Lecture 3

Use cases

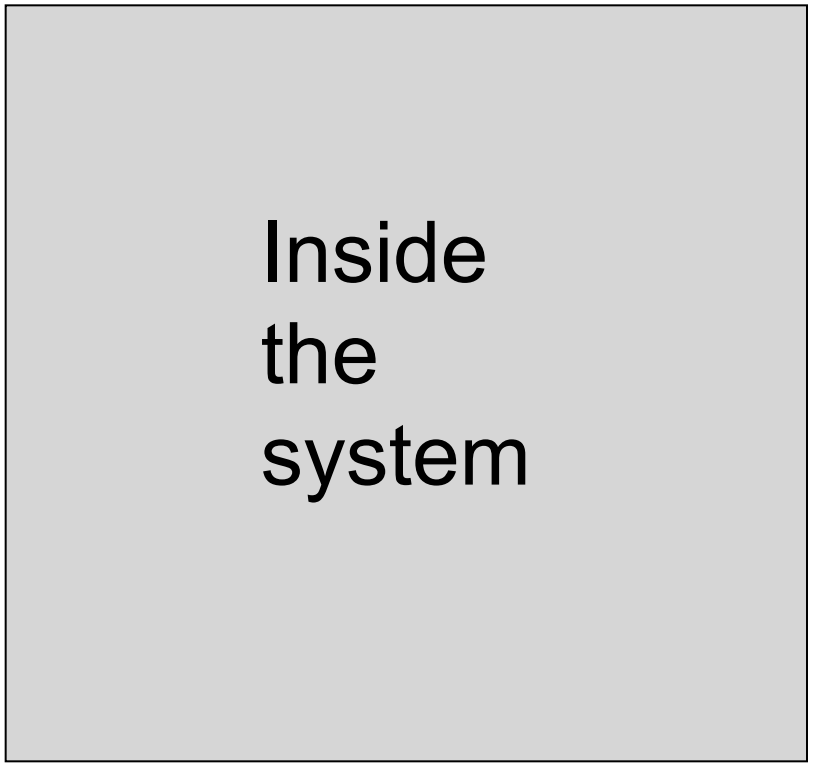
Rogardt Heldal

Objectives

- To be able to
  - Understand the difference between “inside” and “outside” of a system
  - describe the behaviour of a system using *use cases*
  - write use cases

