KAIST

Spring 2025

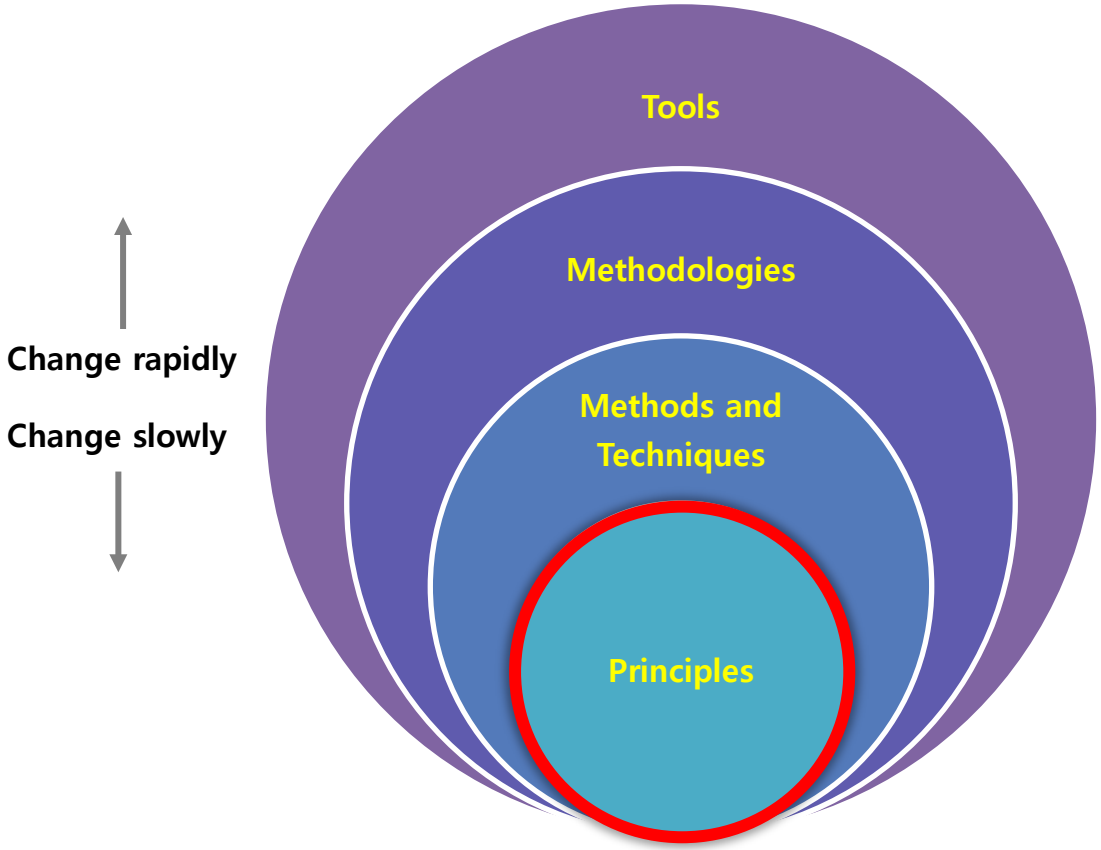
CS30500:
Introduction to Software Engineering

Lecture #13

Prof. In-Young Ko

School of Computing

Software Engineering Principles



Relationship between principles, techniques, methodologies and tool

[GJM02] Ghezzi et al., Fundamentals of Software Engineering

- **Principles**

  "General and abstract statements describing desirable properties of software process and product"

- **Methods**

  "General guidelines that govern the execution of some activity; they are rigorous, systematic, and disciplined approaches"

- **Techniques**

  More technical than methods and have more restricted applicability

- **Methodologies**

  Sets of methods and techniques selected for solving a problem

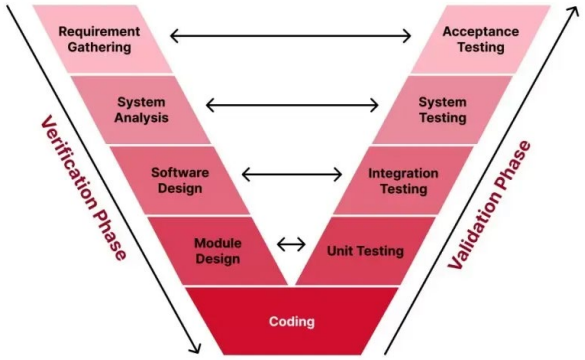
- **Tools**

  Tools to support the application of methodologies

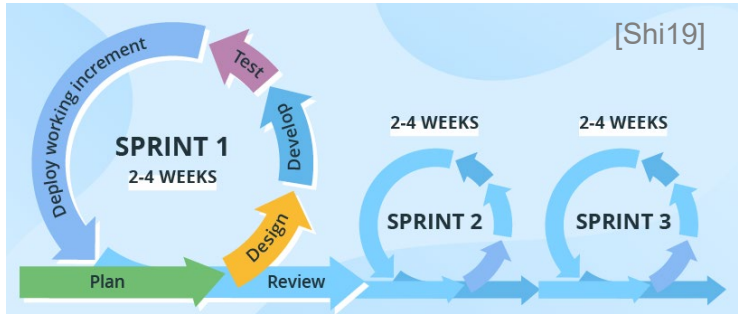
Software Engineering Principles

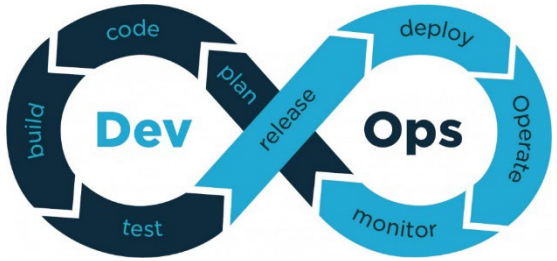
- Rigor and Formality
- Incrementality
- Anticipation of Change
- Generality

- Modularity
- Abstraction
- Separation of Concerns

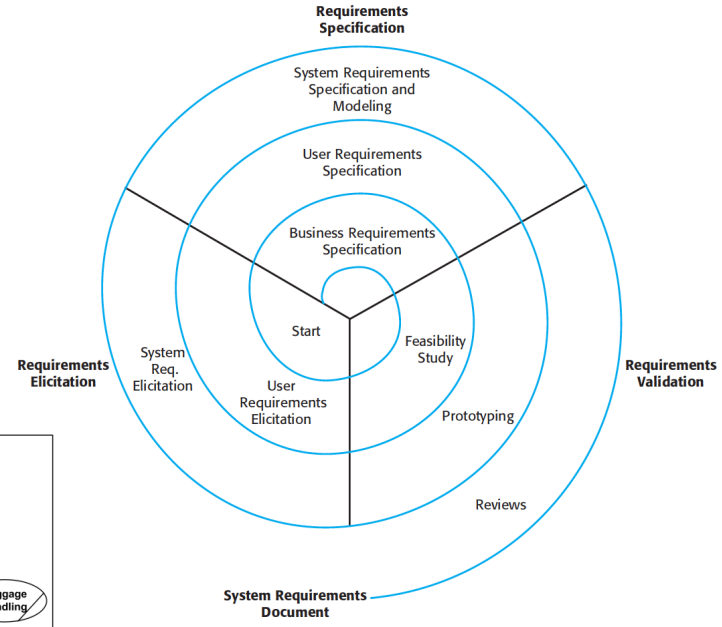


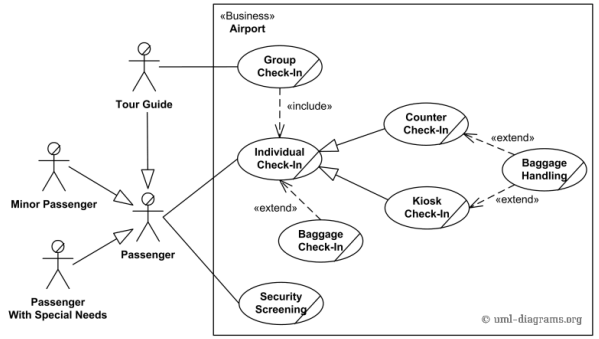
https://reliasoftware.com/blog/v-model





https://www.easyredmine.com/news/devops-for-easy-redmine-a-new-perspective-to-software-development-for-all-redminers





http://www.uml-diagrams.org/airport-checkin-uml-use-case-diagram-example.html

Software Engineering Principles are for …

- Understanding what to develop
- Developing software the right way
- Developing the right software

by effectively collaborating with stakeholders, including customers, users, and developers.

