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).

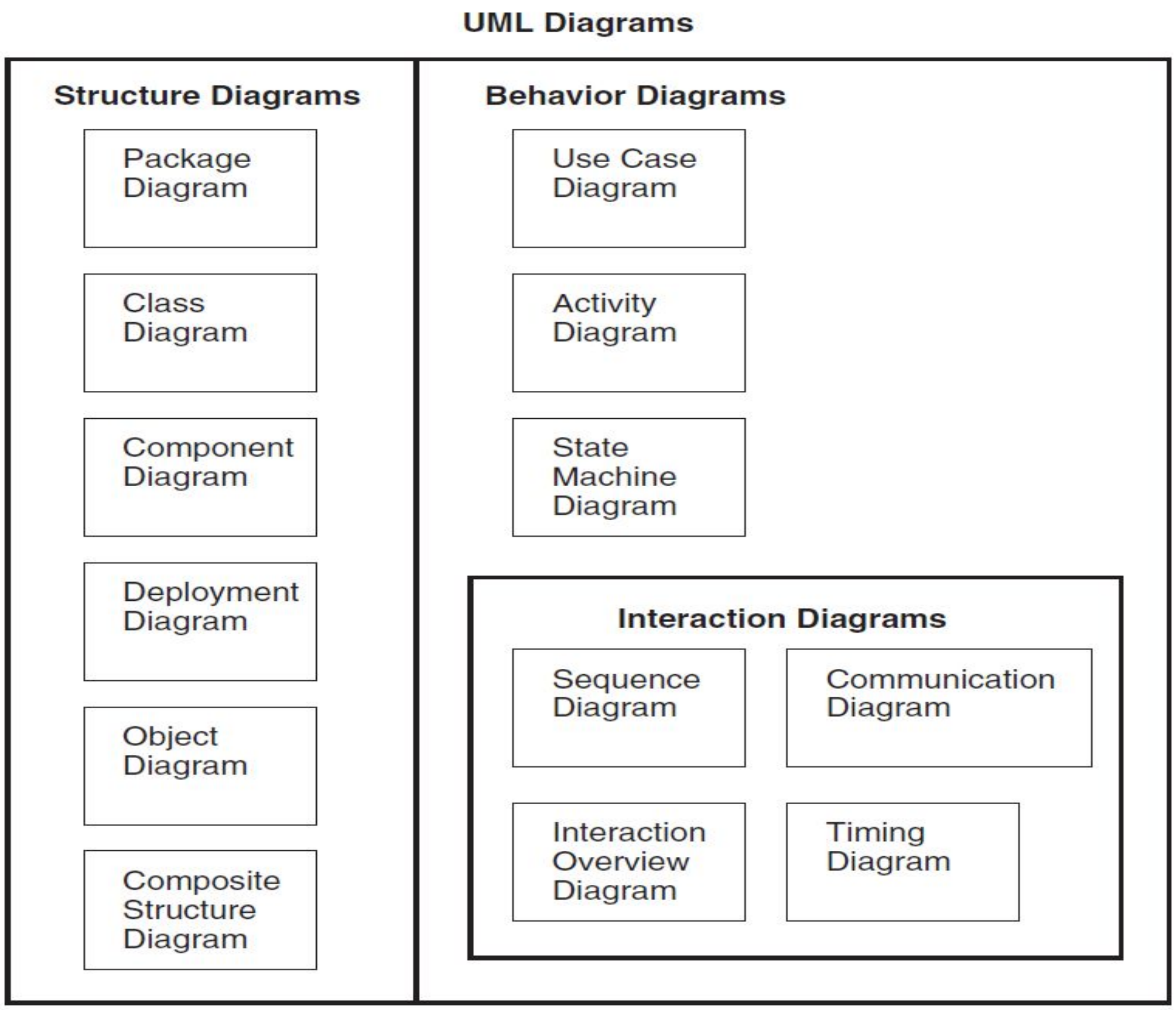
Structure Diagrams

- 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

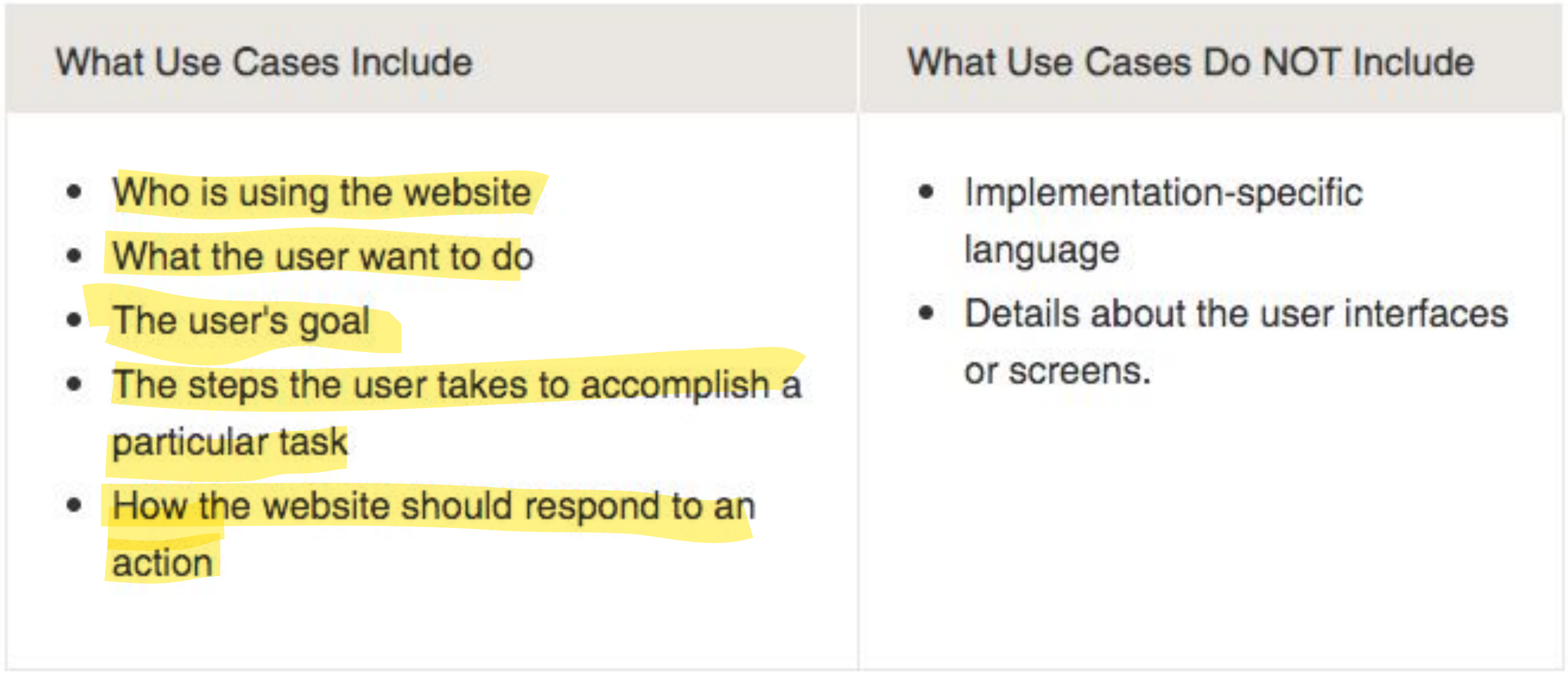
  - The Interaction taking place.
  - The Measurable Benefits to the user interacting with the system
  - The Effect of that Interaction on the system.

- 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 are you 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

