COMP 1531
Software Engineering Fundamentals

Week 02 Wednesday
Requirements Engineering (Part I)
Introduction

What is a requirement?

- *"A condition or capability needed by a user to solve a problem or achieve an objective"* [IEEE]

- Simply stated, a requirement is a statement; a short, concise piece of information that:
  - describes an aspect of what the proposed system should do or describe a constraint
  - must help solve the customer's problem
- Set of requirements as a whole represents a negotiated agreement among all stakeholders

Functional vs Non-Functional requirements

Functional Requirements:

Defines the specific functionality that the software system is expected to accomplish i.e the *services* provided by the system and is typically described as:

- What inputs the system should accept and under what conditions
- The behaviour of the system
- What outputs the system must produce and under what conditions

Non-Functional Requirements:

Describe the *quality attributes* of the software system

- The constraints of the functionality provided by the software-system e.g. security, reliability, maintainability, efficiency, portability, scalability

Functional vs Non-Functional requirements

Consider a cell phone…

- Calling, texting, emailing, taking photos
- Features of the phone based on the user-interaction with the phone – functional requirements

But what do you also look for when you buy a phone?

- Good battery life, access to a network, plenty of internal memory, how easily it can break when you drop it (how the phone should perform in the user's environment – non-functional requirements)

Metrics for Non-Functional requirements

Non-functional requirements are quantifiable and must have a measurable way to assess if the requirement is met (metrics)

- Performance (user response time or network latency measured in seconds, transaction rate (#transactions/sec)
- Reliability (MTBW – mean time between failures, downtime probability, failure rate, availability)
- Usability (training time, number of clicks)
- Portability (% of non-portable code)

What are the top challenges you might face in Requirements Elicitation?

Common Challenges with Requirements

- Limited access to stakeholders
- Conflicting priorities
- Customers don't know what they want
- Customers change their mind
- Getting the RIGHT SMEs
- Missing requirements
- Jumping into the details too early.
- Not thinking outside of the 'current' box
- Too much focus on one type of requirement
- Not separating the What from the How
- Developers don't understand the problem domain
- No clear definition of 'Done'
- Moving from Abstract to Concrete

Requirements Engineering

Key task of requirements engineering:

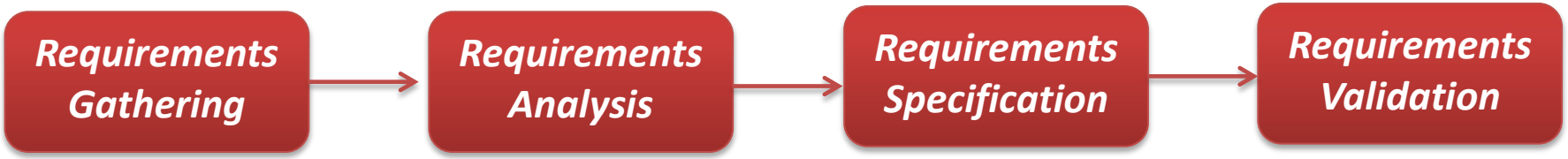
- Formulate a *well-defined problem* to solve
- A well-defined problem consists of:
  - a set of criteria ("requirements") according to which proposed solutions either definitely solve the problem or fail to solve it
  - The description of the resources and components at disposal to solve the problem

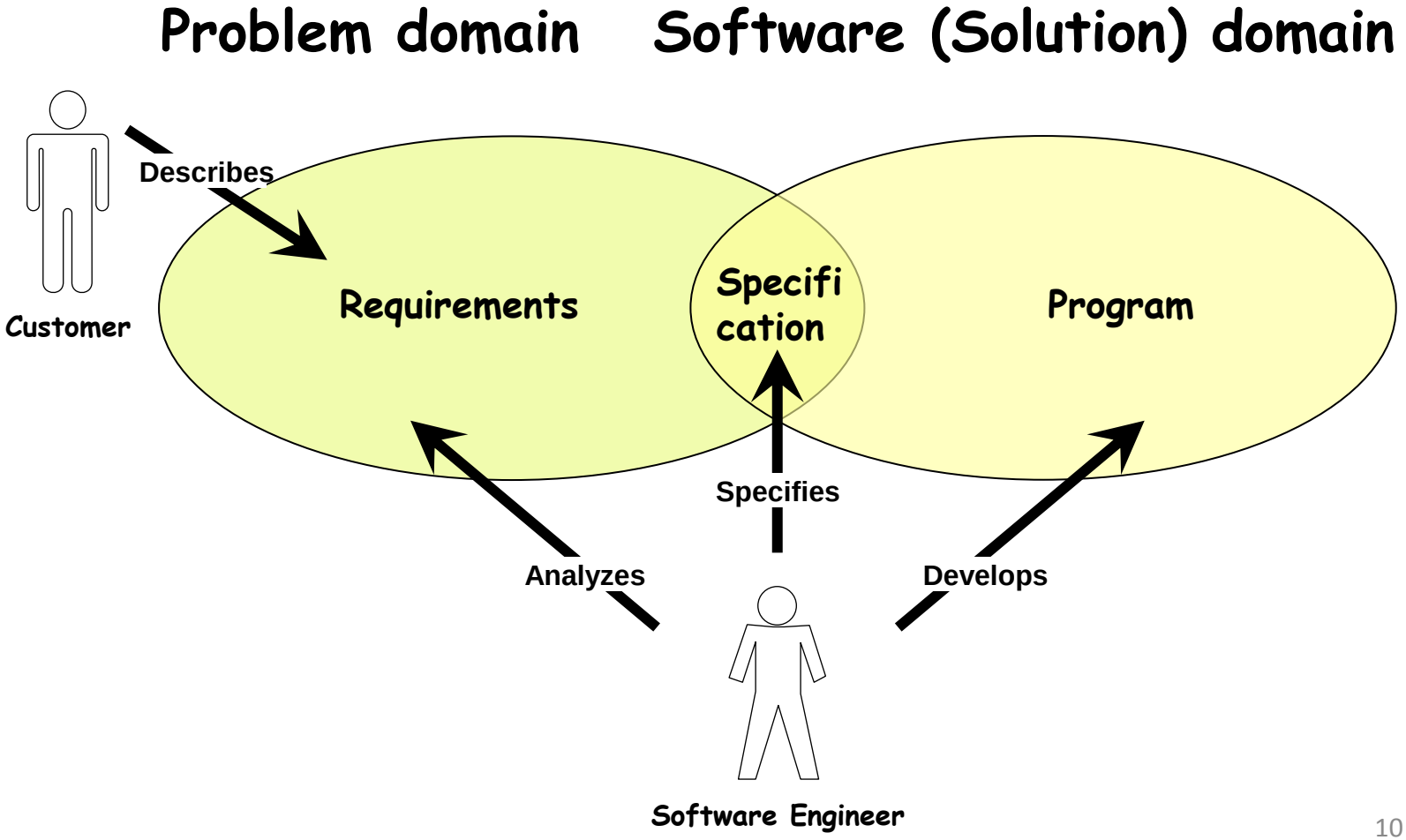
Who is involved in requirements engineering?

