CESE4015 Software Systems

Unified Modeling Language: An Introduction

Guohao Lan
Embedded Systems Group

December 12th 2023

Learning objectives

- At the end of the course, you should be able to:
  - Understand:
    - The purpose of UML (unified modeling language)
    - Three categories of UML diagrams:
      - Structural, behavioral, and interactional.
  - When and how to apply basic UML diagrams to model software systems.

- Assessment:
  - Modeling assignments using UML diagrams. [Group of two]
  - Reflection document on UML-based modeling. [Individual]

Agenda for UML

- Week 5 Lecture:
  - Background of UML
  - Use Case
- Week 5 Lab:
  - Modeling with UML diagrams (part 1)
- Week 6 Lecture:
  - Component, Deployment
  - Class, Sequence
- Week 6 Lab:
  - Modeling with UML diagrams (part 2)

Acknowledgements

- Slides materials are built from different sources:
  - Slides created by Marty Stepp, CSE403 @ UWashington.
  - *UML Distilled, 3rd edition* by Martin Fowler.
  - *The Unified Modeling Language Reference Manual, 2nd edition* by James Rumbaugh, Ivar Jacobson, and Grady Booch.
  - *Practical UML: A Hands-On Introduction for Developers* by Randy Miller.
  - *IBM Rational Software Architect Documentation:* https://www.ibm.com/docs/en/rational-soft-arch/9.5

- Lab platform:
  - PlantUML: https://plantuml.com/
  - A tutorial will be given during the lab sessions.

Agenda for UML

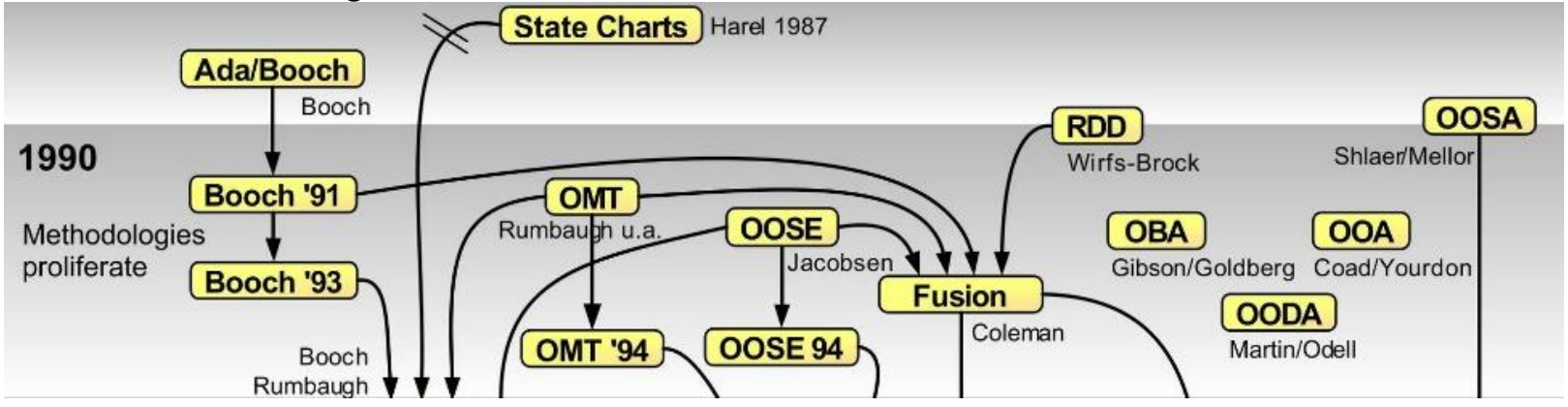
- Week 5 Lecture:
  - Background of UML
  - Use Case
- Week 5 Lab:
  - Modeling with UML diagrams (part 1)
- Week 6 Lecture:
  - Component, Deployment
  - Class, Sequence
- Week 6 Lab:
  - Modeling with UML diagrams (part 2)

Background

- What is the UML?
  - **UML**: A family of standardized graphical notations that helps in describing and designing software systems at a high level of abstraction.
  - It is a graphical design notation:
    - More clear than natural language and code.
    - Simplifies system design process and avoid a lot of details.
  - Help communicating ideas about a system design.
  - It is language and technology independent.

A brief overview of its history

- Driving force:
  - Programming languages do not provide a high enough level of abstraction to facilitate the design.

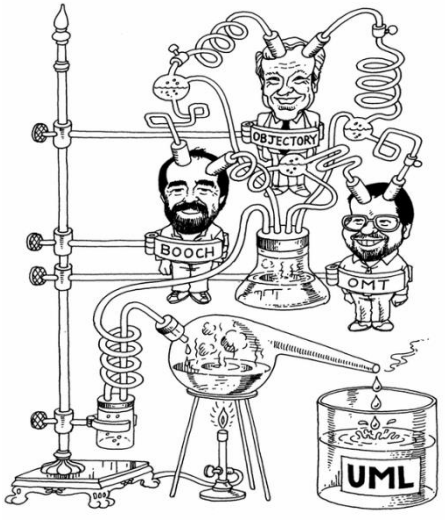


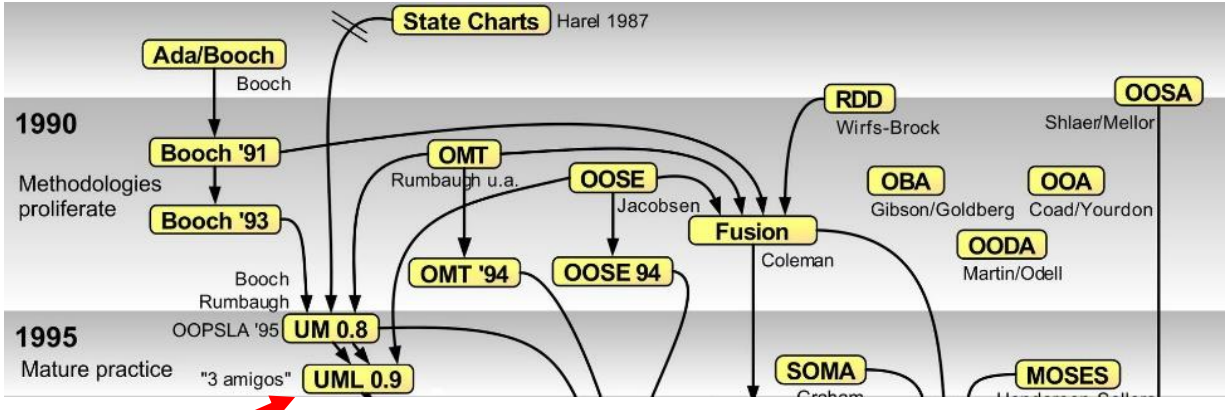
- UML is based on many earlier software design approaches:
  - Evolving since 1990s:
    - The Booch method by Grady Booch
    - The Object-modeling Technique (OMT) by James Rumbaugh
    - The Object-oriented Software Engineering (OOSE) by Ivar Jacobson

Each of these methods had its only notation and approach!

A brief overview of its history (cont.)