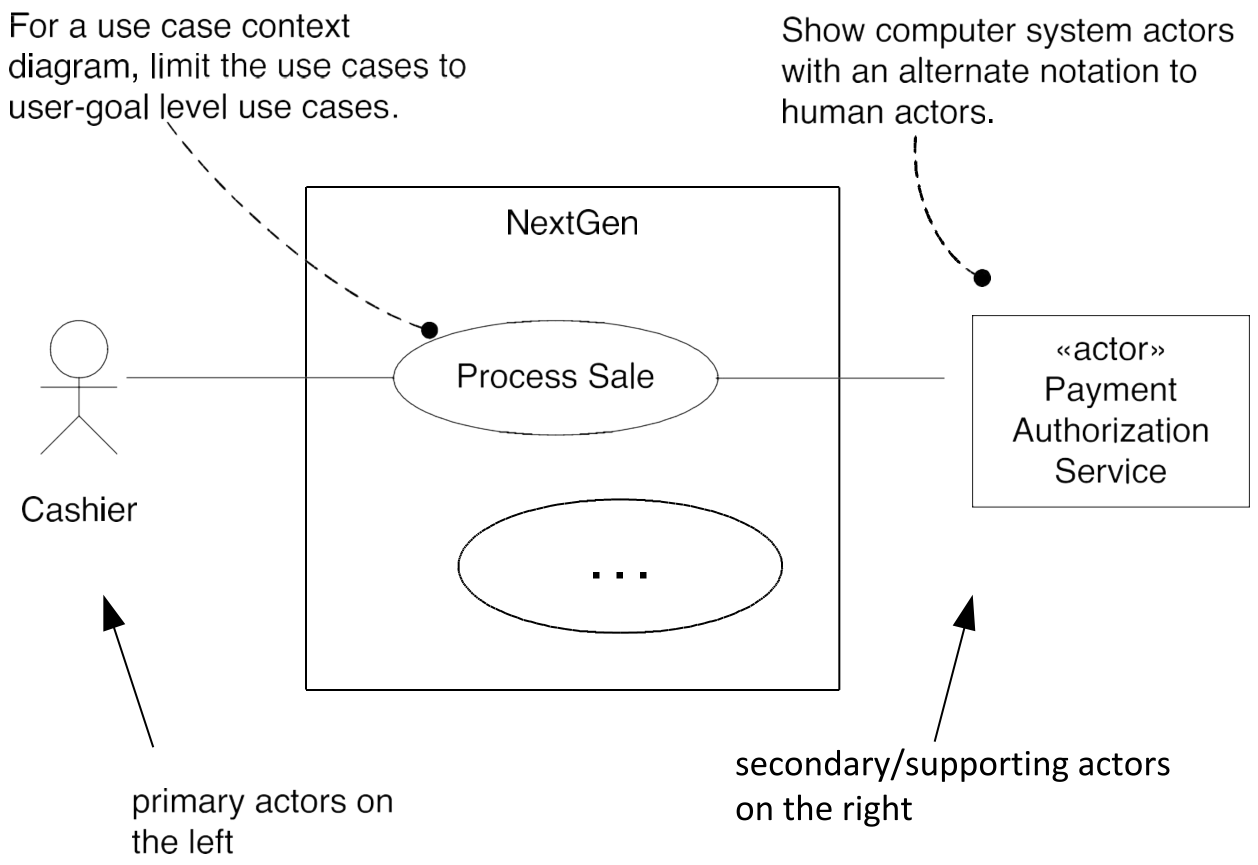
Applying Tests

- 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.