Different stakeholders with varying stakes:

- End users interested in the requested functionality
- Business owner interested in cost and time-lines
- Architects and developers interested in how well the functionality is implemented

Phases of Requirements Engineering





10

Phases of Requirements Engineering (1)

Requirements Gathering: A process through which customers, buyers and end-users articulate, discover and understand their requirements

- Customers specify:
  - What is required?
  - How will the intended system fit into the context of their business
  - How the system will be used on a day-to-day basis?
- Developers to understand the business context through meetings (what if this? What if that?), market analysis, questionnaires etc.
- The problem statement is rarely precise

Phases of Requirements Engineering (2)

Requirements Analysis

- Refining and reasoning about the requirements elicited
- Scope the project, negotiate with the customer to determine the priorities - what is important, what is realistic
- Identify dependencies, conflicts and risk
- Apply user-case modelling – elaborate user-scenarios that describe interaction of user with the system; to ensure developer's understanding of the problem matches the customer's expectations or develop user-stories (Agile requirements analysis)

Phases of Requirements Engineering (2)

Requirements Specification

- Document the function, quality and constraints of software-to-be using formal, structured notations or graphical representation to ensure clarity, consistency and completeness

  *(**Use-cases, User-stories, prototypes, formal mathematical models or a combination of these… OR a formal SRS (System Requirements Specification)***

Requirements Validation

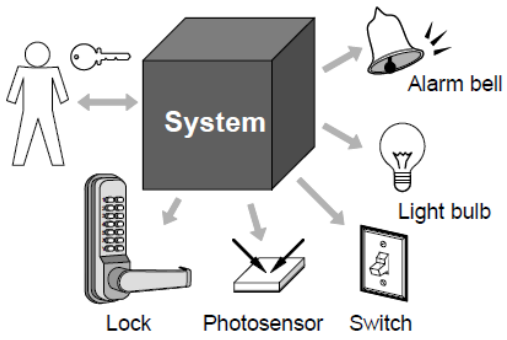
- The process of confirming with the customer or user of the software that the specified requirements are valid, correct, and complete

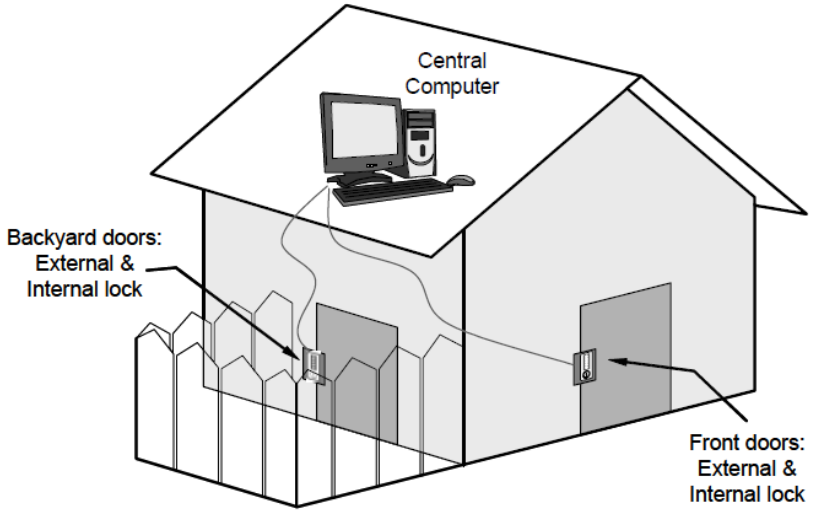
COMP 1531
Software Engineering Fundamentals

Week 02 Thursday
Requirements Engineering (Part I)
Use-Case Modelling

Home Access Case Study

- A home access control system for several functions such as door lock control, lighting control, intrusion detection
- First iteration – Support basic door unlocking and locking functions



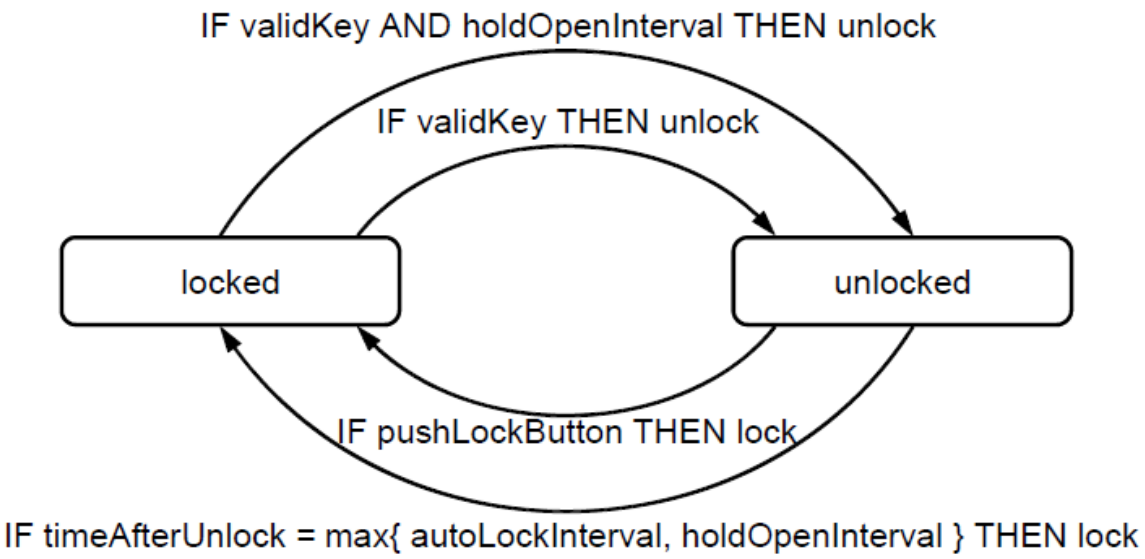


Requirements Analysis Challenges (1)

