or more Customers, so the relationship between Account and Customers is many to many relationship.

- **Loan availed by Customer**
- **M : N** (Assume loan can be jointly held by many Customers).
  One Customer can have more than one Loan and also One Loan can be availed by one or more Customers, so the relationship between Loan and Customers is many to many relationship.



Name
code
address
Date of Establish-ment
Bank
1
has
N
Branch
Branch Name
IFSC CODE
address
1
1
contains
N
Amm-ount
Loan_AccNo
Loan
N
issue
Date of loan
Type
Max_years
M
held
N
M
held
N
Account
Acc_no.
Balance
Type
Date
Customer
Cust_id
Name
Phone
address

Library Management System ER diagram:


Stream
Class
Sname
Email
RegNo
Student
1
Relationship
N
Book
BookID
BName
Author
Publisher
Edition
Price
Quantity
PhoneNo
N
Issues/Returns Book
Issue Date
Return Date
1
1
Maintains
N
Librarian
Maintains
Report
ReportNo
BookNo
Sname
Return
Issue
Fine