System

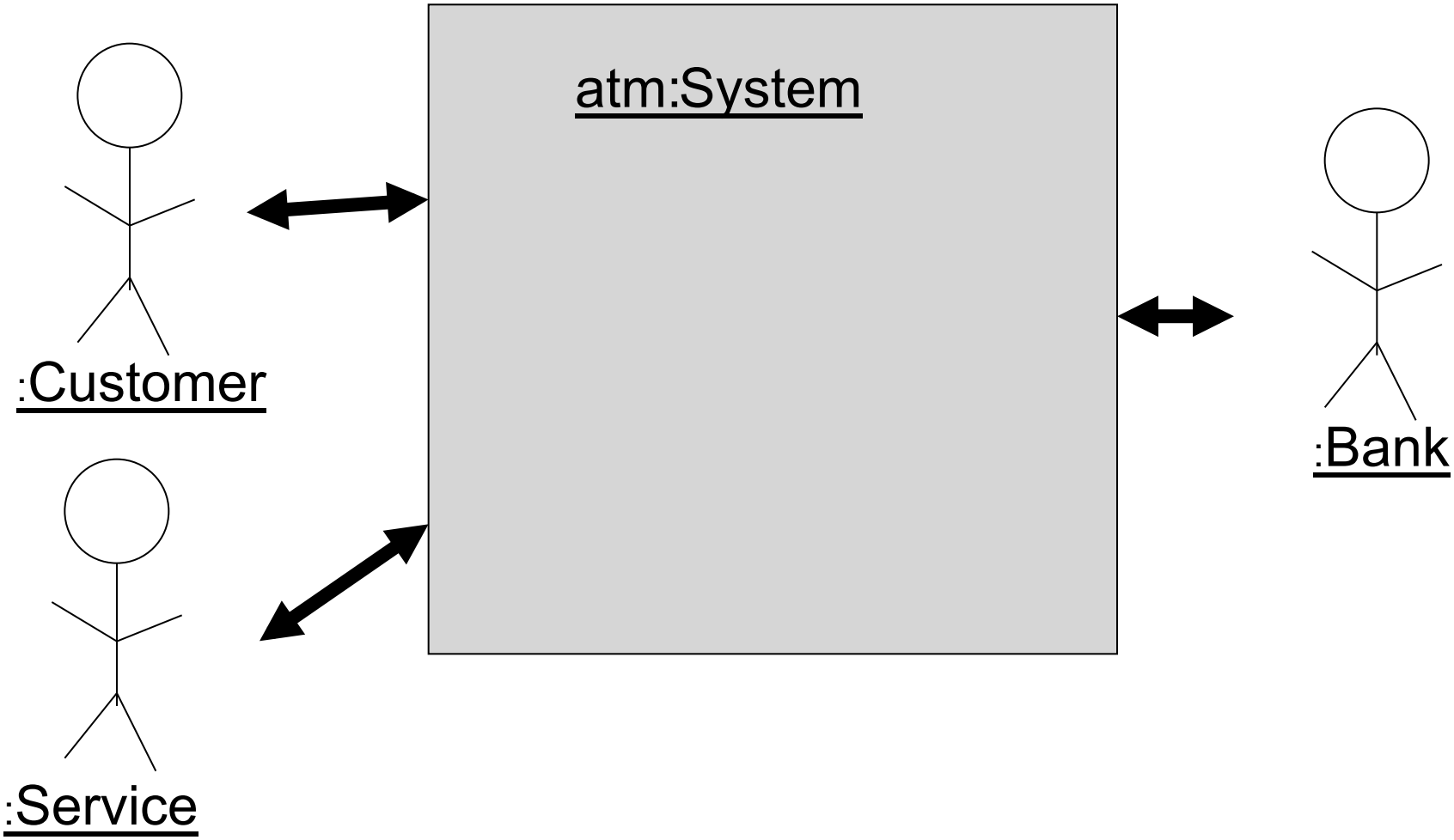
Outside the system



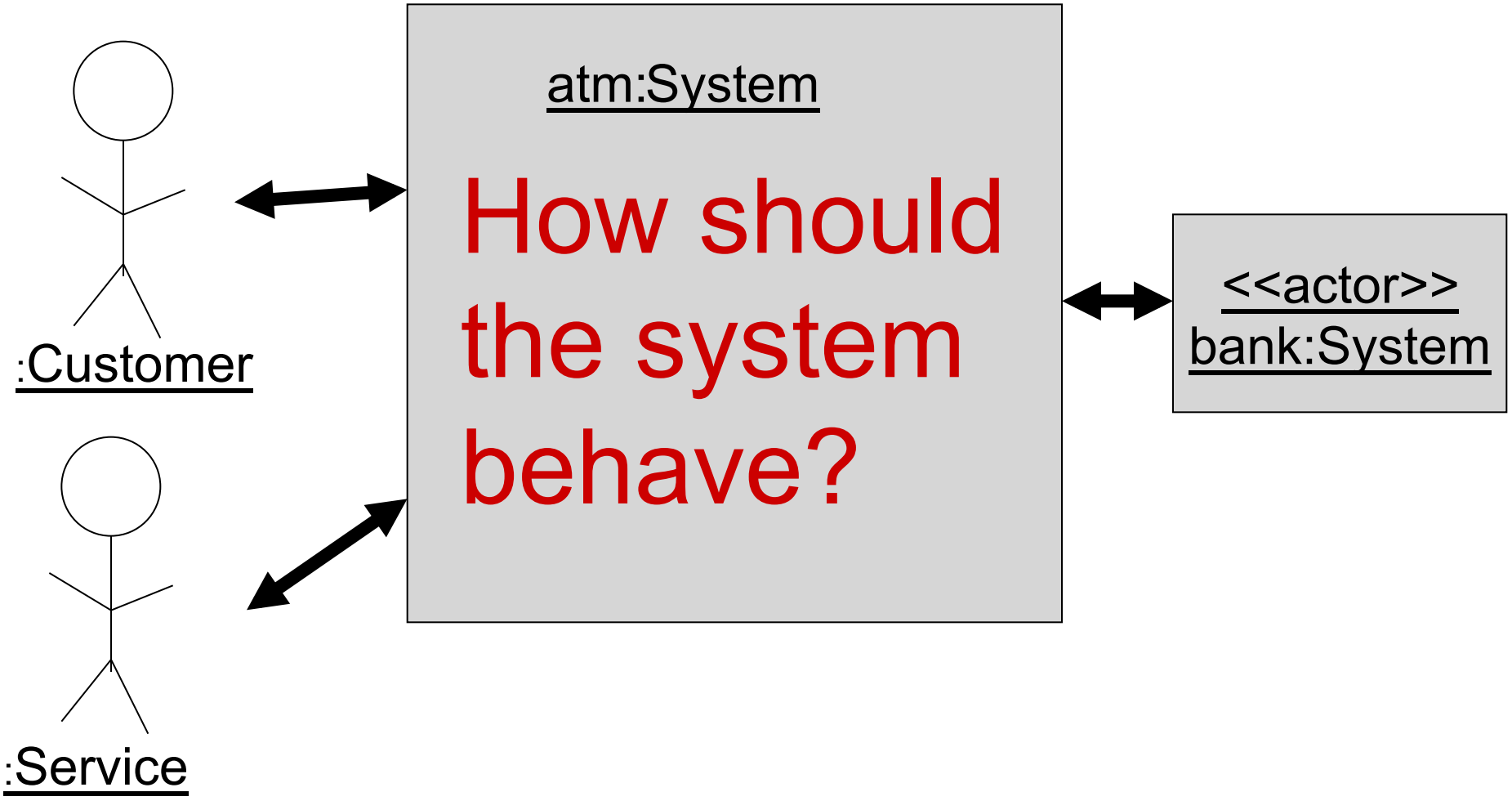
Outside the system

- Find the one which interact with the system:
  - Actors

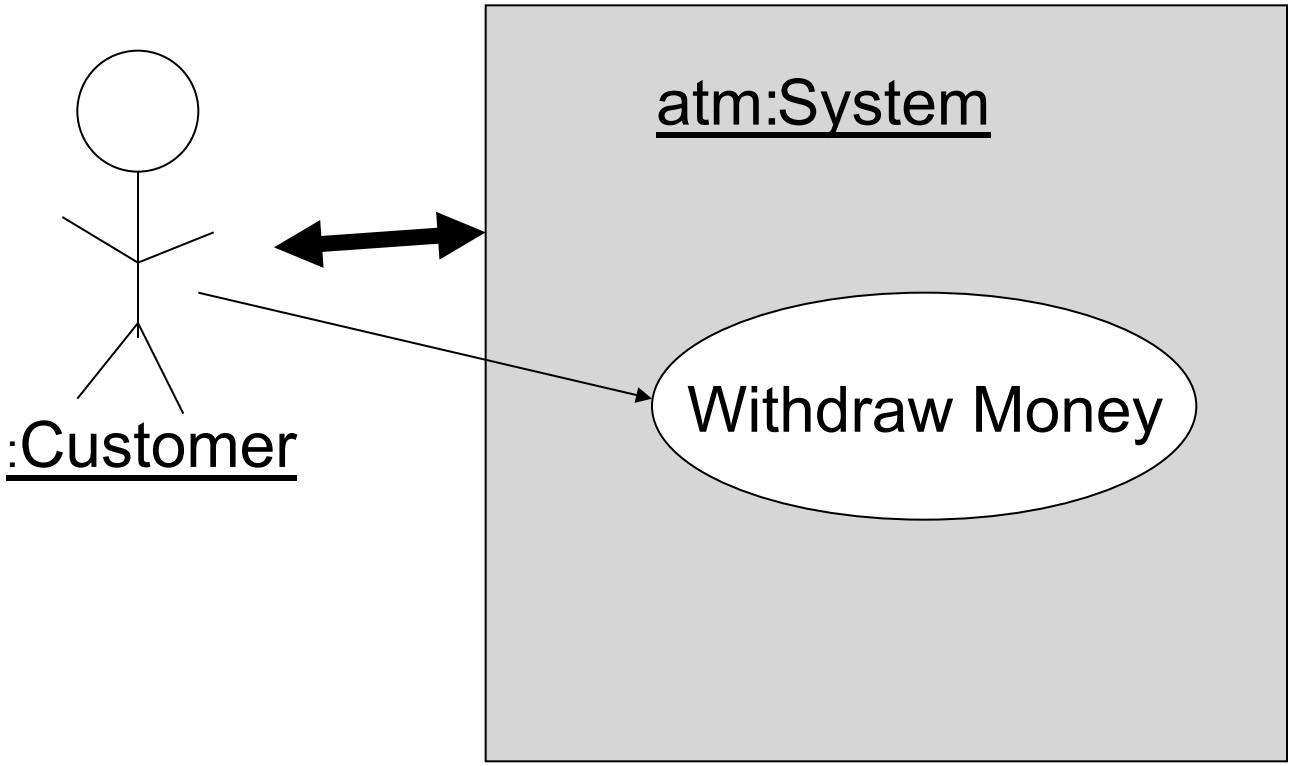
ATM



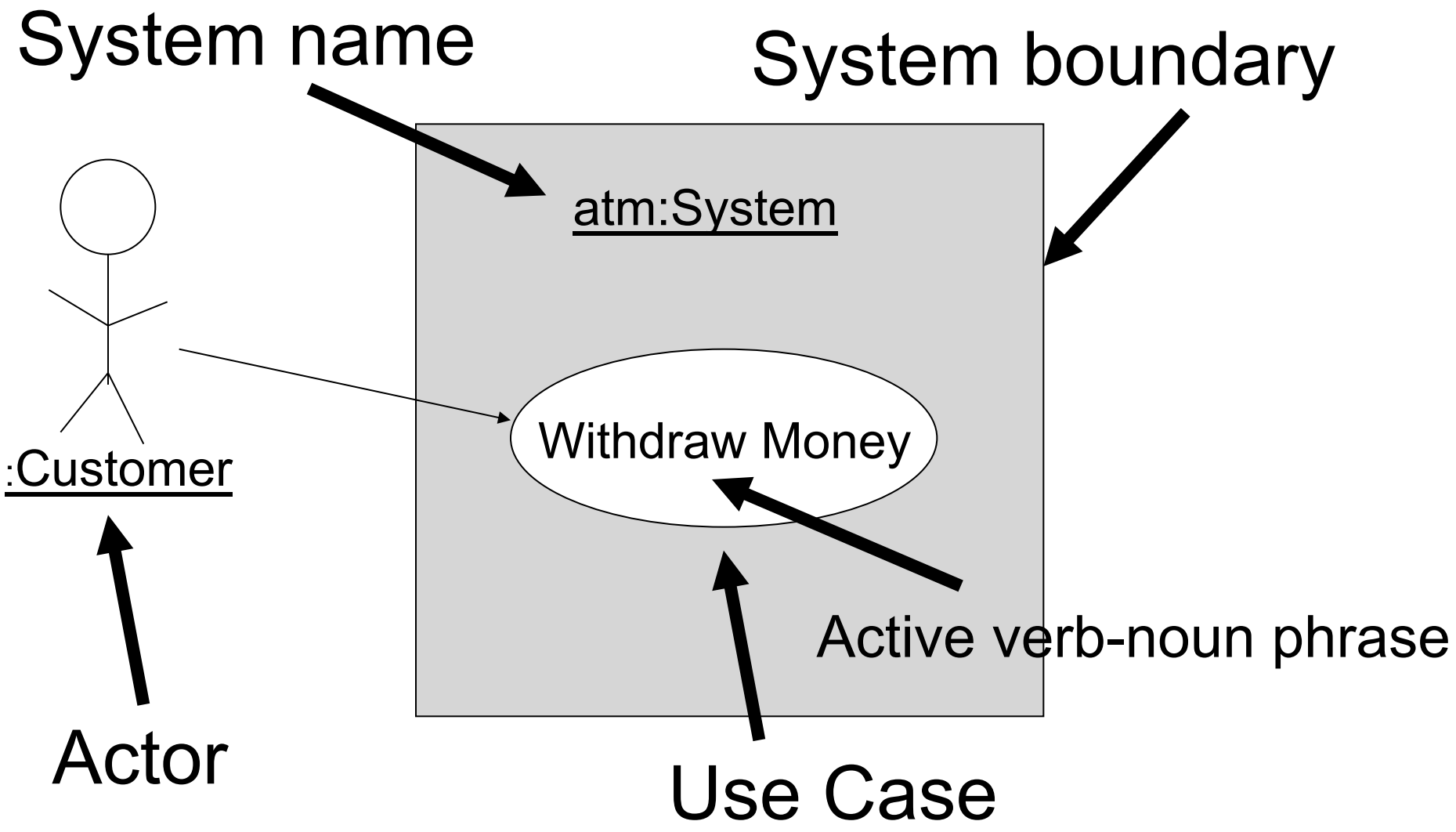
ATM


atm:System
How should the system behave?
:Customer
:Service
<<actor>>
bank:System

