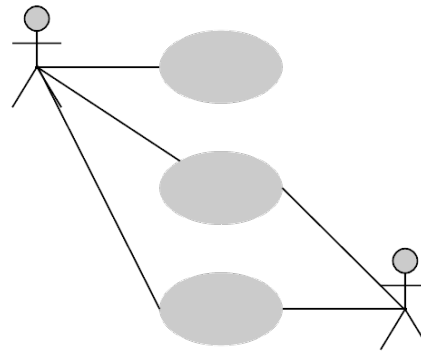


Object Oriented Design

Lecture 02

Use Case Diagram



Why Design Systems ?

“Without good software design, programming is an art of adding bugs to an empty text file” -Louise Srygley

Have you ever developed a software without a design?

- How easy it was to make **changes** to your code?
- Did a small code change produce a **ripple-effect** for changes elsewhere in the code?
- Was your code hard to **reuse**?
- Was the software difficult to **maintain** after a release?

A Good Software Design;

- Allows changes
- More flexible
- Increases reusability
- Easy to understand
- Cost effective

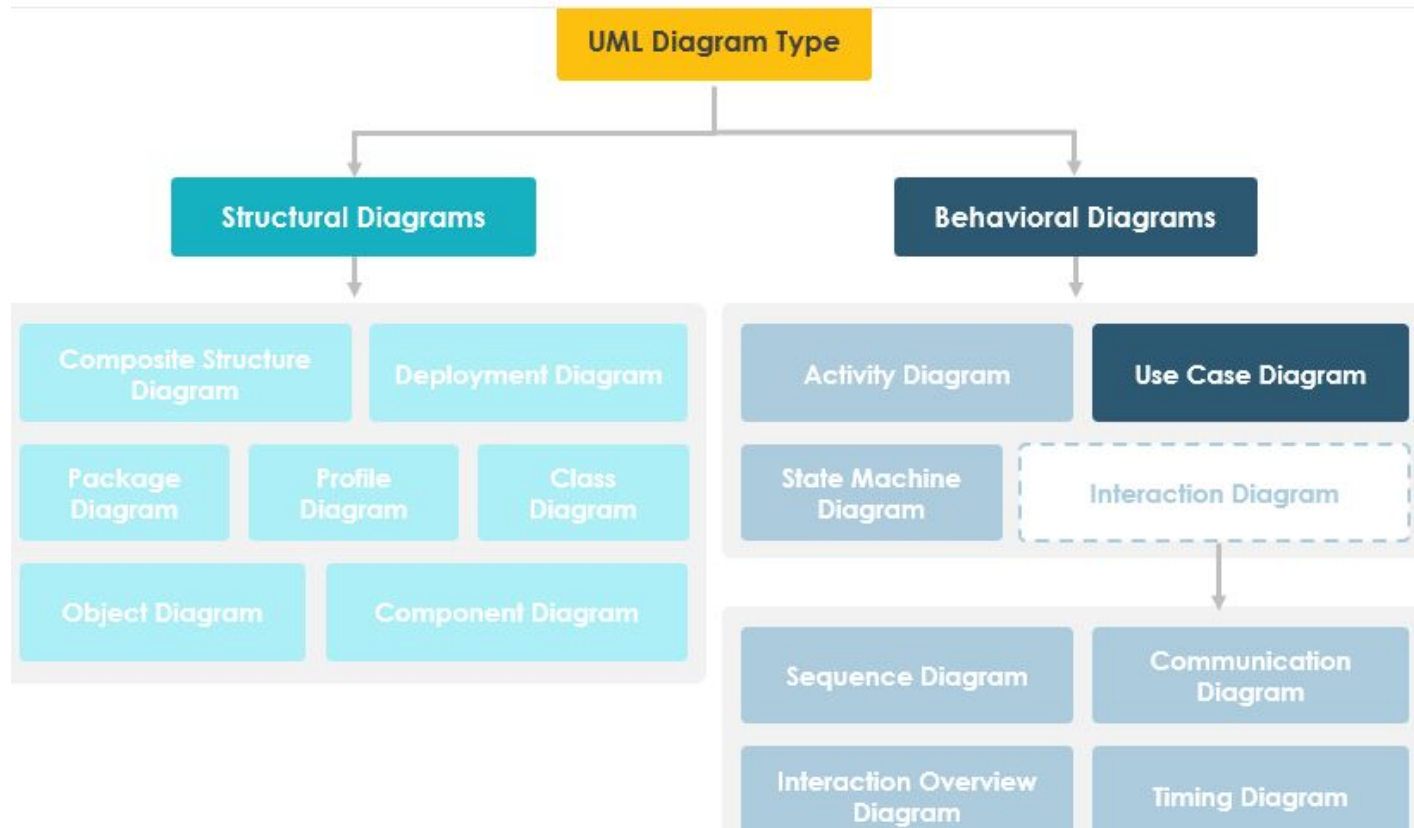
UML (Unified Modeling Language)

Provides a standard way to visualize the design of a system.

UML can be described as the successor of object-oriented (OO) analysis and design.

An object contains both data and methods that control the data.