- Formation of UML:
  - The “three amigos” combined their ideas to create a unified modeling language:



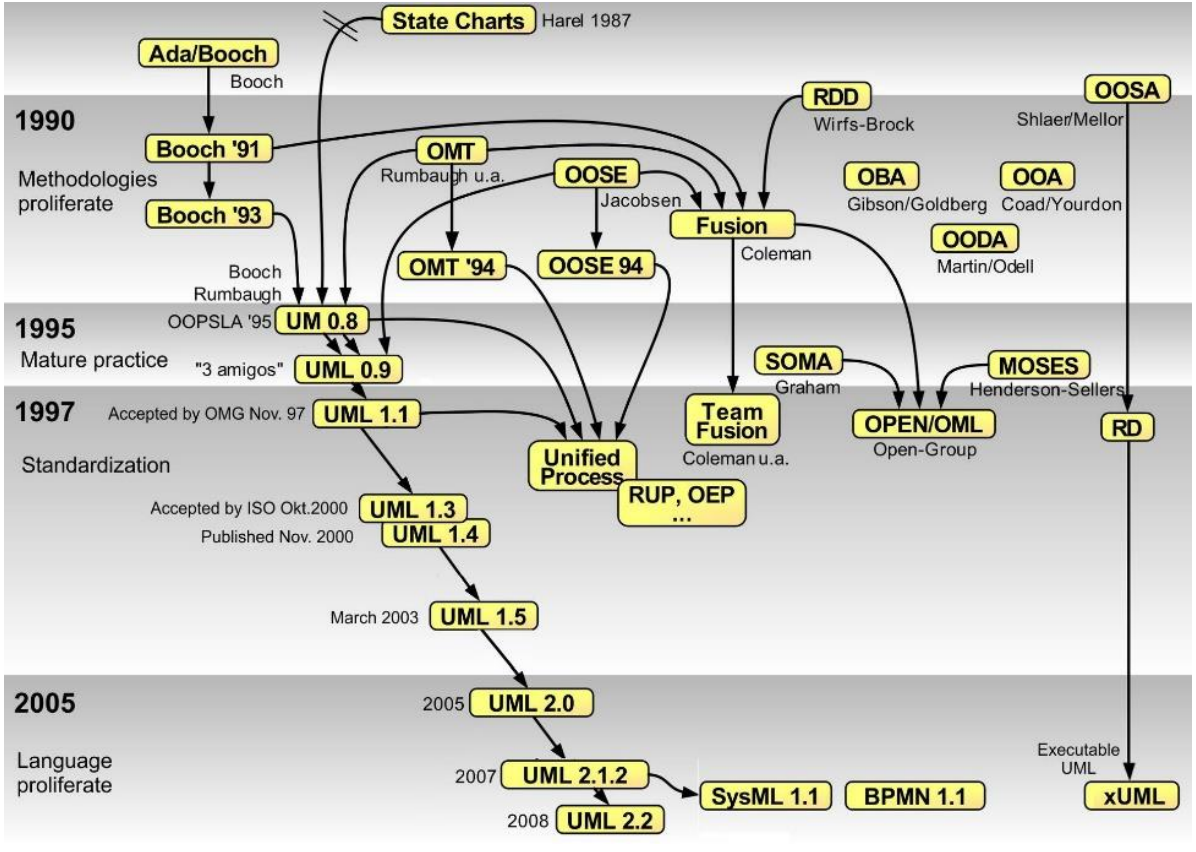


A brief overview of its history (cont.)

- Adoption by OMG and evolution:

UML was adopted as a standard by the Object Management Group (OMG)

Accepted by IOS as a standard and been periodically revised.

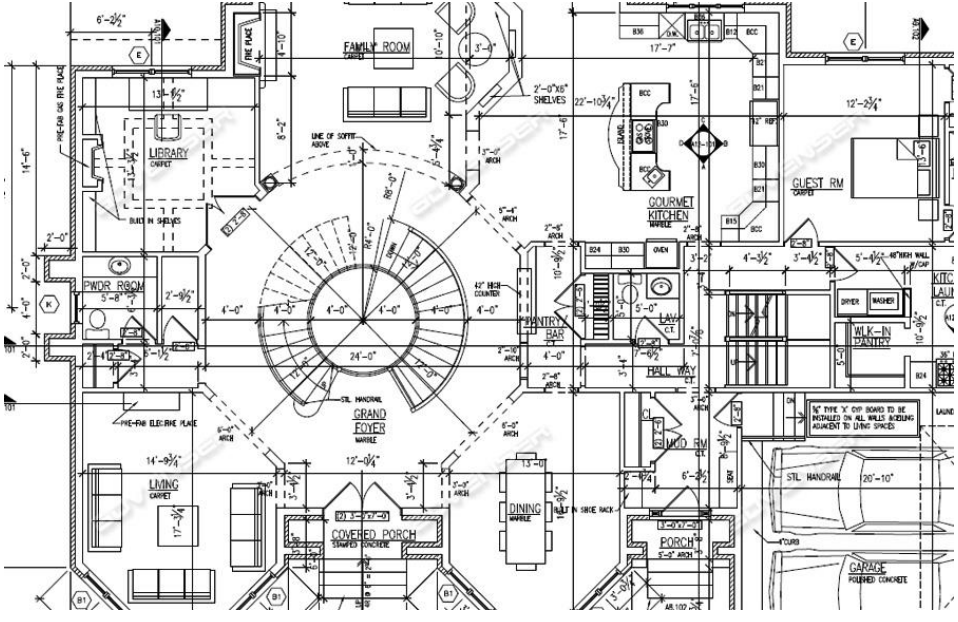


Why bother with the UML?

From the view of building construction:

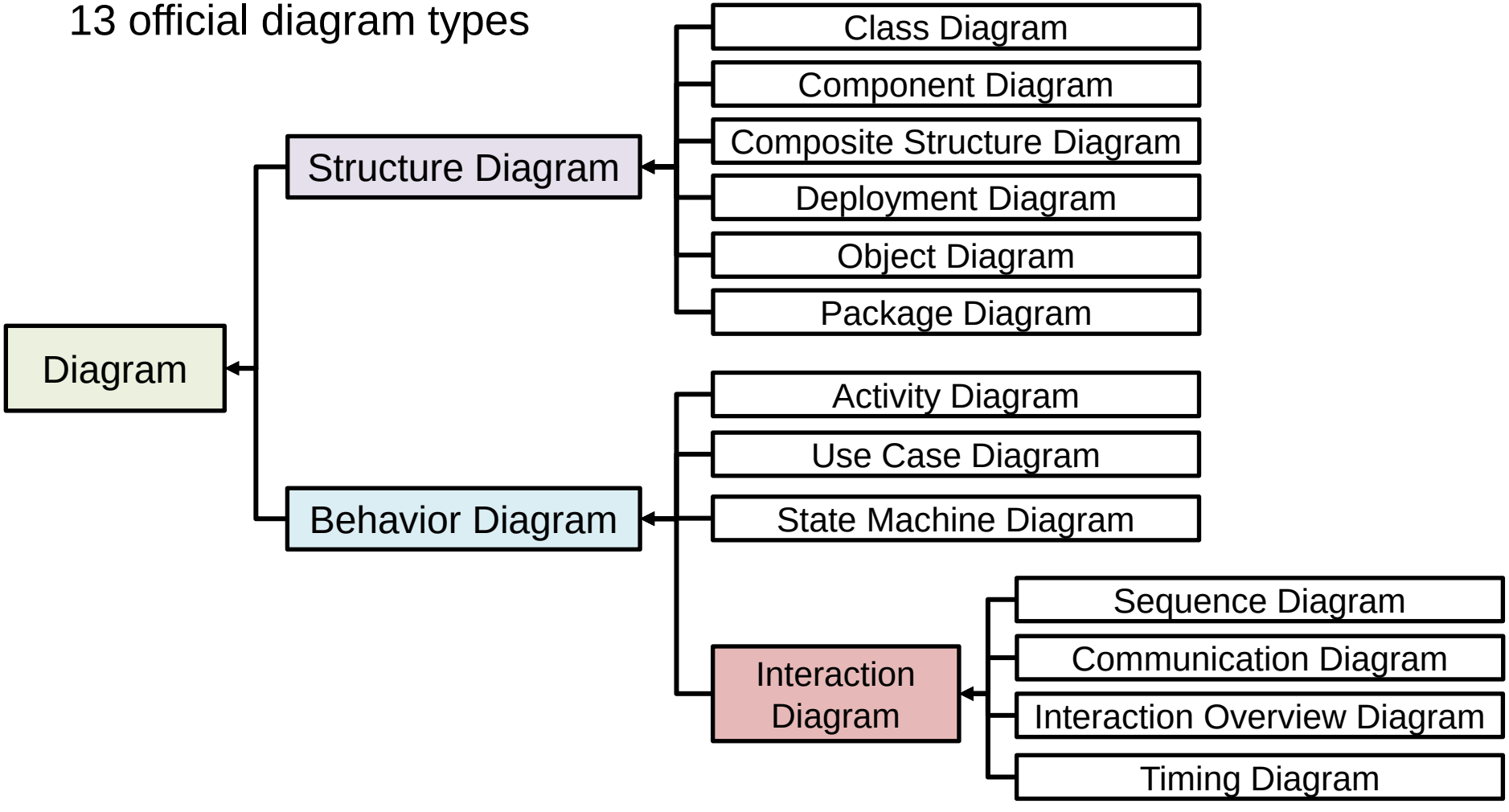
A unified standard that can be understood by architects and builders.

