UML Modeling

Instructor: Mehroze Khan

How are design ideas communicated in a team environment?

- If the software is large scale, employing perhaps dozens of developers over several years, it is important that all members of the development team communicate using a common language.

- This isn't meant to imply that they all need to be fluent in English or C++, but it does mean that they need to be able to describe their software's operation and design to another person.

- That is, the ideas in the head of say the analyst have to be conveyed to the designer in some way so that he/she can implement that idea in code.

- Just as mathematicians use algebra and electronics engineers have evolved circuit notation and theory to describe their ideas, software engineers have evolved their own notation for describing the architecture and behaviour of software system.

- That notation is called UML. The Unified Modelling Language. Some might prefer the title Universal Modelling language since it can be used to model many things besides software.

What is UML ?

- UML is not a language in the same way that we view programming languages such as C++, Java or Python.
- UML is however a language in the sense that it has syntax and semantics which convey meaning, understanding and constraints to the reader and thereby allows two people fluent in that language to communicate and understand the intention of the other.
- UML represents a collection of graphical notations to capture requirements and design alternatives.
- UML is to software engineers what building plans are to an architect and an electrical circuit diagrams is to an electrician.
- UML is suitable for large scale projects

Models and Multiple Views

- UML is used to **model** (i.e., represent) the system being built.
- It is impossible to capture all the subtle details of a complex software system in just one large diagram.
- The UML has numerous types of diagrams, each providing a certain view of your system

Diagram Taxonomy

- UML diagrams can be classified into two groups: **structure diagrams** and **behavior diagrams.**
- System complexity is driven both by the number and organization of elements in the system (i.e., structure) and the way all these elements collaborate to perform their function (i.e., behavior).

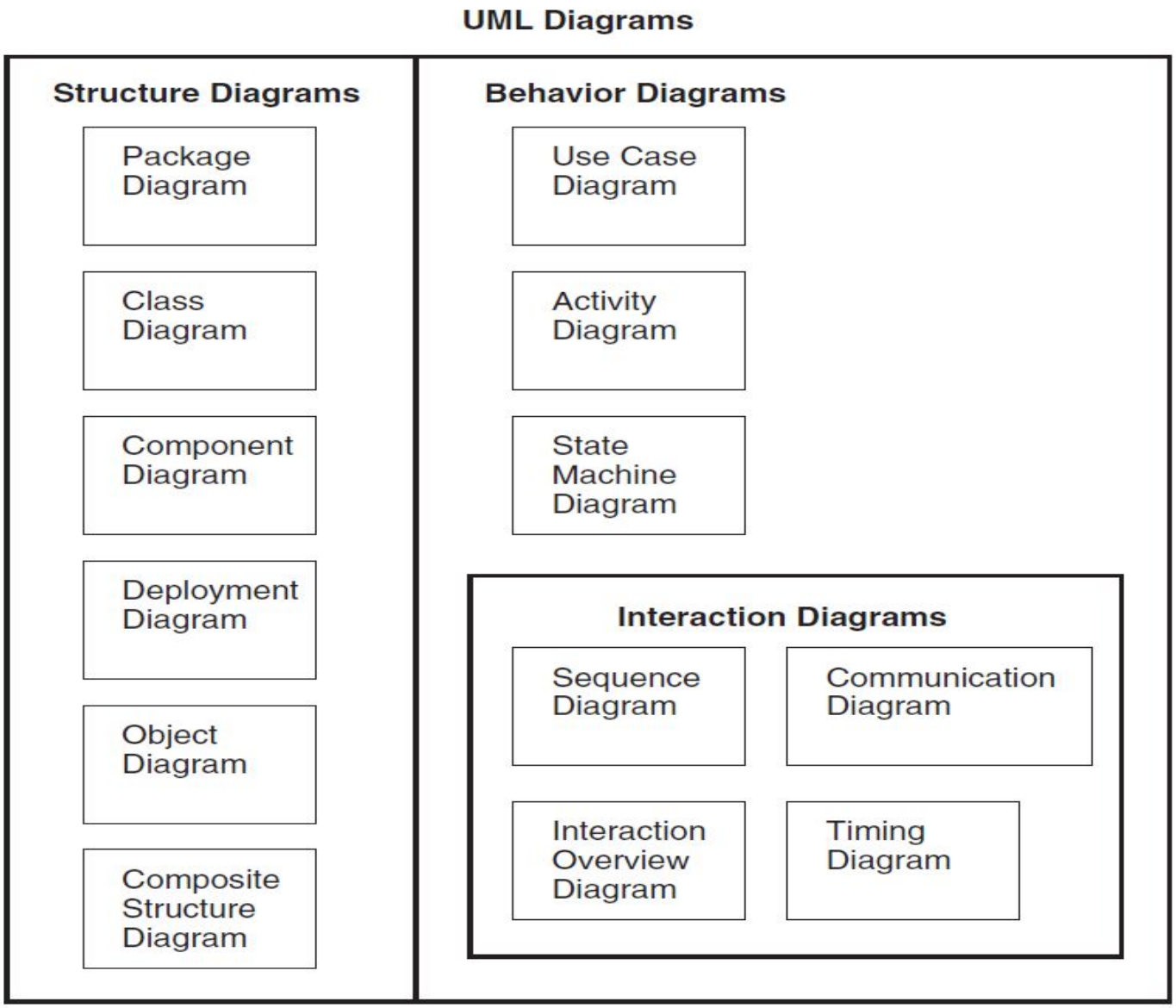
Structure Diagrams

- These diagrams are used to show the **static structure** of elements in the system, **architectural organization** of the system, the **physical elements** of the system, its **runtime configuration**.
- Structure diagrams are often used in conjunction with behavior diagrams to depict a particular aspect of your system. Each class may have an associated state machine diagram that indicates the event-driven behavior of the class's instances.
- Following are the Structure Diagrams in UML:
  - Package diagram
  - Class diagram
  - Component diagram
  - Deployment diagram
  - Object diagram
  - Composite structure diagram

Behavior Diagrams

- Events happen **dynamically** in all software-intensive systems: Objects are **created and destroyed**, objects **send messages** to one another in an orderly fashion, external events **trigger operations** on certain objects.
- Following are the Behavior Diagrams in UML:
  - Use case diagram
  - Activity diagram
  - State machine diagram
  - Interaction diagrams
    - Sequence diagram
    - Communication diagram
    - Interaction overview diagram
    - Timing diagram

UML Diagrams



Definition of a Use-Case

- A process or procedure, describing a user's interaction with the system (e.g. library) for a specified, identifiable purpose. (e.g. borrowing a book).

- As such, each Use-Case describes a step-by-step sequence of operations, iterations and events that document

  - The Interaction taking place.
  - The Measurable Benefits to the user interacting with the system
  - The Effect of that Interaction on the system.

- It is important to document these use-cases as fully as possible as each use-case captures some important functionality that our system will have to provide.

Use Cases in Iterative Development

- Functional requirements are primarily captured in use cases
- Use cases drive the iteration planning and work
- Easy for users to understand

What ‘should’ and what ‘shouldn’t ‘ be included in use cases?

What Use Cases Include	What Use Cases Do NOT Include
• Who is using the website • What the user want to do • The user's goal • The steps the user takes to accomplish a particular task • How the website should respond to an action	• Implementation-specific language • Details about the user interfaces or screens.