UML Diagrams



Use Case Diagram

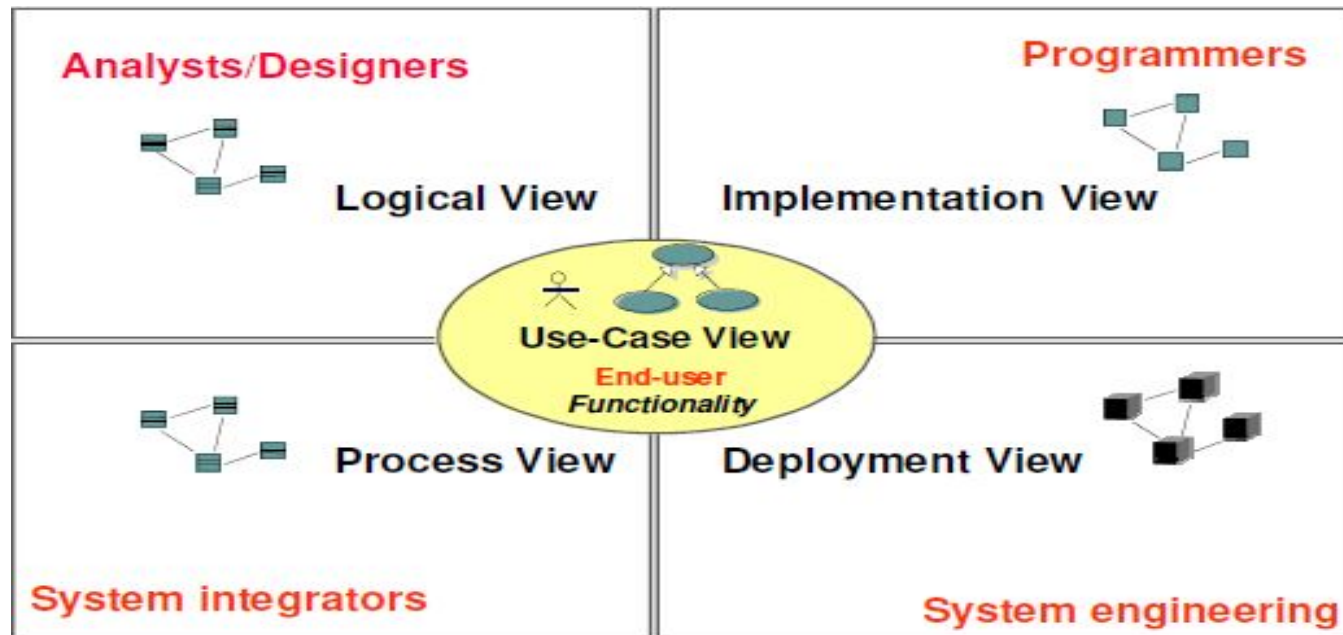
- Captures the **Dynamic** behavior of the system.
- Dynamic behavior shows how the system works when it is running/operating.
- A use case diagram is usually simple. It does not show the detail of the use cases:
 - It only summarizes **some of the relationships** between use cases, actors, and systems.
 - It does **not show the order** in which steps are performed to achieve the goals of each use case.
- Use Case diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.

Hence to model the entire system, a number of use case diagrams are used.

Use Case Diagram

- It is an excellent way to communicate to management customers and other non development people.
- WHAT a system will do when it is completed.
- BUT..it does not go into detail of HOW a system will do anything.
- It illustrates to the development team exactly what is expected from a system.