- **Clarity**: bring clarity to complex design.
- **Communication**: a universal language for diverse teams to understand.
- **Documentation**: a valuable way for future reference.



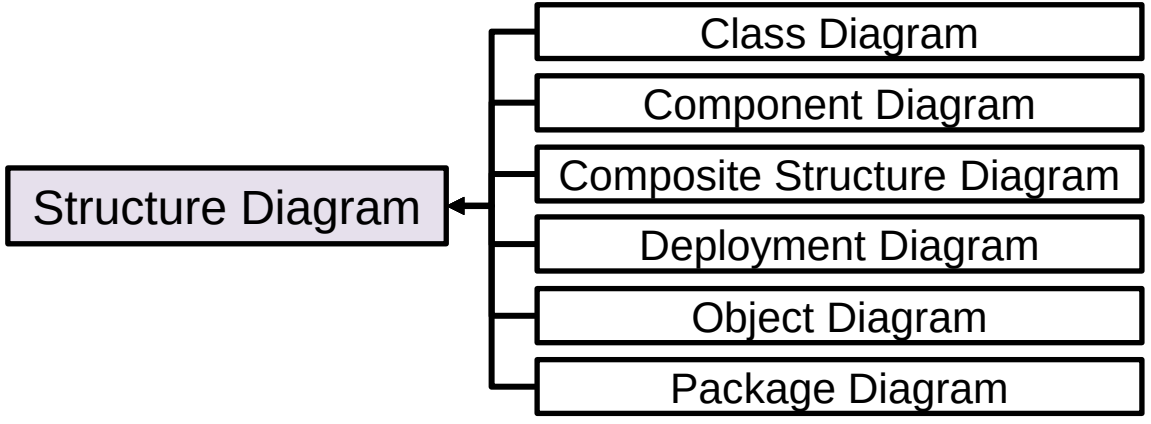
Overview of UML Diagrams

13 official diagram types



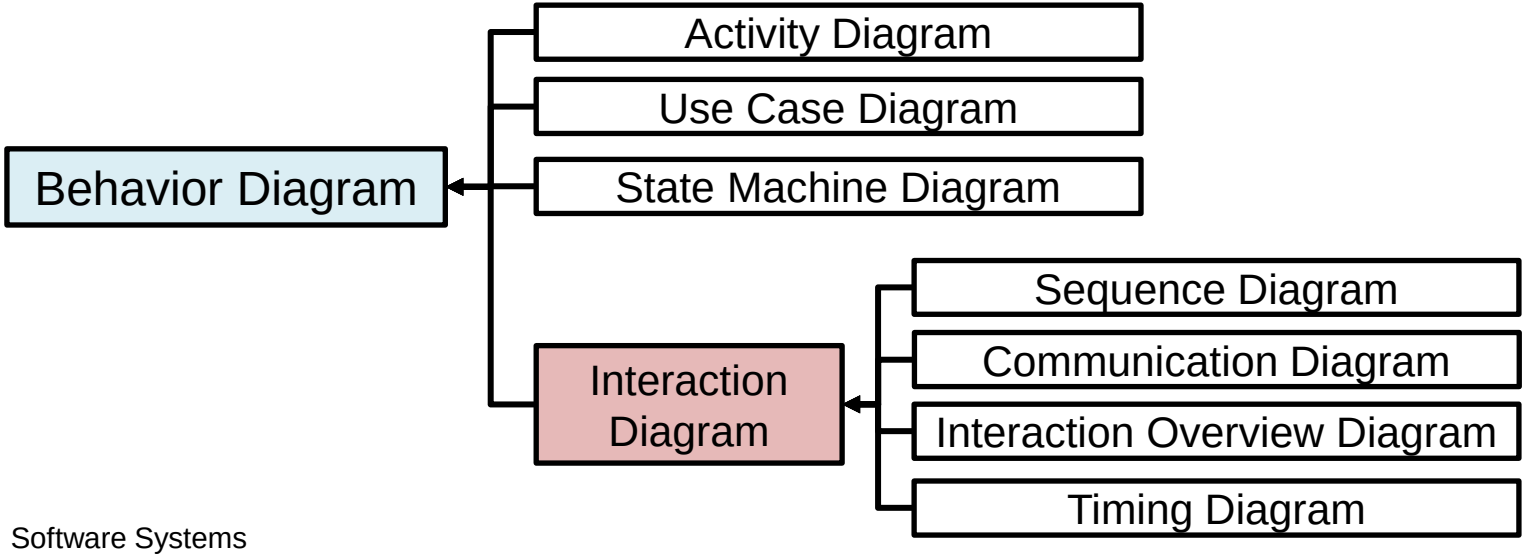
Overview of UML Diagrams (cont.)

- Three types of diagrams:
  - Structural diagrams:
    - Emphasizes the **static structure** of the system and the things that must be presented in the system, including objects, attributes, operations, components, and relationships.
    - Used extensively in documenting the architecture of the software systems.