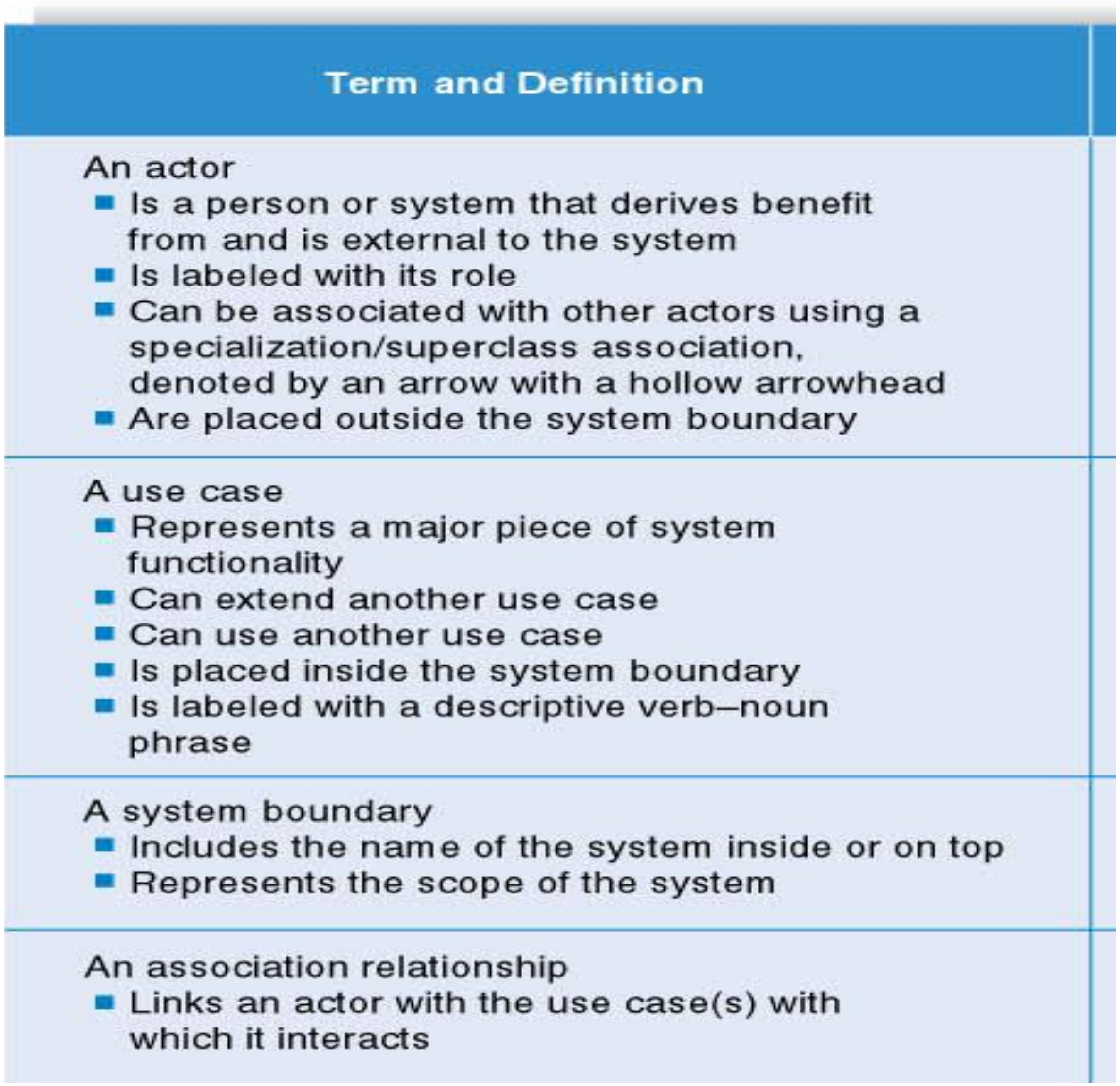
Use Case Diagram Concepts

- 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

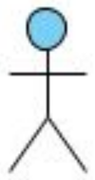

For a use case context diagram, limit the use cases to user-goal level use cases.
Show computer system actors with an alternate notation to human actors.
NextGen
Process Sale
...
Cashier
«actor» Payment Authorization Service
primary actors on the left
secondary/supporting actors on the right

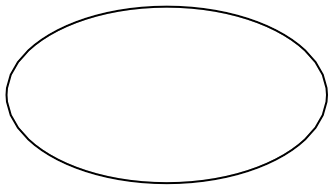
Syntax for Use Case Diagram

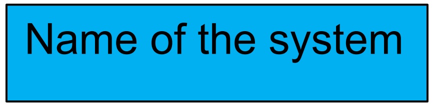


Term and Definition
An actor ■ Is a person or system that derives benefit from and is external to the system ■ Is labeled with its role ■ Can be associated with other actors using a specialization/superclass association, denoted by an arrow with a hollow arrowhead ■ Are placed outside the system boundary
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
An association relationship ■ Links an actor with the use case(s) with which it interacts