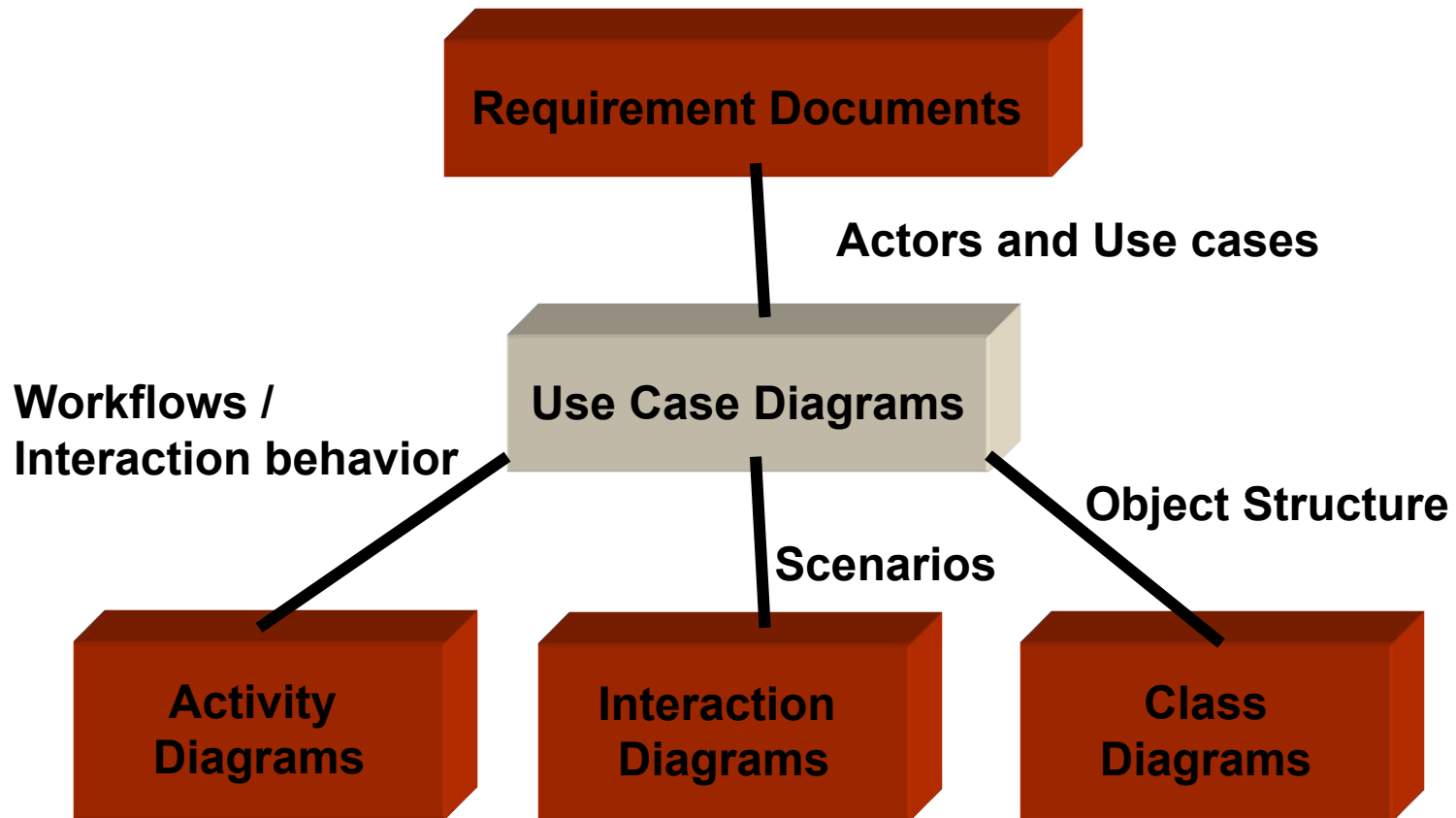
Views of a System



The Use Case View

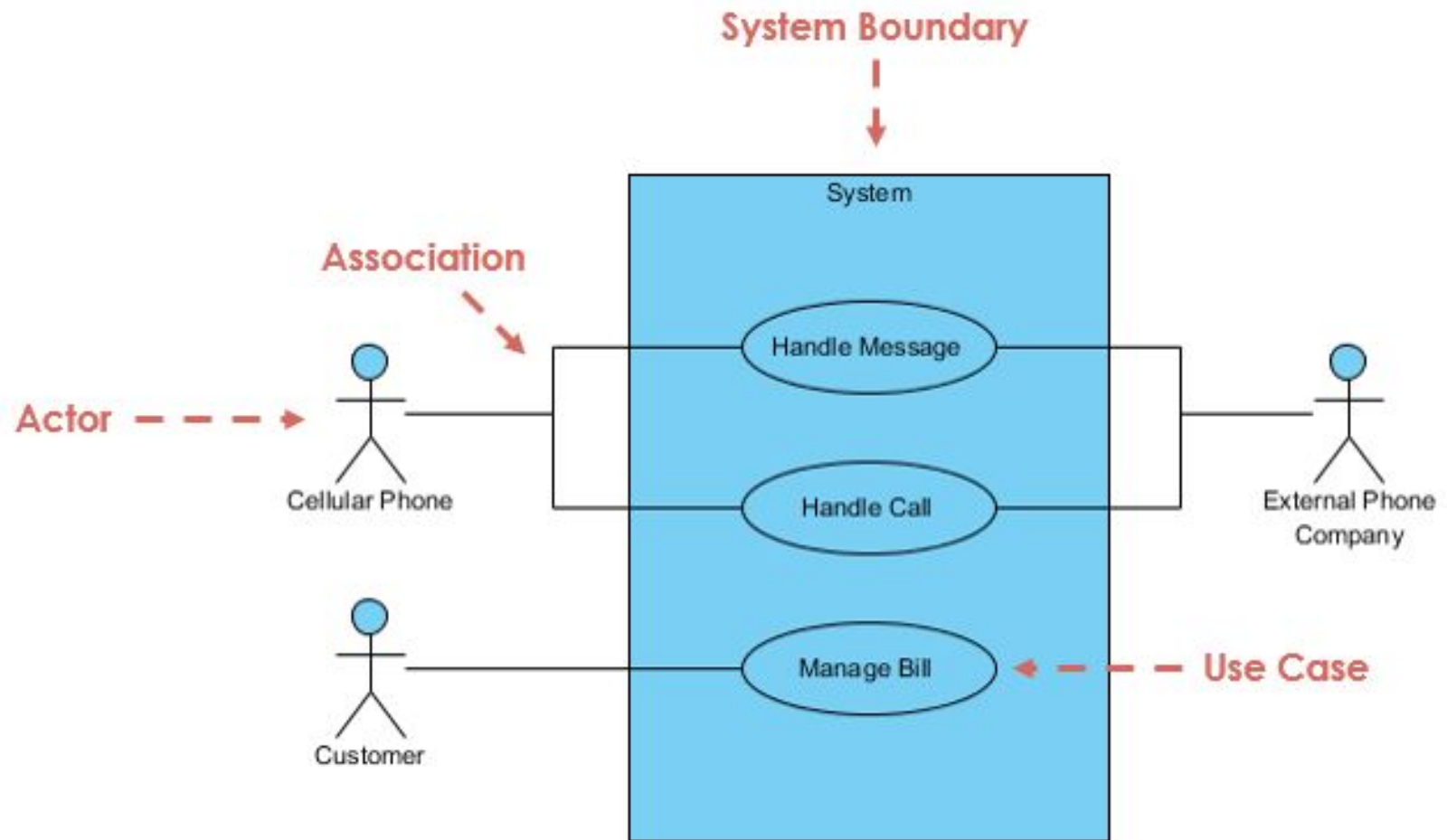
- Use Case View describes the behavior of the system as seen by its end users, analysts and testers.

Role of Use Case Diagrams in UML



Components of a Use Case Diagram

- System : Something that performs function(s).
- Actors : The roles adopted by those participating.
- Use Cases : High level/abstracted activities to be supported by the system.
- Relationships / Links : Which actors are involved in which use cases (dependency, generalization, and association)



System / Subsystem

- A system is a piece or multiple pieces of software that performs some sort of function for its users.
- When modeling a very large system that could benefit from being broken down into more maintainable pieces.

Example :

Banking Application

- Account Creation
- Money Transaction
- Balance Check
- Apply Loan

Actors

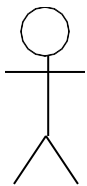
- An Actor represents an entity in the outside world that takes on the role of interacting with the software system.

- An actor can be

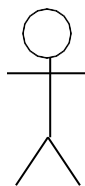
A user (i.e. Human)

Another software system

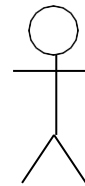
A hardware device (with embedded software)



Customer



Paypal



Printer

How to Determine the Actors?

- who uses the system?
- who gets information from the system?
- who provides information to the system?
- who installs, starts up or maintains the system?

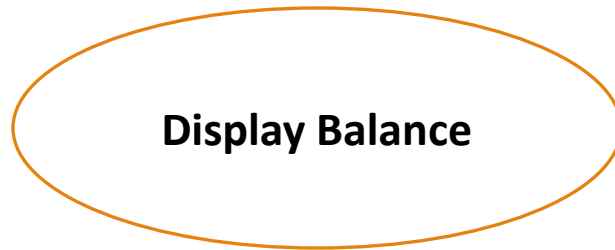
Task one – Identifying Actors

It is required to automate the following manual process on student assessments.