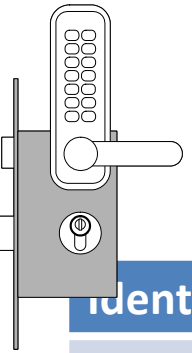
- User-identification: Several choices
  - What you carry on you (physical key or another gadget)
  - What you know (password)
  - Who you are (biometric feature, such as fingerprint, voice, face, or iris)
- Add user-constraints:
  - user should not need to carry any gadgets for identification; and, the identification mechanism should be cheap.
  - rules out a door-mounted reader for magnetic strip ID cards or RFID tags, biometric identification mechanisms
- Solution:
  - simple authentication based on a valid key (memorised by user)
  - Anyone with knowledge of key permitted to enter (no true authentication)

Requirements Analysis Challenges (2)

- But the problem is still complex
  - Handle failed attempts ?
  - Accommodate forgetful users – *autoLock* after #Interval seconds
  - Or perhaps keep door open longer - *holdOpenInterval*



- Hence, clearly, stating the user's goal is critical.



Example System Requirements

Identifier	Priority	Requirement
REQ1	5	The system shall keep the door locked at all times, unless instructed otherwise by an authorised user. When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed) and lights turned off
REQ2	4	The system shall lock the door when commanded by pressing a dedicated button and shut the lights
REQ3	5	The system shall, given a valid key code unlock the door and turn light on
REQ4	3	The system shall permit three failed attempts. However, to resist “dictionary attacks”, after the allowable number of failed attempts, the system will block and an alarm is activated
REQ5	1	The system shall maintain a history log of all attempted accesses for later review
REQ6	2	The system should allow adding new authorised users or removing existing users at run-time

Granularity of requirements

- Some of the requirements in our previous table are relatively complex or compound requirements. Consider REQ2:
  - *The system shall keep the door locked at all times, unless instructed otherwise by an authorised user. When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed)*
- Suppose, testing this requirement fails (the door was found unlocked when it should have been locked)
  - *Did the system accidentally disarm the lock*
  - *Did the auto-lock feature fail?*

(Difficult to tell)

Granularity of requirements

- TDD (Test-driven-development) stipulates writing requirements such that they can be individually tested

- REQ1 can be split into:
  - REQ1a: *The system shall keep the doors locked at all times, unless commanded otherwise by authorized user.*
  - REQ1b: *When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed).*

- Choice of granularity is subject to judgment and experience

Test-Driven Development

- Write User Acceptance Tests (UAT) for requirements during the requirements analysis:
  - capture the customer's assumptions about how the requirement will work and what could go wrong
  - are defined by the customer or developer in collaboration with the customer
- For example, test-cases for REQ1:
  - *Test with the valid key of a current tenant on his or her apartment (pass)*
  - *Test with the valid key of a current tenant on someone else's apartment (fail)*
  - *Test with an invalid key on any apartment (fail)*
  - *Test with the key of a removed tenant on his or her previous apartment (fail)*
  - *Test with the valid key of a just-added tenant on his or her apartment (pass)*

What is UML?

UML stands for **Unified Modelling Language** (http://www.uml.org/)

Programming languages not abstract enough for OO design

An open source, graphical language to model software solutions, application structures, system behaviour and business processes

Several uses:

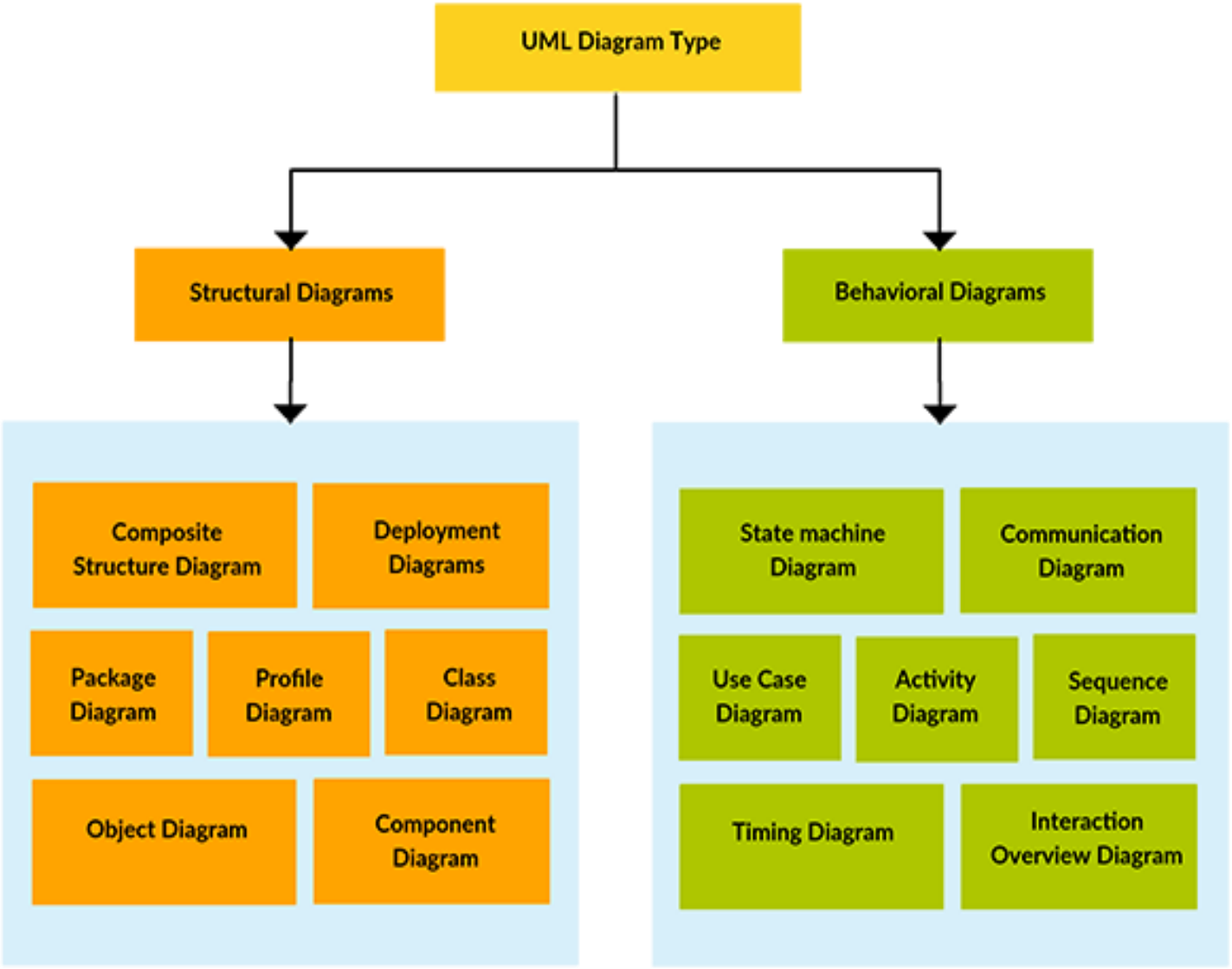
- As a design that communicates aspects of your system
- As a software blue print
- Sometimes, used for auto code-generation