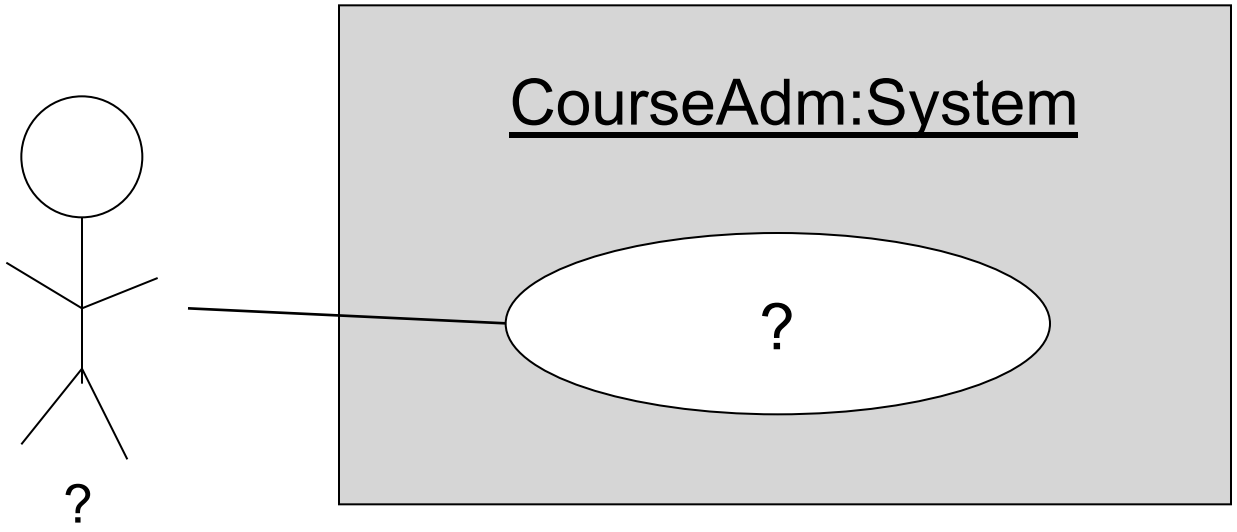
ATM



Use Case Diagram



Problem: Use Case Diagram



1. Find actors for this domain
2. Find use cases for this domain

Brief Use Cases

- A short description of the use case, for example:
  - Name: Withdraw Money
  - Actor: Customer
  - Goal: Take out money from an account
  - Description: The customer identifies himself and requests an amount of money. The ATM gives out money if the customer has sufficient funds in his account.

Problem: Brief Use Cases

- Write a brief use case for “register for course”.
  - Use Case Name: ?
  - Actor Name: ?
  - Goal: ?
  - Description: ?

Complete Use Case

- Template
  - Use Case Name
  - Use Case Goal
  - Actor Names
  - Main Flow of Event
  - Alternative Flows
  - Pre-Condition (if any)
  - Post-Condition
- Different organisations have different templates, but all things in this template should be part of any use case template.

Example: Flow of Events

Main flow of withdraw money:

1. user identifies himself by a card
2. system reads the bank ID and account number from card and validates them
3. user authenticates by PIN
4. system validates that PIN is correct
5. user requests withdrawal of an amount of money
6. system checks that the account balance is high enough
7. system subtracts the requested amount of money from account balance
8. system returns card and dispenses cash

Problem

- Write the main flow of events for the complete Use Case “register to course”.

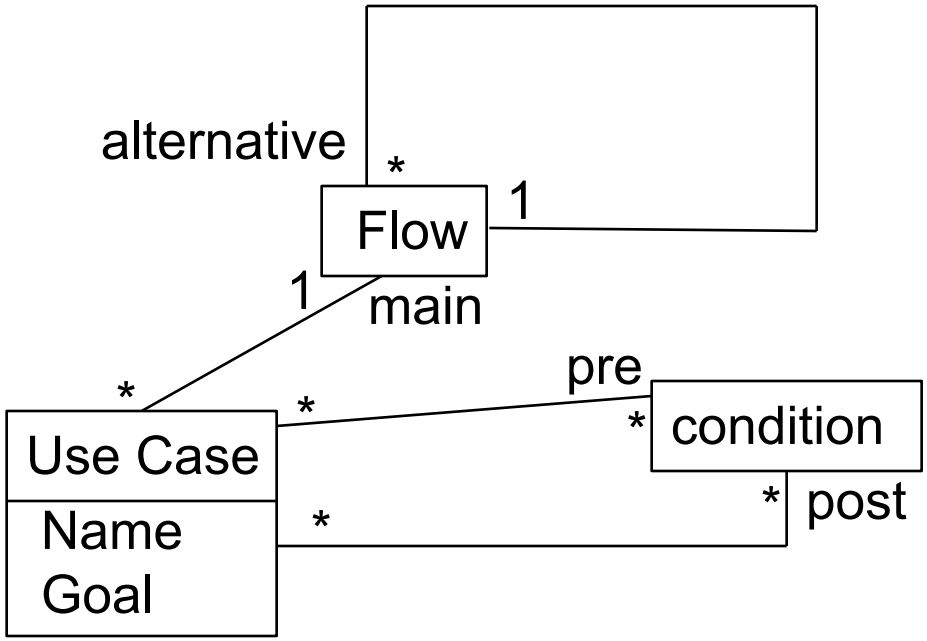
Alternative flows

- Most use cases do not have just one flow, but several alternative flows.
  - Another frequent behaviour of the system
  - Another possible behaviour of the system
  - An error case
  - …
- The alternative ways depend on the input given by the actor, the system state, iterations … .