- Traditionally, a student's performance on each taught module in the Software Engineering course can be assessed in two ways:
 - By written report before, during and after the module, which is marked by the lecturers.
 - By a written exam, which is taken by the students under formal invigilated conditions and then marked by the lecturers.
- The lecturers may choose what weight is given to each form of assessment or, indeed, whether one of the two forms is dropped.
- For audit purposes, copies of all assessment material and marking schemes must be presented to the department's office for safekeeping.
- After the completion of each module's assessment or exam, as appropriate, student marks are presented to the registry.
- To get the student details the registration system is accessed.

Use Cases

- Use Case represents a sequence of events (actions) which combined to form some interaction between the software system and the outside world.

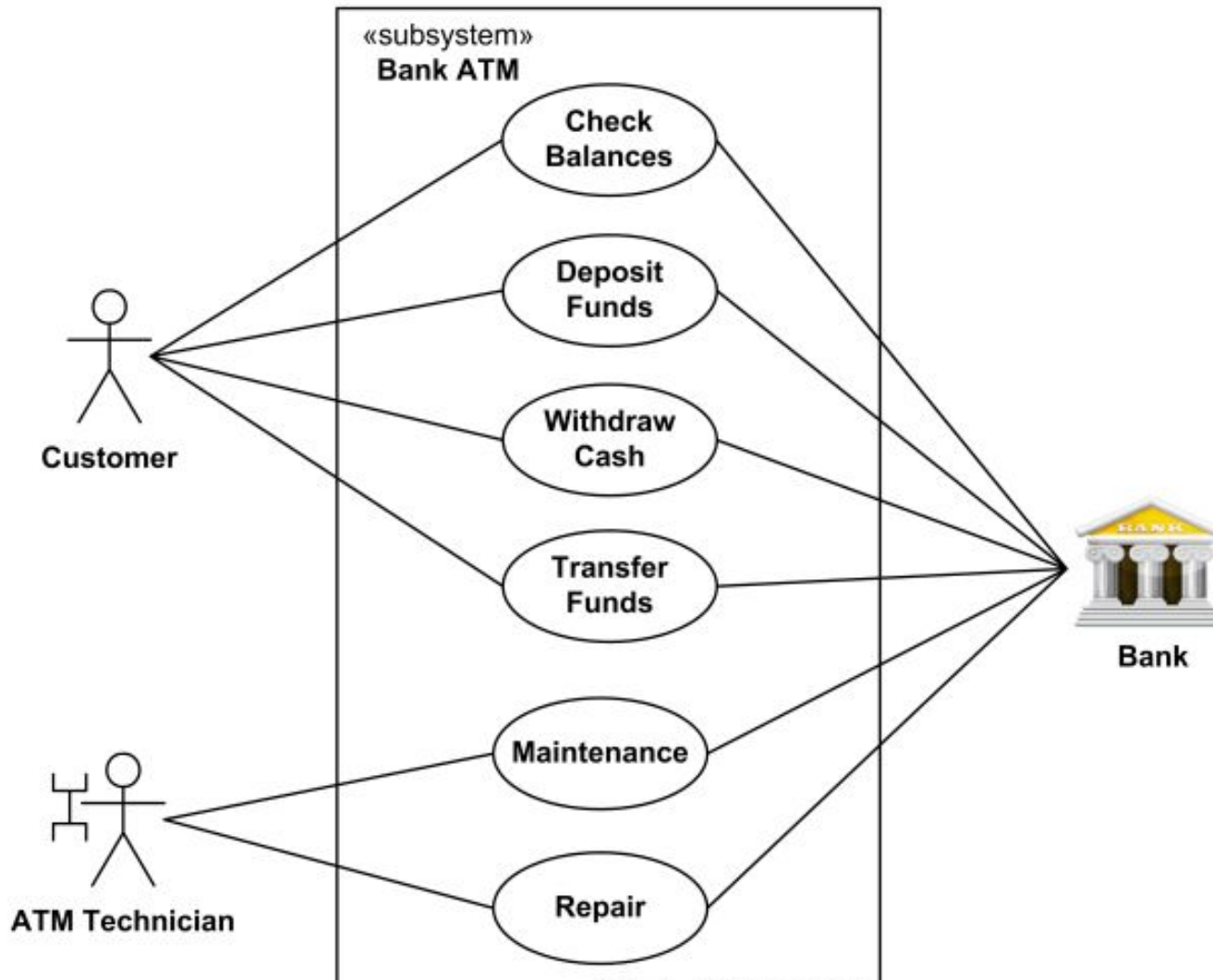


- Every use case must have a name that distinguishes it from other use cases.
- In practice, use case names are short active verb phrases naming some behavior found in the vocabulary of the system you are modeling.

How to Determine Use Cases?

- Which functions/services do actors require from the system ? What does the actor need to do?
- Does the actor need to read, create, destroy, modify or store some kind of information in the system?
- Does the actor need to get notified when some events occur in the system?
- Can some actors daily work be simplified by functionalities in the system? If so, what work?

Example



Task two – Identifying Use Cases

It is required to automate the following manual process on student assessments.

- Traditionally, a student's performance on each taught module in the Software Engineering course can be assessed in two ways:
 - By written report before, during and after the module, which is marked by the lecturers.
 - By a written exam, which is taken by the students under formal invigilated conditions and then marked by the lecturers.
- The lecturers may choose what weight is given to each form of assessment or, indeed, whether one of the two forms is dropped.
- For audit purposes, copies of all assessment material and marking schemes must be presented to the department's office for safekeeping.
- After the completion of each module's assessment or exam, as appropriate, student marks are presented to the registry.
- To get the student details the registration system is accessed.





Use Case Relationships

Relationships may exist;

- Between Use Cases – Include, Extend, Generalization
- Between Actors- Generalization
- Between Actors and Use Cases - Association

Use Case Relationships

1) Relationship Types Between Use Cases;

- Association: Indicates that an actor participates in (i.e. communicate with) the use case.