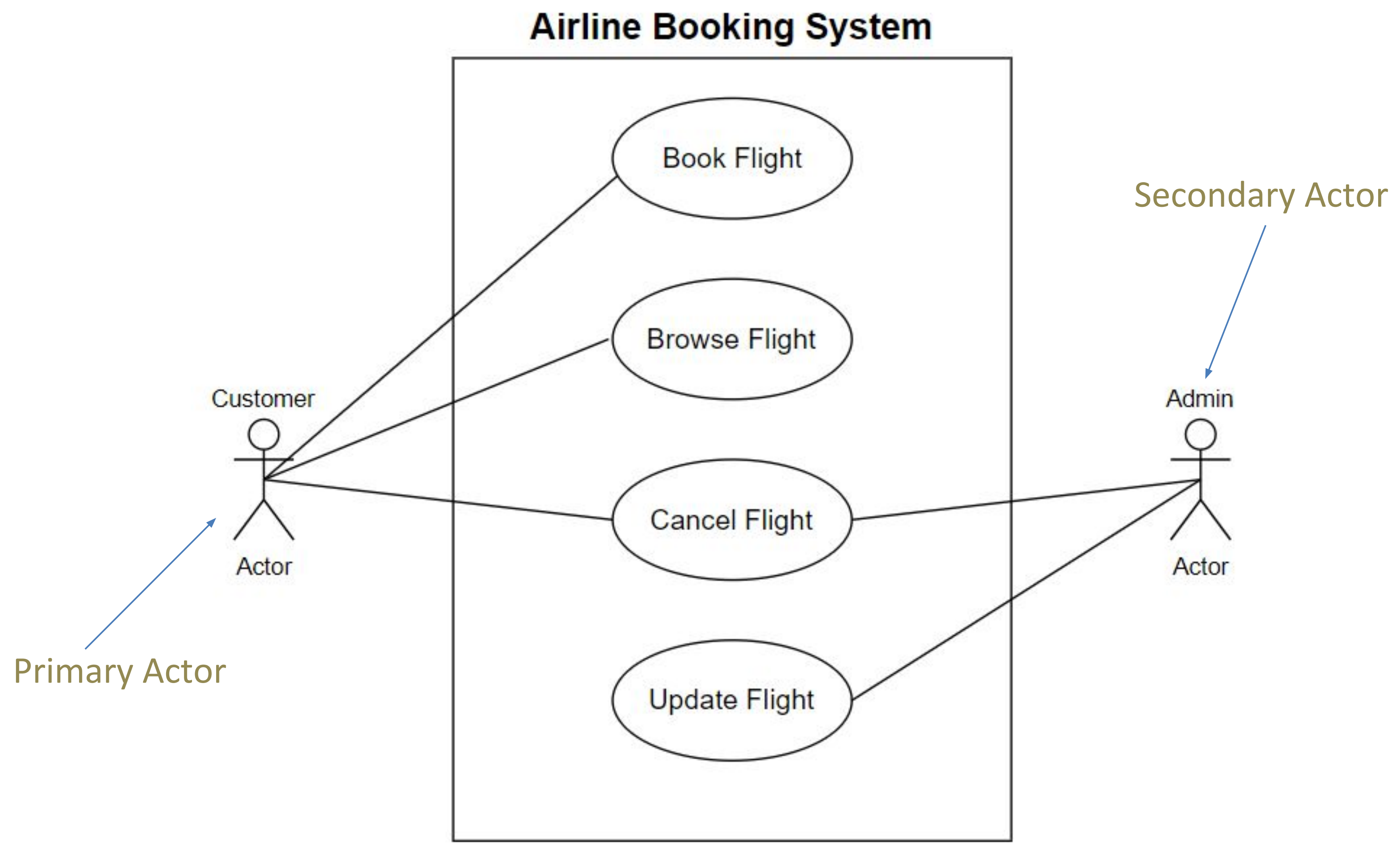
Notations







Use Case Diagram for Airline Booking System

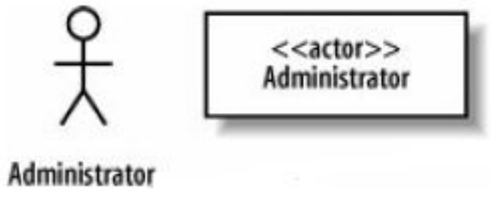


Example

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 service.

The requirement indicates 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.

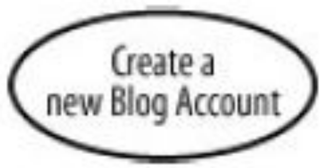
Example

- 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 service.