How to Find Use Cases?

- Choose the system boundary
  - what you are building?
  - who will be using the system?
  - what else will be used that you are not building?
- Find primary actors and their goals
  - brainstorm the primary actors first who starts and stops the system?
  - who gets notified when there are errors or failures?
- Define use cases that satisfy user goals
  - prepare an actor-goal list in general, one use case for each user goal
  - name the use case similar to the user goal

What Tests Can Help Find Useful Use Cases?

- Which of these are valid use cases?
  - Negotiate a Supplier Contract
  - Handle Returns
  - Log in
  - Move Piece on the Game Board

All of these can be use cases at different levels, depending on the system, boundary, actors, and goals

What Tests Can Help Find Useful Use Cases?

- Rather than asking ”What is a valid use case?”

More practical question:

“What is a useful use case?”

- Rules of thumb
  - The Boss Test
  - The EBP Test
  - The size test

What Tests Can Help Find Useful Use Cases?

Boss test

- "What have you been doing all day?"
- Your reply "logging in!"
- Is your boss happy? No value? No good use case!

Elementary Business Process (EBP) test

- A task performed by one person in one place at one time, in response to a business event, which adds measurable business value and leaves data in a consistent state
- Good Examples: Approve Credit or Price Order
- Bad Examples: Delete a line item or print the document

Size test

- Just a single step in a sequence of others -> not good!
- Example: Enter an Item Id

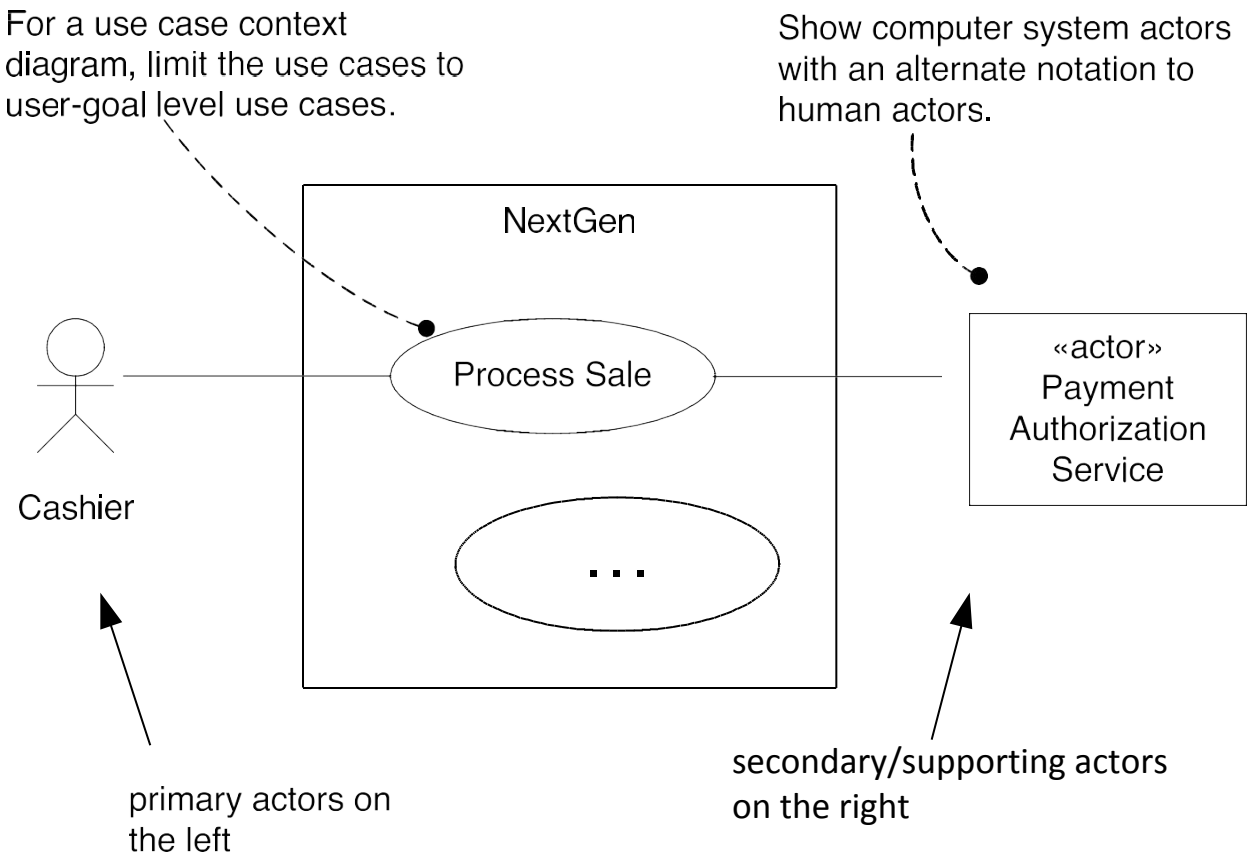
Applying Tests

- Handle returns
  - OK with the Boss. EBP. Size is good.
- Log in
  - Boss is not happy is this is all you do all day!
- Move piece on game board
  - Single step – fails the size test.

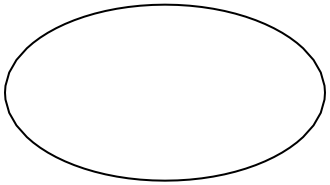
Use Case Diagram Concepts

- Summarizes all use cases (for the part of the system being modeled) together in one picture
- Typically drawn early in the SDLC
- Shows the associations between actors and use cases

