

# **CSC444F: Software Engineering I**

**Mou Hu**  
**[mou.hu@utoronto.ca](mailto:mou.hu@utoronto.ca)**

# Lecture 11: Object-Oriented Analysis and Design (II)

Object-Oriented Analysis and Design Methods  
Object-Oriented Analysis and Design Example  
Object-Oriented Metrics  
Reading: Chapter 12

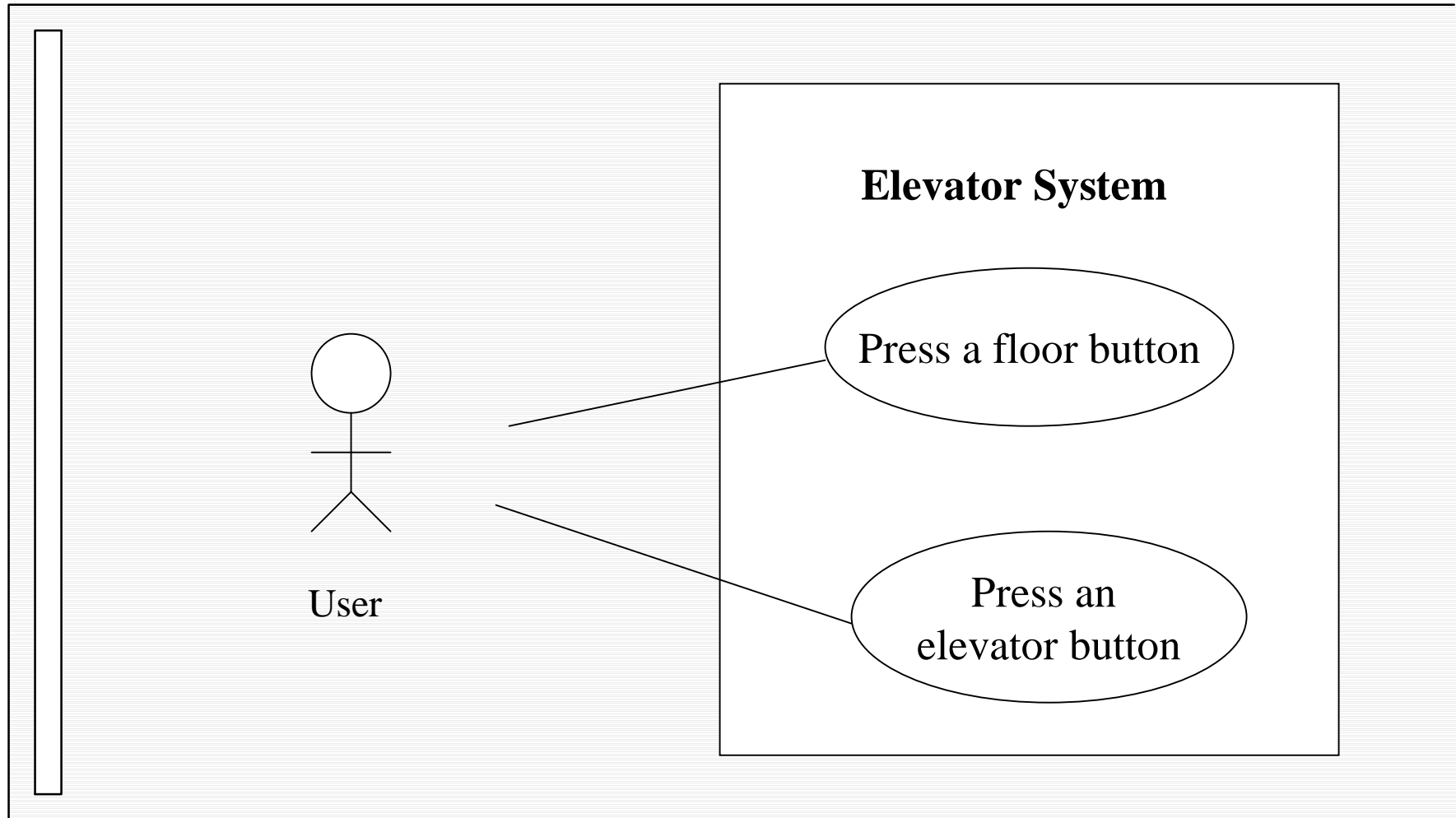
# How to Perform Object-Oriented Analysis