to create a new blog account

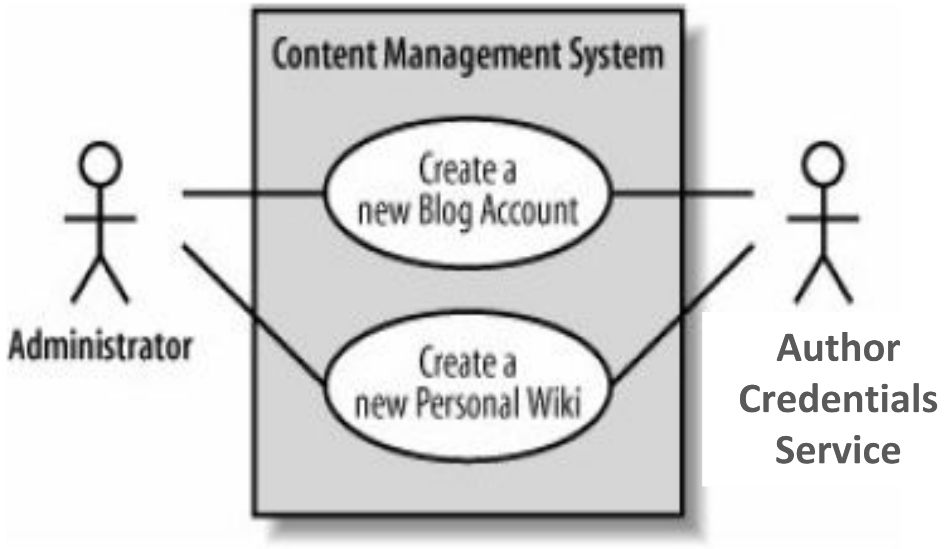
UML Representation



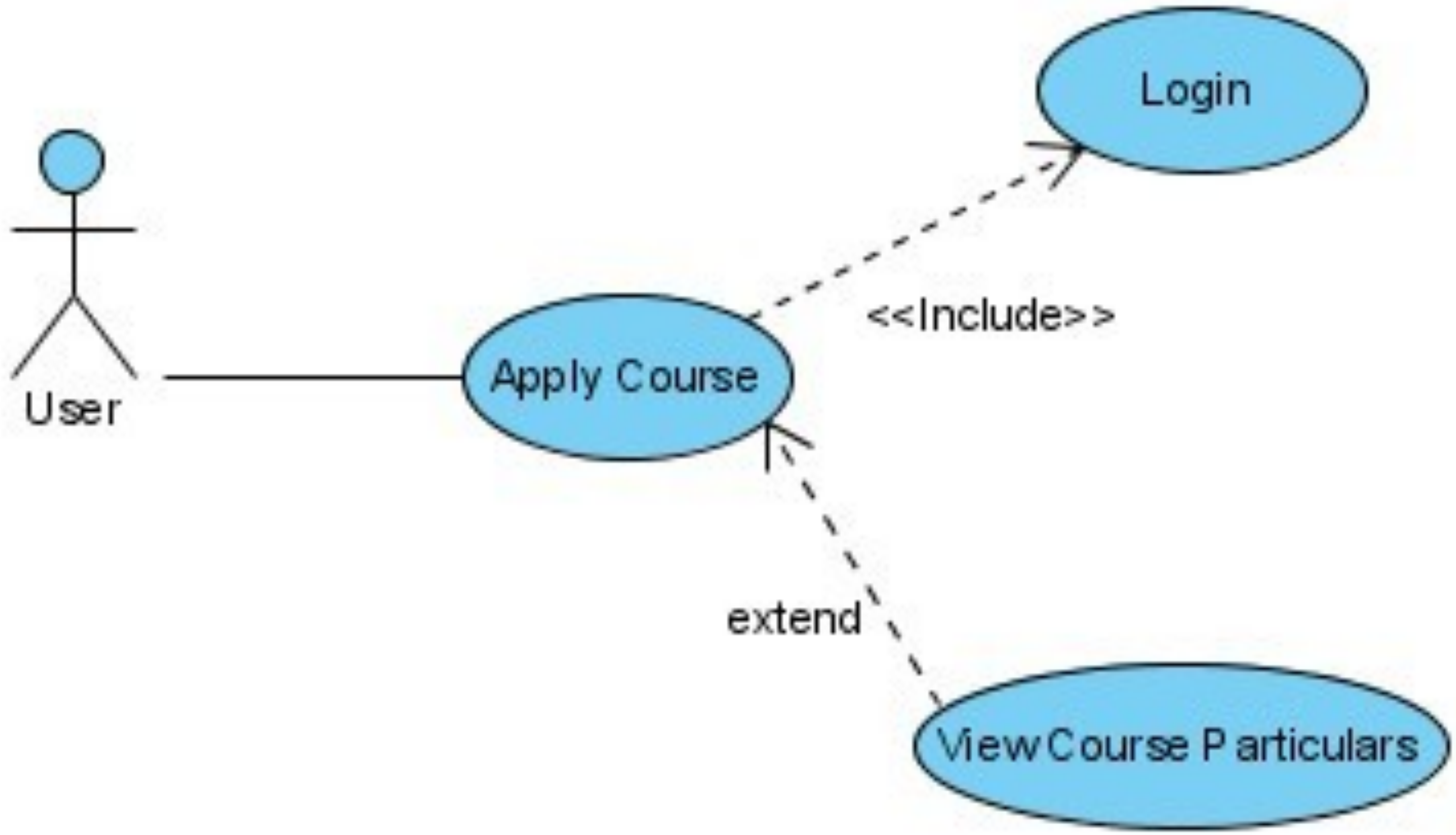
Example (Cont)

Requirement 2

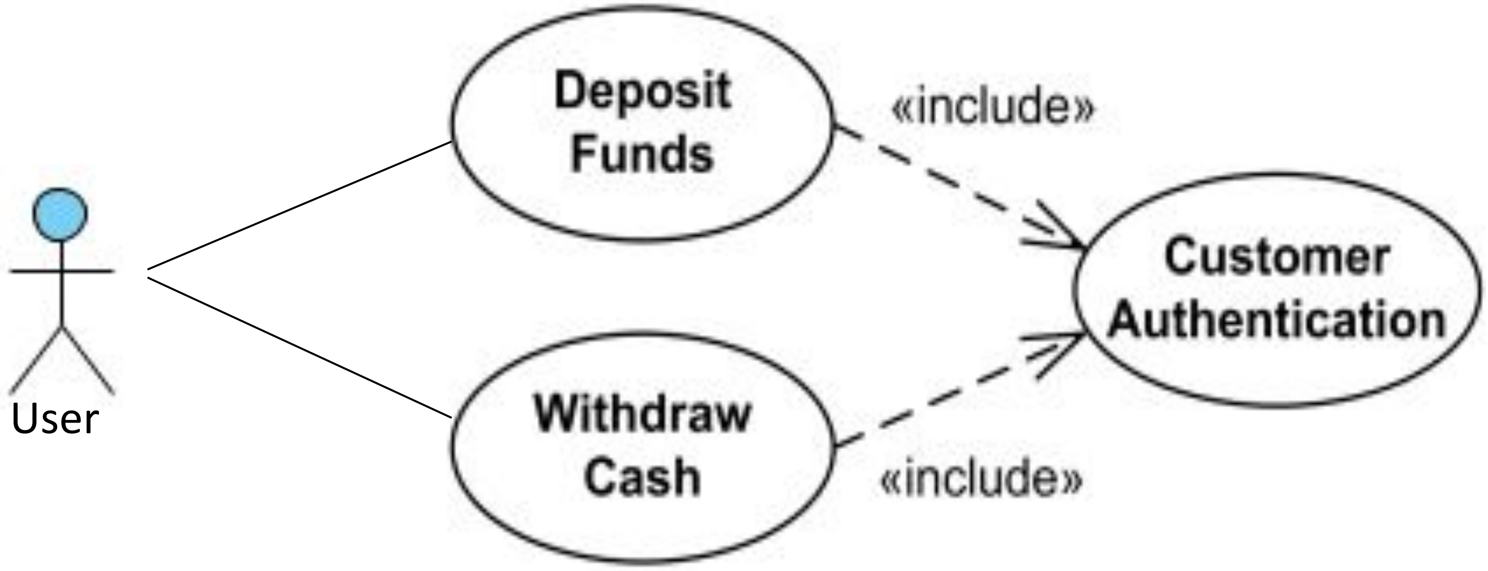
The content management system shall allow an administrator to create a new personal Wiki, provided the personal details of the applying author are verified using the Author Credentials Service.