- Include : Indicates that the behavior of one use case includes the behavior of a another.
 <<include>>

- Extend : Indicates that one use case may be extended by (subjected to specific conditions) by the behavior of another.
 <<extend>>

- Generalization : Indicates that one use case inherits the properties of another with the usual possibilities for overriding and substitution.


Use Case Relationships

2) Relationship Types Between Actors;

- Generalization : Indicates that an instance of one actor can communicate with the same use cases as instance of another.



Use Case Relationships

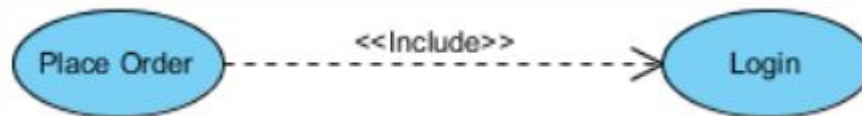
3) Relationship Types Between Actors and Use Cases;

- Association: The ONLY relationship between actors and use cases shown by association.



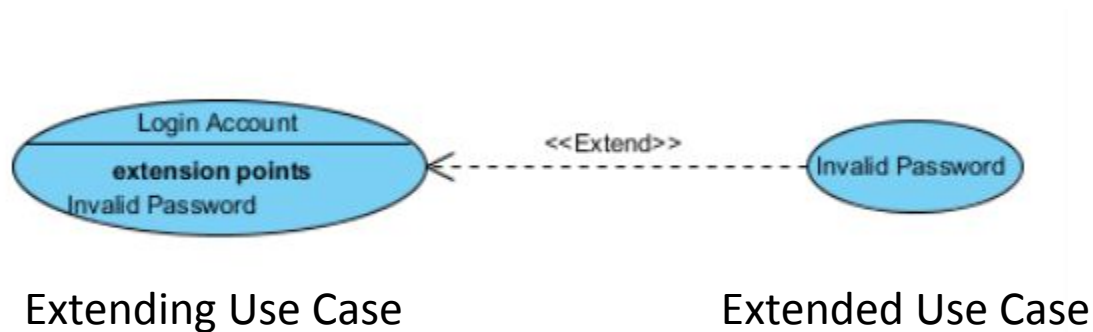
Include (uses) Relationship

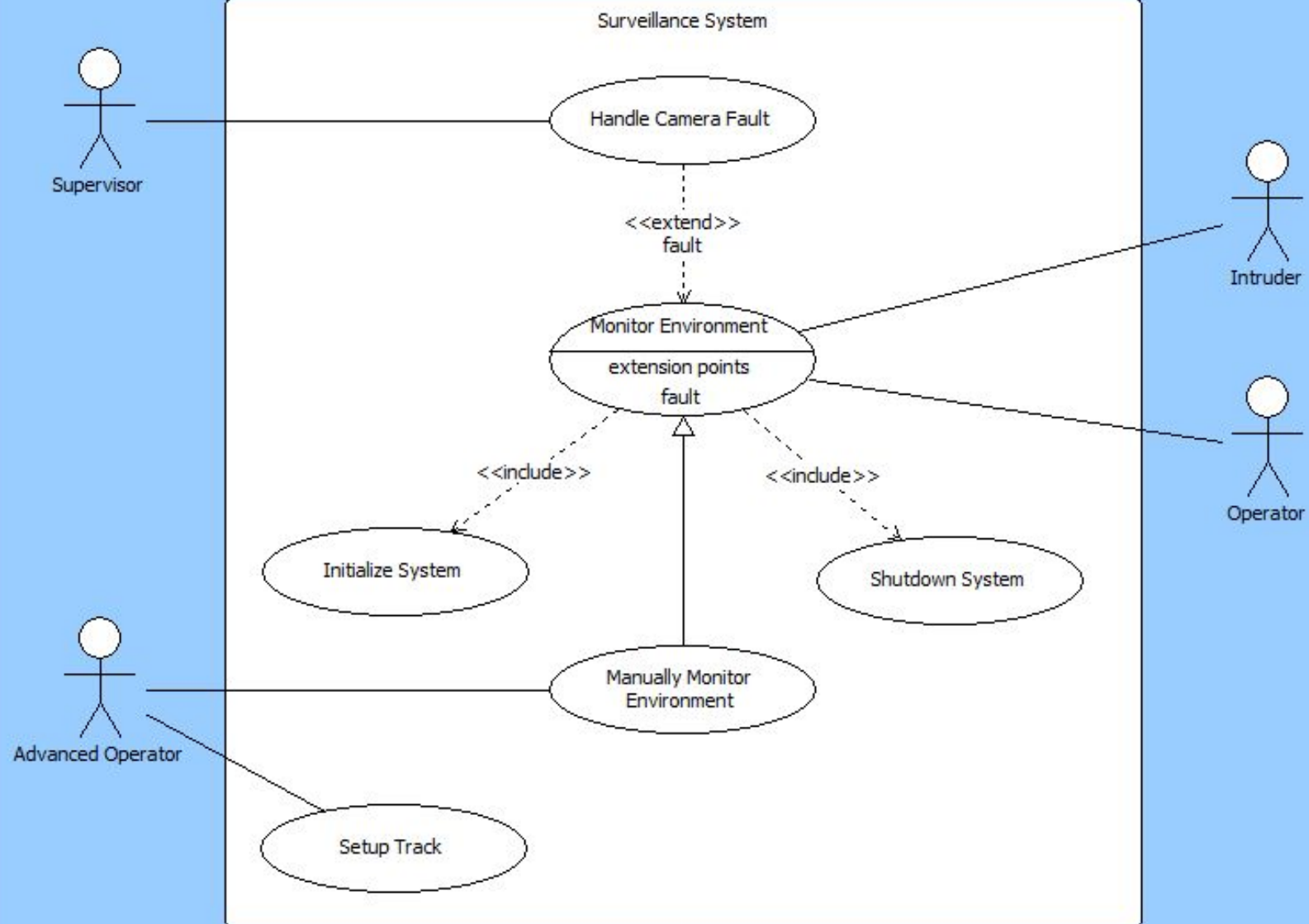
- When a use case is depicted as using the functionality of another use case, the relationship between the use cases is named as include or uses relationship.
- A use case includes the functionality described in another use case as a part of its business process flow.
- A uses relationship from base use case to child use case indicates that an instance of the base use case will include the behavior as specified in the child use case.



