COMP 1531
Software Engineering Fundamentals

Week 02
Requirements Engineering
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 functionality of the "system-to-be" - the set of 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 "system-to-be"

- The constraints of the functionality provided by the 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)

Checkpoint !

Quiz - FR and NFR

What is Requirements Engineering ?

"The hardest single part of building a software system is deciding what to build. No part of the work so cripples the resulting systems if done wrong" (Brooks, 1987)

Requirements Engineering is:

- A set of activities concerned with identifying and communicating the purpose of a software system and the context it will be used
- A negotiation process, where
  - potential "users" of the system explore the requirements, agreeing what they *want* and what they *need* and
  - software engineers formulate a *well-defined problem* to solve, where 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 (Marsic, 2012)

Software Engineering, Ivan Marsic, 2012

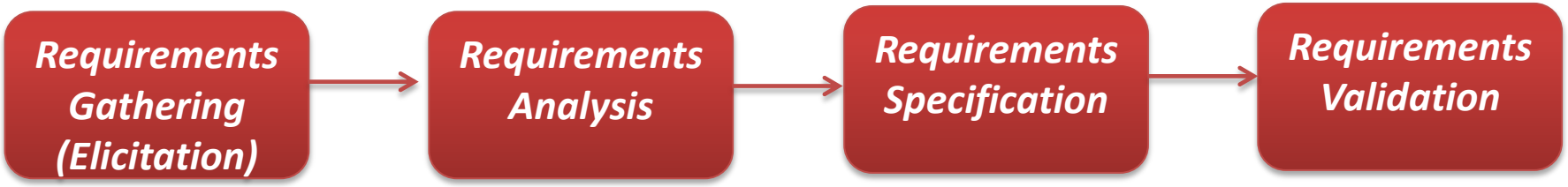
Brooks, F.P. (1987). ***Essence and Accidents of Software Engineering***, *IEEE Computer, Vol. April 1987, pp. 10-19*

Participants in requirements engineering

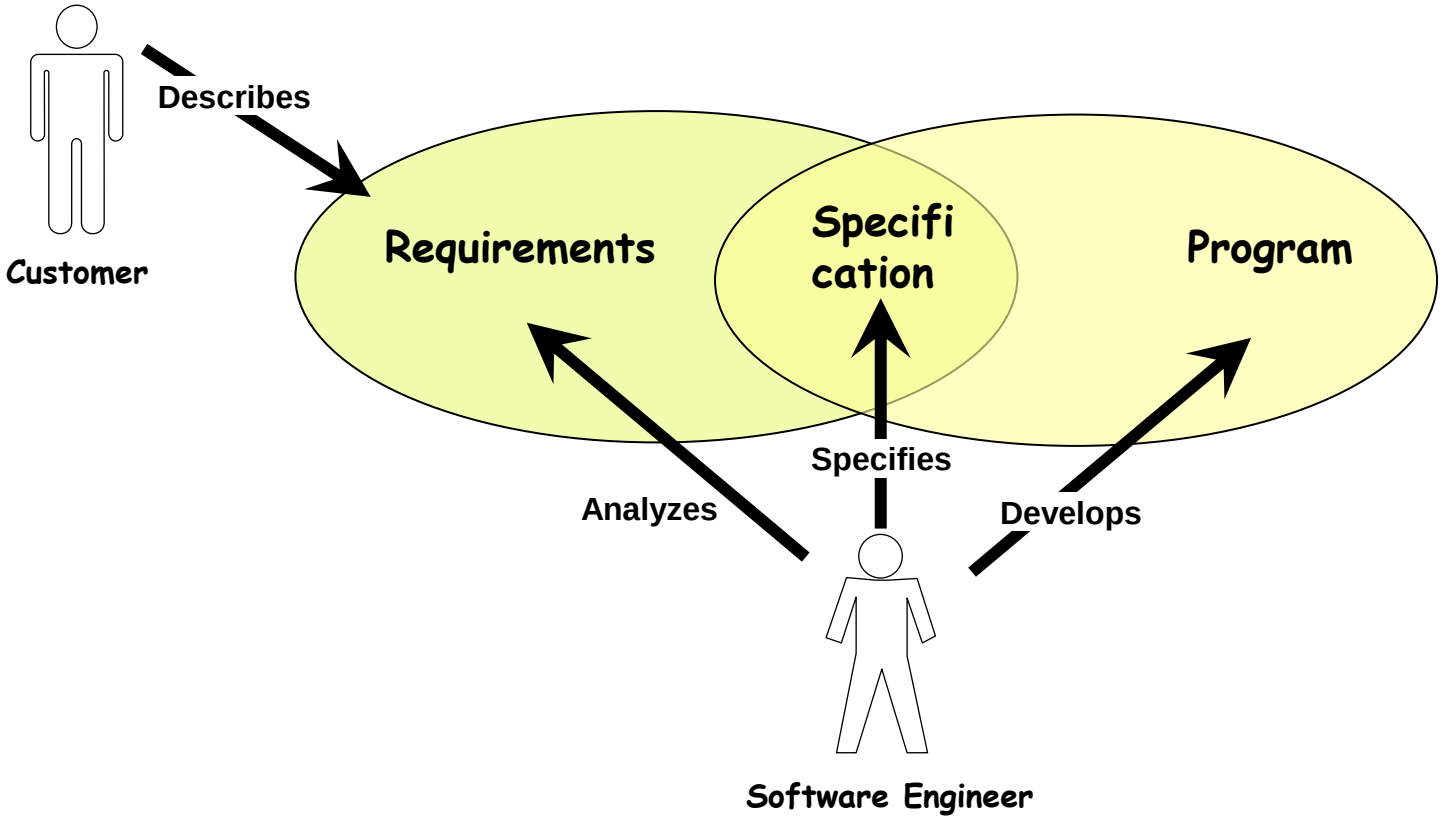
Different stakeholders with varying stakes:

- End users interested in the requested functionality
- Customer (Business owner) interested in cost and time-lines
- Design team (software engineers, architects and developers) interested in how well the functionality is implemented

Phases of Requirements Engineering



Problem domain **Software (Solution) domain**



Phases of Requirements Engineering (1)

Requirements Gathering (Elicitation): A process where customers, 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 understand the business context through meetings (what if this? What if that?), market research, questionnaires, focus groups etc.
- The problem statement is rarely precise

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

Phases of Requirements Engineering (2)

Requirements Analysis

- Start with the customer statement of requirements or the vision statement
- Refine and reason 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

Techniques in Requirements Analysis

Two popular techniques in Requirements Analysis

1. Use-case modelling - Build a set of use-cases, to describe the tasks to be performed by the "system-to-be"
   - Each use-case represents a dialog between user and system, helping the user achieve a goal
   - In each dialog, user initiates an action, system responds with a reaction
   - Build elaborate user-scenarios for each use-case that describe the interaction between user and system
2. Develop user-stories (Agile requirements analysis)

Phases of Requirements Engineering (2)

Requirements Specification

- Document the functional and non-functional (quality and constraints of software-to-be) requirements 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)

Phases of Requirements Engineering (2)

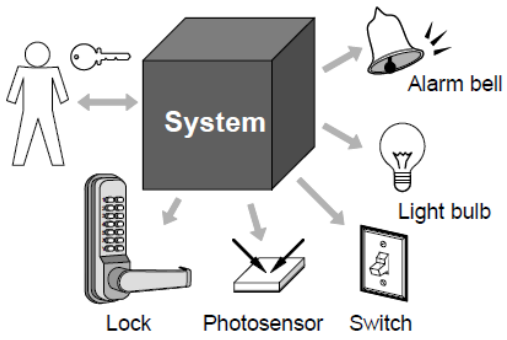
Requirements Validation

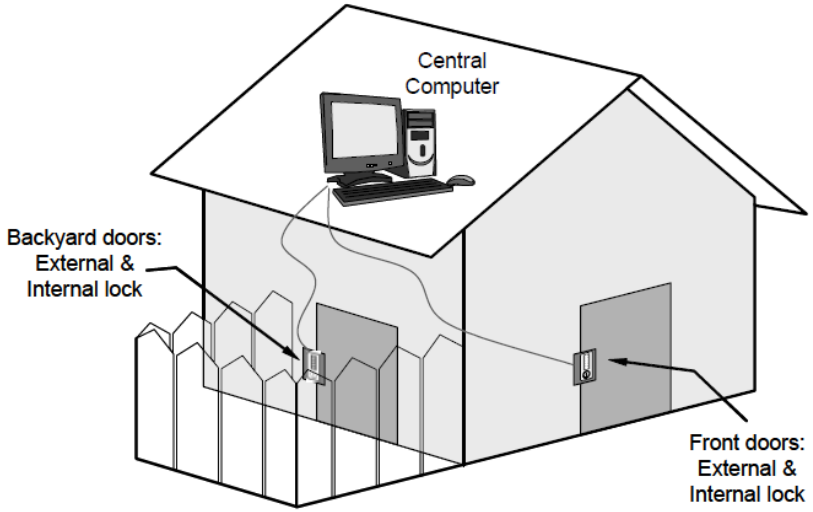
- The process of confirming with the customer or user of the software that the specified requirements are valid, correct, and complete
- Ensure developer's understanding of the problem matches the customer's expectations