https://www.linkedin.com/pulse/go-digital-why-diverse-inputs-multi-stakeholder-feedback-chris-leong/

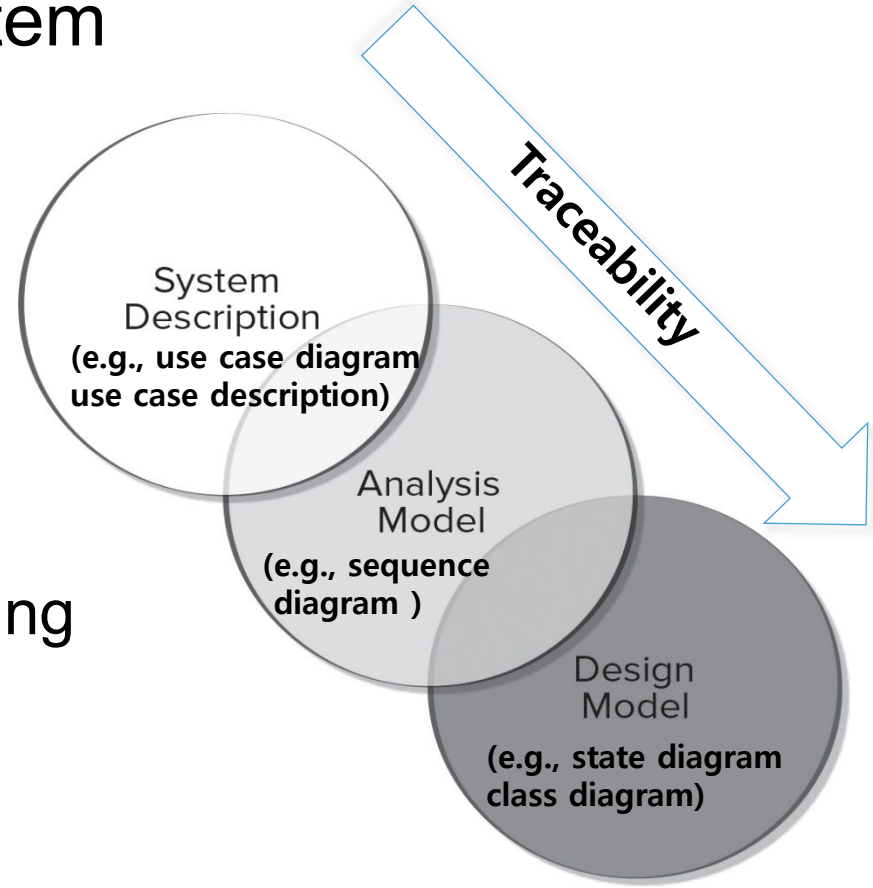
Topics Covered and To-be Covered

- **Understanding what to develop**
  - **Introduction** to Software Engineering (Software Process & Project Management)
  - Software **Requirements** Engineering
- **Developing software the right way**
  - SE Principles: **Separation of Concerns**, **Abstraction, Modularity**, etc.
  - Process Models: Heavy-weight process, Prototyping, **Agile**, **DevOps**,..
  - Basics of **UML** (Unified Modeling Language)

  - Software Design: **Architectural** Patterns, **Resilient Design** Patterns
- **Developing the right software**
  - Software **Quality**: Correctness, Robustness, Safety, Security, etc.
  - Software Testing: **Testing overview**, **Black-box** and **White-box** testing

Requirements Analysis to Design

- Specifies software's operational characteristics
- Indicates software's interface with other system elements
- Establishes constraints that software must meet
- Requirements analysis allows the software engineer to:
  - Elaborate on basic requirements established during earlier requirement engineering tasks
  - Build models that depict the user's needs from several different perspectives

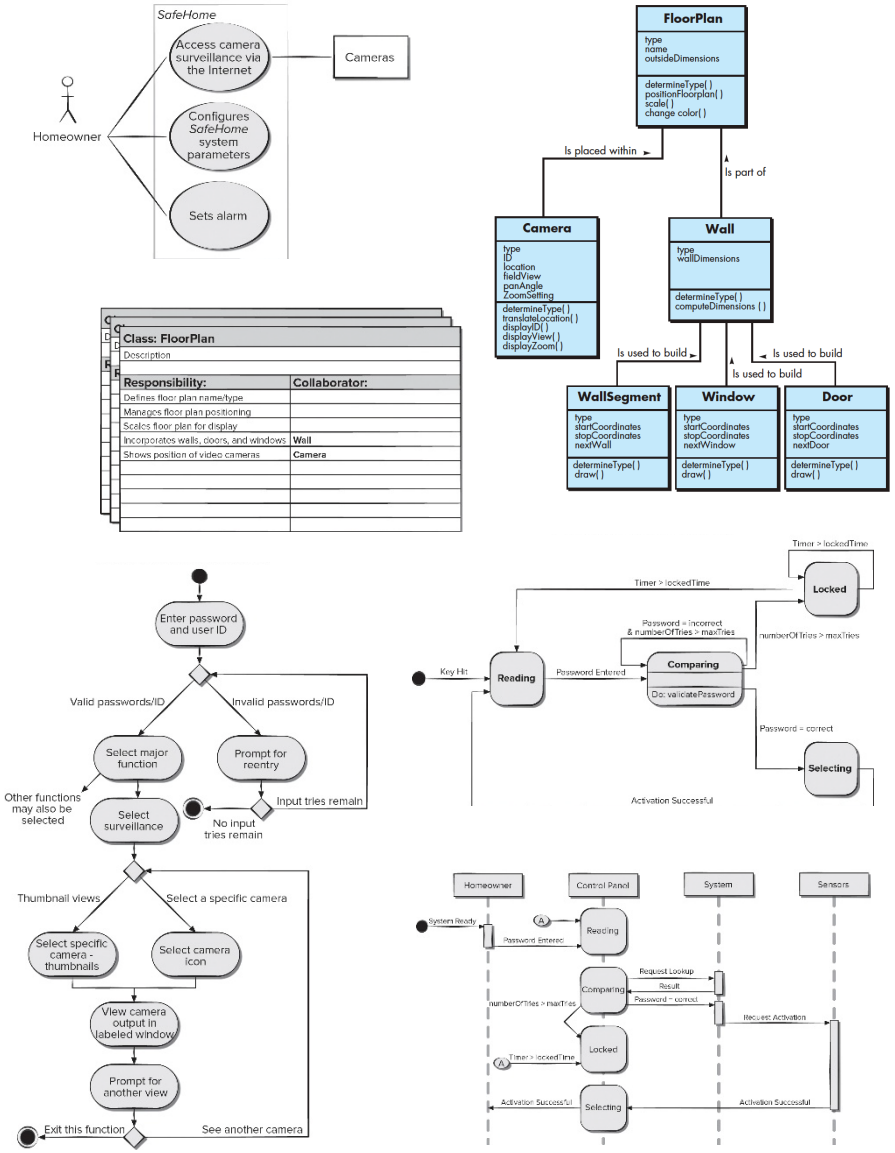


[PrMa20]

Requirements and Design Models

[PrMa20]

- **Scenario-based models**
  - Depict requirements from the point of view of "actors"
  - e.g., storyboards, use case model
- **Class-oriented models**
  - Represent object-oriented classes (attributes and operations) and how classes collaborate
  - e.g., class diagrams, CRC model
- **Behavioral models**
  - Depict how the software reacts to internal or external "events"
  - e.g., control flow diagrams, Petri nets, sequence diagrams, activity diagrams, state diagrams