Content Management System
Create a new Blog Account
Create a new Personal Wiki
Administrator
Author Credentials Service

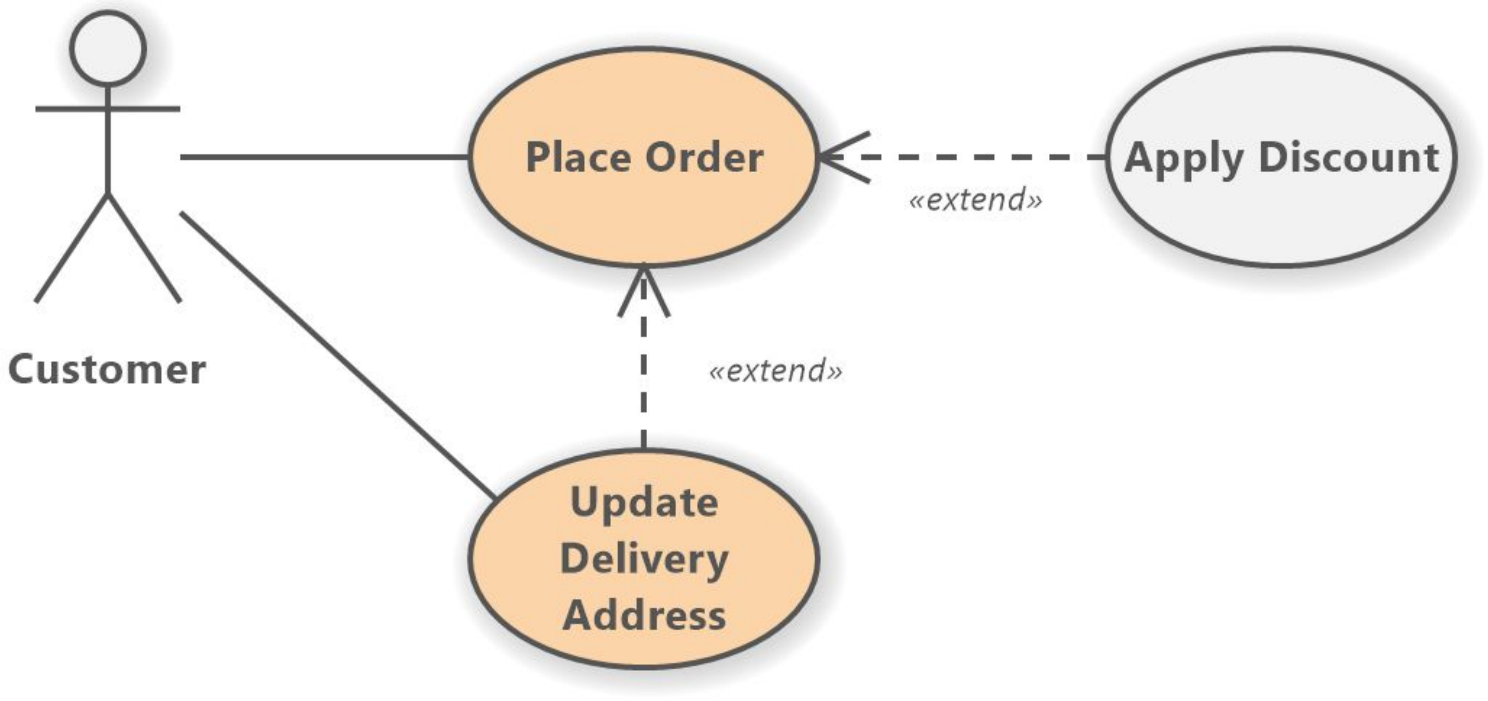
Example of Stereotypes – Include and Extend