The logical ordering of the four activities (elicitation, analysis, specification and validation) does not necessarily imply, they are performed sequentially

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



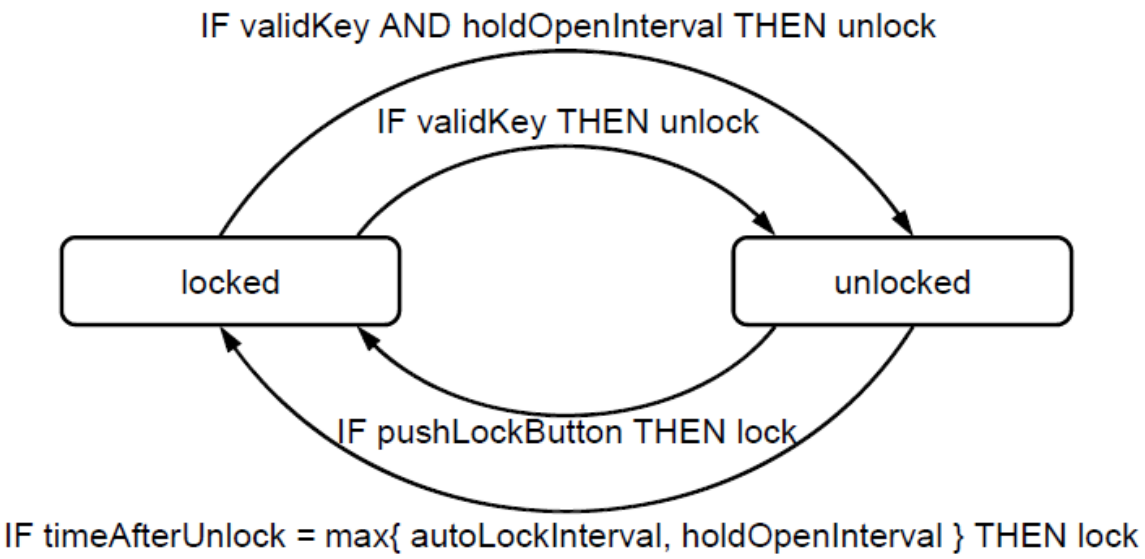


Requirements Analysis Challenges (1)

- Access control system requires 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.

High Level System Requirements (Traditional Requirements Analysis)

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 REQ1:
  - *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

- REQ 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

- TDD (Test-driven-development) stipulates writing tests for the requirements during requirements analysis
- These tests known as User Acceptance Tests (UAT):
  - 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
- The customer can work with developer to write the actual test cases. A **test case** is a particular choice of input values to be used in testing a program and expected output values
- A **test** is a finite collection of test cases

Test-Cases for Case-Study, 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)

Agile Requirements Analysis and Specification With User-Stories

User Story

- An approach used in agile software development to elicit requirements
- A short, simple descriptions of a feature or requirement narrated from the perspective of the person who desires that capability
  - Initial User Story (informal)

    e.g., Student can purchase monthly parking passes online
  - Initial User Story (formal): uses a RGB (Role-Goal-Benefit) template

    As a < type of user >, I want < some goal > so that < some reason >

    e.g., As a student, I want to purchase a parking pass so that I can drive to school

User Story

User-Stories are:

- A reminder to have a conversation with your customer (they shift the focus from writing about features to discussing them)
- Anyone can write user-stories - product owner (*a key stakeholder with a clear vision of requirements and communicates this vision to the developers*) or team member
- But, it is the product owner's responsibility to make sure a product backlog of user stories exist
- Use the simplest tool - Index cards or sticky notes, stored in a shoe-box, and arranged on walls or tables to facilitate planning and discussion

Granularity of a User Story

- **Epic user stories** – covers large amount of functionality and generally too large for an agile team to complete in one iteration
  - e.g., a student can purchase a monthly parking pass OR print student transcripts online

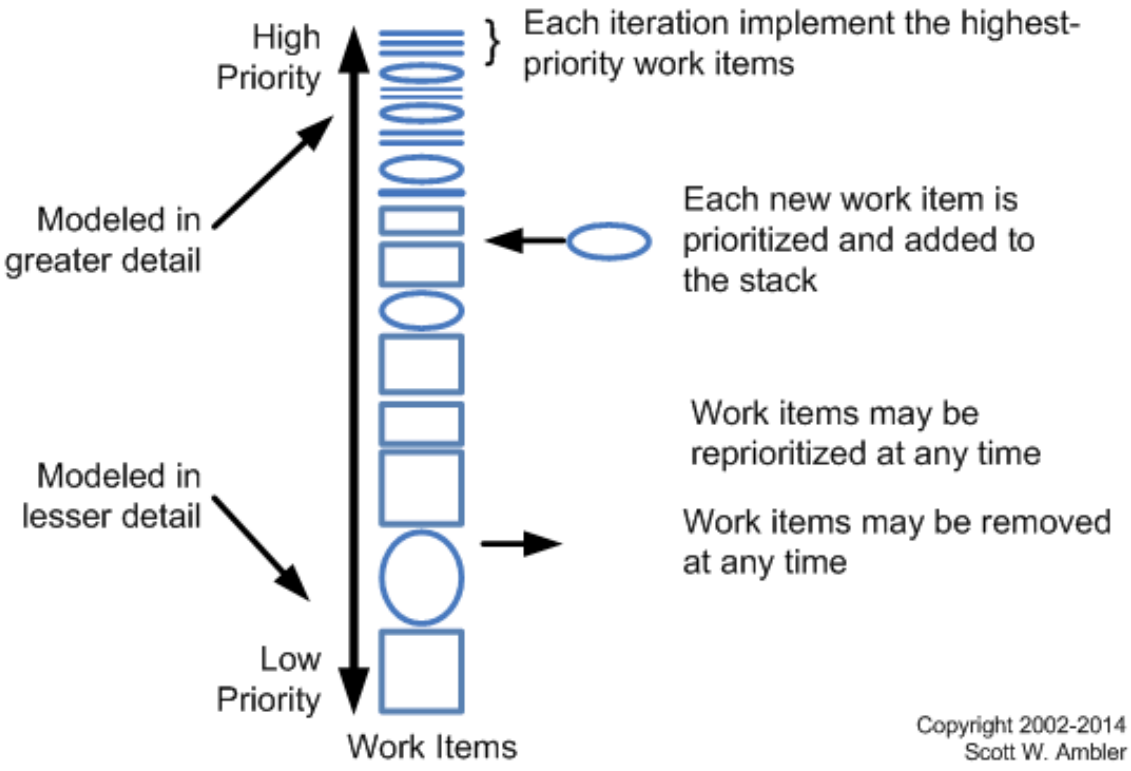
- **Themes** – A collection of related epic user-stories
  - e.g., possible themes for a university registration system - student enrolment, course management, transcript generation

Granularity of a User Story

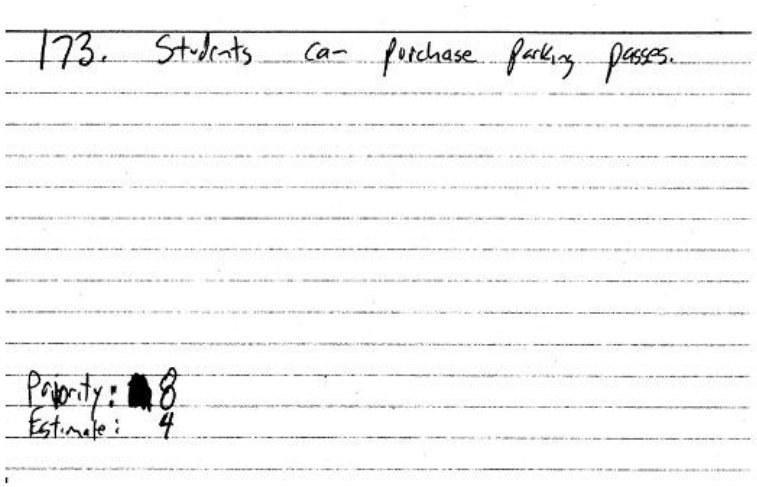
- **User Stories** - split an epic user story into multiple smaller atomic stories, so the story is small enough to be coded and tested in one iteration
  - As a student, I can purchase a monthly or annual parking pass with my credit-card so that I am able to drive to the university
  - As a student, I can purchase a monthly or annual parking pass with PayPal so that I am able to drive to the university
  - As a student, I can order my official transcript online to save time from going in person to student services