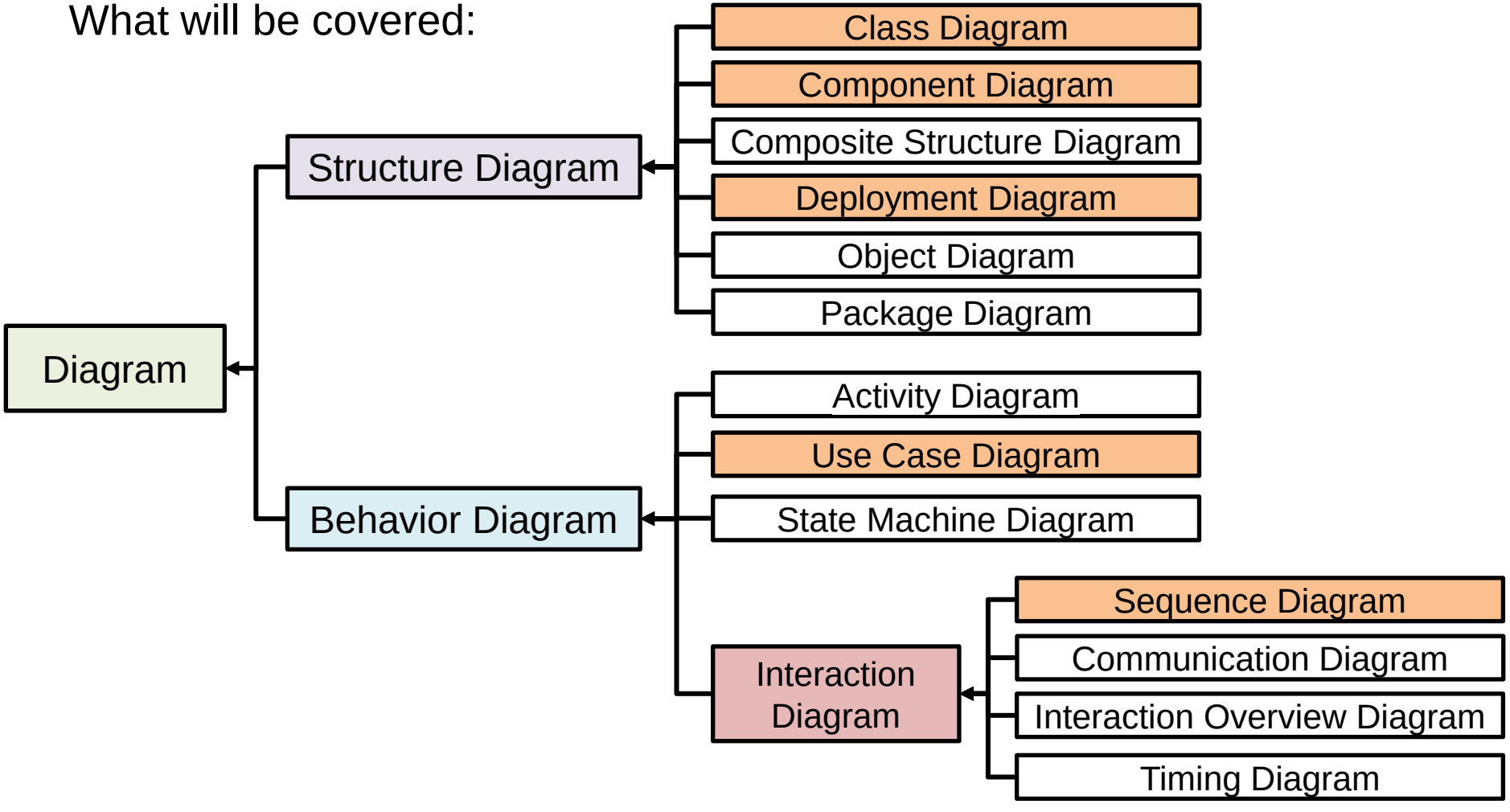
Overview of UML Diagrams (cont.)

- Behavioral diagrams:
  - Focuses on the dynamic behavior of the systems and changes to the internal states of objects.
    - Behavior: how data moves; how does the system change in time; how system behaves with different events.
  - Interaction diagrams:
    - Interaction: emphasize the flow of control, showing collaborations among objects; how objects communicate;

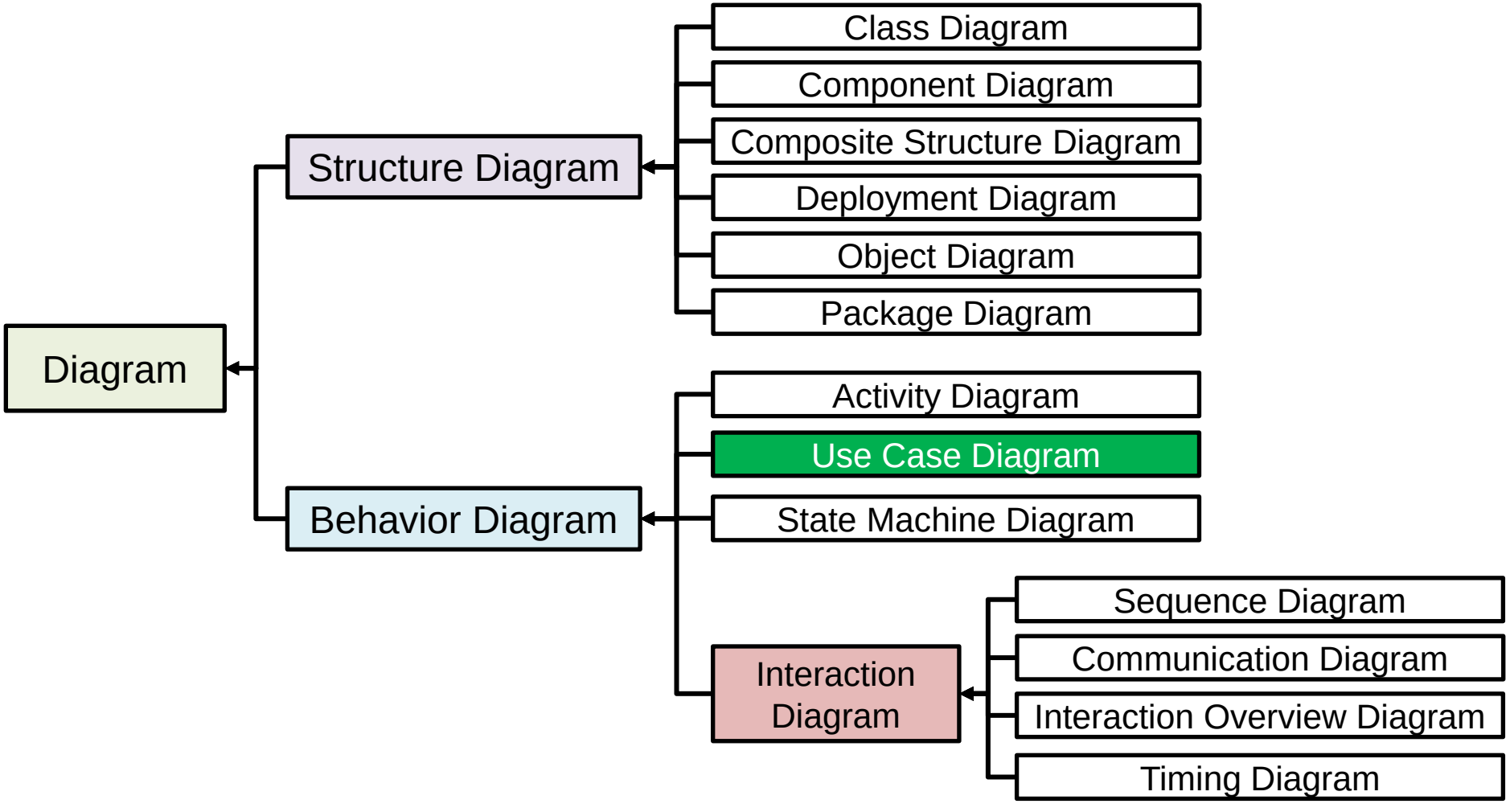


Overview of UML Diagrams (cont.)

What will be covered:



Use Case Diagram



Use Case Diagram (cont.)