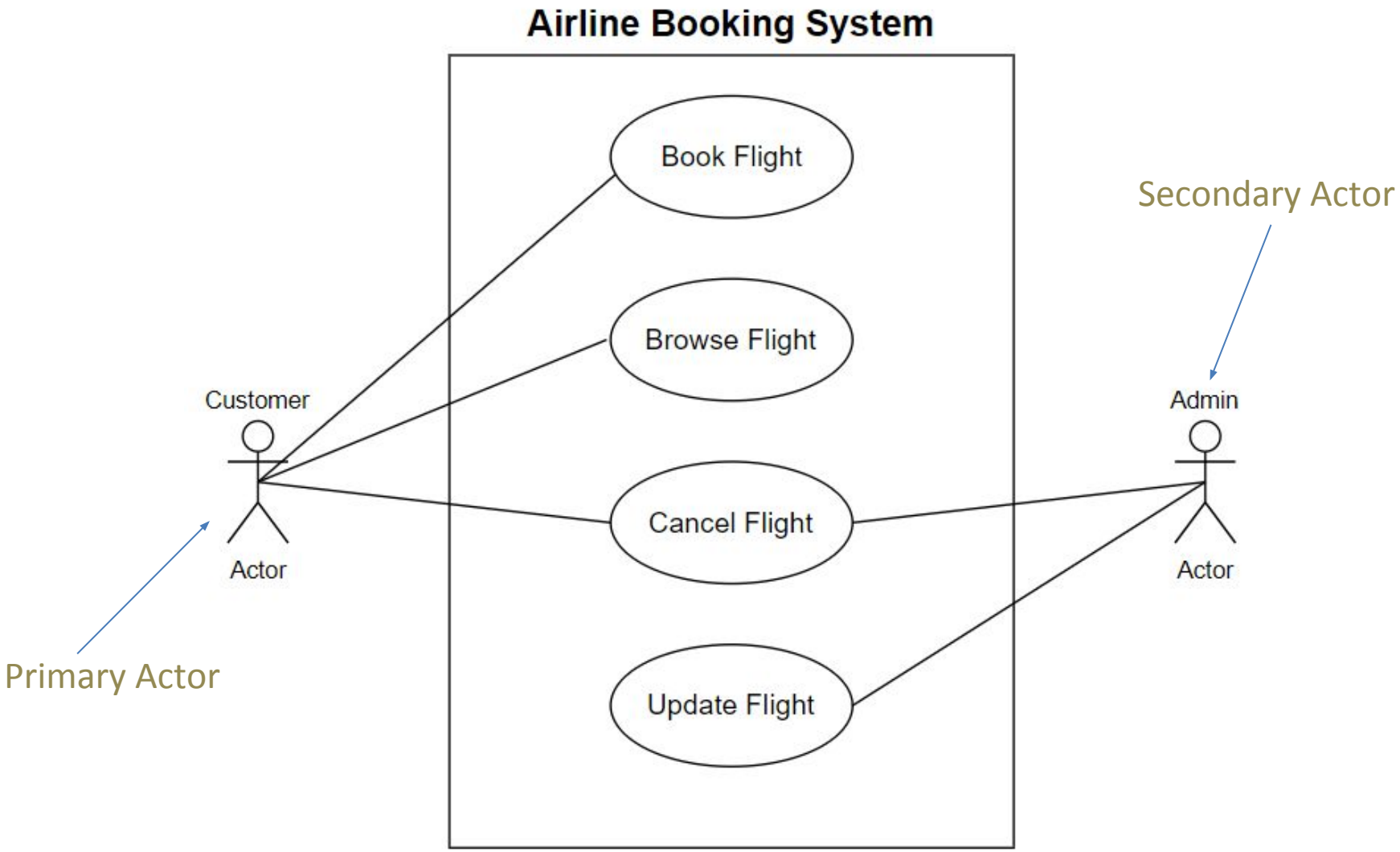
Use Case (Context) Diagrams: Suggested Notation



Syntax for Use Case Diagram

Term and Definition	Notations
An actor ■ Is a person or system that derives benefit from and is external to the system ■ Is labeled with its role ■ Can be associated with other actors using a specialization/superclass association, denoted by an arrow with a hollow arrowhead ■ Are placed outside the system boundary	
A use case ■ Represents a major piece of system functionality ■ Can extend another use case ■ Can use another use case ■ Is placed inside the system boundary ■ Is labeled with a descriptive verb–noun phrase	
A system boundary ■ Includes the name of the system inside or on top ■ Represents the scope of the system	Name of the system
An association relationship ■ Links an actor with the use case(s) with which it interacts	.....

Use Case Diagram for Airline Booking System



Example

Requirement 1
The content management system shall allow an *administrator* to create a new blog account, provided the personal details of the new blogger are verified using the author credentials database.

The requirement indicates that that the ***Administrator*** interacts with the system to create a new blogger's account

The ***Administrator*** *interacts with* the system and is *not part of* the system; thus, the Administrator Is an ***Actor***.

It's actually worth being very careful when naming your actors. The best approach is to use a name that can be understood by both your customer *and* your system designers.

Example

- What can be the use case in our Requirement 1?

Requirement 1
The content management system shall allow an *administrator* to create a new blog account, provided the personal details of the new blogger are verified using the author credentials database.

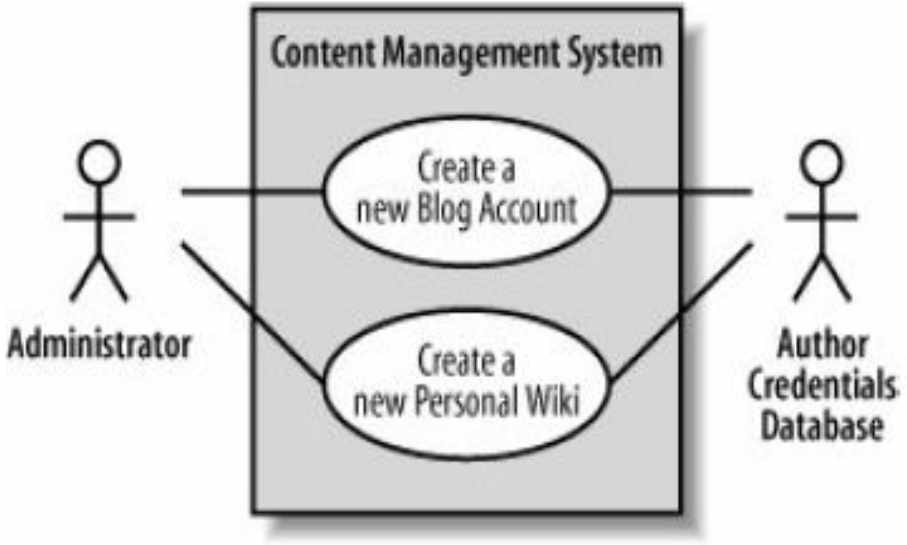
to create a new blog account

UML Representation

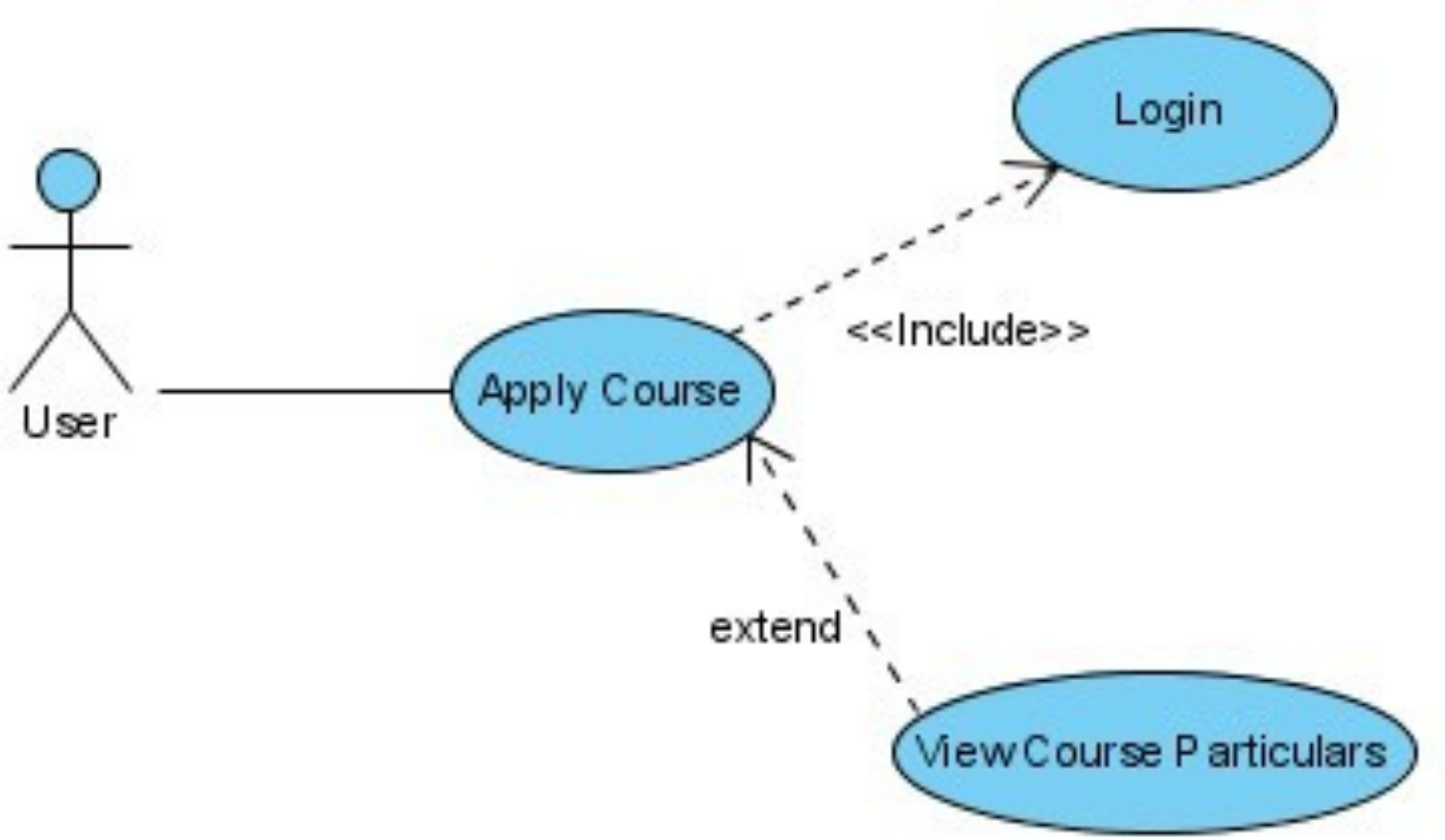
Example (Cont)