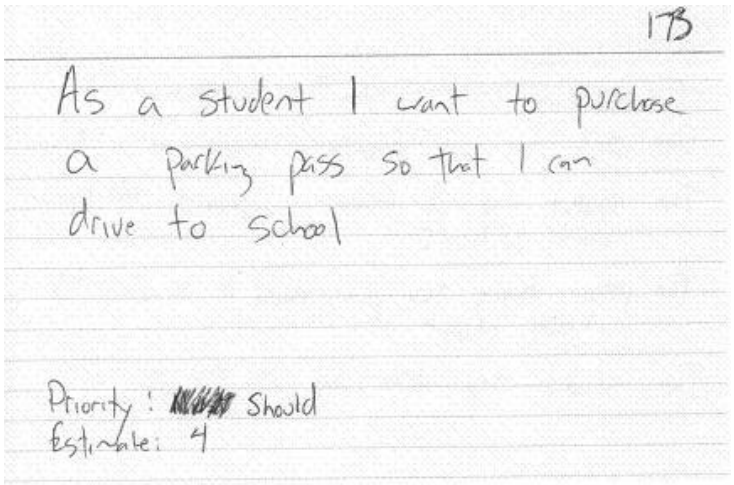
User Story and Planning

- User-stories not only capture the user's vision but also impact the planning process in two key areas; estimating and scheduling



Important considerations for User Stories

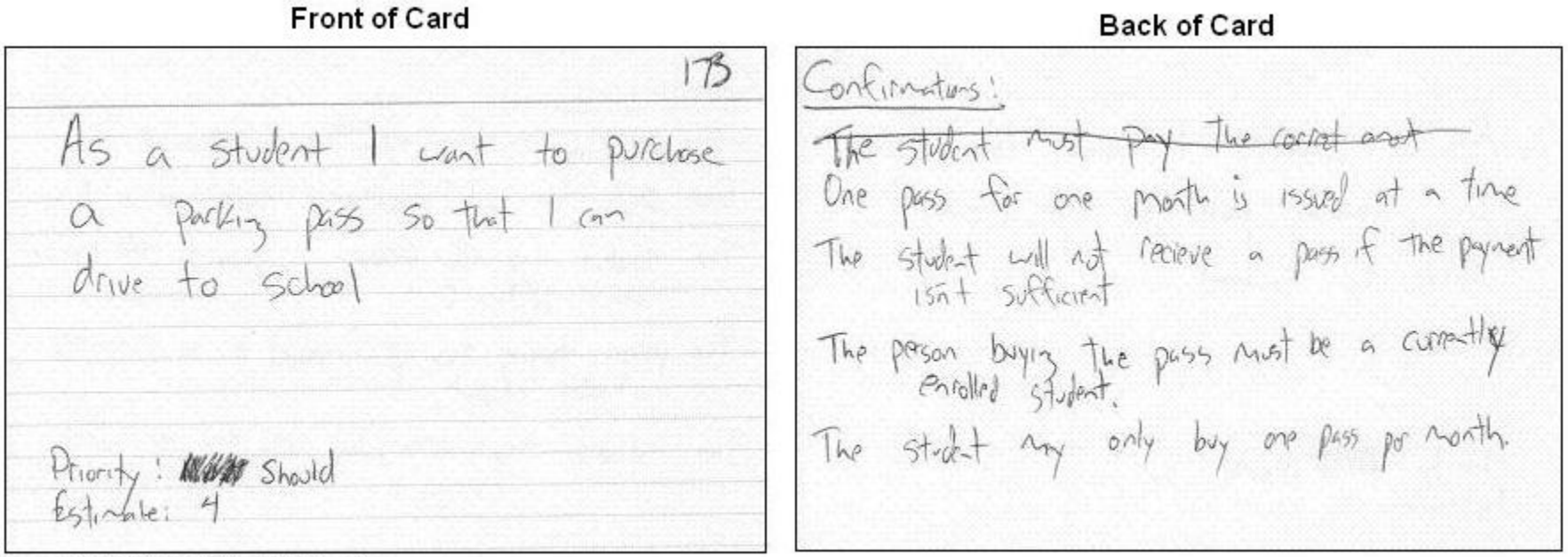




- Assign each user-story a **unique identifier** e.g., US – 12
- Remember non-functional requirements (e.g., the *Students can purchase parking passes online* user story is a usage requirement similar to a use case whereas the *Transcripts will be available online via a standard browser* is closer to a *technical requirement*)
- **Indicate the estimated size** - assign user story points to each card, a relative indication of how long it will take a pair of programmers to implement the story. (e.g., if a user-story point = 2.5 hours, user story (above) will take around 10 hours)
- **Indicate the priority** (e.g., on a scale of 1 – 10)

Detailing a user-story

- User-story with confirmations or acceptance-criteria (similar to test-cases in traditional requirements analysis), that define when a story is “completed”



Techniques to write a User Story (1)

A common technique adopted is the **Role-Feature-Reason** template or **RGB (Role, Goal, Benefit)**, developed by Mike Cohn of Mountain Goat Software, 1991

> *As a **< type of user >**, I want **< some goal >** so that **< some reason >***
>
> e.g.,
>
> As a *librarian*, I want *to have the facility to search for a book by different criteria such as author, title and ISBN* so that *I will save time to serve our customer*.
>
> As a *student*, I'd like to *be able to search the course offerings*, so that *I'll be able to find an offering that most interests me*.

RGB Template

Keeps the focus on the *who*, *what* and *why*

- **Role (who):** describes *who* will be benefited by the feature, must clearly identify the specific type of user e.g., a manager, administrator, librarian, trainer, student etc.
- **Goal (what):** describes ***what*** the user wants from the perspective of the user and **not** from the perspective of the developer who will be coding it e.g., feature = “search for a book”, “search the course offering”
- **Benefit (why):** states why the user wants this feature. What benefit the user will get out of this feature? e.g. goal = “improve customer service”, “find an offering that interests me”. (If the value or benefit can’t be articulated, it might be something that’s not necessary)

Techniques to write a User Story (2)

The three C's model (Ron Jeffries, 2001), identifies three primary components of a user-story that helps business and technical team reach an agreement on the meaning of the user-story.

- a ***"Card"***, a simple statement usually written in the RGB format, addressing the "who", "what" and "why" on an index-card
- a ***"Conversation"***, detailing the simple requirement; *conversation* can take place at different times in the project, enables development team to obtain a clearer understanding of how the feature will work in different situations, including error conditions, the value being provided; conversation is largely verbal but most often supplemented by documentation
- the ***"Confirmation"***, developers need to get *confirmation* regarding the acceptance criteria from the product owner; define the acceptance tests, that will be used to show that the story has been implemented correctly

Developing User Stories For Home Access Control

User Stories are similar to the high-level system requirements, but shift the focus to user benefits, from system features..