Extend Relationship

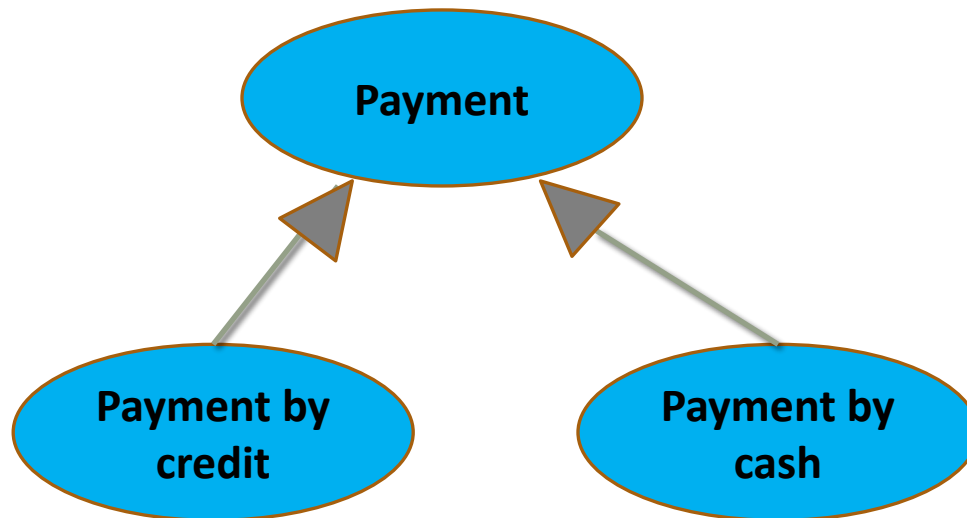
- The base Use Case is a complete Use Case in its own right, which may have its behavior extended by another Use Case when specific conditions arise.
- An extend relationship is to model the part of a use case the user may see optional system behavior.
- In this way, separating optional behavior from mandatory behavior.





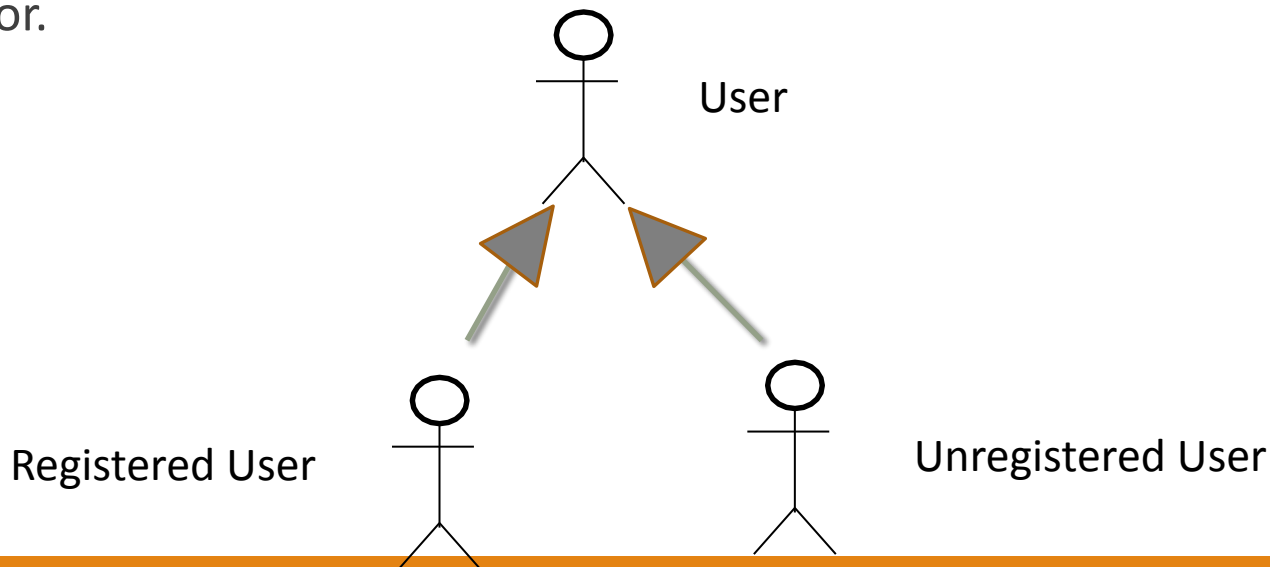
Generalization Relationship

- A generalization relationship is a parent-child relationship between use cases.
- The child use case is an enhancement of the parent use case.