UML diagram categories

- **Structure Diagrams**
  - show the static structure of the system and its parts and how these parts relate to each other
  - they are said to be static as the elements are depicted irrespective of time (e.g. class diagram)

- **Behaviour Diagrams**
  - show the dynamic behaviour of the objects in the system i.e. a series of changes to the system over a period of time (e.g. use case diagram or sequence diagram)
  - a subset of these diagrams are referred to as interaction diagrams that emphasis interaction between objects (e.g., an activity diagram)

UML Diagram Types



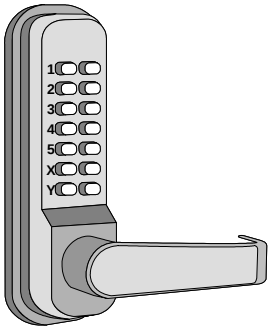
UML Use-Case Diagrams

Use Cases

- Used for Functional Requirements Analysis and Specification
- A ***use case*** is a step-by-step description of how a user will use the system-to-be to accomplish business goals
  - Written as *usage scenarios* visualizing a sequence of actions and interactions between the external actors and the system-to-be

Deriving Use Cases from System Requirements

REQ1: Keep door locked and auto-lock
REQ2: Lock when "LOCK" pressed
REQ3: Unlock when valid key provided
REQ4: Allow mistakes but prevent dictionary attacks
REQ5: Maintain a history log
REQ6: Adding/removing users at runtime



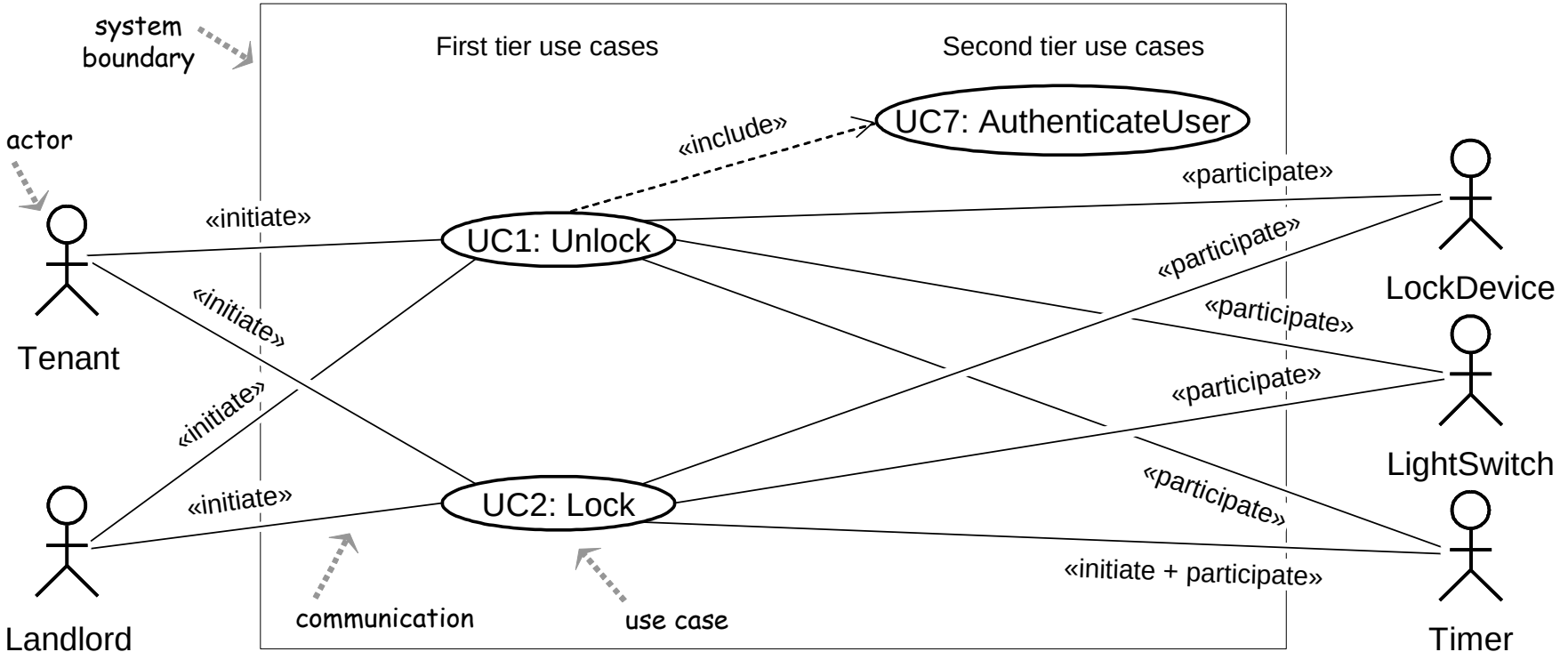
Actor	Actor's Goal (what the actor intends to accomplish)	Use Case Name
Landlord	To disarm the lock and enter, and get space lighted up.	**Unlock (UC-1)**
Landlord	To lock the door & shut the lights (sometimes?).	**Lock (UC-2)**
Landlord	To create a new user account and allow access to home.	**AddUser (UC-3)**
Landlord	To retire an existing user account and disable access.	**RemoveUser (UC-4)**
Tenant	To find out who accessed the home in a given interval of time and potentially file complaints.	**InspectAccessHistory (UC-5)**
Tenant	To disarm the lock and enter, and get space lighted up.	**Unlock (UC-1)**
Tenant	To lock the door & shut the lights (sometimes?).	**Lock (UC-2)**
Tenant	To configure the device activation preferences.	**SetDevicePrefs (UC-6)**
LockDevice	To control the physical lock mechanism.	**UC-1, UC-2**
LightSwitch	To control the lightbulb.	**UC-1, UC-2**
[to be identified]	To auto-lock the door if it is left unlocked for a given interval of time.	**AutoLock (UC-2)**