As a tenant, I can unlock the doors so that I can enter my apartment.

user-role (benefactor) | goal | business-value or benefit

Acceptance Criteria:

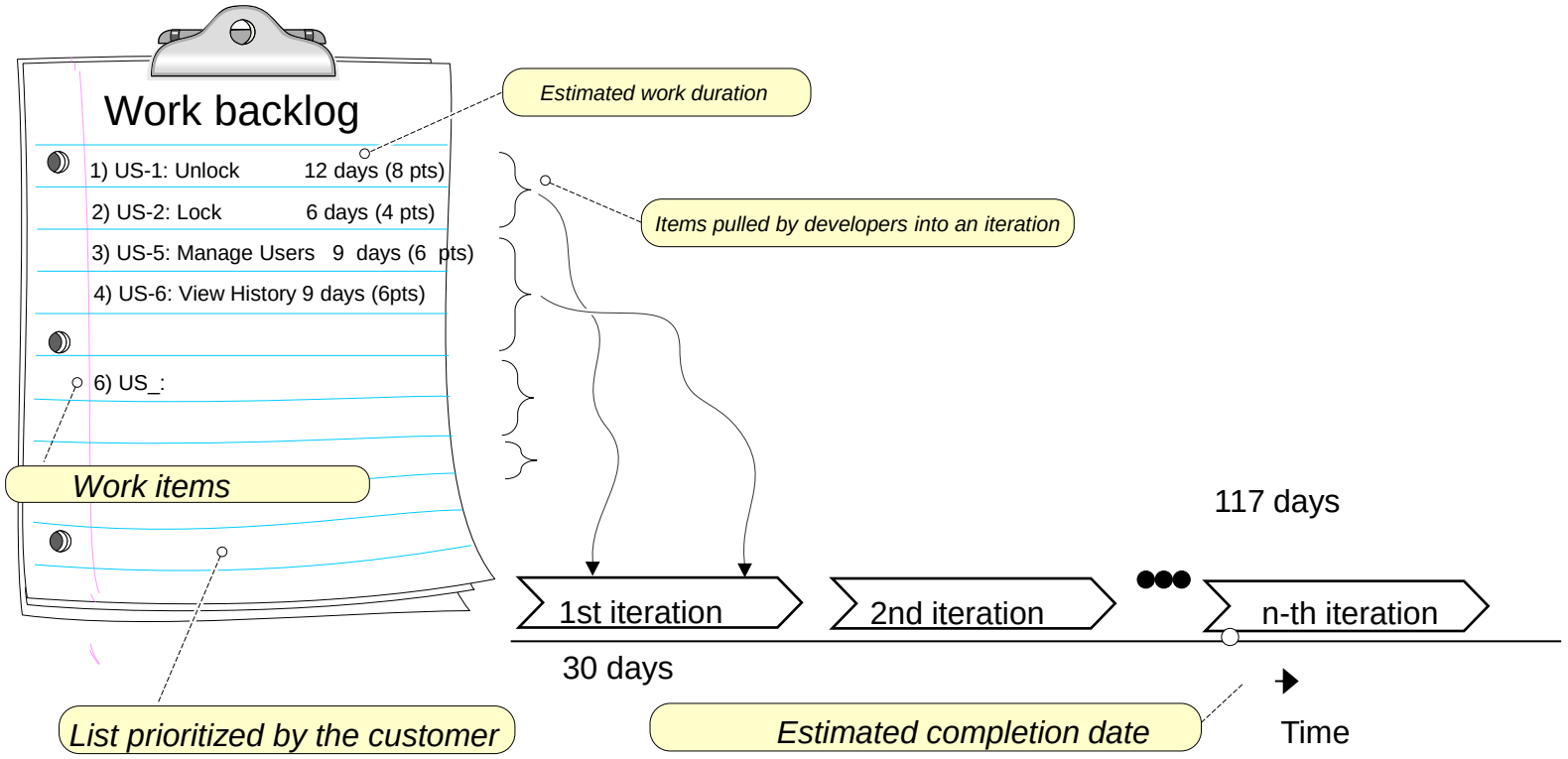
- A tenant is able to enter successfully into his/her apartment after supplying a valid pin code
- A tenant is able to enter an invalid pin code and be prompted to try again with a valid pin code
- A tenant is prevented from entering a pin code after three attempts and an alarm is raised
- A tenant is prevented from entering into another apartment by using his/her pin code

Example User Stories

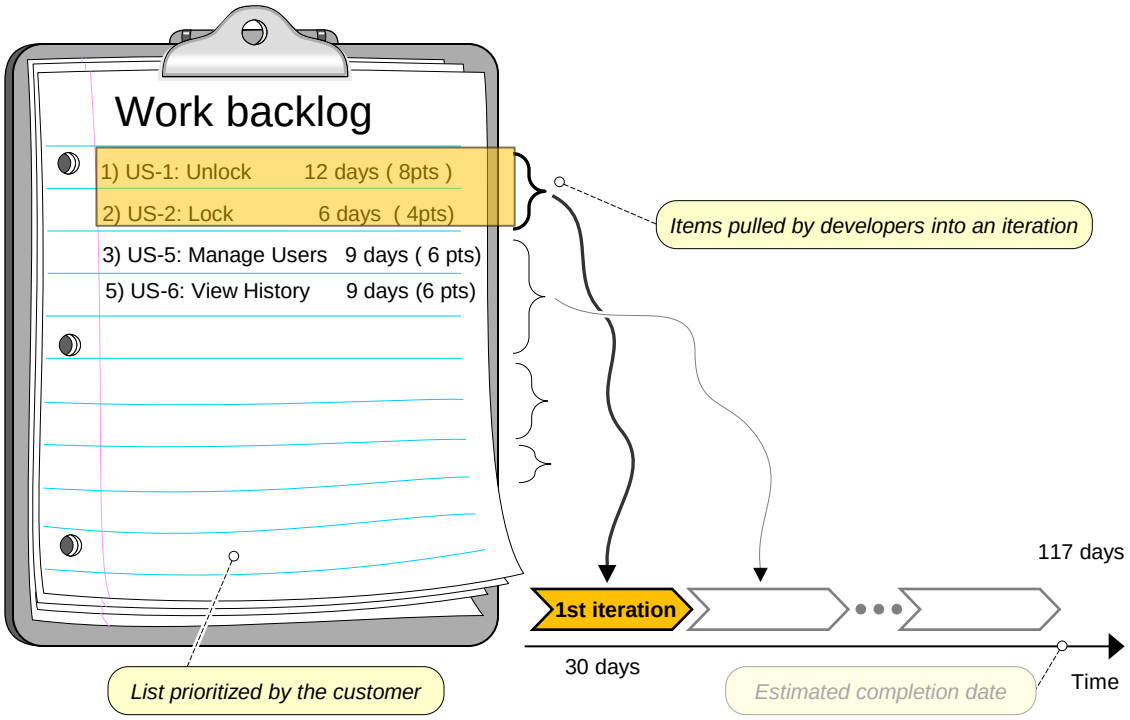
Identifier	User Story	Size
US-1	As an authorized person (tenant or landlord), I can unlock the doors by providing a valid key code to enter the apartment	8 pts
US-2	As an authorized person (tenant or landlord), I can initiate locking of the doors on demand by pressing on a button, to lock the apartment	4 pts
US-3	The system shall initiate a count-down once the door is opened, at the end of which the lock will be automatically locked to ensure automatic lock-down and prevent intruders from entering the apartment	3 pts
US-4	As an authorized person (tenant or landlord), I can keep the doors locked at all times unless explicitly disarmed to prevent intruders	4 pts
US-5	As a landlord, I can at runtime manage (add and remove) authorized tenants to authorise access for new tenant and remove old tenants	12 pts
US-6	As an authorized person (tenant or landlord), I can view past accesses.	8 pts
US-7	As a tenant, I can file complaint about suspicious accesses	4 pts

