UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SOFTWARE ENGINEERING – LECTURE 05

Requirements Engineering: Fully-dressed Use Cases

Prof. Sergio Di Martino

sergio.dimartino@unina.it

https://www.docenti.unina.it/sergio.dimartino

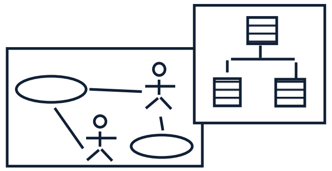
The Software Life Cycle

Requirements Engineering → System Design → Software and UI/UX Design → Implemen-tation → Testing → Operation and Maintenance

Requirements collected via:

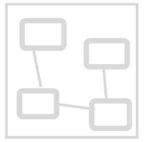
- Interviews with Stakeholders
- Personas
- Stories and Scenarios

Specified using:
- Use Cases
- Natural Language
- Domain Models
- Mock-ups



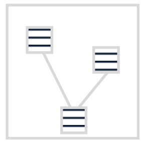
Define System Architecture

- Requirements are allocated to software sub-systems
- Sub-systems are allocated to hardware resources
- Architectural Patterns



Define Subsystems

- Objects required to realize each subsystem are defined.
- Software Design Patterns
- Usability Engineering
- High-fidelity Wireframing

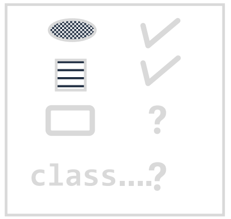


Each Subsystem is implemented

- Source code and other artifacts
- Clean Code
- Frameworks and ORMs
- Focus on Software Quality

Ensure the Software satisfies customers

- Code inspections
- Functional Testing (unit, integration, system testing)
- Usability Testing

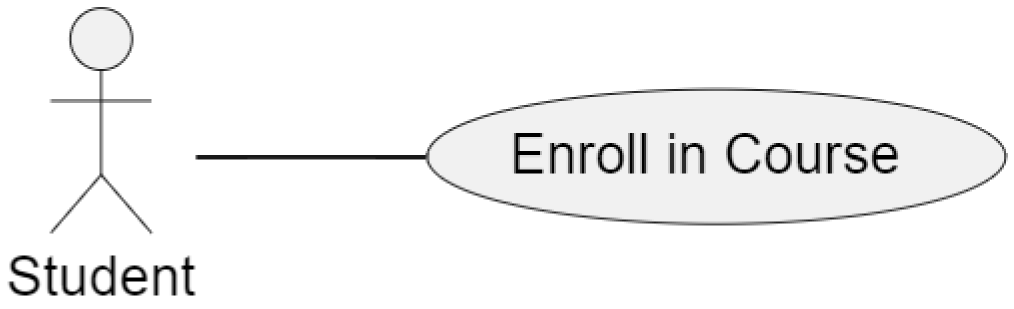


System is put into practical use

Maintance will be required at some point
- To fix errors that were not discovered in previous phases
- To adapt the software to changes in requirements on in its environment

Previously, on Software Engineering

- We’ve discussed Requirement Specification phase in detail
- We’ve seen that different approaches exist, ranging from unstructured natural language to formal specifications
  - We’ve seen how these different approaches are typically applied to different kinds of software systems
- We’ve seen in detail one of such approaches: Use Case Diagrams (UCDs)



Specifying Use Cases

- The UCD provides a very high-level overview of the functional requirements of the systems. It is not detailed enough to establish system requirements
- For **each** UC in the UCD a detailed specification is needed
- The goal is to specify every aspect and detail of the interaction, from the Actor's point of view.
  - Each possible scenario and variation should be described

Text Descriptions of a Use Case

- A use case description generally include:
  1. A description of what the system and users expect when the use case begins
  2. A description of the normal flow of events in the Use Case (**main scenario**)
  3. A description of what can cause errors and how the resulting problems can be handled
  4. A description of the state of the system after the Use Case is complete.