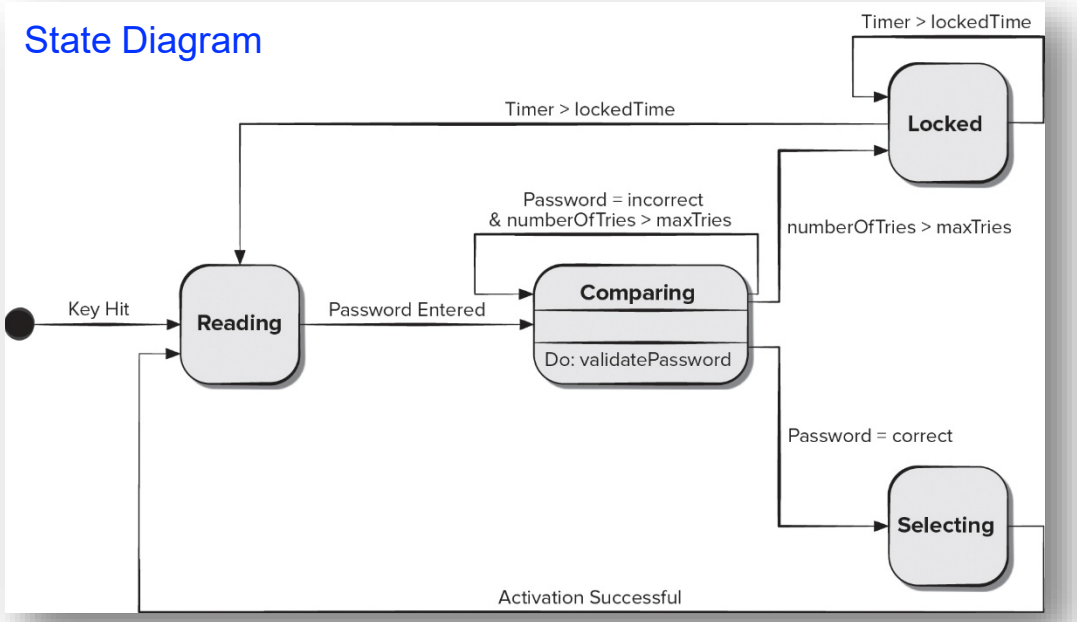
Today's Plan

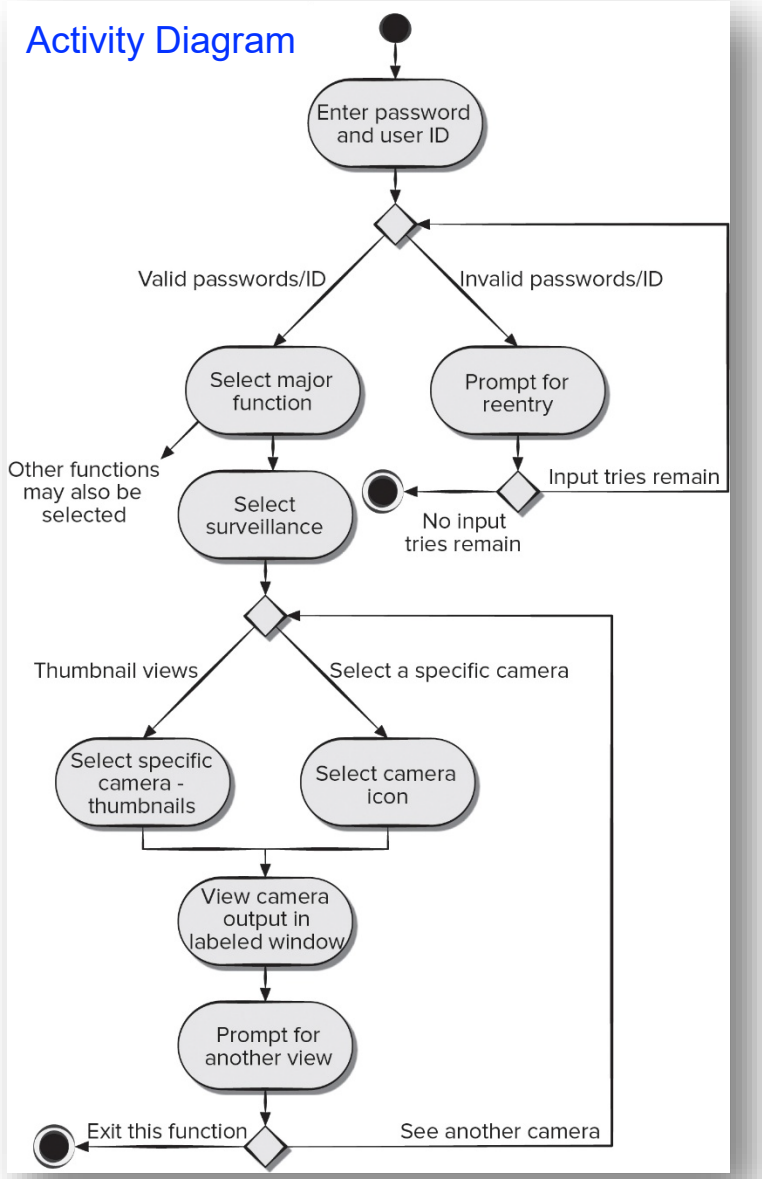
- Requirements Models
  - Scenario-based modeling
  - Class-based modeling
  - Behavioral modeling
- Domain modeling
- Traceability

BEHAVIORAL MODELING

Behavioral Modeling

- A ***behavioral model*** indicates how software will respond to internal or external events or stimuli
  - UML state diagrams can be used to model how system elements respond to external events
  - UML activity diagrams can be used to model how system elements respond to internal events





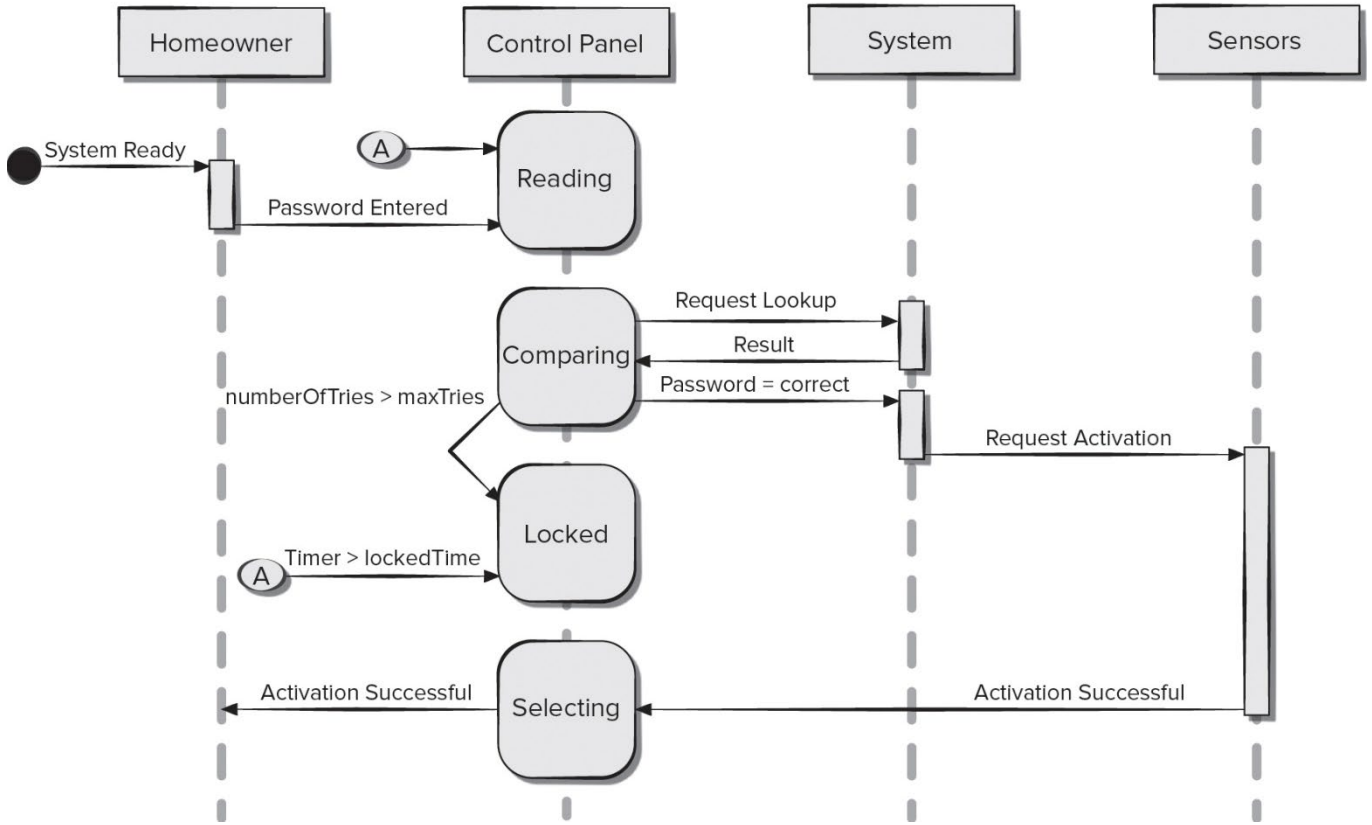
Creating Behavioral Models

1. Evaluate all **use cases** to fully understand the sequence of interaction within the system
2. Identify **events** that drive the interaction sequence and understand how these events relate to specific **objects**
   - An event occurs whenever the system and an actor exchange information
3. Create a sequence diagram for each use case
   - Events are used to trigger state transitions
4. Build a state diagram for the system
5. Review the behavioral model for accuracy and consistency

Name	Make reservation
Summary	Reservation clerk reserves a room for a guest
Actor	Reservation clerk
Precondition	Reservation clerk has logged into the system
Description	**1. Guest** gives personal details to reservation clerk, such as name, room type, number of occupants, and dates and times of arrival and departure **2. Reservation clerk** enters information into **the system** and looks up availability and room reservation rate 3. If room is available, clerk requests the guest to guarantee the reservation with a credit card number 4. The clerk enters the guest's credit card number to guarantee the reservation. 5. If credit card number is valid, the clerk updates the reservation as guaranteed.
Alternatives	Room not available: Line 3: If a room is not available, clerk requests the customer for another date and / or time of arrival and departure Invalid card: Line 5: If the credit card number is not valid, the clerk requests the customer for another credit card number
Postcondition	The reservation is created for the guest.