Complete Use Case

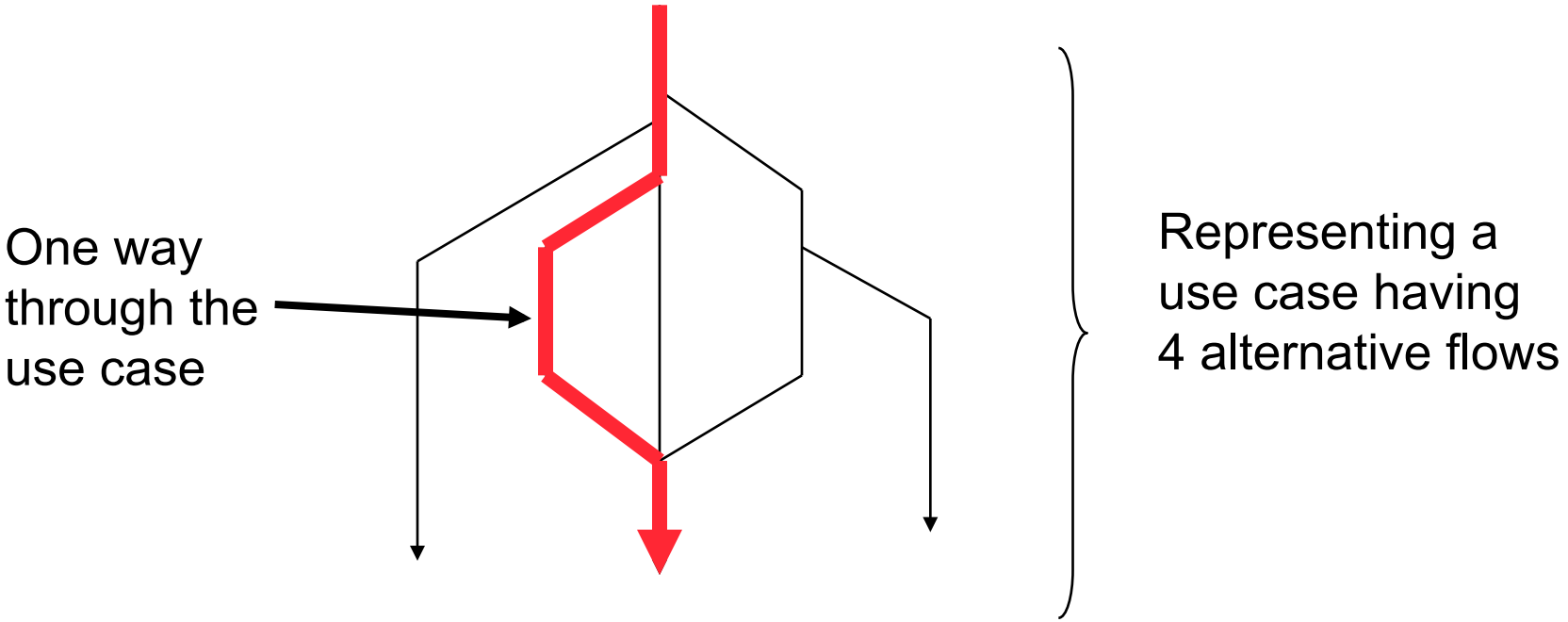
- Template
  - Use Case Name
  - Use Case Goal
  - Actor Names
  - Main Flow of Event (sequence of action steps)
  - Alternative Flows (sequence of action steps)
  - Pre-Condition (if any)
  - Post-Condition

Domain Model (Meta-Model)



Scenarios

- One way through the use case.



One Flow of Events

Main flow of withdraw money:

1. user identifies himself by a card
2. system reads the bank ID and account number from card and validates them
3. user authenticates by PIN
4. system validates that PIN is correct
5. user requests withdrawal of an amount of money
6. system checks that the account balance is high enough
7. system subtracts the requested amount of money from account balance
8. system returns card and dispenses cash

Example: Alternative Flow

Fragment of the use case “Withdraw Money”

…

7. User requests withdrawal of an amount of money
8. System checks that the account balance is high enough
9. System subtracts from account the amount taken out from the ATM
10. System gives back card and dispenses cash

8-10a: Not enough money on account:

1. System does not change the account
2. System returns card

Numbering of Alternative Flows

1. …
2. …
3. …
4. …

3a – instead of point 3 in the main flow do this …

3b - instead of point 3 in the main flow do this …

2-3a – instead of point 2 to 3 in the main flow do this …

2-4a – instead of point 2 to 4 in the main flow do this …

2-4b – instead of point 2 to 4 in the main flow do this …

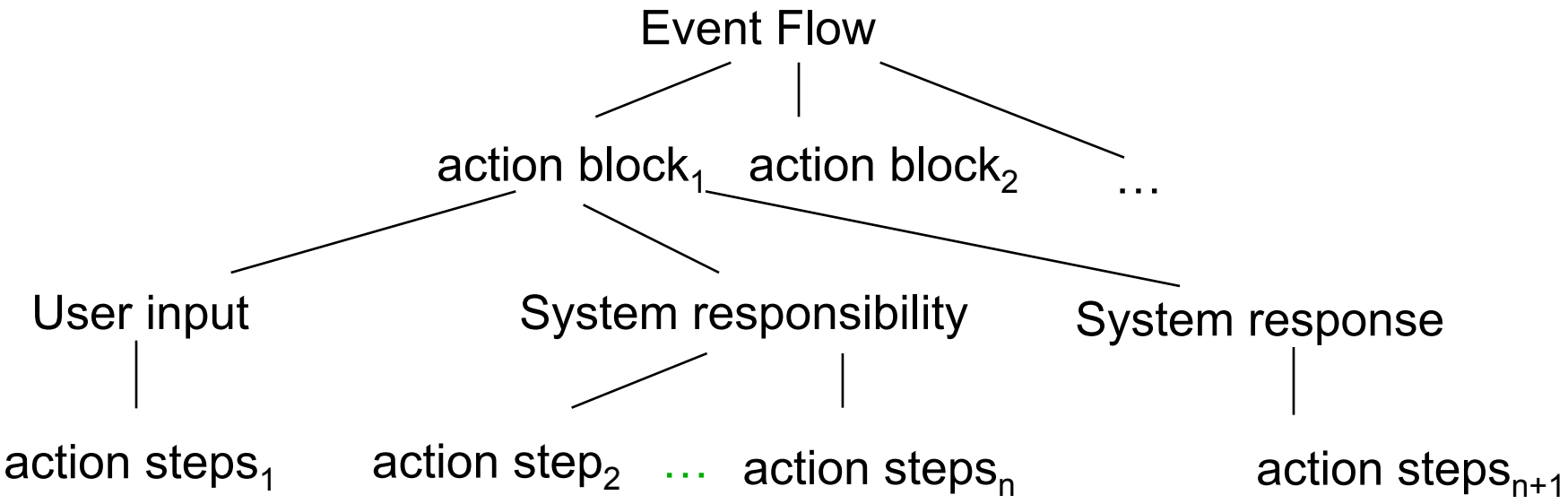
Problem

- In the case of a ATM, a user is permitted to give the wrong pin 3 times.
- If pin is given wrongly 3 times the card is kept.
- Write alternative ways for withdraw money which take this into account. Do not use WHILE loops!

Solution

- 4a. Wrong pin less than 3 times:
  - 1. System updates number of tries
  - 2. start from action step 3
- 4-8a. Wrong pin 3 times:
  - 1. System keeps the card

Action Block parts



Then one can have “assume” action steps when needed and a “choice” steps after return if necessary.

Withdraw Money

Only main flow:

Action block

1. user identifies himself by a card
2. Assume: bank ID and account number are the same

Action block

3. user authenticates by PIN
4. Assume: PIN is correct

Action block

5. user requests withdrawal of an amount of money
6. Assume: that the account balance is high enough
7. system subtracts the requested amount of money from account balance
8. Assume: that the ATM can communicate with the bank
9. system returns card and dispenses cash

User input

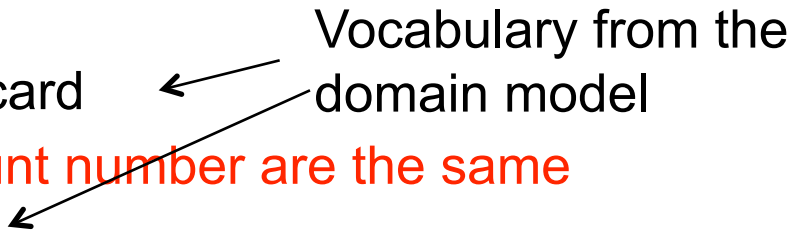
System responsibility

System response

Withdraw Money

Only main flow:

1. user identifies himself by a card
2. Assume: bank ID and account number are the same
3. user authenticates by PIN
4. Assume: PIN is correct
5. user requests withdrawal of an amount of money
6. Assume: that the account balance is high enough
7. system subtracts the requested amount of money from account balance
8. Assume: that the ATM can communicate with the bank
9. system returns card and dispenses cash



Assume and Choice

- Only the places where one have an assume or a choice can one an alternative way.
- For example if we have this action step in the main flow:
  - Assume: PIN is correct
- Then we can expect an alternative flow
  - Assume: PIN wrong three times

Withdraw Money (Choice)

Only main flow:

1. …
2. …
3. …
4. System return a price of the bike
5. Choice: customer want to buy the bike
6. Customer give the name, home address, card info

Action Block

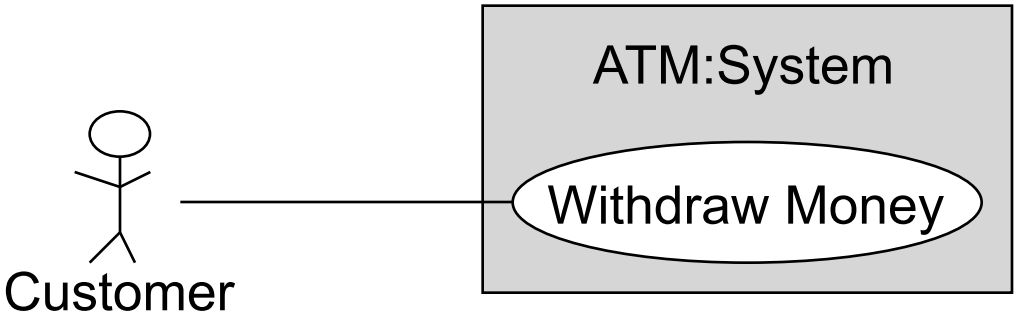
- A pattern for writing the event flows of use cases.