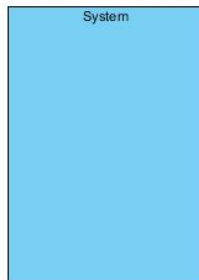
Actor Generalization

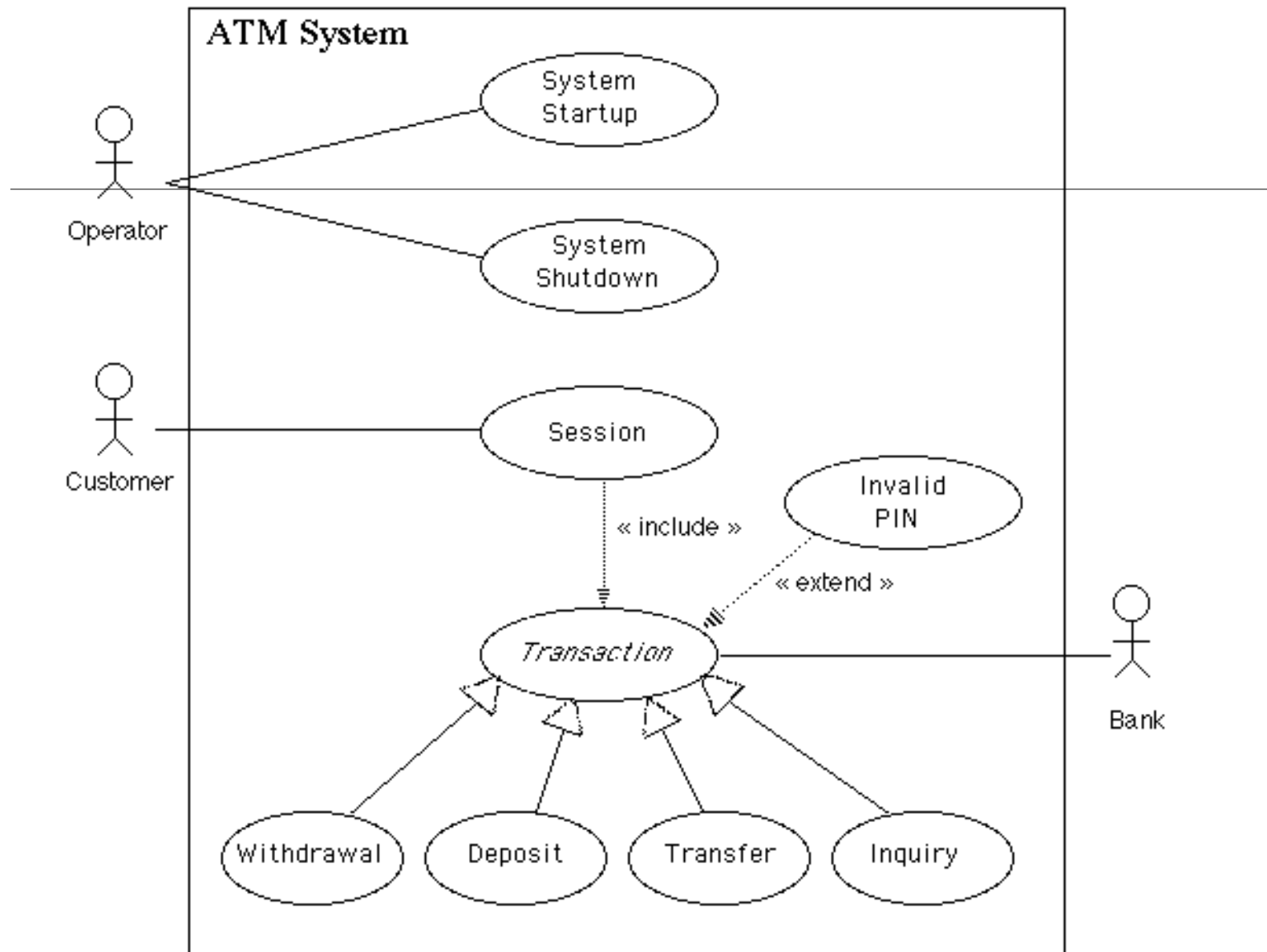
- Actor Generalization is drawn from the concept of inheritance in Object Oriented Programming.
- Child actor inherits all of the characteristics and behavior of the parent actor.
- Can add to, modify or ignore any of the characteristics of the parent actor.



System Boundary

- The system boundary is potentially the entire system as defined in the requirements document.
- For large and complex systems, each module may be the system boundary.
- For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting, etc.
- Can form a 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





Case Study 1 – Theatre Ticketing System

- The following is a description of the activities of a box office clerk interacting with a theatre ticket booking system.
- On receiving a request for tickets from a customer, the clerk must firstly set up an order on the system.
- She then asks the customer if the order is a single order or a subscription order. Members of the theatre are entitled to place subscription orders.
- If a single order, the clerk assigns the seats on the system and charges the customer's credit card.
- If on the other hand, it is a subscription order, the clerk assigns the seats and debits the customer's account at the same time as awarding the customer a member's bonus. In both cases, the

Case Study 2 - The Bank Account

- You are asked to design a system to handle
- current and savings accounts for a bank.
- Accounts are assigned to one or more customers, who may make deposits or withdraw money.
- Each type of account earns Interest on the current balance held in it.
- Current accounts may have negative balances (overdrafts).
- On a savings account there is a maximum amount that can be withdrawn in one transaction.

Case Study - The Bank Account

- Bank employees may check any account that is held at their branch. They are responsible for invoking the addition of interest and for issuing statements at the correct times.
- A money transfer is a short lived record of an amount which has been debited from one account and has to be credited to another.
- A customer may create such a transfer from their account to any other.
- Transfers within a branch happen immediately,
- while those between branches take three days.

Your Turn..