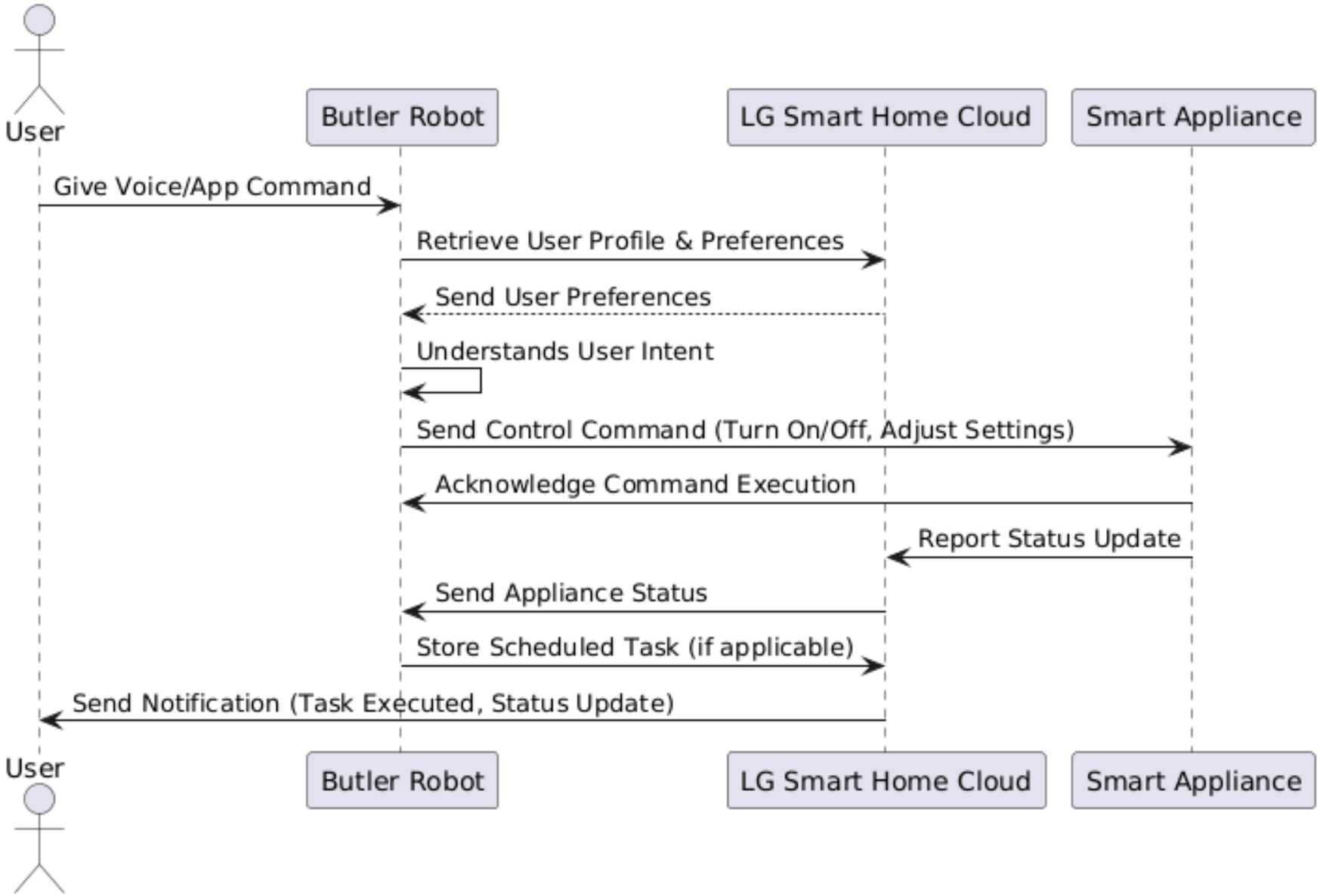
Sequence Diagram

- Sequence diagrams can be used to show how **events** cause transitions from **object to object**
- Sequence diagram is a shorthand version of a use case



[PrMa20]

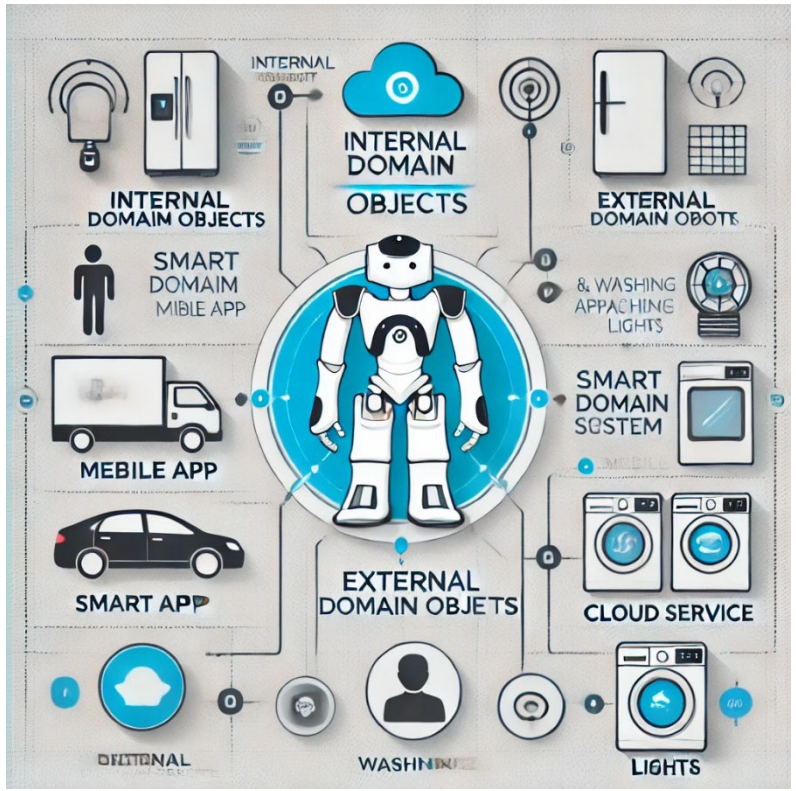
Sequence Diagram Generated by ChatGPT + UML Diagram Creator



DOMAIN MODELING

As a Software Engineer…

- Your task is to design and develop a smart home system where a butler robot understands the user's needs and seamlessly controls and coordinates home appliances.



[Generated by ChatGPT 4o]

External and internal domain objects?

Domain Objects Found by ChatGPT

Domain Object	Description
User	Represents the homeowner or resident interacting with the system.
VoiceCommand	A user-issued verbal instruction that the system needs to interpret.
MobileApp	The user interface through which users control and monitor the system.
ButlerRobot	The central AI system that interprets, coordinates, and executes commands.
Intent	The parsed and interpreted meaning of a user command (e.g., "turn on lights").
Command	A system-generated instruction sent to an appliance or subsystem.
Schedule	Represents a time-based automation or routine for appliances.
Room	A physical location (e.g., kitchen, bedroom) where appliances are placed.
Device (Appliance)	Base class for smart appliances like AC, lights, washer, etc.
├── **SmartLight**	A light that can be turned on/off, dimmed, or scheduled.
├── **SmartAC**	An air conditioner with temperature and mode controls.
├── **SmartFridge**	A refrigerator capable of reporting temperature, status, etc.
├── **WashingMachine**	A smart washer with status reporting and scheduling capabilities.
DeviceStatus	Real-time or historical state of an appliance (on, off, current temp, etc.).
UserProfile	Contains user preferences, access level, language, etc.
Notification	Alerts or messages sent to the user via mobile or voice.
AuthenticationToken	Represents user identity verification for access control.
WiFiConnection	Represents network details and status for appliances and the robot.
CloudService	Backend services handling API routing, storage, and integration.
LogEntry	A record of commands, errors, or events for auditing and monitoring.