Requirement 2

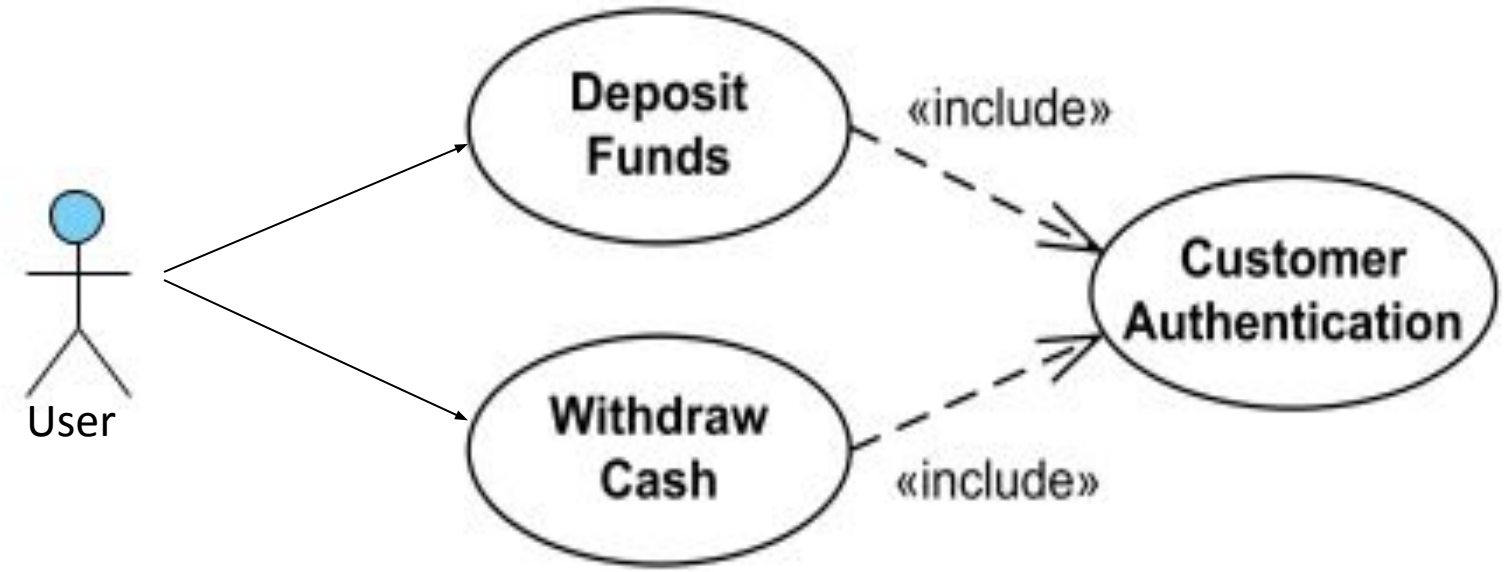
The content management system shall allow an administrator to create a new personal Wiki, provided the personal details of the applying author are verified using the Author Credentials Database.



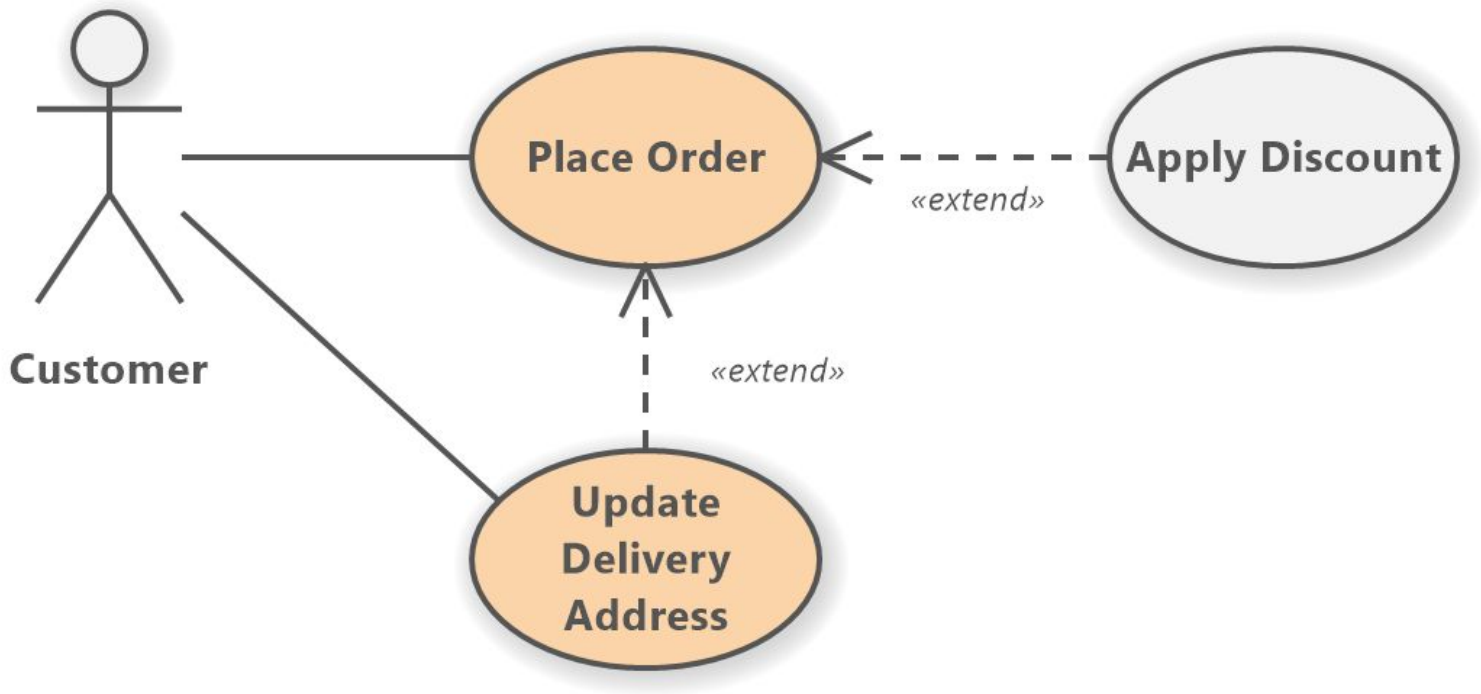
Example of Stereotypes – Include and Extend