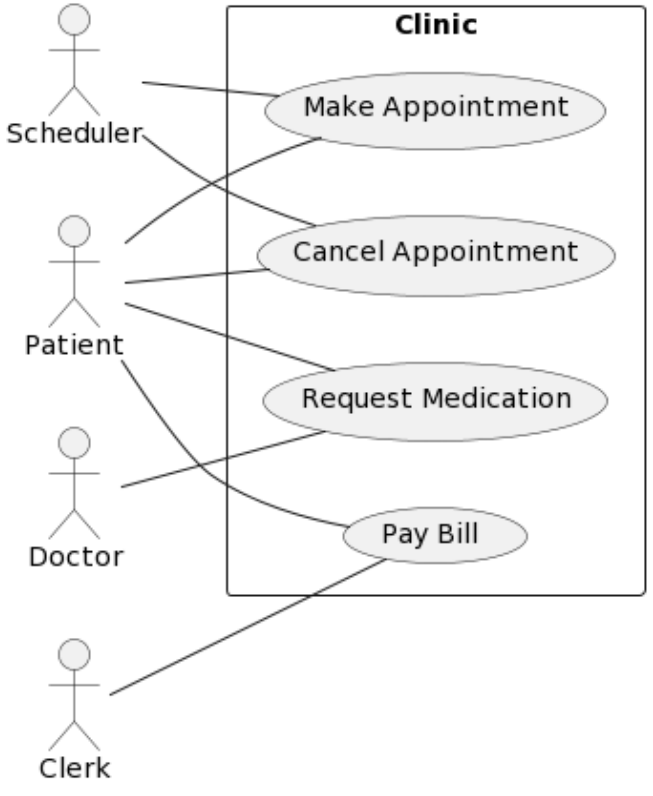
- What is the Use Case Diagram?

- **Use Case Diagram:** a collection of actors, use cases, and their associations that describes what a system does from the standpoint of an external observer.

Use Case Diagram (cont.)

- What is the Use Case Diagram?

Clinic management system

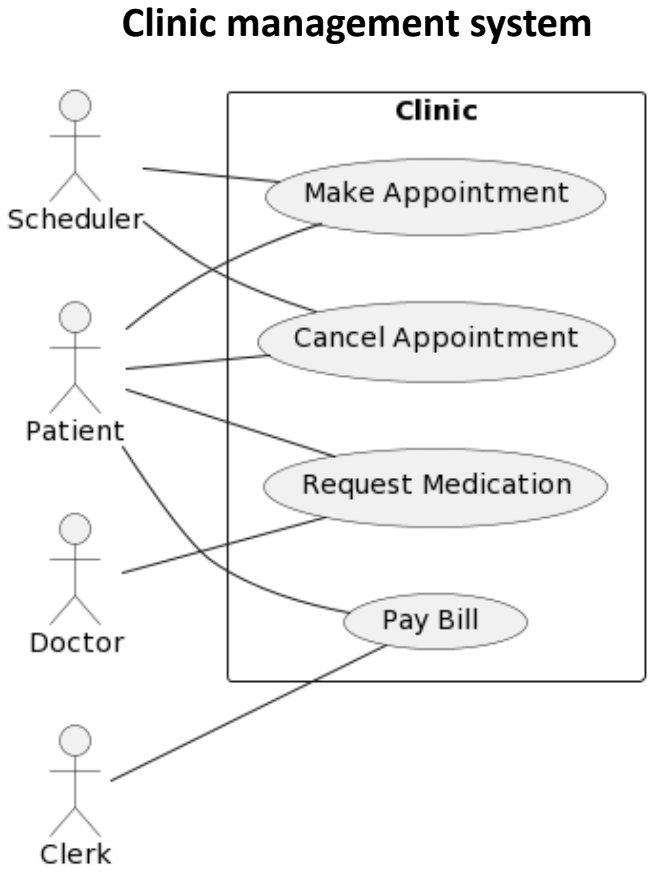


- **Discussion**:
  - What do you see in this diagram?
  - What are the elements in this diagram?
  - What message(s) this diagram may try to deliver?

Think/write → Pair → Share

Use Case Diagram (cont.)

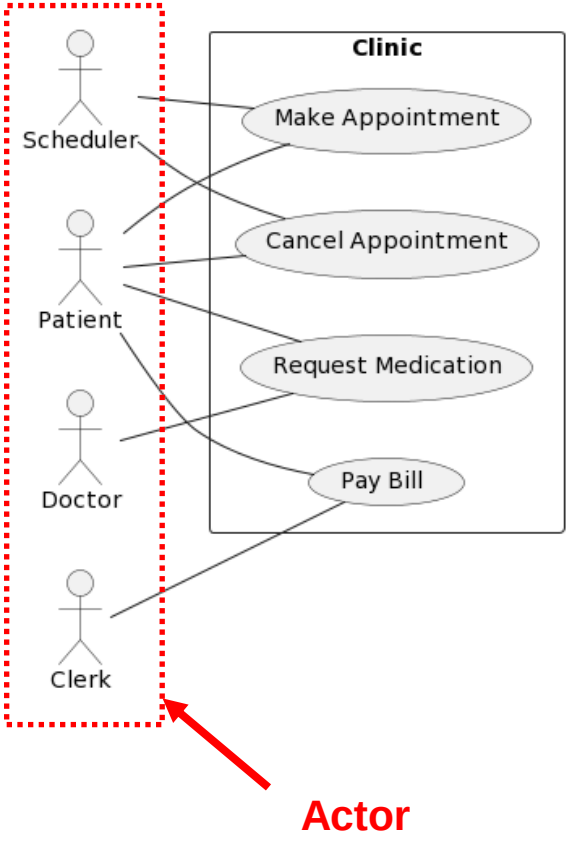
- A Use Case Diagram:
  - Presents the users of the system and their interactions with the system.
  - Shows high-level overview of relationship between use cases, actors, and the system.
  - Does not provide a lot of details.



Use Case Diagram (cont.)

- Elements in the Use Case Diagram:

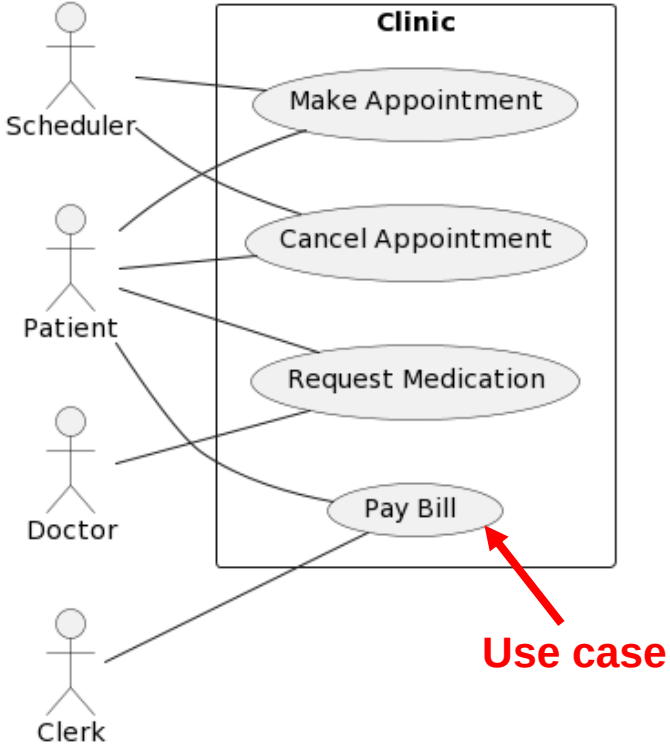
- **Actor**: represents a **role** played by a user or any other system that interacts with the system being modeled.
  - Focus: who will use (interact with) the system?
  - Represented by stick figures.
  - Actors must be external entities that produce or consume data (interact with the system).
  - Actor is different from the concept of user – a user can act as different actors.