Agile Project Effort Estimation for case-study

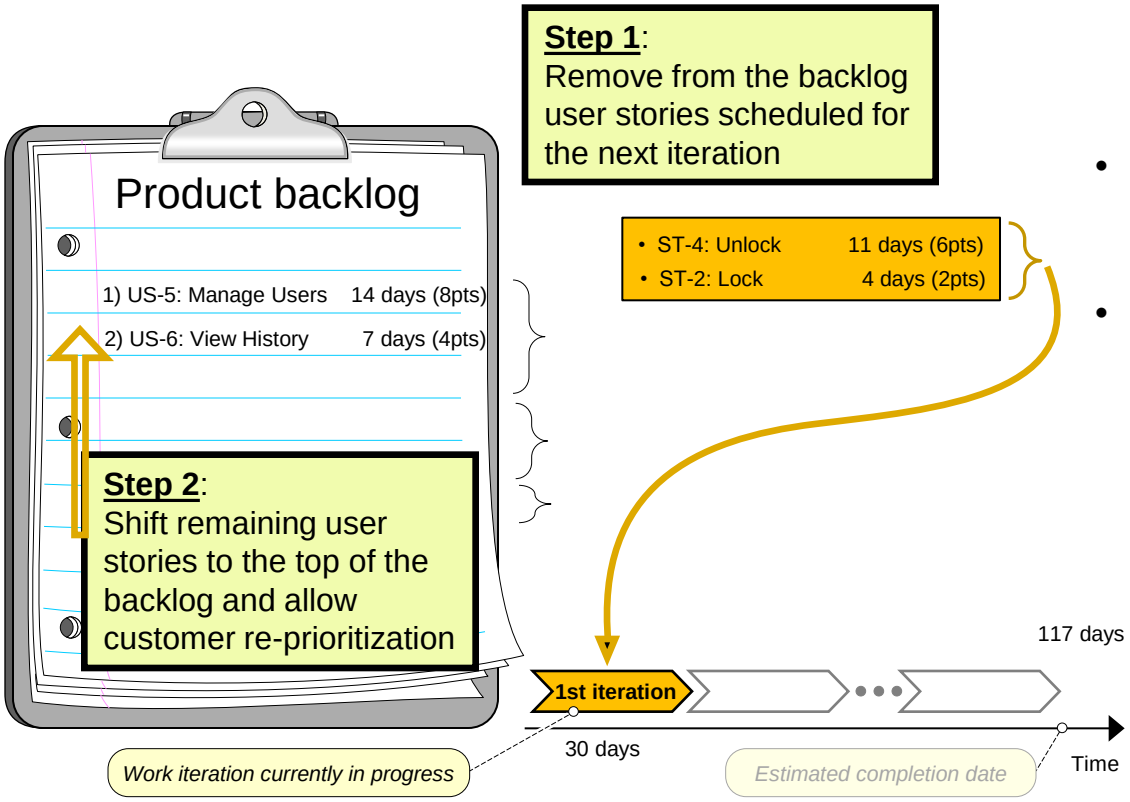
Total project effort is estimated based on the cumulative story points of all user-stories



Agile Prioritization of Work



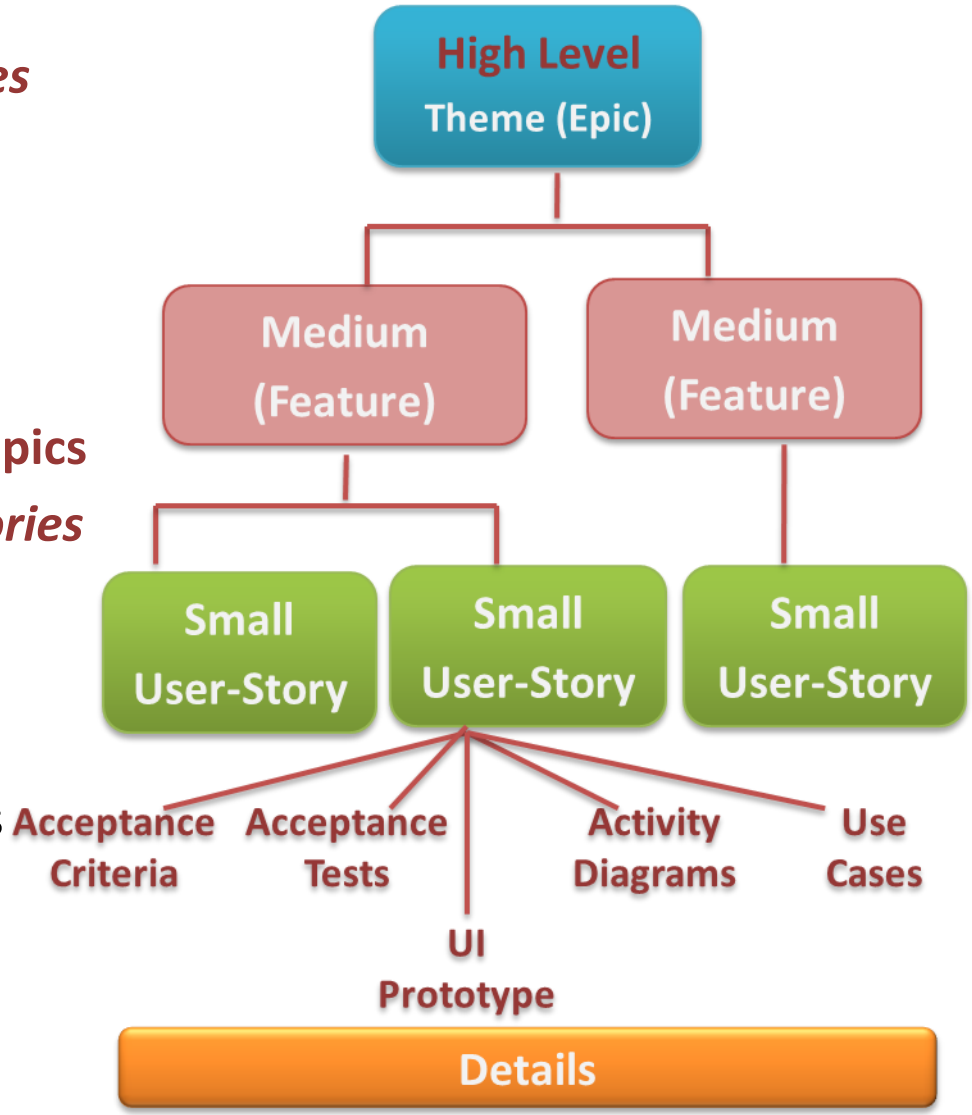
Update Product Backlog between iterations



- Items pulled by developers into an iteration are not subject to further customer prioritization
- Developers have a **steady goal** until the end of the current iteration
- Customer has **flexibility** to change priorities in response to changing market forces

Summarising Agile Requirements Engineering

- Start with ***visioning***
  - To identify the ***Theme*** or ***Epic Stories***
  - What are the key features?
  - Who are the target users
  - What are the selling points of this product? (3-5 selling objectives)
- ***Brainstorm*** to identify features of the **epics**
- ***Breakdown*** features into small ***user-stories***
- ***Detail*** user-stories to yield iteration deliverables
  - Acceptance criteria
  - Screen sketches, Use case diagrams
  - UI Prototypes, Wire Frames (UI focussed projects)
  - Activity Diagrams (Process centred projects)
- A final list of user-stories or ***product backlog*** of user-stories is created



Key points: Agile Requirement Engineering

- Design upfront a ***process*** for collaborative requirements gathering
- Identify and engage a ***product owner*** and knowledgeable subject ***SMEs***
- Focus on ***breadth early, on depth later***
- Break down/slice epic stories to the right level, so the team has clarity on the requirement, but keep a ***'Just Enough For the Next Step'*** attitude

Exercise: User-Stories for an event management system

Problem Statement: A training centre, wishes to build an online system where trainers trainer are provided with the ability to add a course and student are provided with the ability to register for the course online

Write a set of user-stories for above problem statement

Exercise: User-Stories for an event management system

Epic Story 1:

As a trainer, I would like to be able to add a new course, so that I have the potential to attract new students

Epic Story 2:

As an attendee, I would like to search for the different types of course-offerings and also register for a particular course-offering online, to save time

Ex: User-Stories for an event management system

Epic Story 1 is atomic enough, so does not need to be broken down into more finer user-story. It can now be written as a user-story along with acceptance criteria as:

US1: As a trainer, I'd like to be able to add a new course, so that I'll have the potential to attract new students.

Acceptance Criteria:

1. A trainer is displayed a form to add a new course by entering course title, abstract, dates and times, location and trainer name and trainer email address and clicks on an "add course" button to add the course successfully
2. If any fields are missing or dates or times are invalid, an error message, "…." will be displayed to the user
3. Save the added course to the database
4. Email the trainer upon successfully adding the new course
5. Once the course has been successfully added, display the list of courses (along with the new course) currently offered

Ex: User-Stories for an event management system

Epic Story 2 is large, could possibly be broken into three smaller user-stories (US2, US3, US4)

US2: As an attendee, I would like to able to search the course-offerings, so that I would be able to find an offering that most interests me

1. A course attendee is able to search for all a list of all current course offerings
2. If no courses currently running, a message is displayed to the attendee: “No course offered currently. Please check again later”.
3. The search result (if successful) will list all current course offerings. For each course offering, the date, time and location of the event is also displayed.
4. An attendee can click on any course from the display results to obtain an abstract about the course

Ex: User-Stories for an event management system

US3: As an attendee, I'd like to be able to search for a particular course offering by title, so that I'll be able to find to quickly locate an offering that I am interested in.

Acceptance Criteria:

1. Able to search for a particular course offering by title or just specifying part of the course title in a search field
2. If the "search by title" does not correspond to a valid course offering, then a message is displayed to the user as: "No such course is being offered currently. Please check again later" message will be displayed on the webpage.
3. The search if successful, will list all the matching results. For each course offering, the date, time and location of the event is also displayed.
4. An attendee can click on any course from the display results to obtain an abstract about the course. The abstract should display the course title, description of the course, duration, convenor, date, venue and cost

Ex: User-Stories for an event management system

US4: As an attendee, I want to be able to register online, so that I can register quickly and cut down on paperwork.