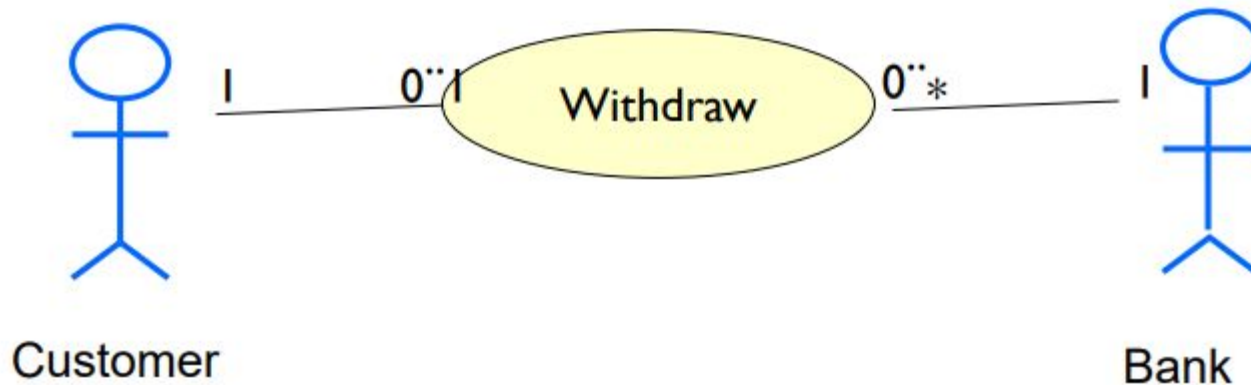
Draw the Use Case Models

Case Study – Library System

Librarian can add books to the library catalogue when new books are available in the library. Librarian can remove books from the library catalogue when needed. Member can reserve books which he/she wishes to borrow. Library Manager can take a list of books available in the Library. Furthermore he/she can add or remove any books from the Library catalogue when needed. When member is reserving the books he/she has to login to the system. Member can renew the books he/she has borrowed. When renewing if book has exceeded the loan period a fine will be calculated. For renewing purposes the member should login to the system. Library Manager can generate reports of the Borrowed books, Overdue books at the end of each month.

Multiplicity in Use Case

The uses connector can optionally have multiplicity values at each end, as in the following diagram which shows that a customer may only have one withdrawal session at a time, but a bank may have any number of customers making withdrawals concurrently.



Create use cases for different levels

1. **White** is the highest level, like clouds. This is the enterprise level, and there may only be four to five for the entire organization. Examples might be to advertise goods, sell goods to customers, manage inventory, manage the supply chain, and optimize shipping.
2. **Kite** is lower than white but still a high level, providing an overview. The kite use case may be at the business unit or department level and is a summary of goals. Examples would be to register students, or if working with a travel company: make an airline, hotel, car, or cruise reservation.

Create use cases for different levels

1. **Blue** is at sea level, and is usually created for user goals. This often has the greatest interest for users and is easiest for a business to understand. It is usually written for a business activity and each person should be able to do one blue level activity in anywhere from 2 to 20 minutes. Examples are register a continuing student, add a new customer, place an item in a shopping cart, and order checkout.
2. **Indigo or fish** is a use case that shows lots of detail, often at a functional or subfunctional level. Examples are choose a class, pay academic fees, look up the airport code for a given city, and produce a list of customers after entering a name.

Use Case Scenarios

Use Case Scenarios

- A use case description is a **text-based narrative of a functionality comprised of detailed, step-by-step interaction between the actor and the system**. It describes the outcomes of an action taken to accomplish a specific goal.

Use Case Scenarios

- A Scenario is a formal description of the flow of events that occur during the execution of a Use Case instance. It defines the specific sequence of events between the system and the external actors.
- There is usually a **Main scenario**, which describes what happens when everything goes according to the plan. It is written under the assumption that everything is okay, no errors or problems occur, and it leads directly to the desired outcome of the use-case.
- **Other scenarios** describe what happens when variations to the main scenario arise, often leading to different outcomes.

Sample Template

1. Use Case ID and name
2. Characteristic Information
 - ☐ Goal in Context
 - ☐ Scope
 - ☐ Level
3. Pre-Conditions
4. Primary Actor
5. Main Success Scenario Steps
6. Extensions
7. Optional Information

Key Terms

Scenario - Describes one way that an actor can accomplish a specific goal. It is a *single* path written as a series of steps performed by actors or the solution. A collection of scenarios constructs a use case.

Actor - Any person or system external to the system that uses or interacts with that system to achieve a goal. Each actor is given a role to represent their interactions with the solution. The names of people actors should be in the form of a role and should not specify actual names. Actors generally fall into the category of primary, secondary or stakeholder.

- **Primary Actor** –An actor that initiates a use case. Ex: Card Holder
- **Secondary Actor** – An actor that reacts or responds to actions made by a primary actor. Ex: Bank who do the authentication of a transaction
- **Stakeholder** - *Off-stage* actors who don't interact directly with the use case but have an interest in the outcome of the use case. Ex: Receiving a notification

Key Terms

Flow of Events (Paths) – The set of steps that must be taken by the actor and the solution while the use case is being performed. Generally, use cases consist of the main success path (also called basic or primary), alternate paths and exception paths.

- *Main success path* – The inputs from the actors and system responses that represent the *most* common successful path to accomplish the actor's goal
- *Alternate path* - The inputs from the actors and system responses that represent other *less* common paths to accomplish the actor's goal
- *Exception path* - The inputs from the actors and system responses that represent an unsuccessful path when the actor's goal is unachievable.

Name – Unique name that represents the goal of the use case. It generally includes a verb (action by actor) and a noun (what is being done). For example: "Assign Task".

Pre-Conditions - Conditions that are assumed to be in place before the use case can begin. May act as a constraint on the completion of the use case.

Key Terms

Post-Conditions (Guarantees) - Conditions in the form of *guarantees* that must be in place when the use case is completed for all possible flows including the main success path, the alternate path as well as the exception path. Guarantees for a successful use case is called a success guarantee, while the guarantee for an unsuccessful use case is called a minimal guarantee.

- *Success Guarantee* – Conditions that must be in place when the main success path or an alternate path are successfully executed.
- *Minimal Guarantee* - Conditions that must be in place even when an exception path is taken and the actor's goal is not achieved. May address non-functional requirements such as security.

Trigger - An event that initiates the flow of events in a use case. Triggers can either be an actor action or a temporal event.

- *Actor Actions* - The most common trigger when an action taken by the primary actor (input) initiates the use case.
- *Temporal Event* – Time-related events that may trigger a use case. Commonly used when actions need to be executed during a specific time of day or on a specific date.

Ex: The cash needs to be restocked to the ATM every morning at 6.00 am.

Exercise 01

Write down Use case scenario for withdraw money from an ATM.

Include / Extend : Textual Description

- The text description of the **Base Use Case** contains references to any included Use Cases and any extension point(s).
- Eg:- The RentVideo Use Case begins when the member is identified.

Basic flow:

1. include:: (Identify Member).
2. extend:: (late fine).
3. The rental transaction is then performed.
4. include:: (Process Rental).

Exercise 01 Con.