Objective:

- Write structured and informative use cases

When to use the Pattern

- Complete use cases
  - Contain all the important parts, in particular main and alternative flows.
- Essential style
  - Does not contain implementation issues
  - No interface details
  - …
- System use cases (not business use cases), for example:



Alternative

2a Assume login is valid name and password is not correct and the same login name has been used less than four times

1. Ask for a new login name and password
2. System remember how many times the login name has been used with wrong password.
3. Go to 2

2-8 Assume login name is valid and password is not correct and the same login name has been used four times

1. Inform that the student has used more than four tries with the same login
2. Make the login name not valid

2-8 Assume login name not valid

1. Inform that it is not possible to login

Numbering of Alternative Flows

1. …
2. …
3. …
4. …

3a – instead of point 3 in the main flow do this …

3b - instead of point 3 in the main flow do this …

2-3a – instead of point 2 to 3 in the main flow do this …

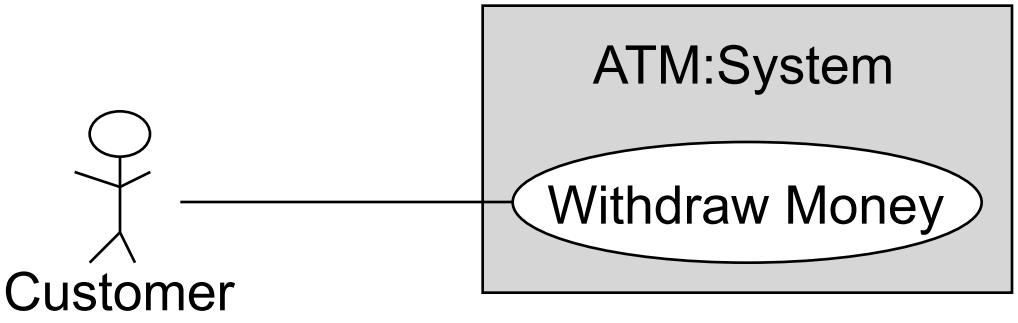
2-4a – instead of point 2 to 4 in the main flow do this …

2-4b – instead of point 2 to 4 in the main flow do this …

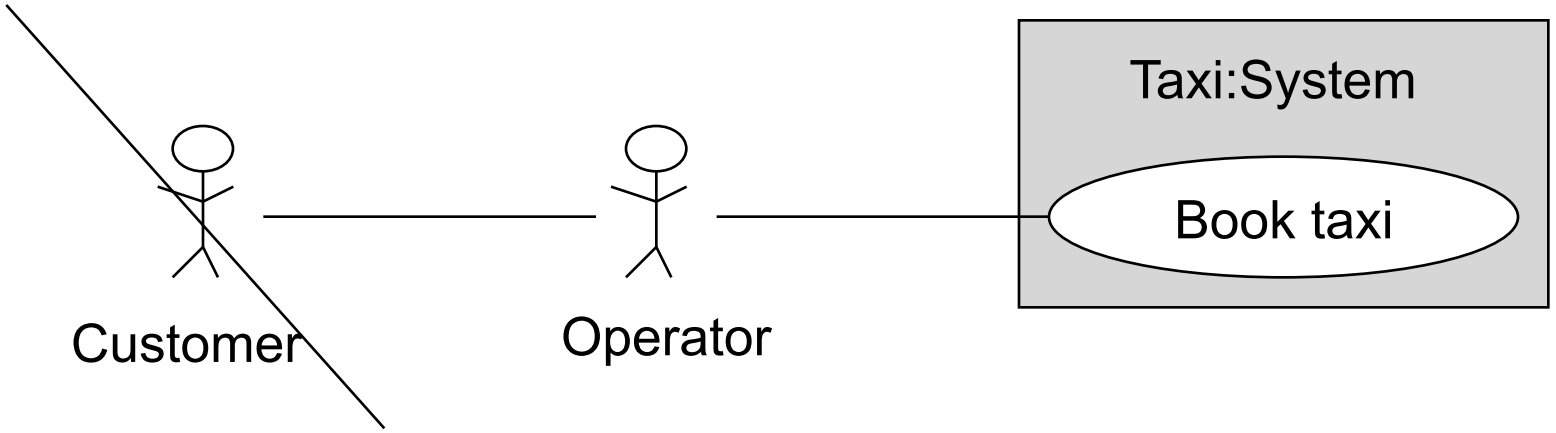
* Whenever

When to use the Pattern

- Complete use cases
  - Contain all the important parts, in particular main and alternative flows.
- Essential style
  - Does not contain implementation issues
  - No interface details
  - …
- System use cases (not business use cases), for example:



Not good style



Not the way of doing it:

1) Customer calls operator
2) Customer give bla bla to operator
3) Operator enter info bla bla into the system
4) Operator gives info bla bla back to customer

Only include interaction between actor and system!

Complete Use Case

- Template
  - Use Case Name
  - Use Case Goal
  - Actor Names
  - Main Flow of Event (sequence of action steps)
  - Alternative Flows (sequence of action steps)
  - Pre-Condition (if any)
  - Post-Condition

Pre- and Post-condition

- Pre-condition is what holds, whenever the use case takes place, *before* the action steps happen.
  - For example “Withdraw Money”:
    - Customer has account
    - Customer has a bank card
    - …
  - But: Usually trivial stuff is left out
- Post-condition is what holds after the use case has taken place.
  - Examples later
- Beware: Conditions and action steps have to fit together!