Iterate the following three steps:

- **Use-case modeling.** Determine how the various results are computed by the system (without regard to sequencing). Results of this step are use-case diagram and associated scenarios.
- **Class modeling.** Determine the classes, their attributes, and the interrelationships between the classes. Result is the class diagram.
- **Dynamic modeling.** Determine the actions performed by or to each class or subclass. Results are state diagrams.

Source: Ref. [2]

# Use Case Modeling for Elevator Problem



# A Normal Scenario

1. User A presses the Up floor button at floor 3 to request an elevator. He wishes to go to floor 7.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains user B, who has pressed the elevator button for floor 9.
4. The Up floor button is turned off.
5. The elevator door opens.
6. The timer starts. User A enters the elevator.
7. User A presses the elevator button for floor 7.
8. The elevator button for floor 7 is turned on.
9. The elevator door closes after a timeout.
10. The elevator travels to floor 7.
11. The elevator button for floor 7 is turned off.
12. The elevator door opens.
13. The timer starts. User A exits from the elevator.
14. The elevator door closes after timeout.
15. The elevator proceeds to floor 9 for user B.

# An Exception Scenario

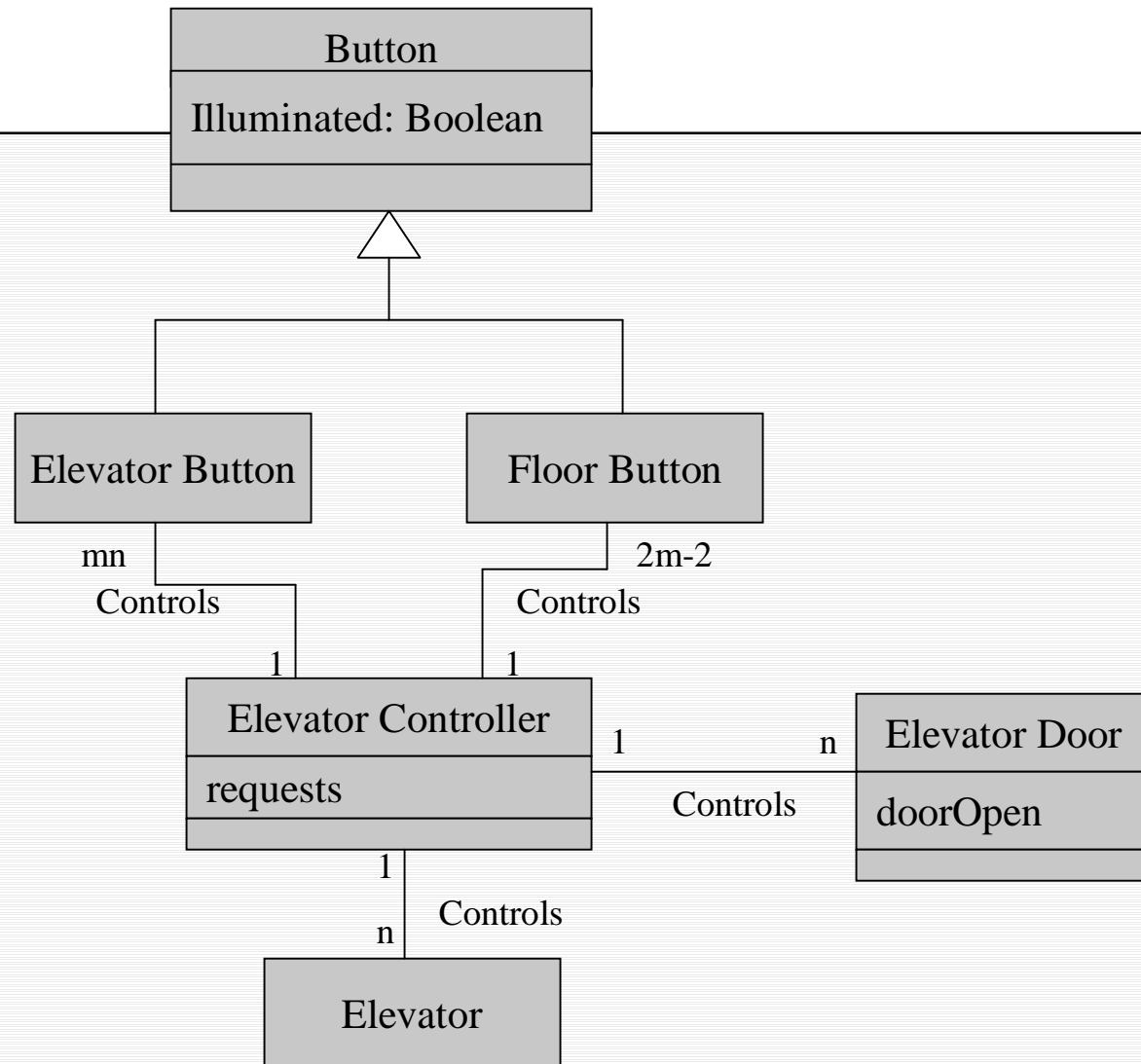
1. User A presses the Up floor button at floor 3 to request an elevator. He wishes to go to floor 1.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains user B, who has pressed the elevator button for floor 9.
4. The Up floor button is turned off.
5. The elevator door opens.
6. The timer starts. User A enters the elevator.
7. User A presses the elevator button for floor 1.
8. The elevator button for floor 1 is turned on.
9. The elevator door closes after a timeout.
10. The elevator travels to floor 9.
11. The elevator button for floor 9 is turned off.
12. The elevator door opens.
13. The timer starts. User B exits from the elevator.
14. The elevator door closes after timeout.
15. The elevator proceeds to floor 1 for user B.

# Class Modeling

- Study all scenarios and use noun extraction to identify candidate classes.
- Remove abstract nouns and nouns lie outside the problem boundary, e.g., user and floor for this example.
- Add, delete, or modify classes as necessary. For example, we add an Elevator Controller class, and let Timer be part of this class.
- Coad & Yourdon suggest each class should satisfy (most of) the following criteria:
  - Retained information: Does the system need to remember information about this class?
  - Needed Services: Does the class have identifiable operations that change the values of its attributes?
  - Multiple Attributes: If the class only has one attribute, it may be better represented as an attribute of another class