Types of Actors

- ***Initiating actor*** (also called *primary actor* or simply "user"): initiates the use case to achieve a goal
- ***Participating actor*** (also called *secondary actor*): participates in the use case but does not initiate it.
  - helps the system-to-be to complete the use case

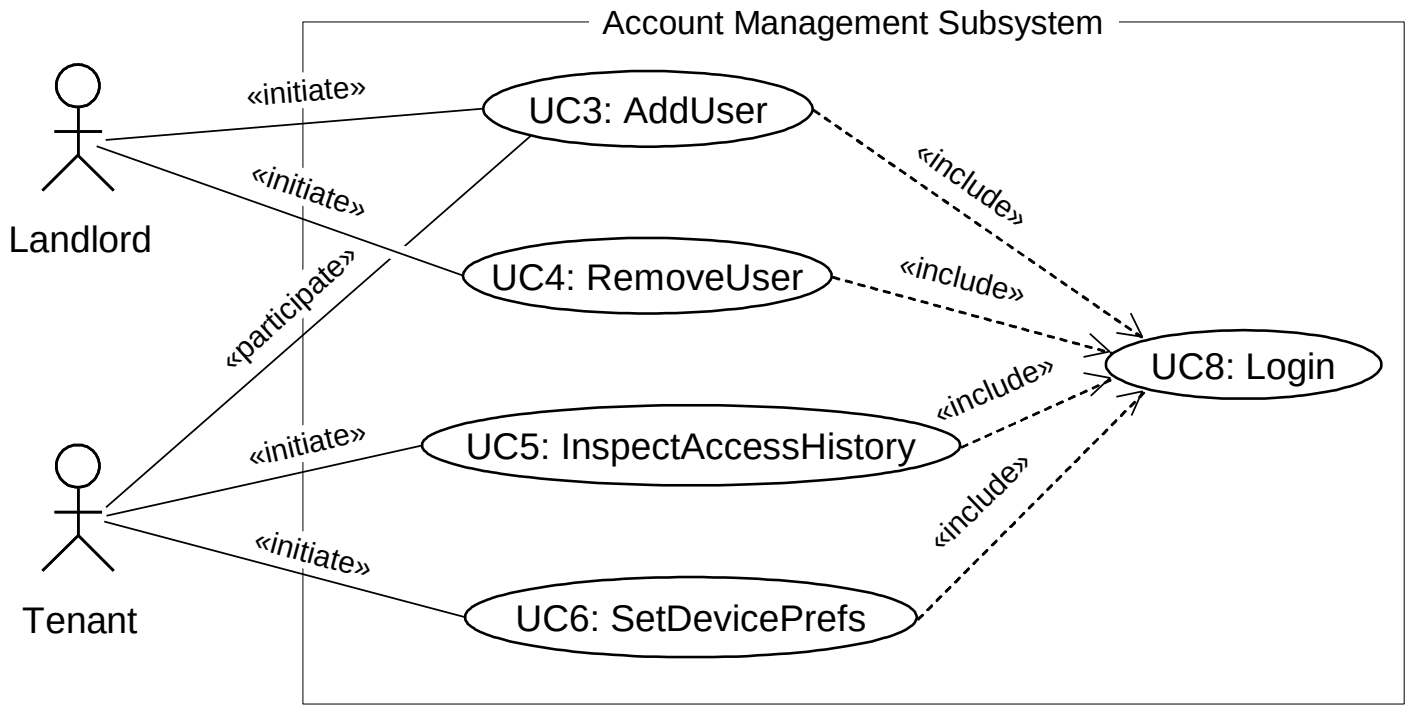
Use Case Diagram: Device Control

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login



Use Case Diagram: Account Management

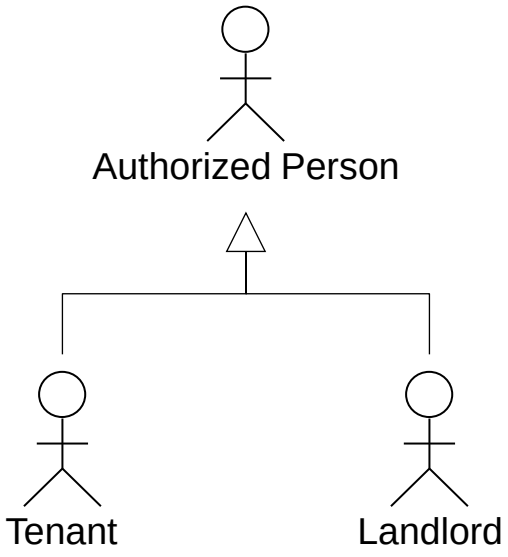
UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login