[Generated by ChatGPT 4o]

Domain Modeling (1/2)

- Why? — To understand how system-to-be will work
  - Requirements analysis determined how users will interact with system-to-be (*external behavior*)
  - Domain modeling determines how elements of system-to-be interact (*internal behavior*) to produce the external behavior
- Where? — Based on the following sources:
  - Knowledge of how system-to-be is supposed to behave (from requirements analysis, e.g., use cases)
  - Studying the work domain (or, problem domain)
  - Knowledge base of software designs
  - Developer's past experience with software design

Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

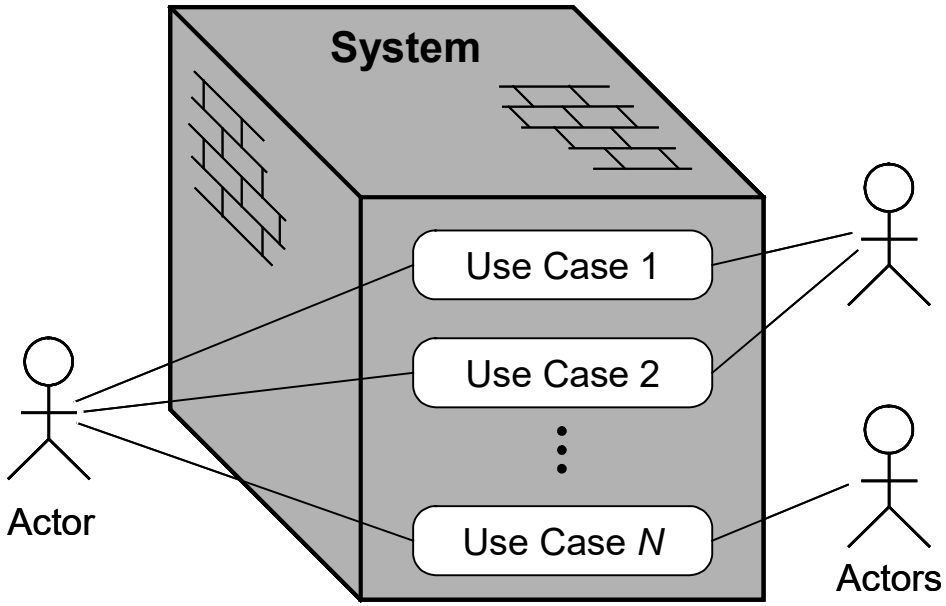
Domain Modeling (2/2)

- What?
  - Illustrates meaningful *conceptual classes* in a problem domain
  - Is a representation of real-world concepts, not software artifacts
  - Is the most important objects/classes in OO development
    - Decompose the system into a set of objects, rather than functions!
- How?
  - Identify nouns and noun phrases in textual descriptions of the domain
    - Use **use case descriptions**
  - Add associations for relationships that must be retained
  - Add attributes necessary for information to be preserved

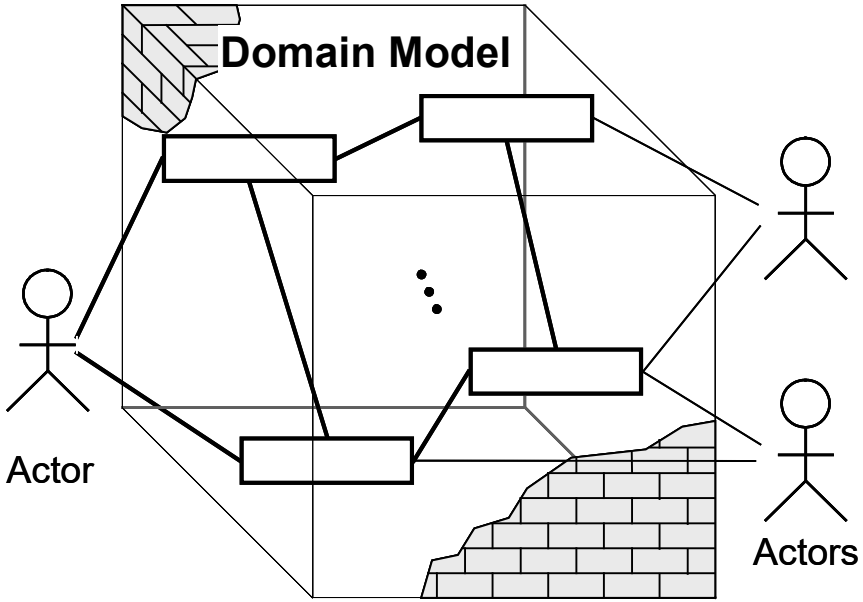
Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Use Case Modeling vs. Domain Modeling

In **use case analysis**, we consider the system as a "**black box**"

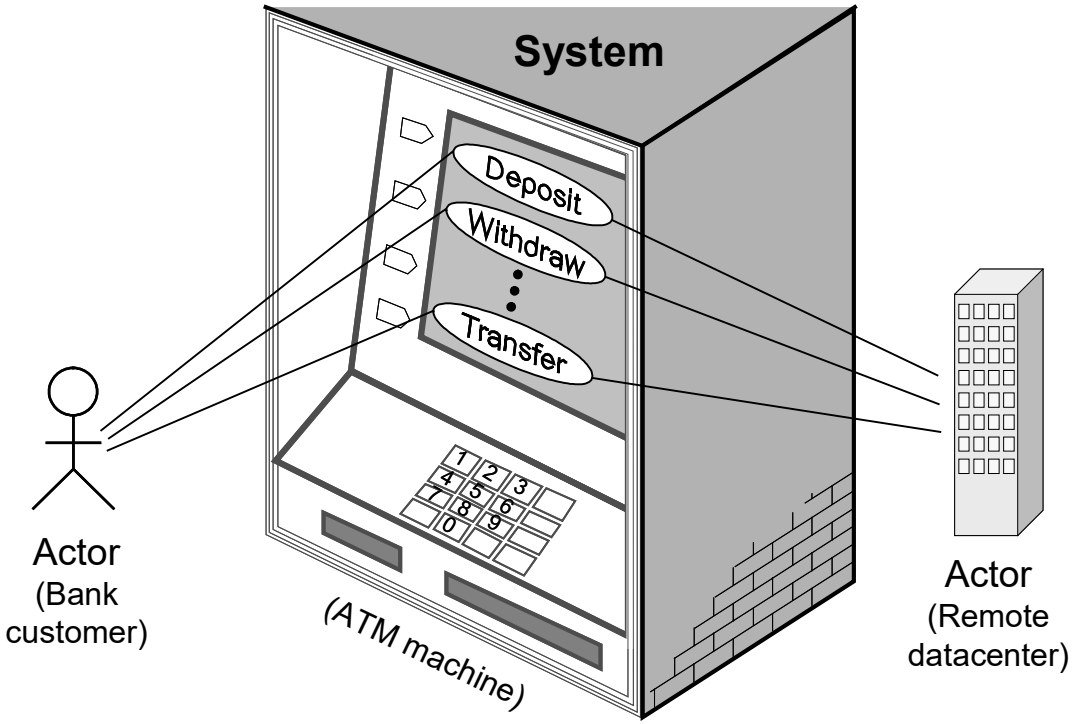


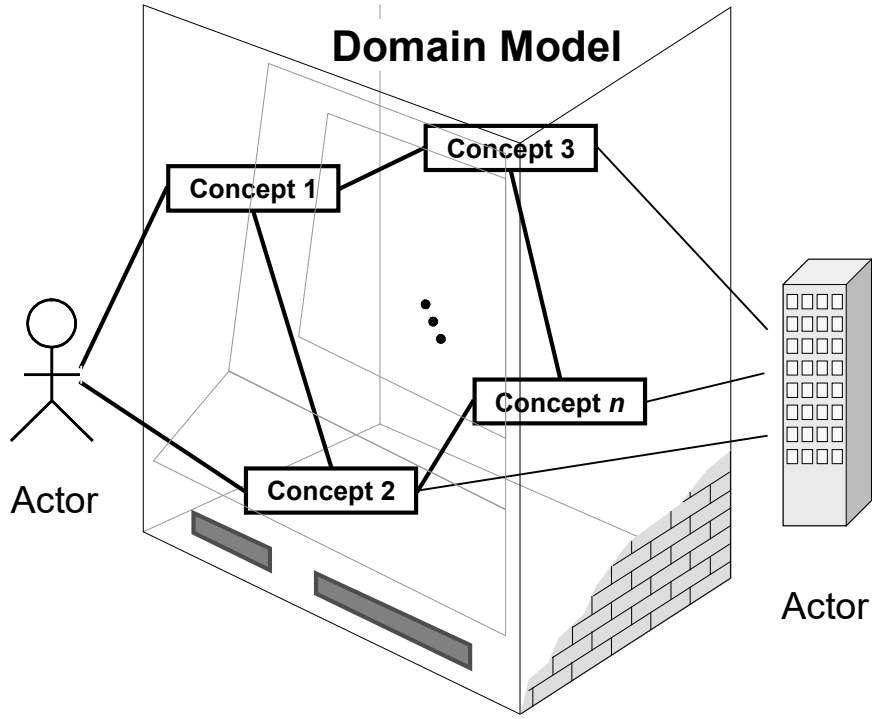
In **domain analysis**, we consider the system as a "**transparent box**"



Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Example: ATM Machine





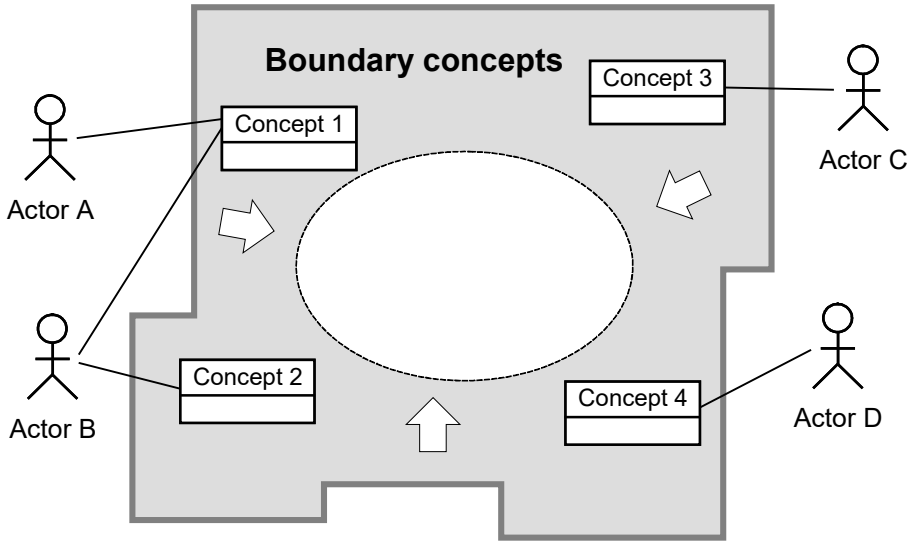
Identifying Classes for ATM

1. Identify all the nouns or noun phrases from use case descriptions
   - Customer, bank, ATM, (bank) account, funds, cash, keypad, screen, cash dispenser, cash deposit slot, user, checking account, saving account, card, password, transaction, receipt, bookkeeping, request.
2. Eliminate redundant, irrelevant, implementation constructs, attributes, etc.

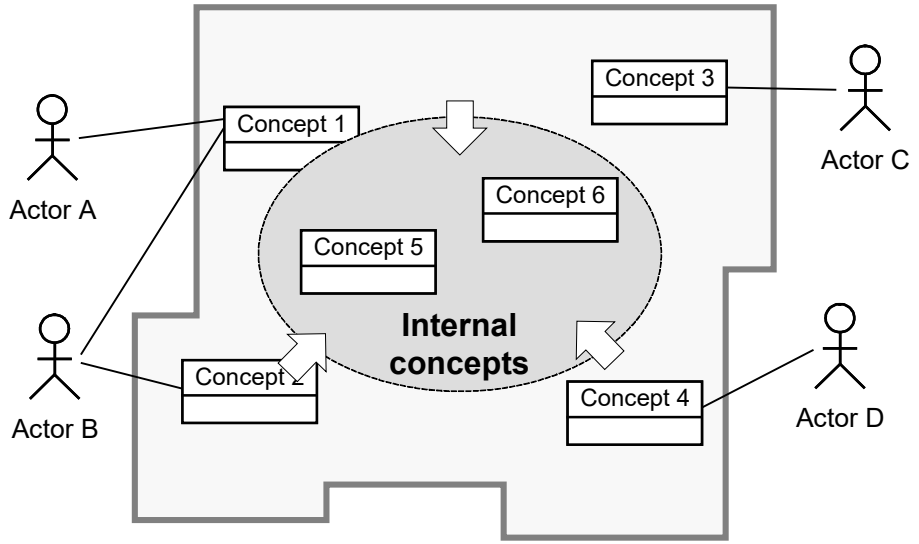
Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Building Domain Model from Use Cases

Step 1: Identifying the boundary concepts



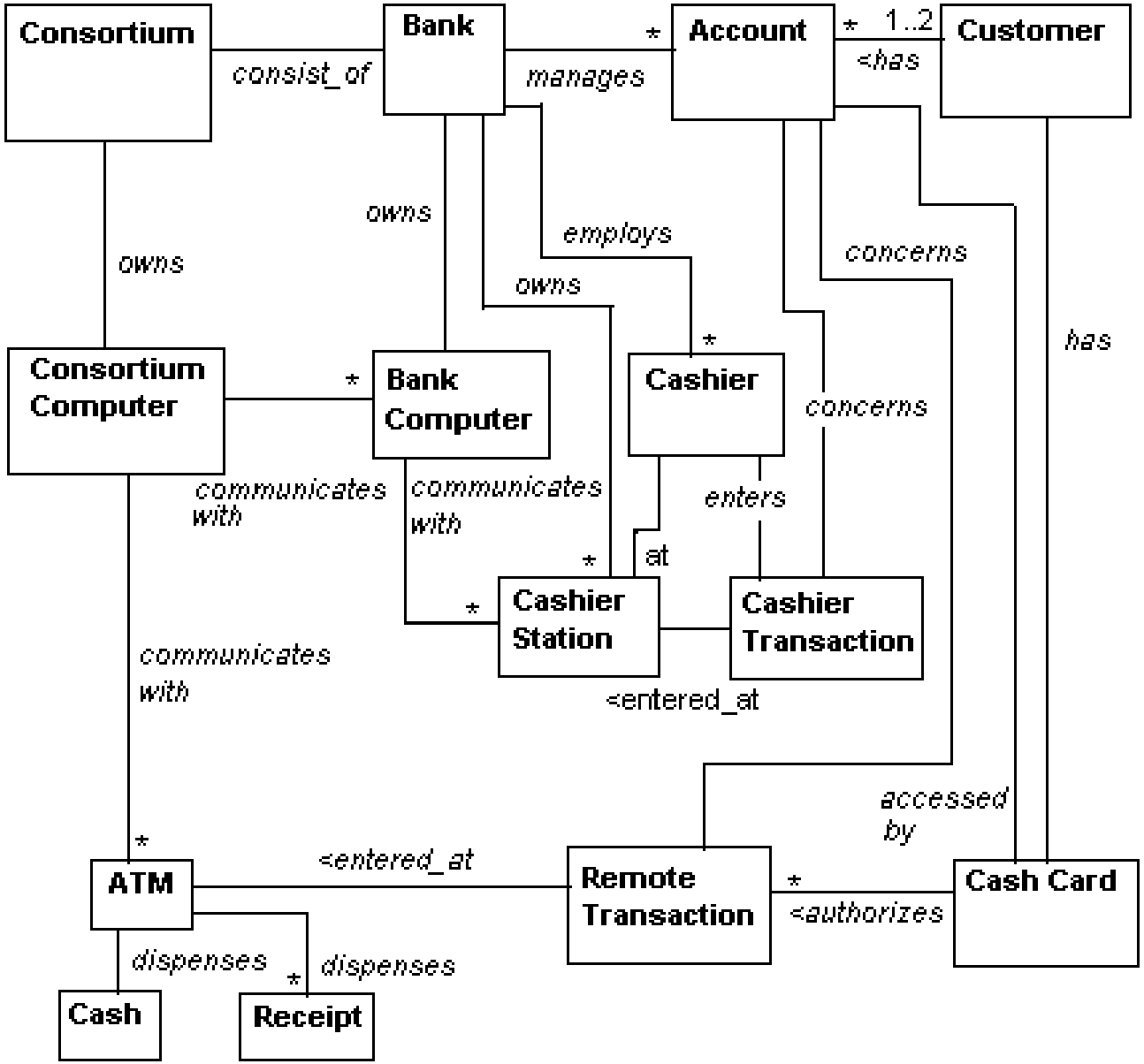
Step 2: Identifying the internal concepts



Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

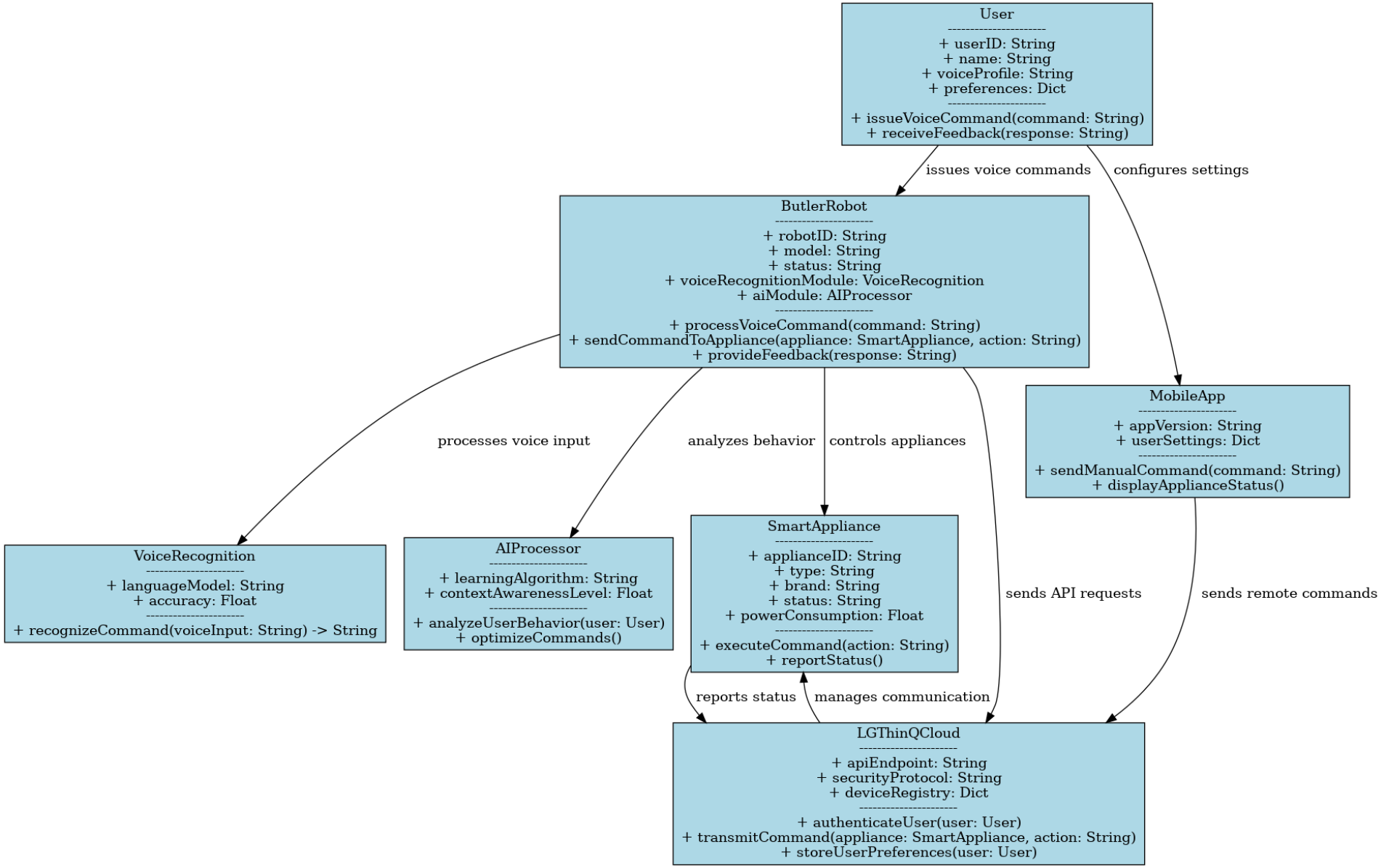
Initial Class Diagram (Domain Model) for ATM

- Objects: ATM, Keypad, Deposit slot, Cash dispenser, Screen, Transaction, Bank, Account, Customer
- Draw a class diagram with only class names and relationships



Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

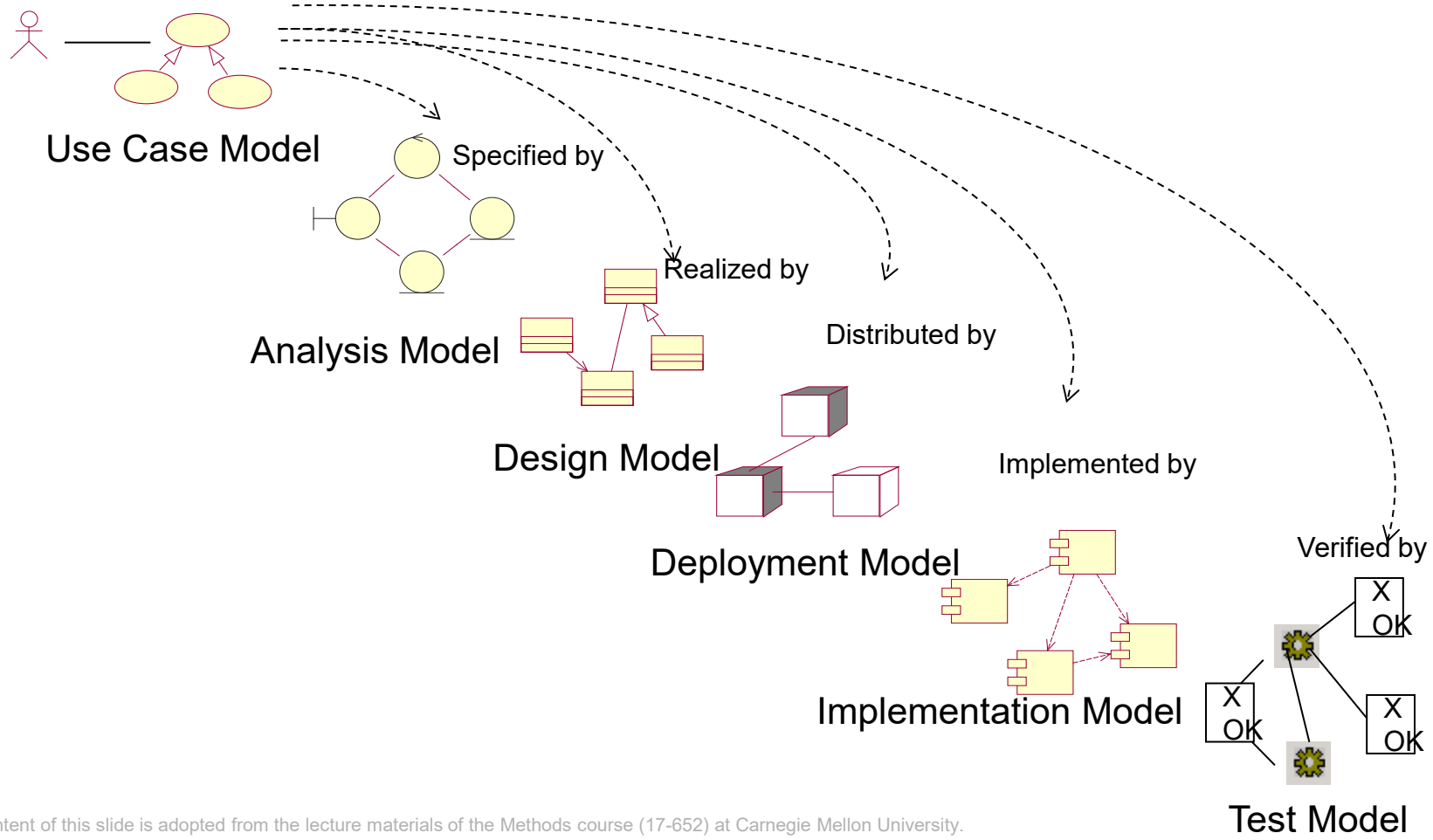
Domain Model Generated by ChatGPT



TRACEABILITY

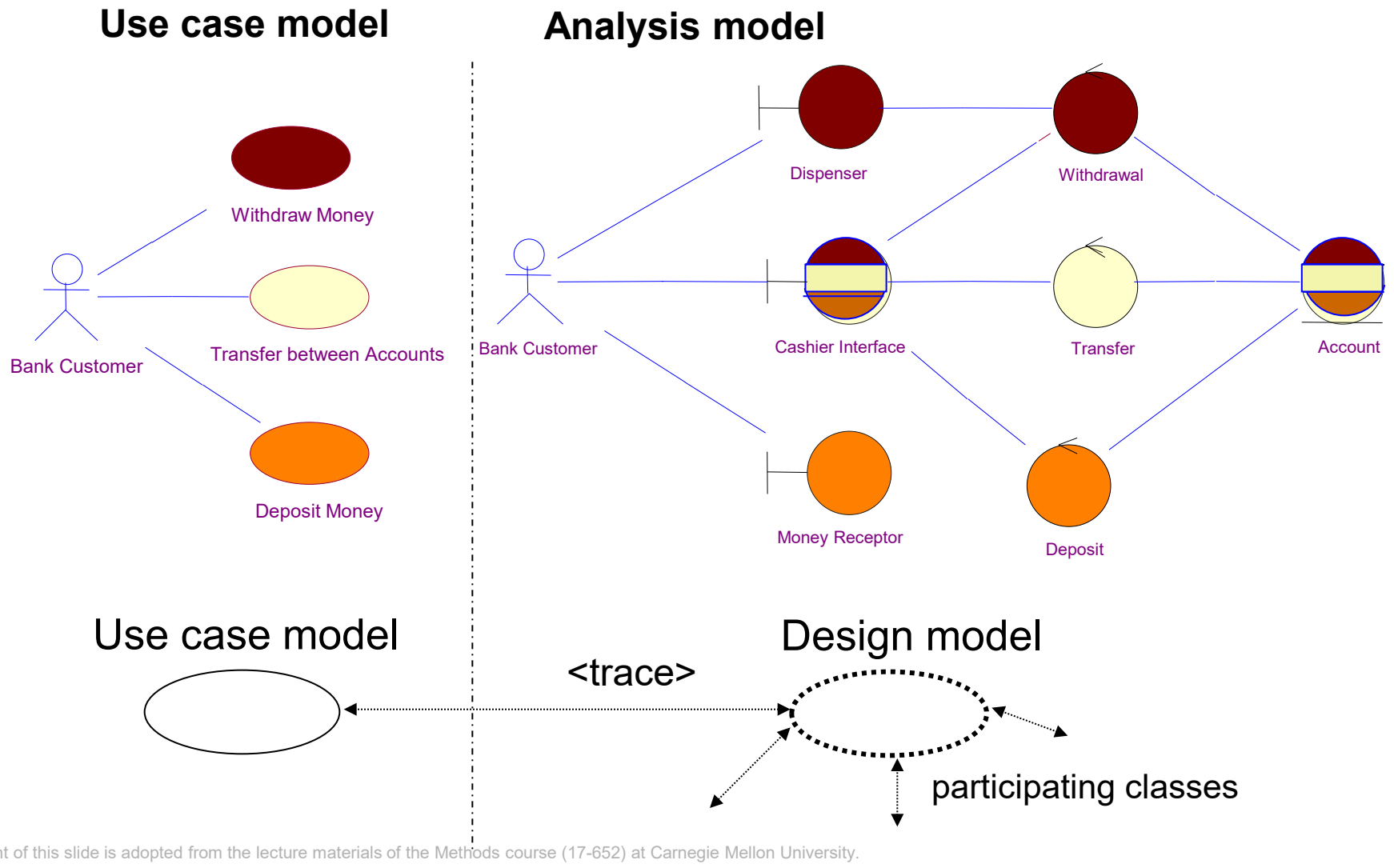
From Use Cases to Objects (2/2)

- Jacobsen, Booch, Rumbaugh view [JBR 1999] - RUP



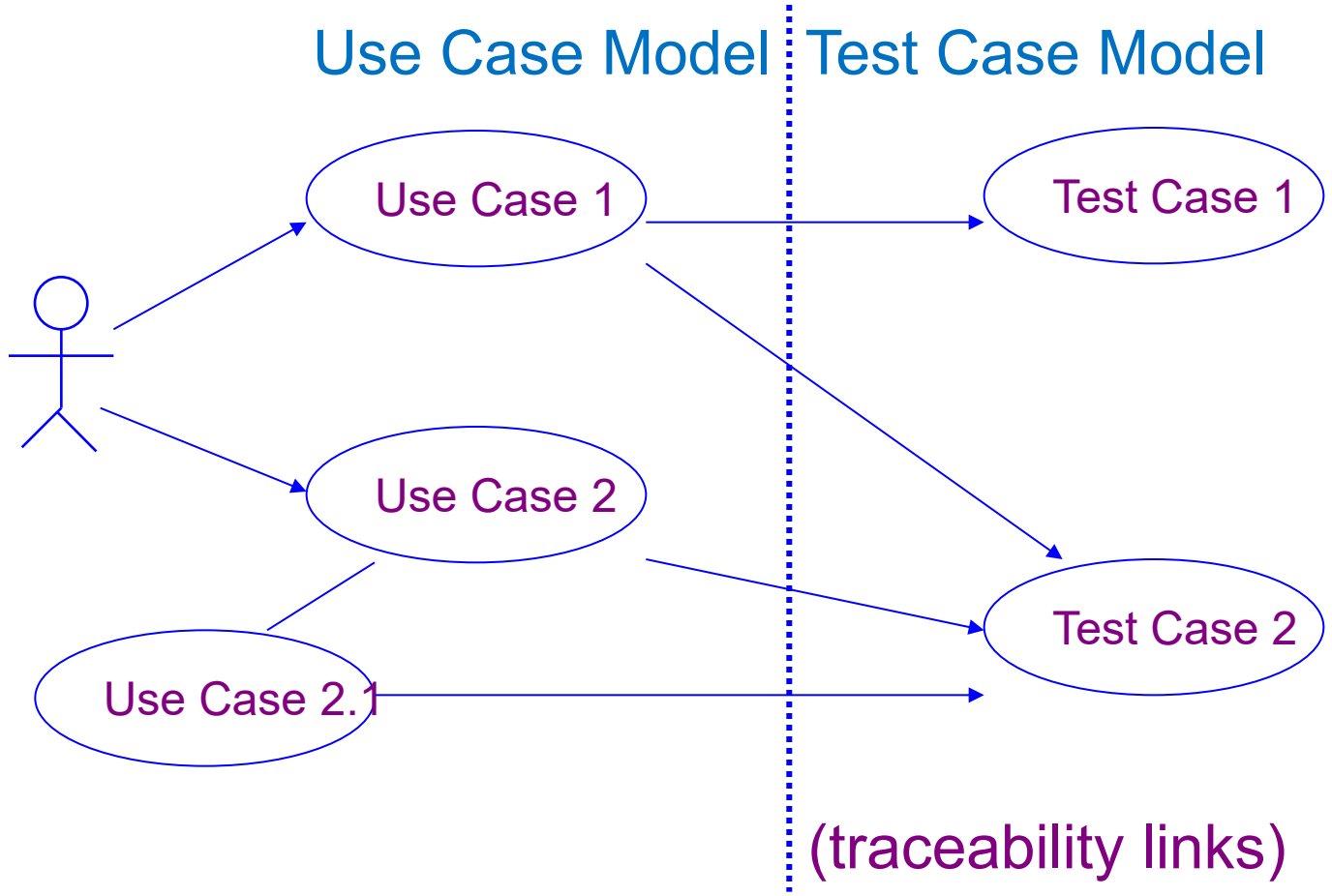
The content of this slide is adopted from the lecture materials of the Methods course (17-652) at Carnegie Mellon University.

Use Case Realizations



The content of this slide is adopted from the lecture materials of the Methods course (17-652) at Carnegie Mellon University.

Test Cases



The content of this slide is adopted from the lecture materials of the Methods course (17-652) at Carnegie Mellon University.

Test Case Design

Test case for ‘Control Light’ use case

Test Case ID	Event description	Input 1	Input 2	Expected Result
		Basic Flow		
2001	Resident presses control switch	Any enabled button	Light was on before button pressed	Light is turned off
2002			Light was off	Light is turned on
2003	Resident releases button in less than 1 second	Light on		Stays off
2005	Resident releases button in less than 1 second	Light off		Stays on

The content of this slide is adopted from the lecture materials of the Methods course (17-652) at Carnegie Mellon University.

Traceability Relationship

- The degree to which a relationship can be established between two or more products of the development process
- The degree to which each element in a software product establishes its reason for existing
- Predecessor-successor relationships
- Master-subordinate relationships

The content of this slide is adopted from the lecture materials of the Methods course (17-652) at Carnegie Mellon University.

Neglecting Traceability Results in

- Loss of knowledge
- Misunderstandings, miscommunication
- Losing even the simplest, but very crucial aspects of the overall software development life cycle

  e.g., Where did this use case come from?
- Unnecessary revisions, thus increased cost and time

The content of this slide is adopted from the lecture materials of the Methods course (17-652) at Carnegie Mellon University.

QUESTIONS?