Example – Include and Extend



Example – Include and Extend



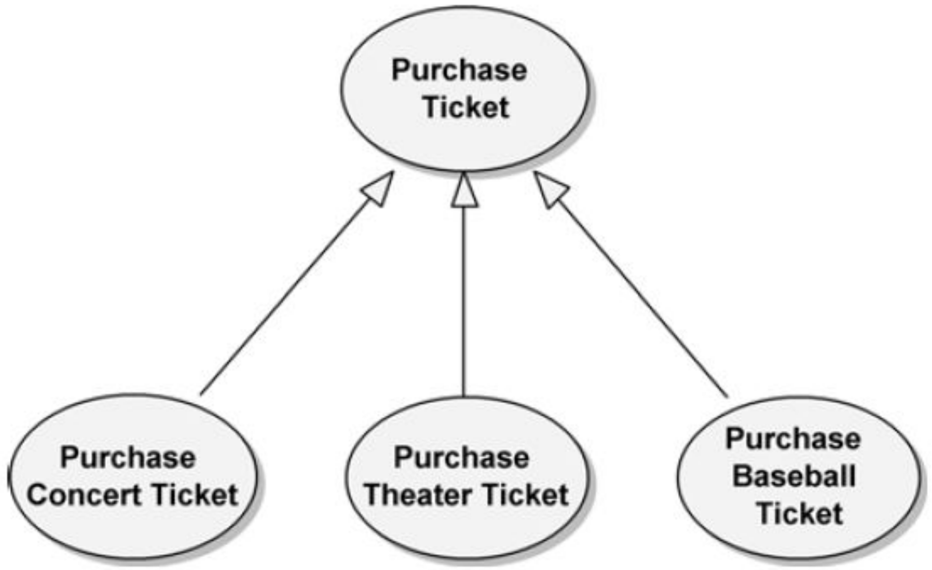
Include and Extend Relationships

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

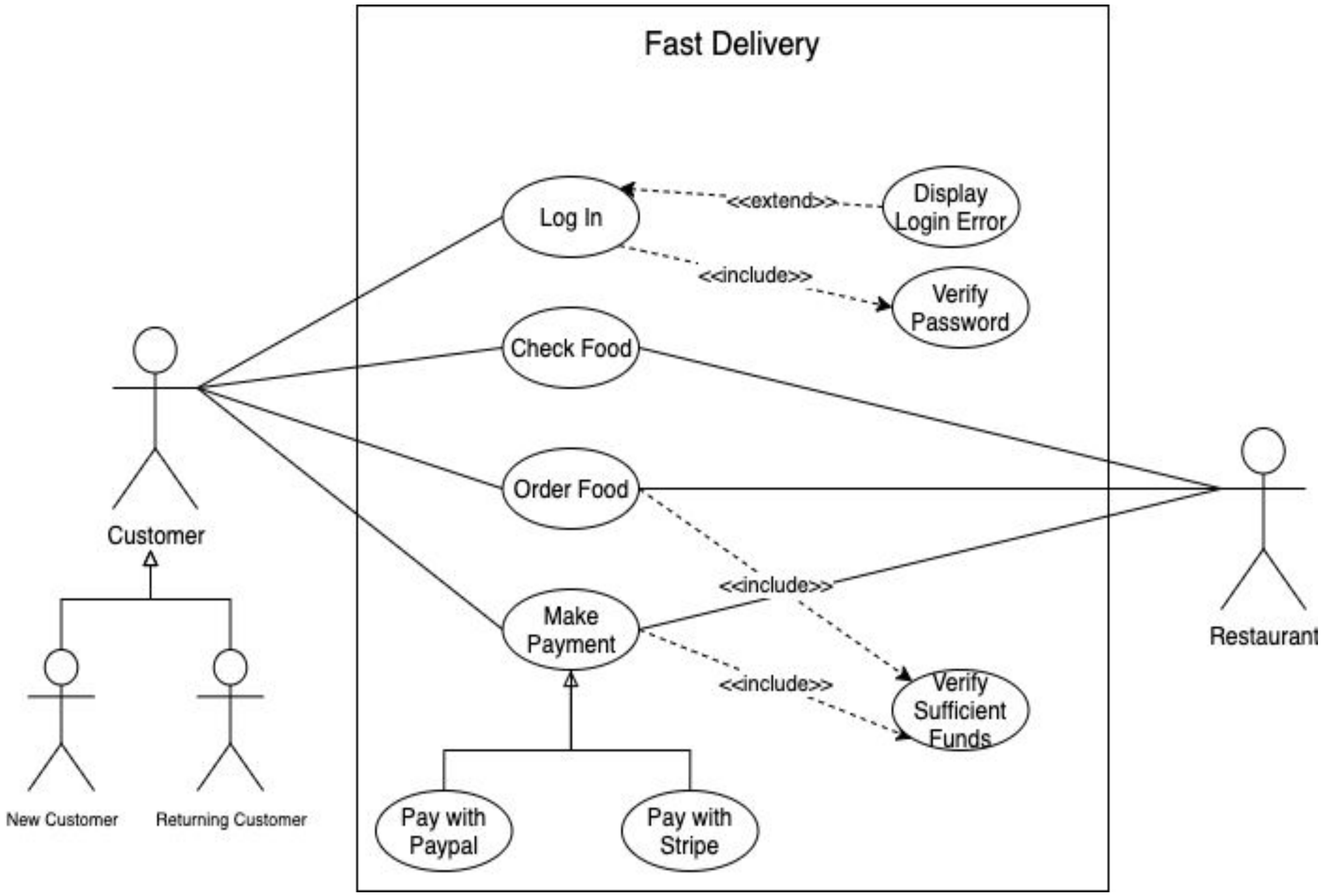
- Only use them if they are necessary.

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

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.