Acceptance Criteria:

1. An attendee can select any course from the displayed results and register online by providing the name of the attendee, a valid email address, a phone number through an online form and proceed to the payment options
2. Payment for a course can be made via credit-card
3. Payment for a course can be made via pay pal.
4. Upon successful payment, data about the registered attendee is stored in the database.
5. Upon successful payment, a confirmation email is sent to the attendee.

Checkpoint !

Quiz - User Stories

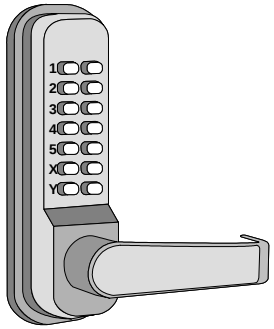
Use-Case Modelling

Use Case Modelling

- Use Case Modelling is building a set of use-cases that describe the tasks to be performed by the “system-to-be” and the relation between these tasks and the outside world
- A use-case description represents a dialog between the user and the system, with the aim of helping the user achieve a goal
- Use cases signify what the system needs to accomplish, not how;

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

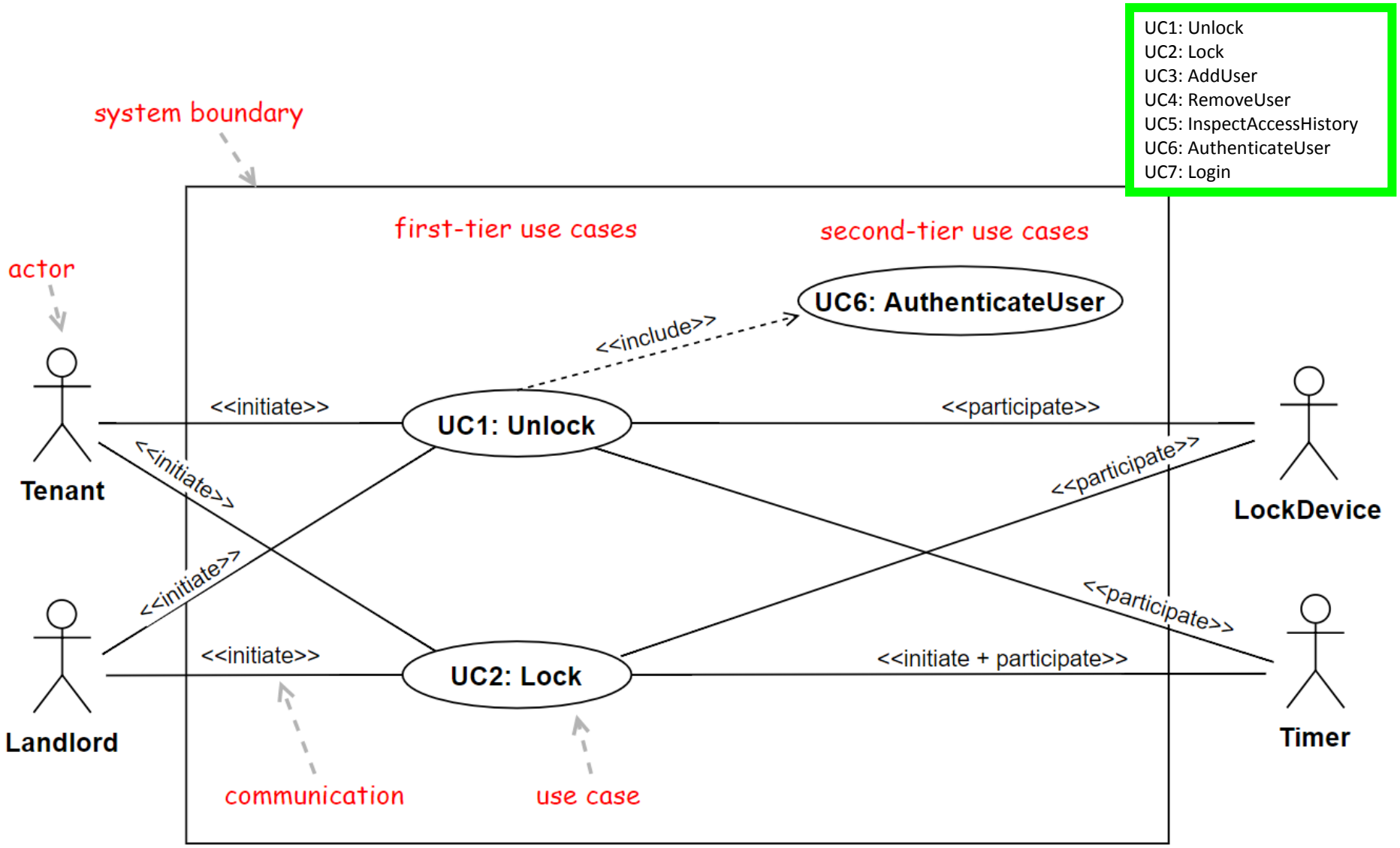


Actor	Actor's Goal (what the actor intends to accomplish)	Use Case Name
Landlord	To disarm the lock and enter	**Unlock (UC-1)**
Landlord	To lock the door	**Lock (UC-2)**
Landlord	To create a new user account and allow access to home	**AddUser (UC-3)**
Landlord	To retire an existing user account and disable 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	**Unlock (UC-1)**
Tenant	To lock the door	**Lock (UC-2)**
LockDevice	To control the physical lock mechanism	**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)**

Use Case

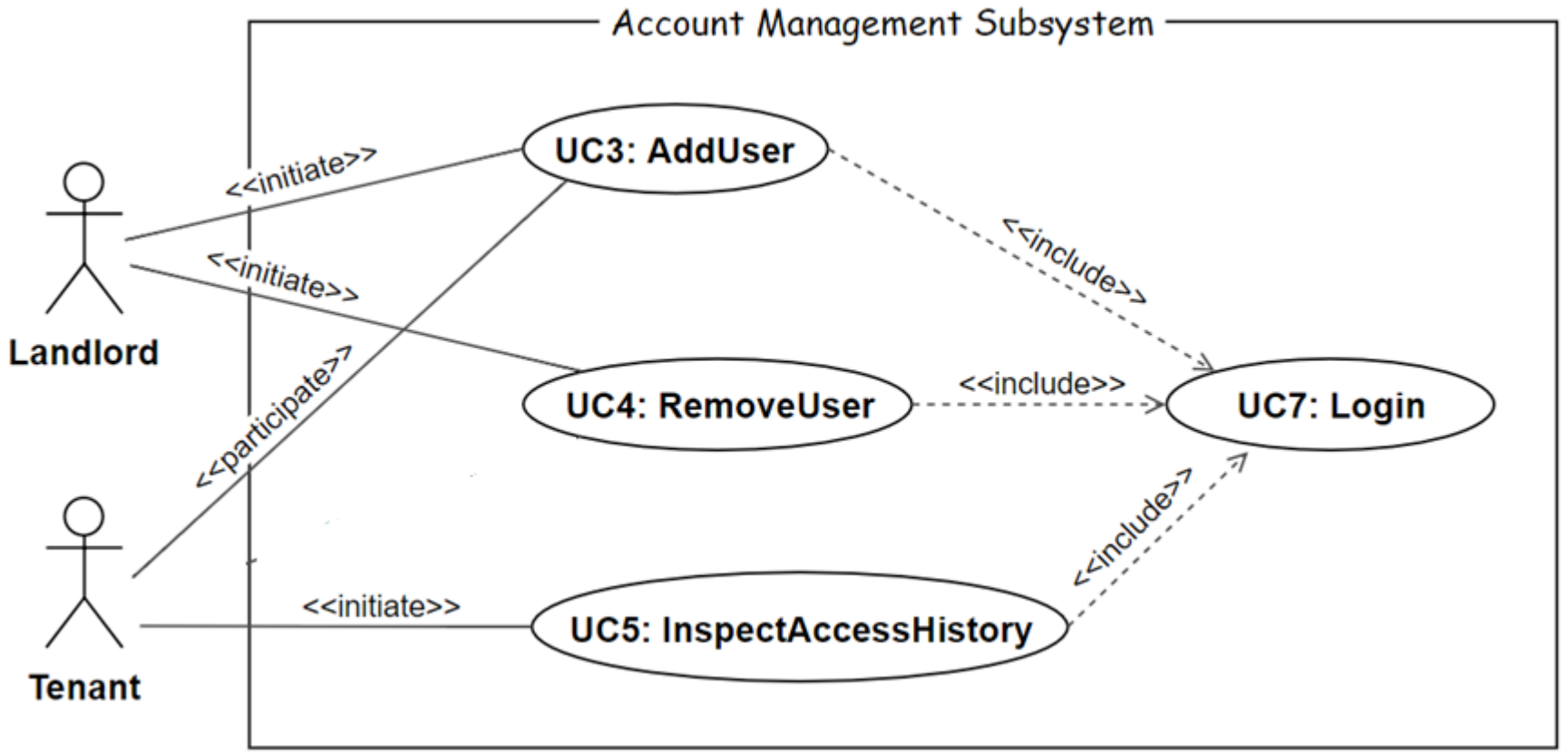
- As use cases helps a user to achieve goals, each use-case name must include a "verb" capturing the goal achievement e.g., "withdraw cash"
- Each use-case description represents a dialog, where the user initiates actions and the system responds with reactions
- Each use-case specifies what information must pass the boundary of the system in the course of a dialog (without considering what happens inside the system)