Example – Include and Extend



Example – Include and Extend



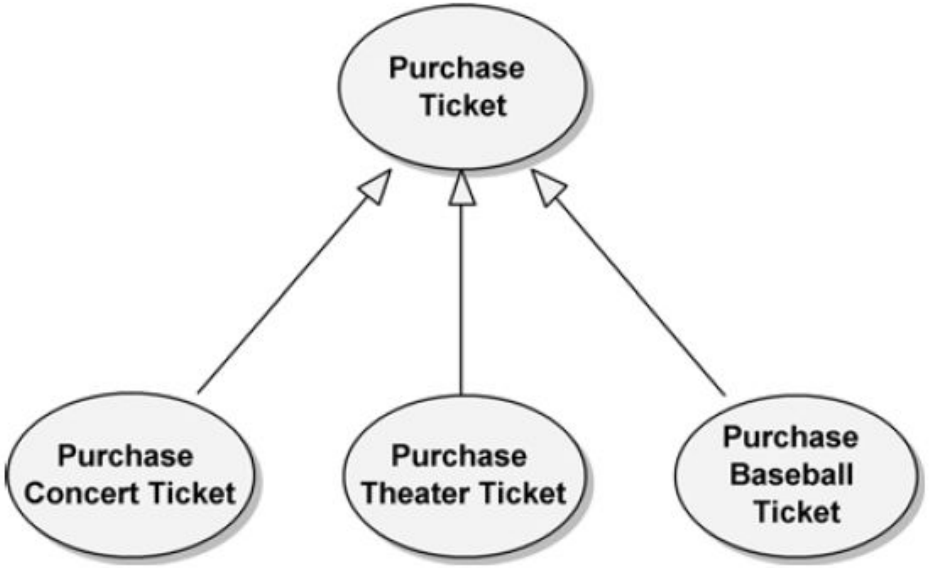
Include and Extend Relationships

- The use of include and extend is discouraged simply because they add unnecessary complexity to a Use Case diagram.

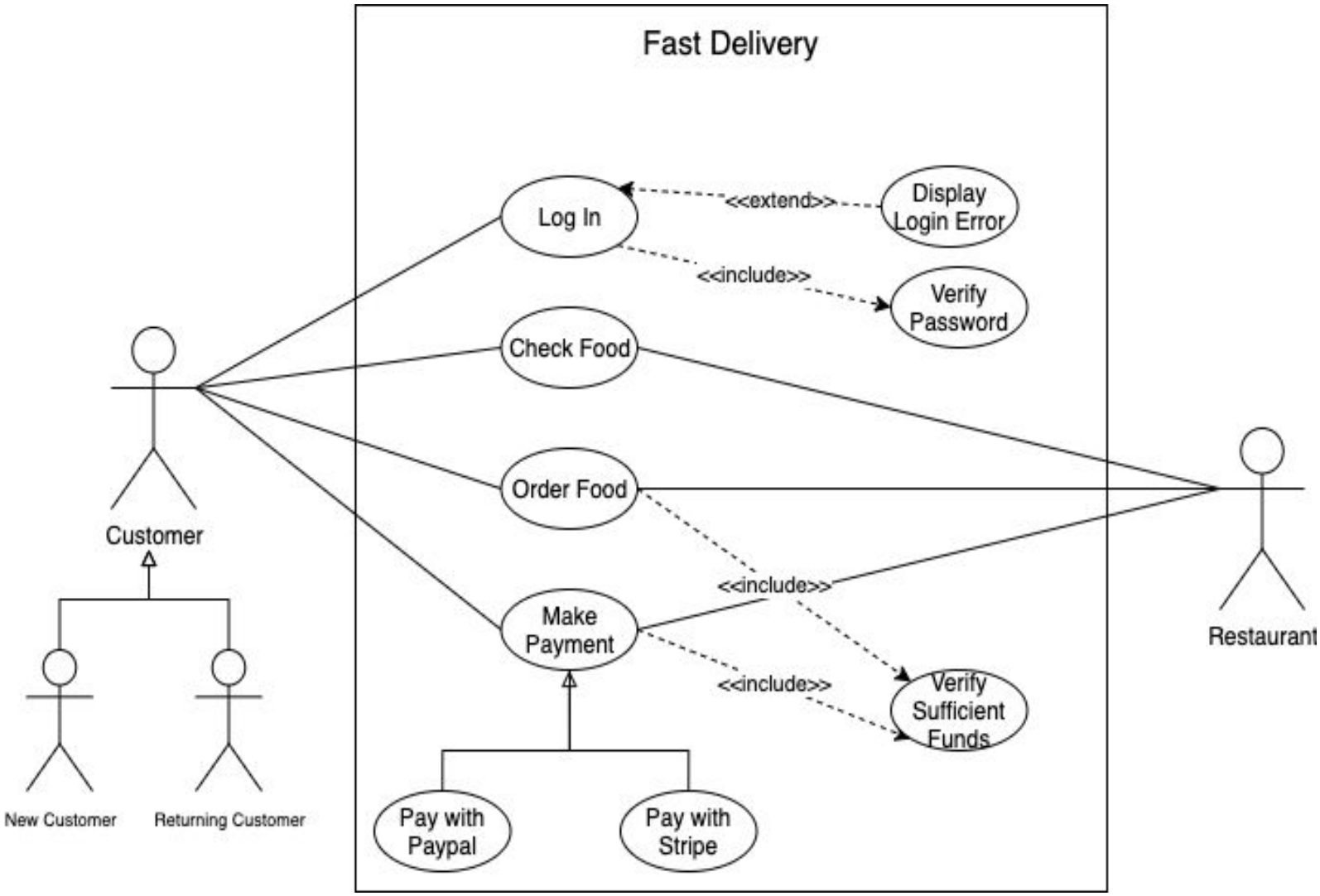
- Only use them if they are necessary.

Generalization

- Generalization relationships can also be used to **relate use cases**. As with classes, use cases can have common behaviors that other use cases (i.e., child use cases) can modify by adding steps or refining others.
- Purchase Ticket contains the basic steps necessary for purchasing any tickets, while the child use cases specialize Purchase Ticket for the specific kinds of tickets being purchased.