Use Case Formats

Use cases can be written in different formats and levels of formality:

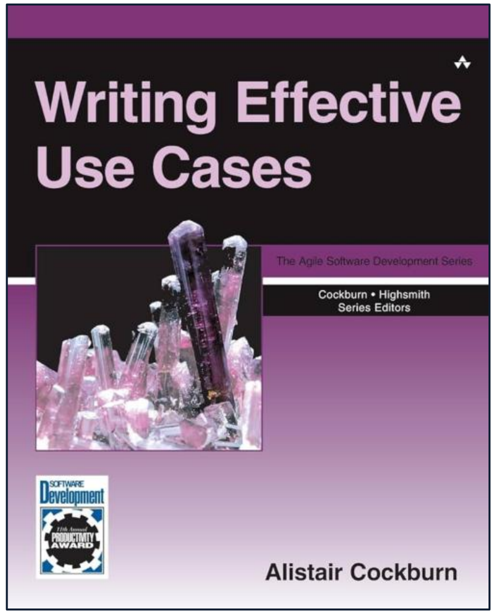
- **Brief**: Terse one-paragraph summary, usually of the main success scenario.
- **Casual**: Informal paragraph format. Multiple paragraphs that cover various scenarios.
- **Fully-dressed description**: All steps and variations are written in detail, and there are supporting sections, such as preconditions and success guarantees.

Use Case Formats

- **Brief** and **Casual** descriptions can be used in the early stages of requirements specification, to get a quick sense of subject and scope
- **Fully-dressed** descriptions may be developed later on, to serve as a basis for a contract and specify in greater detail the behaviour of the system to be developed

Fully-dressed Use Case Descriptions

- Different formats for fully-dressed use case descriptions have been proposed
- We'll see a template based on the one proposed by **Alistair Cockburn**


Writing Effective Use Cases
The Agile Software Development Series
Cockburn • Highsmith Series Editors
Software Development Productivity Award
Alistair Cockburn

Cockburn’s Template

USE CASE #X	**Name of the Use Case**			
Goal in Context	Description of the objective of this UC			
Preconditions	All the conditions that must apply to start the UC			
Success End Condition	State of the system if the UC was successful			
Failed End Condition	State of the system if the UC failed			
Primary Actor	Primary actor of the UC			
Trigger	Action of the primary actor that initiates the UC			
Main Scenario	**Step n.**	**Actor 1**	**Actor n**	**System**
	1	Trigger action		
	2			Response
	..	Action 2		
	..		…	…
	n			Final action

Cockburn's Template (cont.)

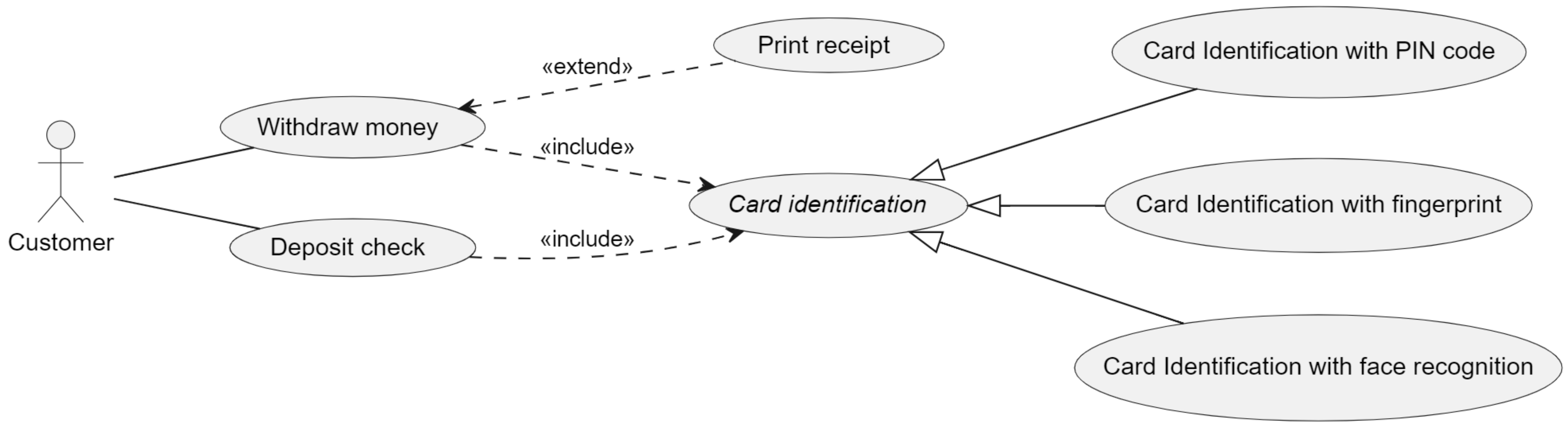
Extension #1 (short description)	**Step**	**Actor 1**	**Actor n**	**System**
	x <condition>	**…**	**…**	**…**
	…	**..**	**…**	**…**
	…			Final action (possibly return to a step of the main scenario)
Extension #n (short description)	**Step**	**Actor 1**	**Actor n**	**System**
	y <condition>	**…**	**…**	**…**
	…	**..**	**…**	**…**
	…			Final action
Open Issues	List all the aspects that still need to be clarified. At the delivery of the doc must be empty			