Use Case Diagram: Device Control



Use Case Diagram: Account Management

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



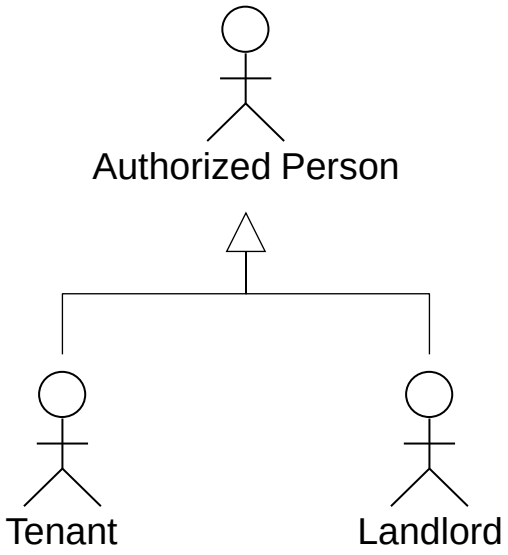
Types of Actors

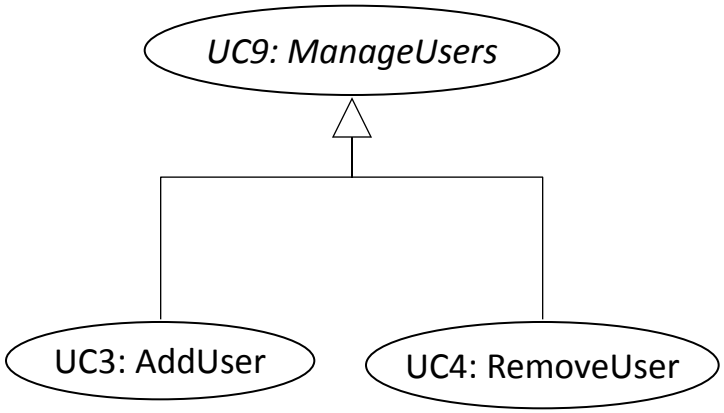
- **Initiating actor** (also called *primary actor* or simply “user”): initiates the use case to achieve a goal
- **Participating actor** (also called *secondary actor*): participates in the use case but does not initiate it.
  - helps the system-to-be to complete the use case

Use Case Generalizations

- More abstract representations can be derived from particular representations

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





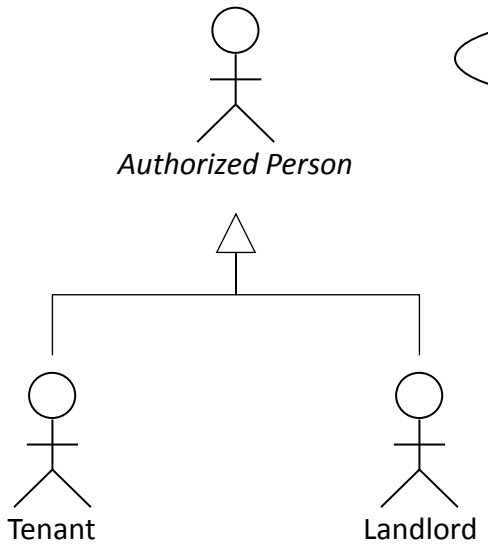
Actor Generalization

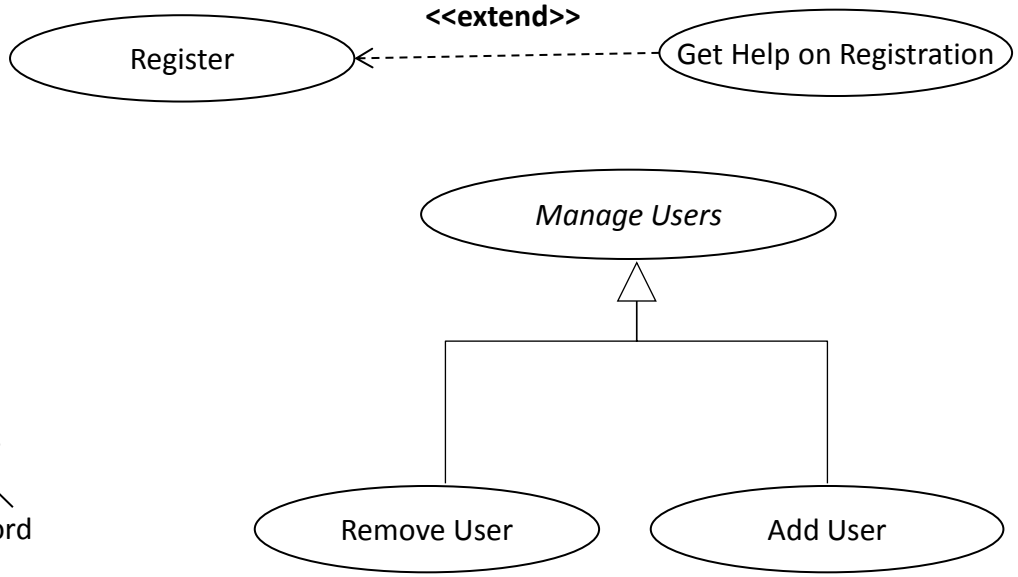
Use Case Generalization

Use Case Generalizations

- More abstract representations can be derived from particular representations

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



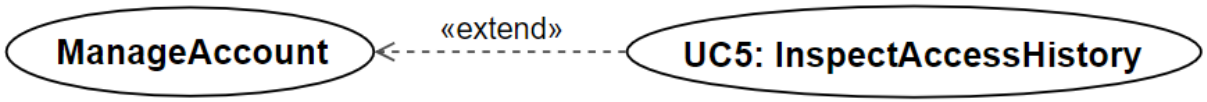


Actor Generalization

Use Case Generalization

Optional Use Cases: «extend»

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



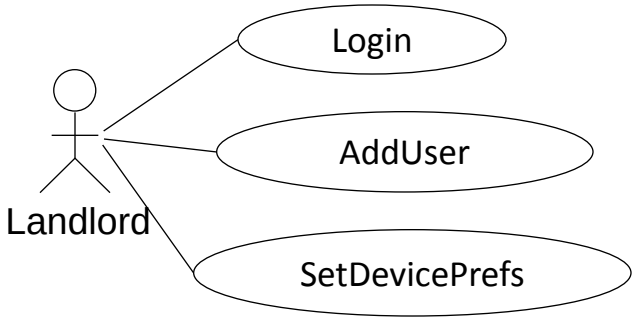
Key differences between **«include»** and **«extend»** relationships

	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

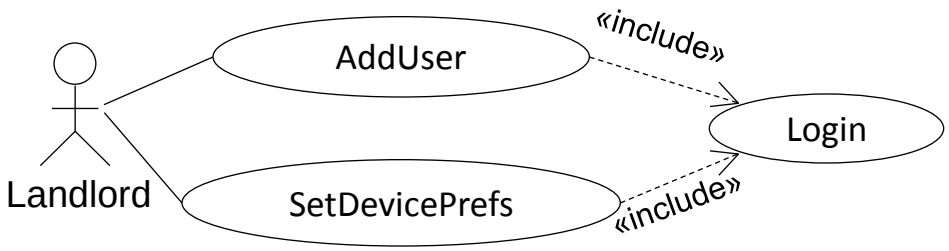
[Source: Robert Maksimchuk & Eric Naiburg: ***UML for Mere Mortals***, Addison-Wesley, 2005.]