Example


Fast Delivery
Log In
<<extend>>
Display Login Error
<<include>>
Verify Password
Check Food
Order Food
Make Payment
<<include>>
<<include>>
Verify Sufficient Funds
Pay with Paypal
Pay with Stripe
Customer
New Customer
Returning Customer
Restaurant

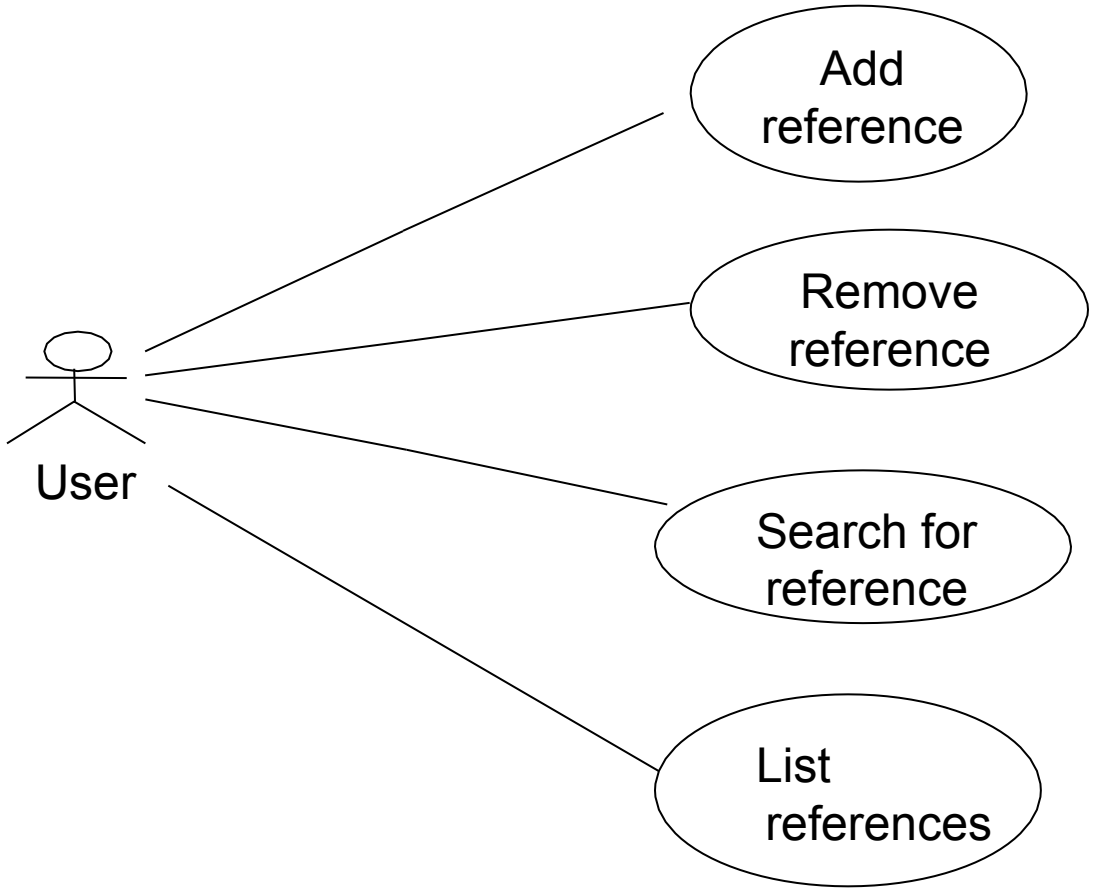
Use Case Diagram– Guidelines & Caution

- Actors should be **Nouns**
- Use cases should ideally begin with a **verb** – i.e. generate report.
- Use cases should **NOT be open ended** – i.e. Register (instead should be named as Register New User)

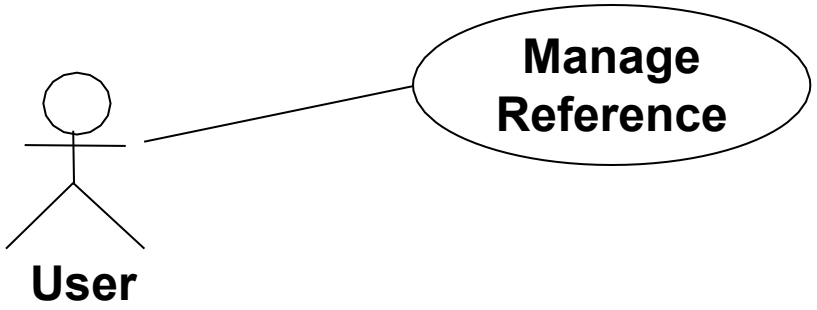
•Avoid showing communication between actors.

- named as **singular**. i.e., student and NOT students. **NO names** should be used – i.e., John, Sam, etc.
- **Do NOT show behavior** in a use case diagram instead only depict system functionality.
- Use case diagram **does not show sequence** – unlike DFDs.

Granularity of Use Cases



Granularity of Use Cases



Generally, a use case should embody sufficient levels of granularity without which the use case may not be rendered as useful.

Identify Actors

- Driver
- Hostess
- Customer
- Company Staff/ Administrator

Driver and hostess cannot be an Actor. Why?

Consider an online reservation system for a bus company. The bus company includes several buses and manage trips to different cities. Each bus is identified by its plate number and a separately assigned bus number. The trips are based on a predefined schedule and stop at predefined bus stations. Each bus can have only one trip per day. Each bus includes a driver and one hostess. For long trips, the bus will have breaks at service and rest areas. There are two types of trips, normal trips and express trips. Express trips do not stop at intermediate stations and get faster at the destination. Seats can be reserved by customers on the web site of the bus company. The customer has the option to directly pay for the seat through the website. In that case, the seat cannot be cancelled (neither by the customer nor by the bus company). If the customer has not paid for the seat, the bus company can cancel the seat if the customer does not show up one hour before the trip. When the reservation is cancelled, the seat will become free and can be sold to another customer. Both the customer and the company staff must register themselves for performing operations with the system.

Identify Verb / Goal/Use Case

- Reserve Seat
- Pay for the Seat
- Cancel Seat
- Register
- View Seat

Consider an online reservation system for a bus company. The bus company includes several buses and realizes trips to different cities. Each bus is identified by its plate number and a separately assigned bus number. The trips are based on a predefined schedule and stop at predefined bus stations. Each bus can have only one trip per day. Each bus includes a driver and one hostess. For long trips, the bus will have breaks at service and rest areas. There are two types of trips, normal trips and express trips. Express trips do not stop at intermediate stations and get faster at the destination. Seats can be reserved by customers on the web site of the bus company. The customer has the option to directly pay for the seat through the website. In that case, the seat cannot be cancelled (neither by the customer nor by the bus company). If the customer has not paid for the seat, the bus company can cancel the seat if the customer does not show up one hour before the trip. When the reservation is cancelled, the seat will become free and can be sold to another customer. Both the customer and the company staff must register themselves for performing operations with the system.