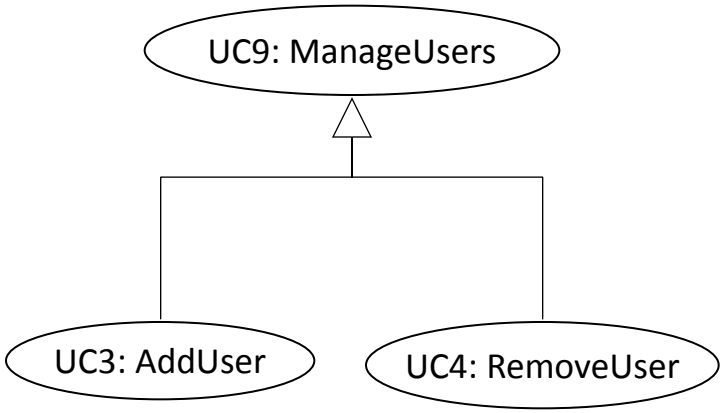
Use Case Generalizations

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login

- More abstract representations can be derived from particular representations



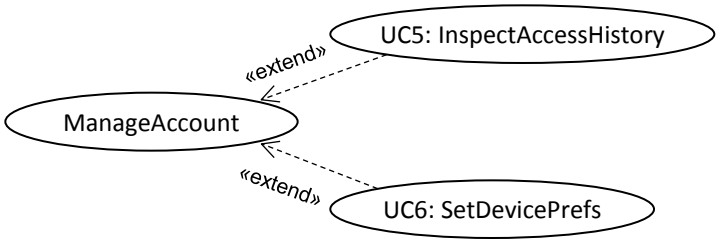
Actor Generalization



Use Case Generalization

Optional Use Cases: «extend»

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login



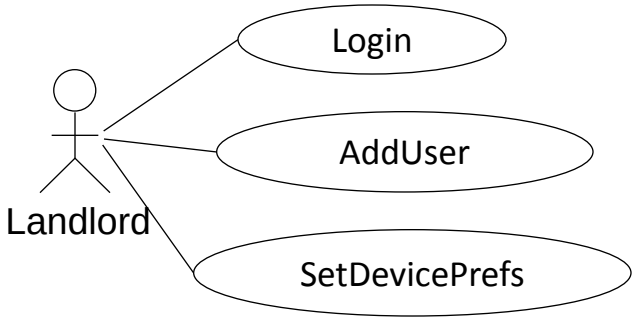
Key differences between **«include»** and **«extend»** relationships

	Included use case	Extending use case
Is this use case optional?	No	Yes
Is the base use case complete without this use case?	No	Yes
Is the execution of this use case conditional?	No	Yes
Does this use case change the behavior of the base use case?	No	Yes

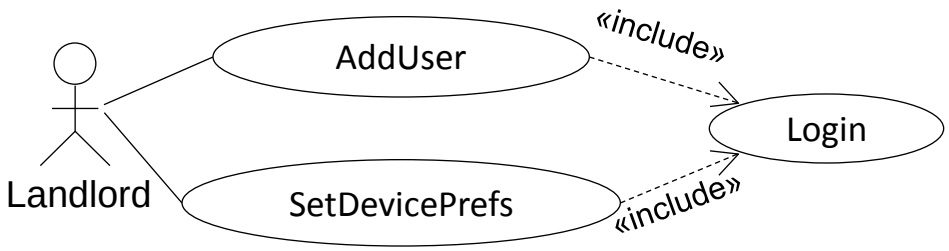
[Source: Robert Maksimchuk & Eric Naiburg: ***UML for Mere Mortals***, Addison-Wesley, 2005.]

Login Use Case?

BAD:



GOOD:



Traceability Matrix

Mapping: System requirements to Use cases

REQ1: Keep door locked and auto-lock
REQ2: Lock when “LOCK” pressed
REQ3: Unlock when valid key provided
REQ4: Allow mistakes but prevent dictionary attacks
REQ5: Maintain a history log
REQ6: Adding/removing users at runtime

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login

Req’t	**PW**	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8
REQ1	5	X	X						
REQ2	2		X						
REQ3	5	X						X	
REQ4	4	X						X	
REQ5	2	X	X						
REQ6	1			X	X				X
REQ7	2						X		X
REQ8	1					X			X
REQ9	1					X			X
Max PW		5	2	2	2	1	5	2	1
Total PW		15	3	2	2	3	9	2	3

Continued for domain model, design diagrams, …

Traceability Matrix Purpose

- **Traceability** refers to the property of a software artefact (use-case, a class etc) of being *traceable* to the original requirement that motivated its existence
- Traceability matrices are continued through the domain model, design diagrams etc…
- In the context of use-cases, the matrix serves to:
  - To check that all requirements are covered by the use cases
  - To check that none of the use cases is introduced without a reason (i.e., created not in response to any requirement)
  - To prioritize the work on use cases

Schema for Detailed Use Cases

Use Case UC-#:	**Name / Identifier [verb phrase]**
Related Requirements:	List of the requirements that are addressed by this use case
Initiating Actor:	Actor who initiates interaction with the system to accomplish a goal
Actor's Goal:	Informal description of the initiating actor's goal
Participating Actors:	Actors that will help achieve the goal or need to know about the outcome
Preconditions:	What is assumed about the state of the system before the interaction starts
Postconditions:	What are the results after the goal is achieved or abandoned; i.e., what must be true about the system at the time the execution of this use case is completed

Flow of Events for Main Success Scenario:		
→	1.	The initiating actor delivers an action or stimulus to the system (the arrow indicates the direction of interaction, to- or from the system)
←	2.	The system's reaction or response to the stimulus; the system can also send a message to a participating actor, if any
→	3.	…

Flow of Events for Extensions (Alternate Scenarios): What could go wrong? List the exceptions to the routine and describe how they are handled		
→	1a.	For example, actor enters invalid data
←	2a.	For example, power outage, network failure, or requested data unavailable
		…

The arrows on the left indicate the direction of interaction: → Actor's action; ← System's reaction

Use Case 1: Unlock

Use Case UC-1:	**Unlock**
Related Requirem'ts:	REQ1, REQ3, REQ4, and REQ5 stated in Table 2-1
Initiating Actor:	Any of: Tenant, Landlord
Actor's Goal:	To disarm the lock and enter, and get space lighted up automatically.
Participating Actors:	LockDevice, LightSwitch, Timer
Preconditions:	• The set of valid keys stored in the system database is non-empty. • The system displays the menu of available functions; at the door keypad the menu choices are "Lock" and "Unlock."
Postconditions:	The auto-lock timer has started countdown from autoLockInterval.