Main Scenario And Extensions

- The main scenario is the sequence of actions that occurs when all in the use case goes smooth as intended
- However, there may be different ways to perform an use case
  - Users can authenticate themselves by using the PIN or a fingerprint scanner
  - An error might occur at some point
- When defining the functional behaviour of the system, it is important to describe also these alternative sequences of actions that can happen when performing a use case
  - This is done using **Extensions**
  - Typically, there’s way more text in the Extensions rather than in the Main Scenario

Example

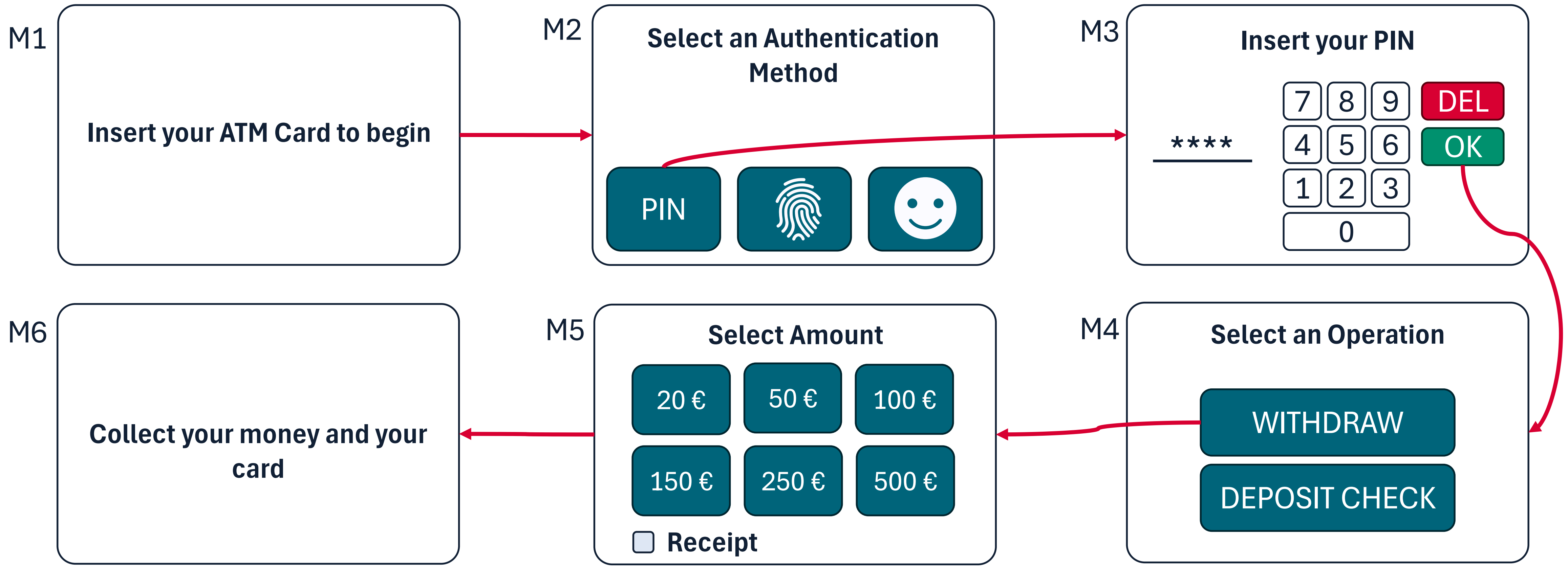
Example: ATM System


Print receipt
Card Identification with PIN code
«extend»
Withdraw money
«include»
Customer
Card identification
Card Identification with fingerprint
Deposit check
«include»
Card Identification with face recognition