So far what we have done

- Actors
  - Customer
  - Bus Company Administrator

- Use Cases
  - Reserve Seat
  - Pay for the Seat
  - Cancel Seat
  - Register
  - View Seat

Start Making a Use Case Diagram with the identified infromation

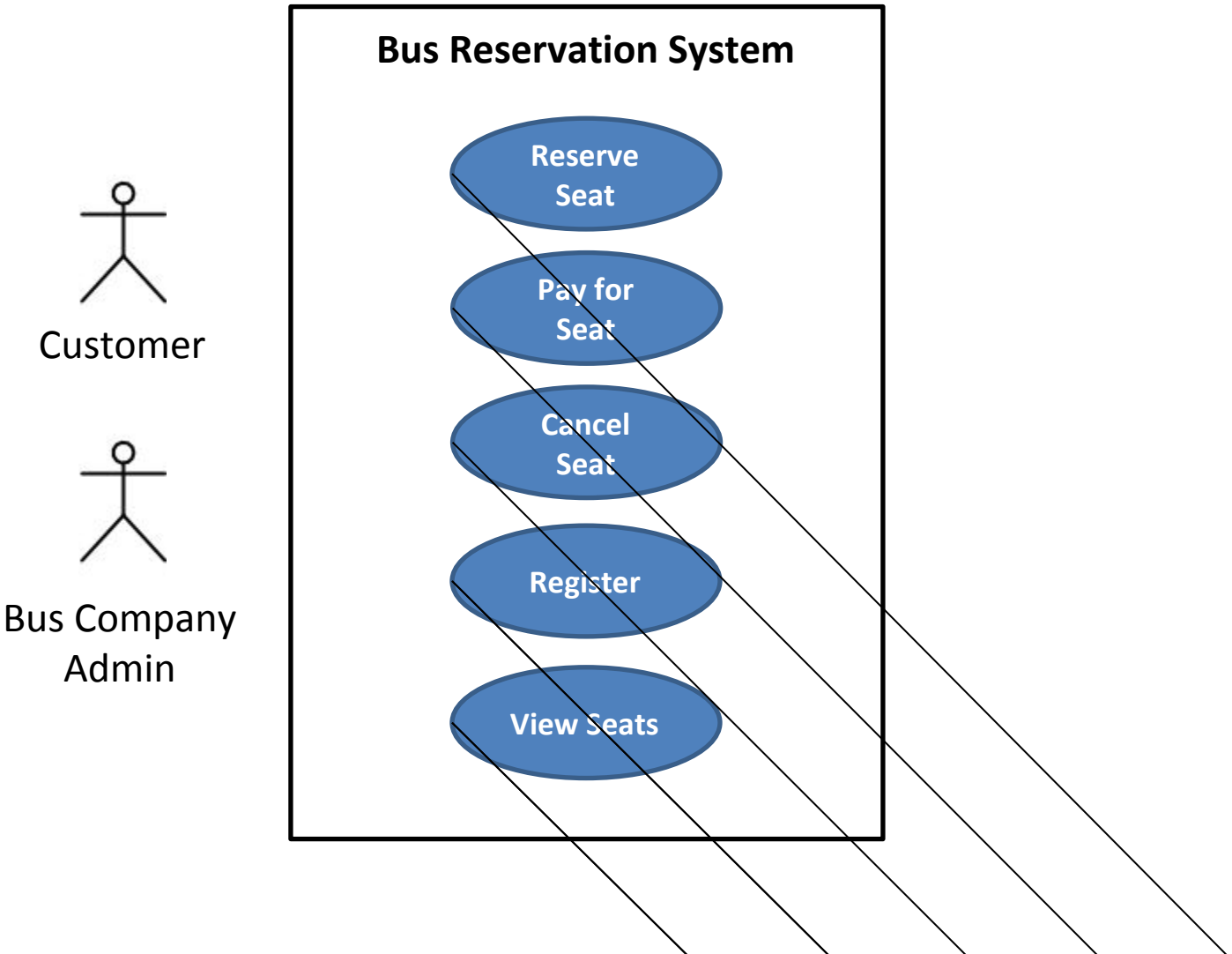
Actor Goals?

Customer Reserves the Seat
Customer Pays for the Seat
Customer & Admin Cancels the Seat
Customer & Admin must Register
Customer & Admin can View the Seat

Complete the Use Case Diagram

Consider an online reservation system for a bus company. The bus company includes several buses and realizes trips to different cities. Each bus is identified by its plate number and a separately assigned bus number. The trips are based on a predefined schedule and stop at predefined bus stations. Each bus can have only one trip per day. Each bus includes a driver and one hostess. For long trips, the bus will have breaks at service and rest areas. There are two types of trips, normal trips and express trips. Express trips do not stop at intermediate stations and get faster at the destination. Seats can be reserved by customers on the web site of the bus company. The customer has the option to directly pay for the seat through the website. In that case, the seat cannot be cancelled (neither by the customer nor by the bus company). If the customer has not paid for the seat, the bus company can cancel the seat if the customer does not show up one hour before the trip. When the reservation is cancelled, the seat will become free and can be sold to another customer. Both the customer and the company staff must register themselves for performing operations with the system.

One Possible Solution for the Case



Case Study: Point of Sale terminal

Case Study: Point of Sale terminal

- Process Sale:
  - A customer arrives at a checkout with items to purchase.
  - The cashier uses the POS system to record each purchased item.
  - The system presents a running total and line-item details.
  - The customer enters payment information, which the system validates and records.
  - The system updates inventory.
  - The customer receives a receipt from the system and then leaves with the items.

Recall: Procedure to Find Use Cases

Step 1: **Choose the system boundary**

- Software application,
- The hardware and application as a unit
- An entire organization, or a dept of the org

Step 2: **Identify the primary actors**

- Those having user goals fulfilled through using services of the system.

Step 3: **Identify Actor (User) Goals**

- For each (primary actors), identify their user goals.

Step 4: **Define use cases**

- Use cases that satisfy user goals.
- Name them according to their goal.

Step 1: Choosing the System Boundary

- POS system itself is the System Boundary;
  - every-thing outside of it is outside the system boundary, including the cashier, payment authorization services, and so on.

- Once the external actors are identified, the boundary becomes clearer.
  - For example, is the complete responsibility for payment authorization within the system boundary? No, there is an external payment authorization service actor.

Step 2: Identify Actors

- We cannot understand a system until we know who will use it
  - Direct users
  - Users responsible to operate and maintain it
  - External systems used by the system
  - External systems that interact with the system

Actors in POS

- Cashier
- Manager
- System Administrator
- Sales Activity System

Step 3: Actors and User Goals List

Actor	Goal	Actor	Goal
Cashier	1. Process sales 2. Process rentals 3. Handle returns 4. Cash in 5. Cash out …	**System Administrator**	1. Add users 2. Modify users 3. Delete users 4. Manage security 5. Manage system tables
Manager	1. Start up 2. Shut down	**Sales Activity System**	1. Analyze sales and performance data
…	…	…	…