Flow of Events for Main Success Scenario:

→	1.	**Tenant**/**Landlord** arrives at the door and selects the menu item "Unlock"
	2.	include::*AuthenticateUser* (UC-7)
←	3.	**System** (a) signals to the **Tenant**/**Landlord** the lock status, e.g., "disarmed," (b) signals to **LockDevice** to disarm the lock, and (c) signals to **LightSwitch** to turn the light on
←	4.	**System** signals to the **Timer** to start the auto-lock timer countdown
→	5.	**Tenant**/**Landlord** opens the door, enters the home [and shuts the door and locks]

Subroutine «include» Use Case

Use Case UC-7:	**AuthenticateUser** (sub-use case)
Related Requirements:	REQ3, REQ4 stated in Table 2-1
Initiating Actor:	Any of: Tenant, Landlord
Actor's Goal:	To be positively identified by the system (at the door interface).
Participating Actors:	AlarmBell, Police
Preconditions:	• The set of valid keys stored in the system database is non-empty. • The counter of authentication attempts equals zero.
Postconditions:	None worth mentioning.

Flow of Events for Main Success Scenario:

← 1. **System** prompts the actor for identification, e.g., alphanumeric key

→ 2. **Tenant**/**Landlord** supplies a valid identification key

← 3. **System** (a) verifies that the key is valid, and (b) signals to the actor the key validity

Flow of Events for Extensions (Alternate Scenarios):

2a. **Tenant**/**Landlord** enters an invalid identification key

← 1. **System** (a) detects error, (b) marks a failed attempt, and (c) signals to the actor

← 1a. **System** (a) detects that the count of failed attempts exceeds the maximum allowed number, (b) signals to sound **AlarmBell**, and (c) notifies the **Police** actor of a possible break-in

→ 2. **Tenant**/**Landlord** supplies a valid identification key

3. Same as in Step 3 above

Acceptance Test Case for UC-7 Authenticate User

Test-case Identifier:	TC-1
Use Case Tested:	UC-1, main success scenario, and UC-7
Pass/fail Criteria:	The test passes if the user enters a key that is contained in the database, with less than a maximum allowed number of unsuccessful attempts
Input Data:	Numeric keycode, door identifier

Test Procedure:	**Expected Result:**
Step 1. Type in an incorrect keycode and a valid door identifier	System beeps to indicate failure; records unsuccessful attempt in the database; prompts the user to try again
Step 2. Type in the correct keycode and door identifier	System flashes a green light to indicate success; records successful access in the database; disarms the lock device

Use Case 2: Lock

Use Case UC-2:	**Lock**
Related Requirements:	REQ1, REQ2, and REQ5 stated in Table 2-1
Initiating Actor:	Any of: Tenant, Landlord, or Timer
Actor's Goal:	To lock the door & get the lights shut automatically (?)
Participating Actors:	LockDevice, LightSwitch, Timer
Preconditions:	The system always displays the menu of available functions.
Postconditions:	The door is closed and lock armed & the auto-lock timer is reset.

Flow of Events for Main Success Scenario:

→ 1. **Tenant/Landlord** selects the menu item "Lock"

← 2. **System** (a) signals affirmation, e.g., "lock armed," (b) signals to **LockDevice** to arm the lock (if not already armed), (c) signal to **Timer** to reset the auto-lock counter, and (d) signals to **LightSwitch** to turn the light off (?)

Flow of Events for Extensions (Alternate Scenarios):

2a. System senses that the door is not closed, so the lock cannot be armed

← 1. **System** (a) signals a warning that the door is open, and (b) signal to **Timer** to start the alarm counter

→ 2. **Tenant/Landlord** closes the door

← 3. **System** (a) senses the closure, (b) signals affirmation to the **Tenant/Landlord**, (c) signals to **LockDevice** to arm the lock, (d) signal to **Timer** to reset the auto-lock counter, and (e) signal to **Timer** to reset the alarm counter

Use Case 3: Add User

Use Case UC-3:	**AddUser**
Related Requirements:	REQ6 stated in **Error! Reference source not found.**
Initiating Actor:	Landlord
Actor's Goal:	To register new or remove departed residents at runtime.
Participating Actors:	Tenant
Preconditions:	None worth mentioning. (But note that this use case is only available on the main computer and not at the door keypad.)
Postconditions:	The modified data is stored into the database.

Flow of Events for Main Success Scenario:

→ 1. **Landlord** selects the menu item "ManageUsers"

2. **Landlord** identification: Include *Login* (UC-8)

← 3. **System** (a) displays the options of activities available to the Landlord (including "Add User" and "Remove User"), and (b) prompts the **Landlord** to make selection

→ 4. **Landlord** selects the activity, such as "Add User," and enters the new data

← 5. **System** (a) stores the new data on a persistent storage, and (b) signals completion