- Suppose we want to describe the **Withdraw money** use case, using a fully-dressed format

Example: Mockups

- It may be useful to design some mockups of the system



Example: Fully-dressed Use Case

USE CASE #1	Withdraw money		
Goal in Context	A customer wants to withdraw money from the ATM		
Preconditions	The customer has an account at the Bank and owns a bank card		
Success End Condition	The system keeps track of the withdrawal operation and erogates the requested money		
Failed End Condition	No transaction is made		
Primary Actor	Customer		
Trigger	Customer walks up to the system and touches the screen to activate it		
Main Scenario	**Step n.**	**Customer**	**System**
	1	Touches screen	
	2		Shows M1
	3	Inserts card	
	4		Shows M2

Example: Fully-dressed Use Case

USE CASE #1	**Withdraw money**		
Main Scenario	**Step n.**	**Customer**	**System**
	5	Touches «PIN» button	
	6		Shows M3
	7	Inserts PIN	
	8		Shows M4
	9	Touches «Withdraw» button	
	10		Shows M5
	11	Touches «50 €» button	
	12		Erogates money, Ejects card, Shows M6

Example: Extensions

- What can go wrong?
    - PIN might not be correct
    - Customer might not have enough money in their account
    - ATM might not have enough cash reserves to erogate the required money
    - Card might be flagged as stolen
    - Card might be unreadable
    - …
- What could go differently?
    - Customers might authenticate themselves using their fingerprint or face recognition
    - Customers might opt-in to get the printed receipt

Example: Extensions

- Each of these scenarios should be detailed using extensions

Extension #1 (customer inserts an invalid PIN)	**Step**	**Customer**	**System**
	7a <wrong PIN is inserted>	Inserts PIN	
	8a		Shows M7 and terminates UC

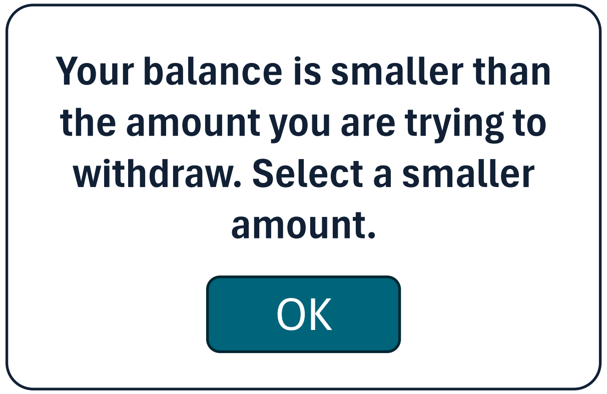
M7

Invalid PIN – Authentication denied.
Recover your card from the tray.

Example: Extensions

Extension #2 (customer does not have enough money)	**Step**	**Customer**	**System**
	11b	Selects «500€» button	
	12b		Shows M8
	13b	Clicks ok	
	14b		Return to step 10 of the Main Scenario

M8



Your balance is smaller than the amount you are trying to withdraw. Select a smaller amount.

Requirements Validation

Requirements Validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
    - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirements Checking

- Validity. Does the system provide the functions which best support the customer’s needs?
- Consistency. Are there any requirements conflicts?
- Completeness. Are all functions required by the customer included?
- Realism. Can the requirements be implemented given available budget and technology
- Verifiability. Can the requirements be checked?

Requirements Validation Techniques

- Requirements reviews
  - Systematic manual analysis of the requirements.
- Prototyping
  - Using a simplified executable model of the system to check requirements.
  - Visual prototyping (i.e., using mockups / wireframes)
- Test-case generation
  - Developing tests for requirements to check testability.

Requirements Reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

Review Checks

- **Verifiability**
  - Is the requirement realistically testable?
- **Comprehensibility**
  - Is the requirement properly understood?
- **Traceability**
  - Is the origin of the requirement clearly stated?
- **Adaptability**
  - Can the requirement be changed without a large impact on other requirements?

Readings and References

- A. Cockburn, *Writing effective use cases*. Pearson, 2008.