# Class Diagram for Elevator Problem

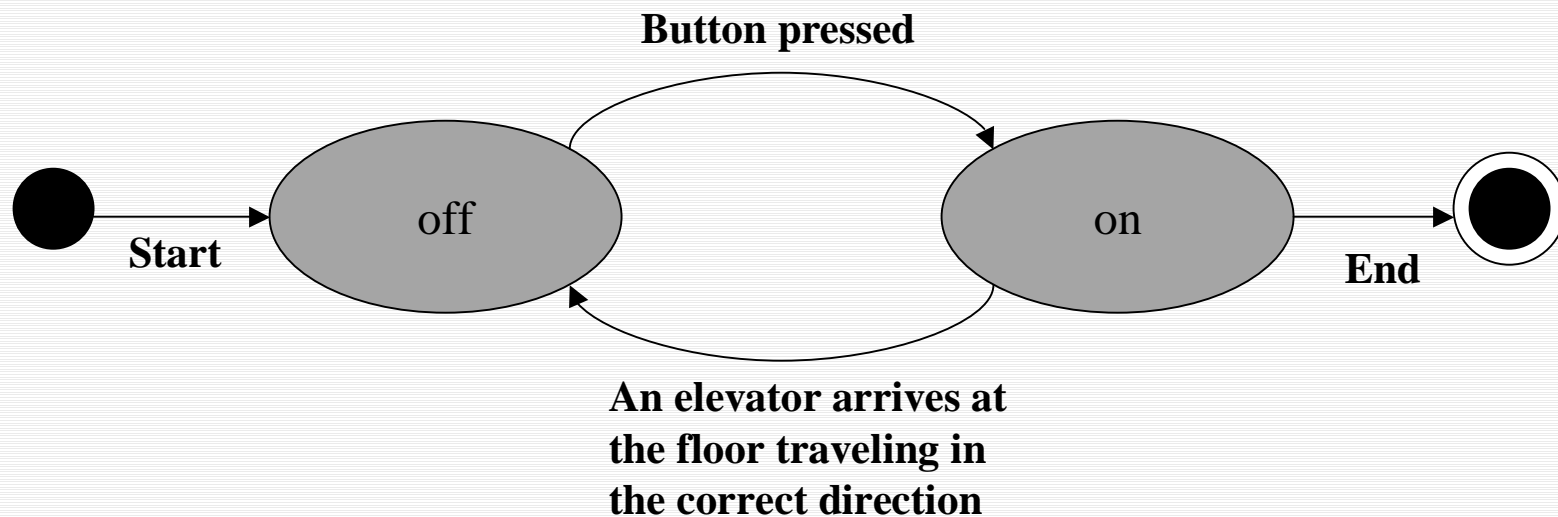
- Assume there are  $m$  floors and  $n$  elevators
- Usually many iterations are needed to reach the final class diagram



# Dynamic Modeling for Elevator Problem

■ For each class, produce a state diagram, which depicts the dynamic behavior of the class.

■ The following is a state diagram for the floor button class



# How to Perform Object-Oriented Design

**Construct interaction diagram** (sequence diagram or collaboration diagram) **for each scenario.**

**Construct the detailed class diagram.** The methods are inserted to the detailed class diagram. Determination of all the actions of the system is performed by examining the interaction diagrams of every scenario.

**Design the system in terms of client of objects.** An object C that sends a message to object O is a client of O. The interaction diagrams are used to draw a diagram showing the clients of each object. A simple main program that essentially starts things off; the objects themselves take over then.

**Proceed to the detailed design.** Use pseudo-code or tabular representation.

Source: Ref. [2]

# Object-Oriented Metrics

- WMC (Weighted Methods per Class) =  $\sum_{i=1}^n C_i$   
where  $C_i$  is the complexity of method  $i$ .

Larger classes are in general less desirable.

- DIT (Depth of class in Inheritance Tree) = the distance of a class to the root of its inheritance tree  
A widely accepted heuristic is to strive a collection of inheritance tree of medium height.
- NOC (Number of Children)  
Higher value of NOC suggests a higher complexity of the class.
- CBO (Coupling Between Object classes) = the number of other classes with which it is coupled  
Higher values of CBO suggest tighter bindings with other components, and is undesirable.
- LCOM (Lack of Cohesion Of a Method)  
= the number of disjoint sets of methods of a class  
The preferred value of LCOM is 1.

# References

- [1] Hans van Vliet, “Software Engineering: Principles and Practice”, John Wiley and Sons, Ltd., 2000.
- [2] Stephen R. Schach, “Object-Oriented and Classical Software Engineering”, McGraw-Hill Companies, Inc., 2002.
- [3] Martin Fowler and Kendall Scott, “UML Distilled: A Brief Guide to the Standard Object Modeling Language”, Addison Wesley Longman, Inc., 2000.