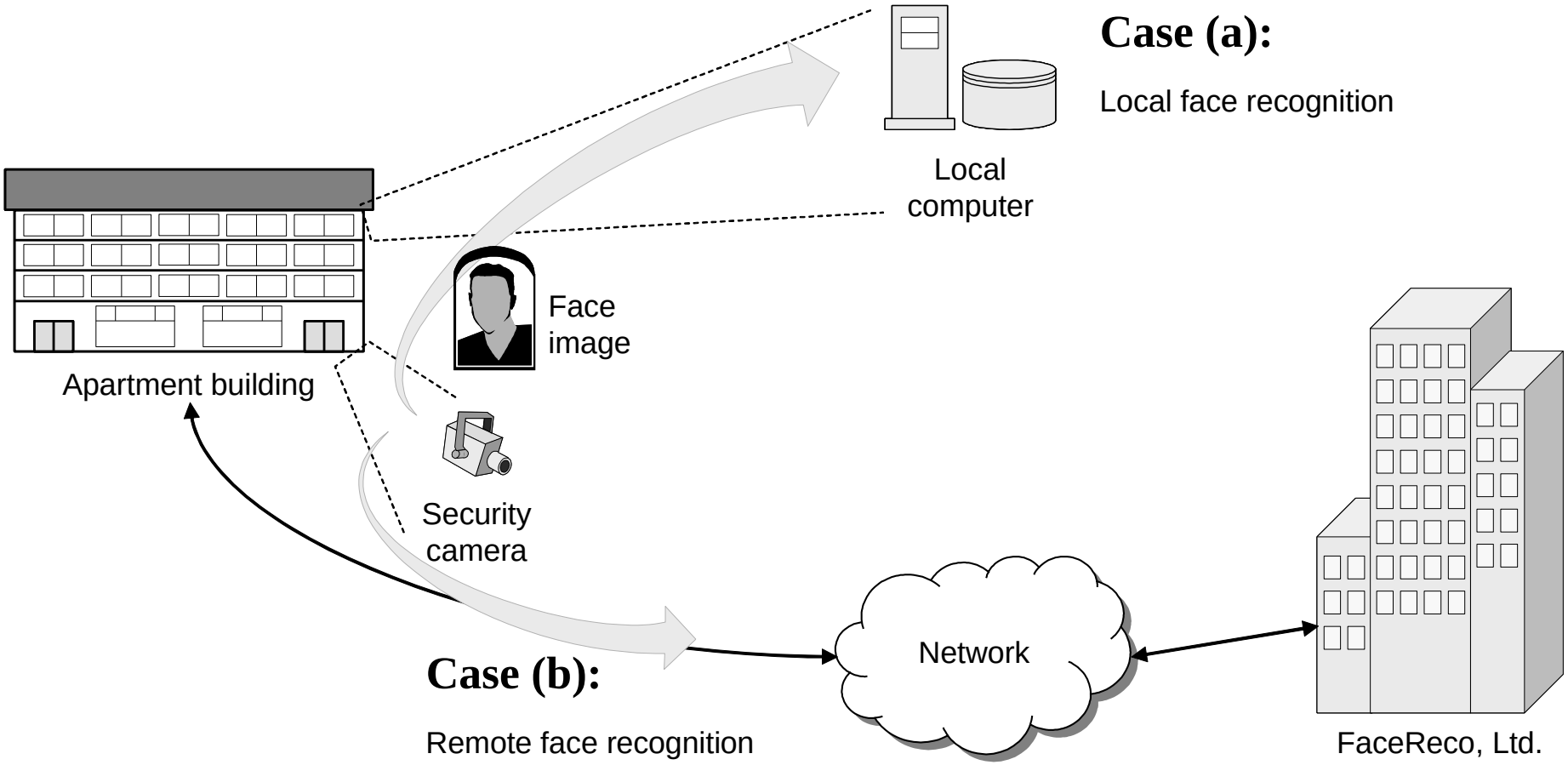
Flow of Events for Extensions (Alternate Scenarios):

4a. Selected activity entails adding new users: Include *AddUser* (UC-3)

4b. Selected activity entails removing users: Include *RemoveUser* (UC-4)

System Boundary & Subsystems

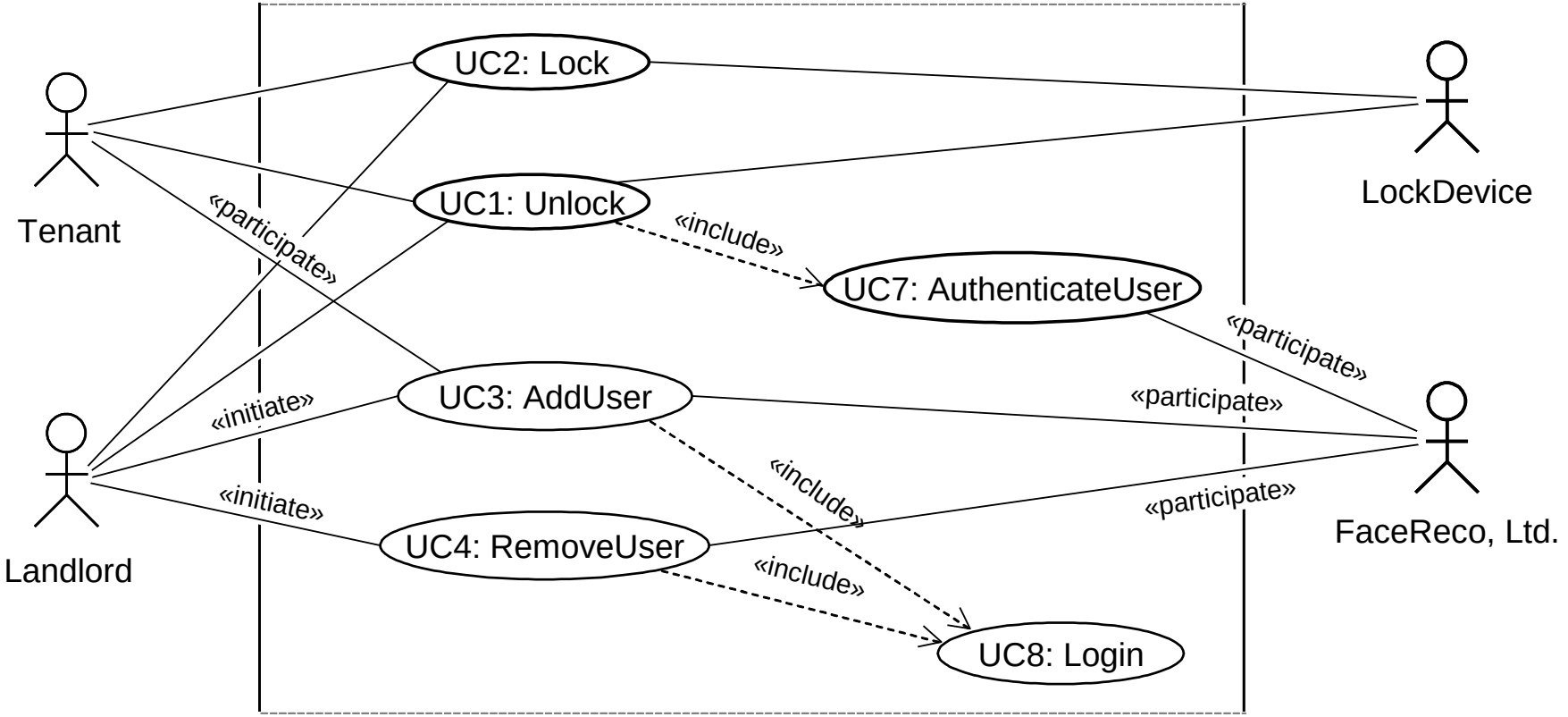
Use Case Variations Example:



Modified Use Case Diagram

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login

Authentication subsystem (FaceReco, Ltd.)
is externalized from the system-to-be:



Next week …

Domain-modelling based on OO design