Use Case Diagram (cont.)

- Common elements in the Use Case Diagram:

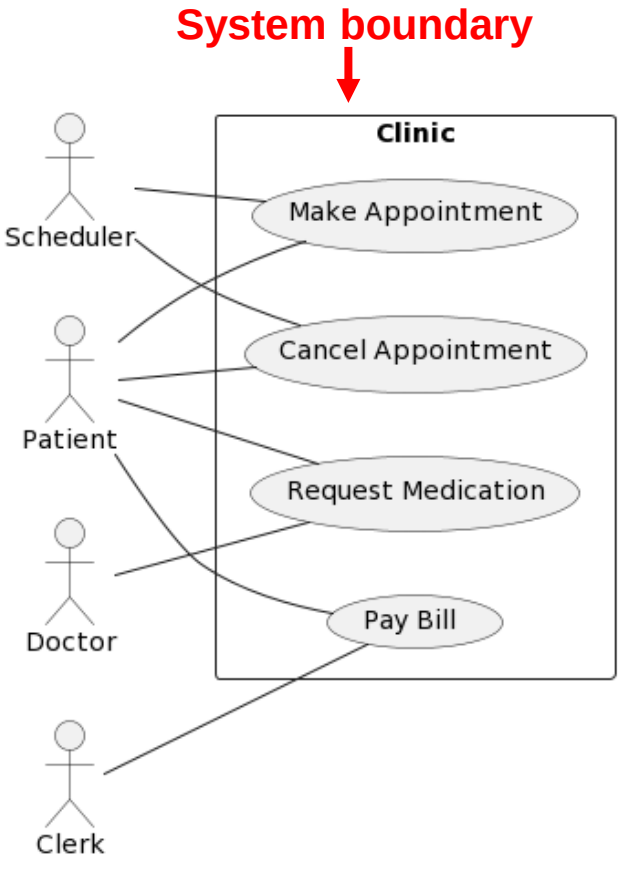
- **Use case:** is a summary of scenarios that describes the typical interaction between the actors of a system and the system itself.
  - Represented by horizontally shaped ovals
  - Typically represent system function from a user's point of view.
  - Focus on what, not how!
    - A simple and descriptive way to show what the system does from the user's perspective.



Use Case Diagram (cont.)

- Common elements in the Use Case Diagram:

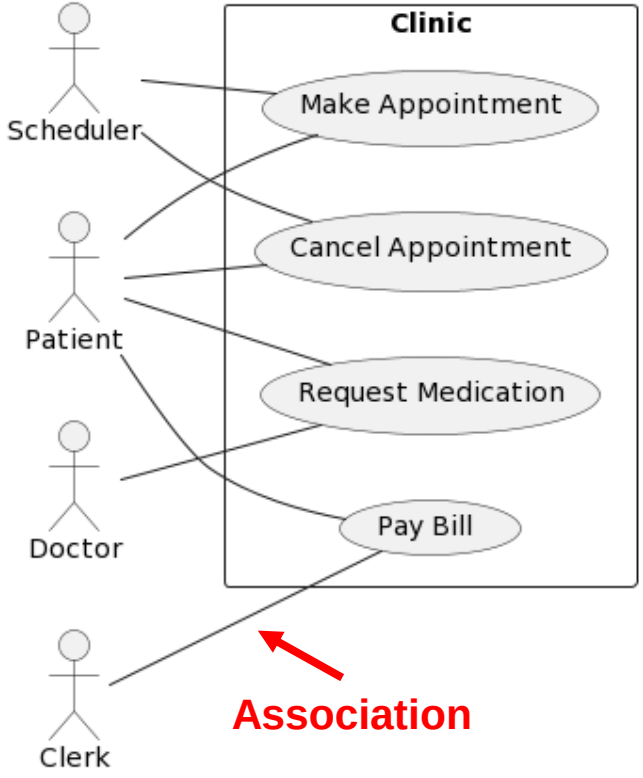
- **System boundary:** a rectangle that separates the system from the external actors.
  - It defines the scope of the system.
  - All use cases outside the boundary box are outside the scope of the system.
  - For large and complex systems, each module may be the system boundary.



Use Case Diagram (cont.)

- Common elements in the Use Case Diagram:

- **Association**: illustrates the relationship between an actor and a use case.
  - A solid line between actor and user case. **[No arrow!]**
  - Shows an actor can initiate or interact with the process defined in the use case.
  - No flow of control. No sequence or timing.



Exercise

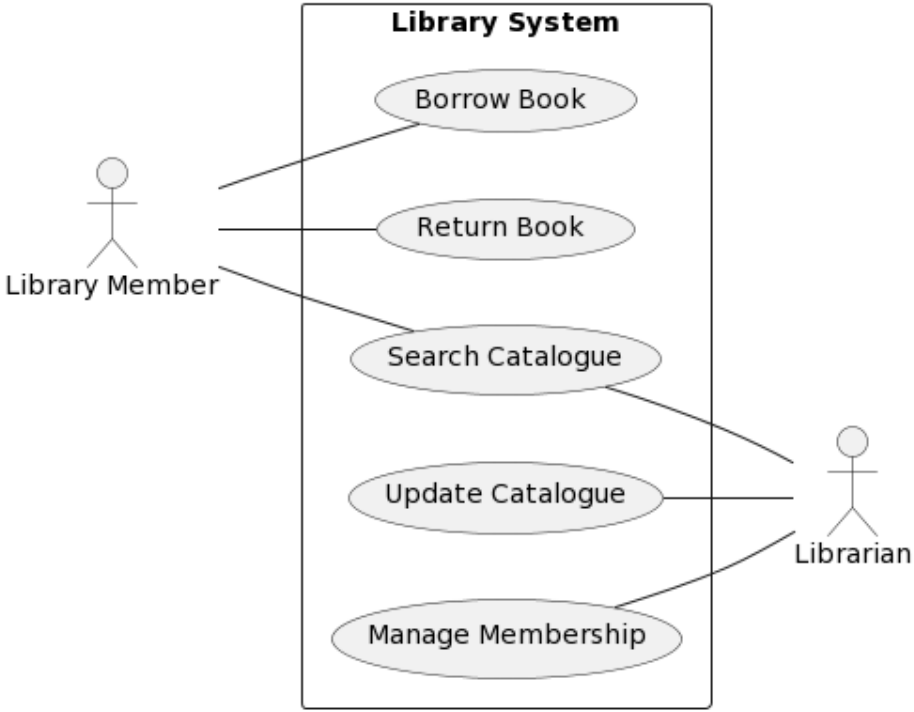
- Draw a use case diagram for the scenario described below:

Imagine a Library System that has two main types of users: library members and librarians. Library Members can use the system to borrow books to take home and return books when they have finished reading. They can also search their book of interest. On the other side, librarians are responsible for updating the catalogue and managing memberships. Both library members and librarians can search books.