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: AuthenticateUser
UC7: Login

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

Usage Scenario: 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, 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
←	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
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-6 Authenticate User

Test-case Identifier:	TC-1
Use Case Tested:	UC-1, main success scenario, and UC-6
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

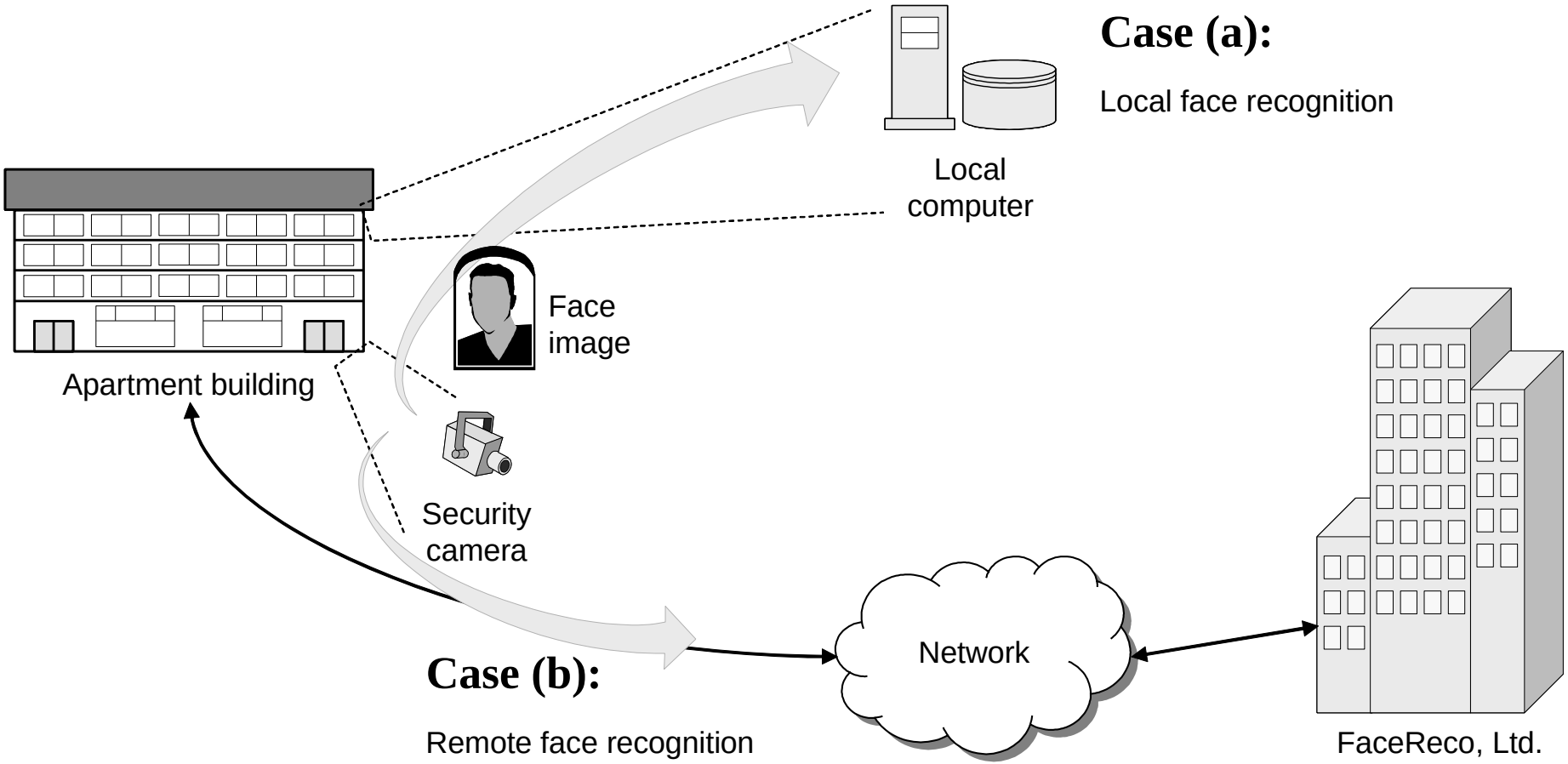
← 1. System (a) signals a warning that the door is open, and (b) signal to Timer to start the alarm counter

→ 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

System Boundary & Subsystems

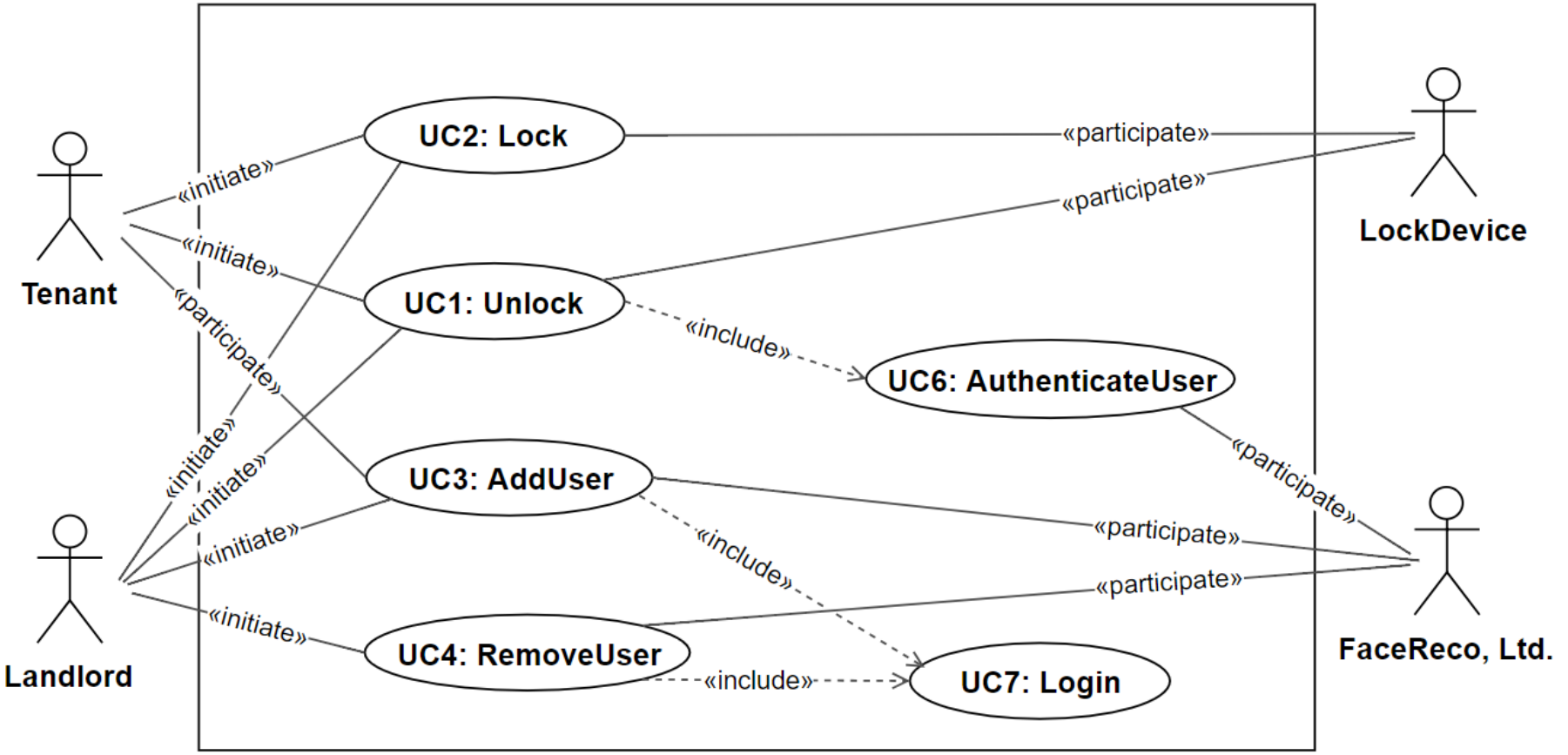
Use Case Variations Example when external authentication mechanisms are used:



Modified Use Case Diagram

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

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



Checkpoint !

Quiz – Coming up soon…

Next week …

- Art of Writing User-Stories
- Domain-modelling using OO design