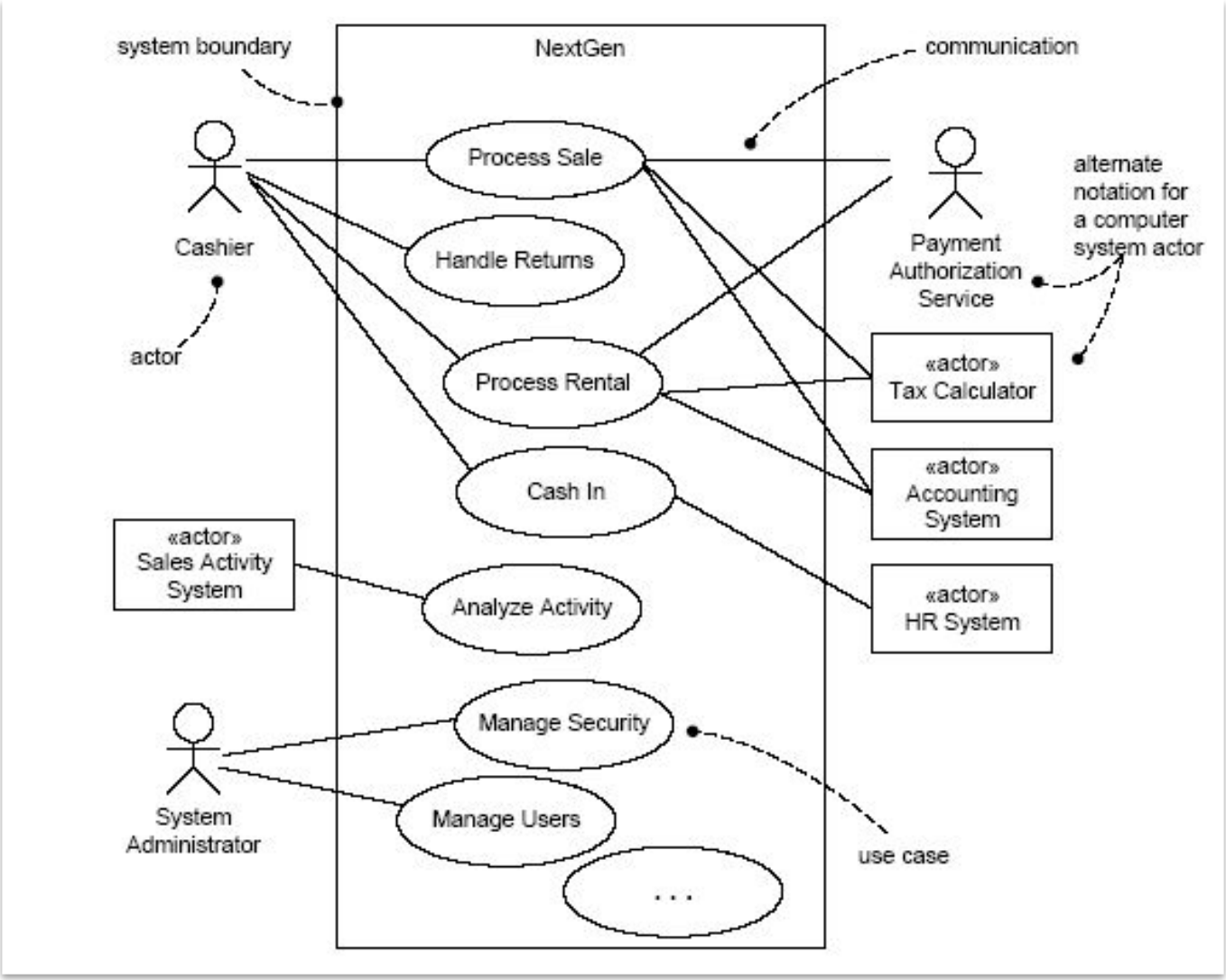
Step 4: Define Use Cases

- In general, define one use case for each user goal.
- Granularity: An exception to one use case per goal is to collapse CRUD (create, retrieve, update, delete) separate goals into one CRUD use case, idiomatically called Manage <X>. For example, the goals —edit user,—delete user and so forth are all satisfied by the Manage Users use case.
- Name the use case similar to the user goal (e.g., Goal: process a sale; Use Case: Process Sale.
- Start the name of use cases with a verb

Write Black-Box Use Cases

- Focus on what the system must do,
  - i.e., the behavior or functional requirements
  - Not on how it will do (the design)
- Examples:
  - Good: The system records the sale
  - Bad: The system writes the sale to the database.
  - Worst: System generates SQL INSERT statement for the sale

Use Case Diagram of POS System



Use Case Diagram

1. A use case diagram for the entire system – a graphic model.

2. Use cases are **not diagrams** they are **text documents.**

3. Use cases are one way of capturing the **functional requirements.**

4. Use cases are **text stories** of some **actors** using a system to **meet goals**.

5. One or more use case narratives for each use case – descriptions in text
   - **high-level/user story**
   - **expanded**

Use Case Description Template

Identifier	Give a unique Identifier for the Use Case
Name	Write the name of use case
Purpose	Brief Description of the process (what is happening?) in the use case
Priority	High/Medium/Low
Actors	List all actors (people, systems etc.) associated with this requirement
Pre-conditions	What must occur before the use case begins
Post-conditions	What has occurred as a result of the use case
Dependencies	List identifiers of use cases on which this use case is dependent

Typical Course of Action (Primary Flow)

S#	Actor Action	S#	System Response
1	List the actor steps here	1.1	List the system steps here
2		2.1	

Alternative Course of Action

S#	Actor Action	S#	System Response
1	List the actor steps here	1.1	List the system steps here

Exceptions: List the sequence of actions that prevents getting to the post condition

Identifier	UC-01
Name	Process Customer Order
Purpose	This use case describes the procedure for processing a customer's order, from placing the order to confirming the payment and shipment.
Priority	High
Actors	Customer, Payment Gateway, Warehouse System
Pre-conditions	1 - The Customer Account should exist. 2 - The Customer is logged in. 3 - The items to be purchased are in stock. 4 - The Payment Gateway is operational.
Post-conditions	The order is placed, payment is confirmed, and the warehouse is notified to ship the items.
Dependencies	Integration with Payment Gateway and Warehouse System.

Typical Course of Action (Primary Flow)

S#	Actor Action	S#	System Response
1	Select items to purchase and proceed to checkout.	1.1	Display the order summary page.
2	Review order and click "Place Order."	2.1	Initiate the payment process by redirecting to the Payment Gateway.
3	Enter payment details and confirm payment.	3.1	Verify payment information with the Payment Gateway.
		3.2	Upon successful payment, save the order to the database.
		3.3	Notify the Warehouse System to prepare the order for shipment.
		3.4	Display an order confirmation message and send a confirmation email to the Customer.

Alternative Course of Action			
S#	**Actor Action**	S#	**System Response**
1	Select items to purchase but decides to edit the cart.	1.1	Allow the Customer to modify the cart and update the order summary.
2	Proceed to checkout after editing.	2.1	Display the updated order summary page and continue with the normal flow.

Exceptions

Payment Failure: If the payment fails, display an error message and prompt the Customer to retry the payment or choose a different payment method. The order is not saved until the payment is successful.

Out of Stock: If an item goes out of stock during checkout, notify the Customer and allow them to either remove the item from the cart or choose a different item before proceeding.

System Error: If there is a system error during order processing, an error message is displayed, and the order is not placed. The Customer is prompted to retry later.

References

- Object-Oriented Analysis and Design with Applications, Grady Booch et al., 3rd Edition, Pearson, 2007.
- Timothy C. Lethbridge, Robert Laganaiere, Object-Oriented Software Engineering (2nd Edition), McGraw Hill, 2005
- Object-Oriented Modeling and Design with UML, Michael R. Blaha and James R. Rumbaugh, 2nd Edition, Pearson, 2005.
- Larman, Craig. Applying UML and patterns: an introduction to object oriented analysis and design and interative development. Pearson Education India, 2012.