Think/draw → Pair → Share

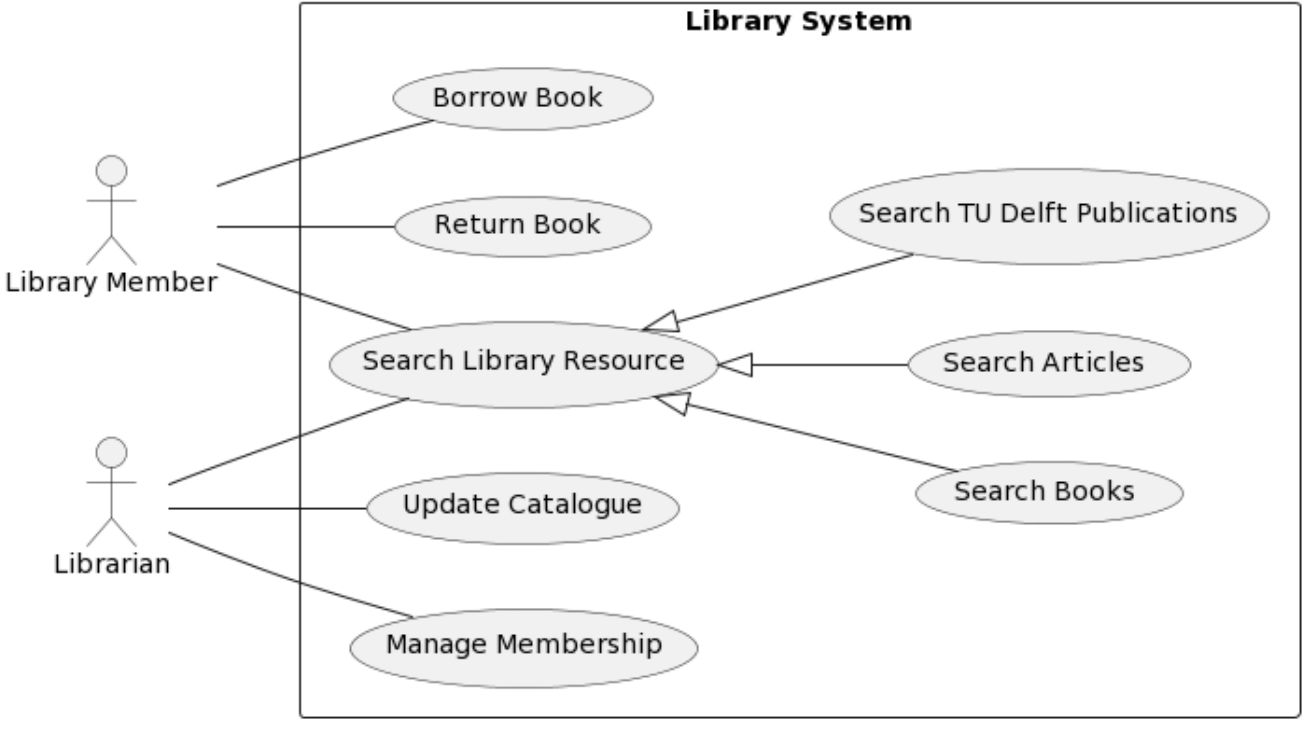
Exercise (cont.)

Imagine a Library System that has two main types of users: **Library Members** and **Librarians**. Library Members can use the system to **borrow books** to take home and **return books** when they have finished reading. They can also **search their book** of interest. On the other side, librarians are responsible for **updating the catalogue** and **managing memberships**. Both library members and librarians can search books.



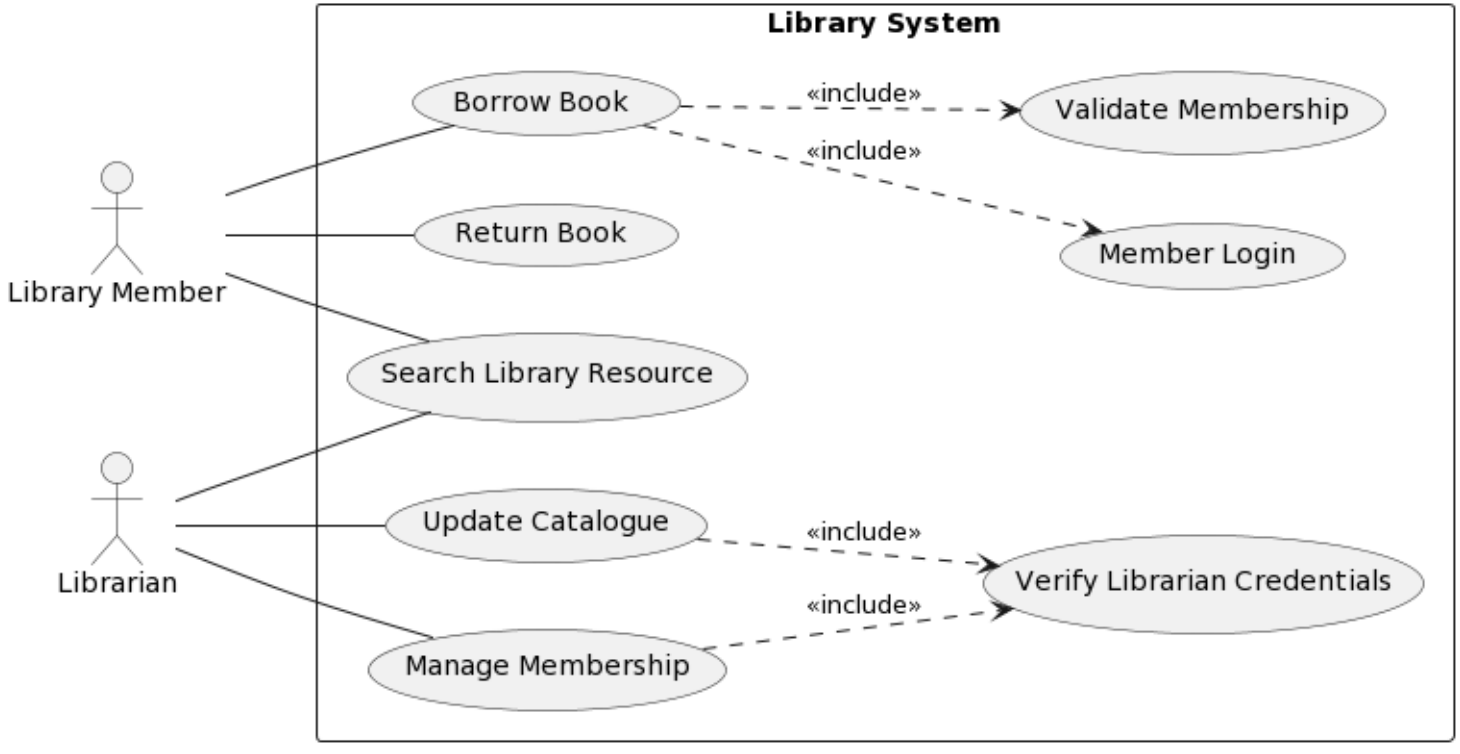
Use Case Diagram (cont.)

- Use Case Relationship:



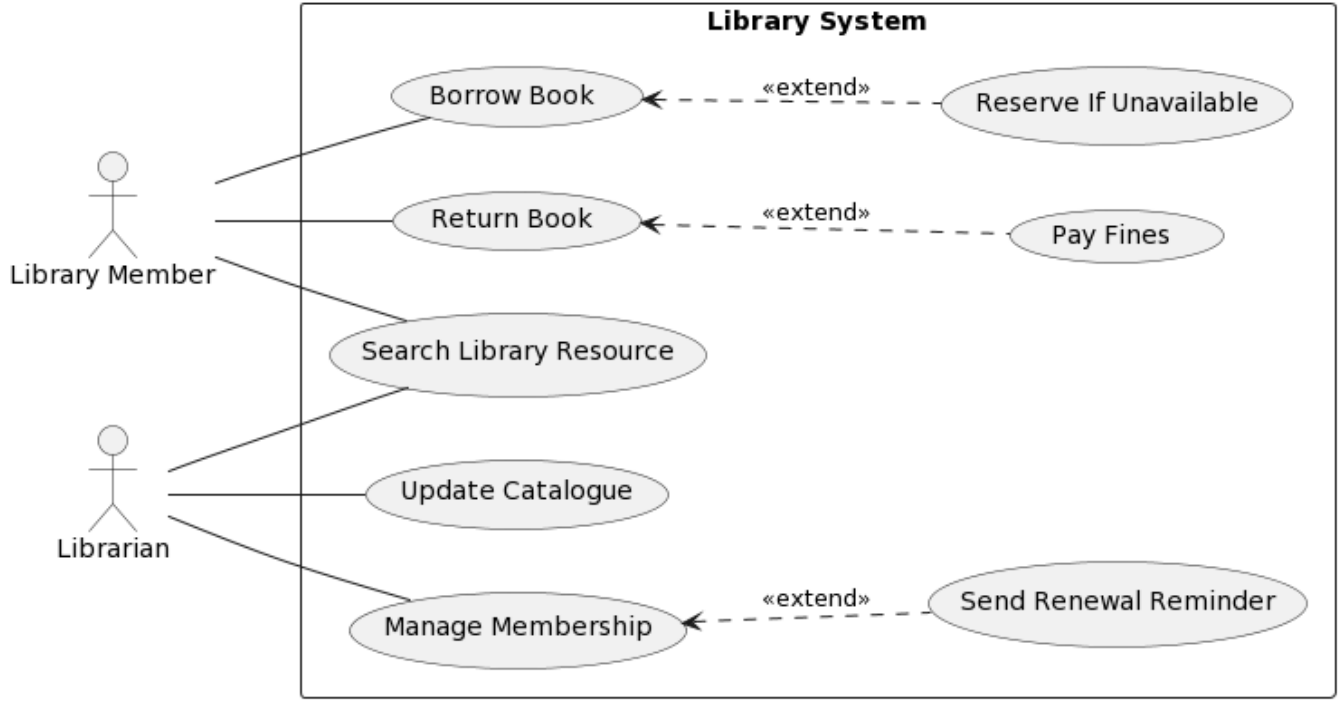
- **Generalization**:
  - Indicate one use case is **a special kind** of the other.
  - Represented by a directed arrow with a triangle arrowhead.
  - Generalization is used when we find **two or more use cases that have commonalities** in behavior, structure, or purpose.

Use Case Diagram (cont.)



- **Include**:
  - Indicates one use case (the base use case) is using the functionality of another use case (the inclusion use case).
  - Represented by a directed arrow with dotted line.
  - The stereotype "<<include>>" identifies the include relationship, where the base use case includes the functionality of the inclusion use case.
  - Include relation is used to support the reuse of functionality.

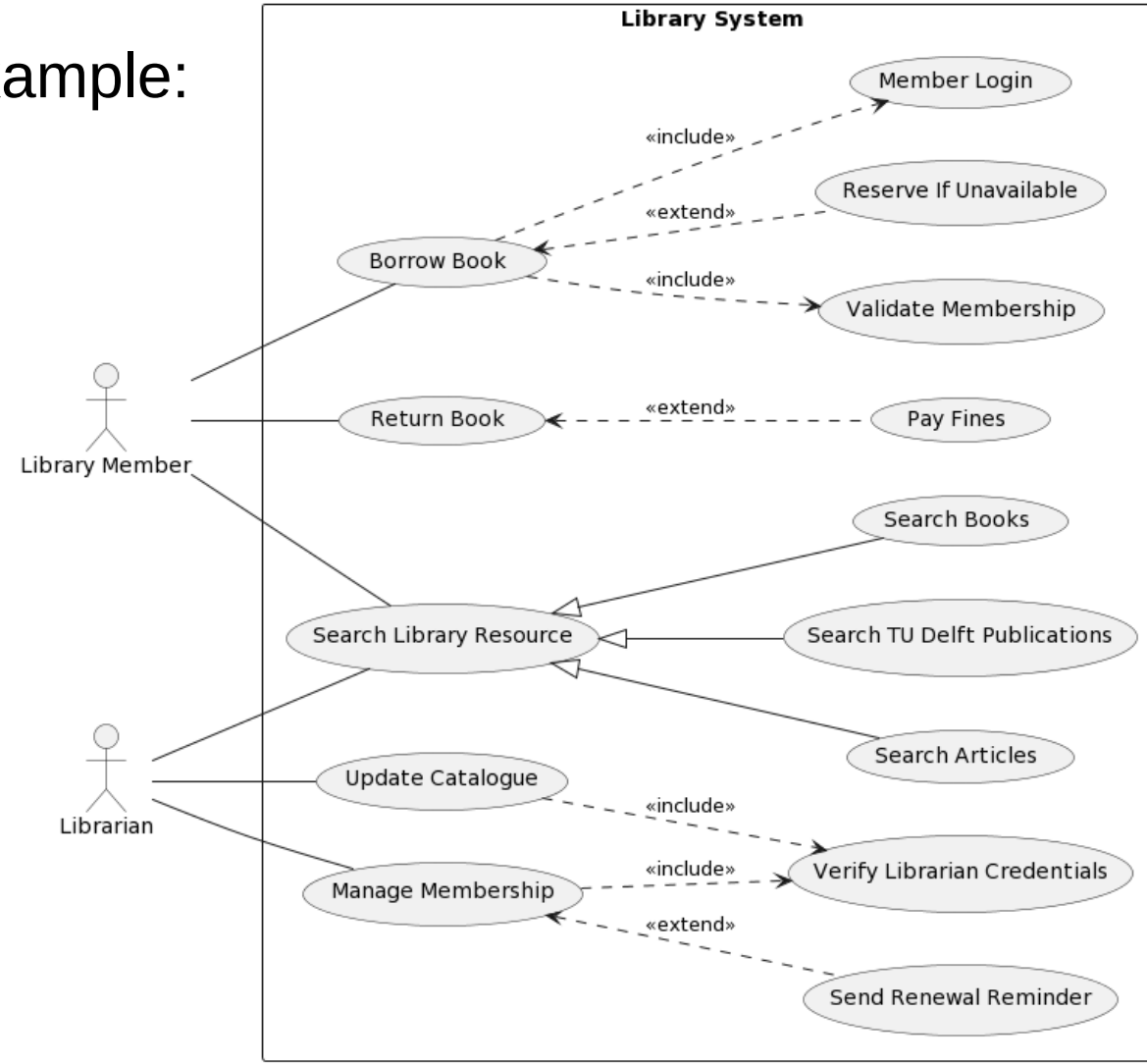
Use Case Diagram (cont.)



- **Extend**: specify that one use case (extension) extends the behavior of another use case (base).
  - Represented by a directed arrow with dotted line. The stereotype "<<extend>>" identifies the extend relationship.
  - We use extend relationship to show:
    - A use case is an optional system behavior.
    - A use case is executed only under certain conditions.

Use Case Diagram (cont.)

- An overall example:



Use Case Diagram (cont.)

- When to use the Use Case Diagram?
  - To represent the system-user interactions.
  - To define and organize the functional requirements of a system.
  - Is typically used in the early phase in system design.

Closing remarks

- In the Lab session:
  - Download and install PlantUML.
  - Go over the tutorial for the use case diagrams:
    - URL: https://cese.pages.ewi.tudelft.nl/software-systems/part-2/tutorials/uml/use-case.html
    - Get familiar with the system mentioned in the modeling assignments;