Problems with pre-conditions

- Let us consider having “the customer entered the correct PIN” as a pre-condition:
  - This means that we will not specify what happens if the customer enters a wrong PIN.
  - Making the use case very weak!
- One should avoid pre-conditions as much as possible in use cases, because the usual understanding of a pre-condition is that the post-condition need only be guaranteed if the pre-condition is met before the use case.

Examples for post-conditions

- Post-condition for “Withdrawal”
  - If the customer entered the PIN on the Card, and the customer's balance was greater or equal to the requested amount, then the customer got the requested amount and the amount was deducted from the balance.
  - If the customer entered the wrong PIN three times, the card was retained.
  - If the customer requested too much money, the card was returned to the customer.

Problem

- Write a post-condition for “register for course”?

Primary Actor

- A use case is always started by some actor : most often the primary actor
- There are also other types of actors:
  - Secondary actor
  - Helper actor
  - Time

Requirements on Use Cases

- Shall be a complete process
- Shall result in a given goal

External and Internal View

External View

User	System
identify himself	
	present choices
choose	
	dispense money
take money	

Can be viewed as a role play between user and system.

Example: Internal View

User	System
identify himself	
	verify identity
	present choices
choose	
	dispense money
take money	

Example: Internal View

1. User identify himself
2. System verify identity
3. System present choices
4. User choose
5. System dispense money
6. User take money

Action Block Details

- Two types:
  - Black box
    - User intention
    - System response
  - White box
    - User intention
    - System responsibility ← This is extra
    - System response
- In the white box case system response can often be left out if it is clear from the system responsibility how the system will respond.

Abstraction level

Real Use Cases

- Contain design details.
- Different aspects:
  - Internal view of the system
  - Also gives information about user interface (i.e., not essential use case)
  - Consider also technology issues (like databases)

Real Use Case?

- Is this a real use case?
  - User authenticates himself by PIN
  - System validates that PIN is correct
- Less abstract than
  - Customer identifies himself
  - System verifies identity
- But the first fragment makes more sense for people working in the banking industry.
- The second fragment is often too general, can be used in several contexts:
  - ATM
  - Library system
  - Student Registration system

Which abstraction level to use?

- During analysis, one should write essential use cases.
- Later, during design, essential use cases can be refined to real use cases.
- But we believe that there are better ways of defining the design than using real use cases

Process for Brief Use Cases

- Find actors
- Consider the goals of each actor
- Write brief use cases, based on the goals of the use case:
  - Give a name of the use case
  - Give the actor(s)
  - Write the goal as one sentence
  - Write a brief description

Prioritize Use Cases

- Based on the importance of a brief use case, a complete use case should be written.
- For ordering use cases take into consideration:
  - How important is the use case for the business?
  - Does the use case have important consequences for the system (technological, architectural)? These use cases should come early.

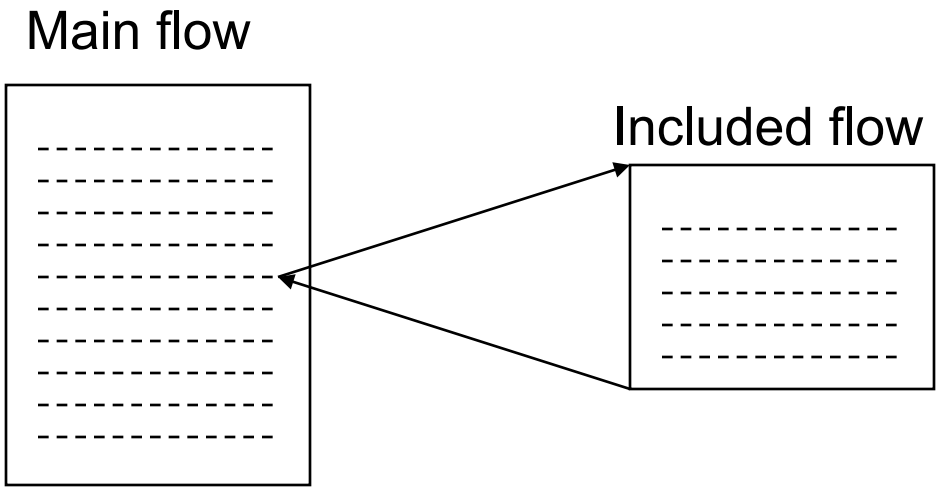
Process for Complete Use Case

- Based on a brief use case
  - Write the main flow of control using
  - Write the alternative
  - Write pre-conditions (if any)
  - Write the post-conditions

Advance topics

”Include”

- Often used to catch common action steps

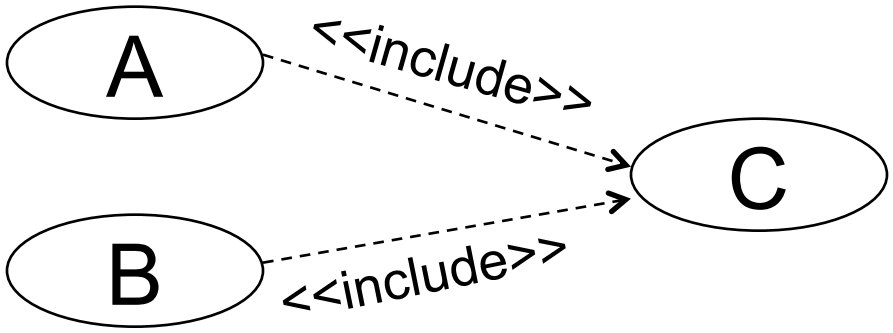


- Important: one has to leave and come back to the main flow at the same place.

- In the flow which is included add action step:
- include (the name of the included use case)

UML-syntax

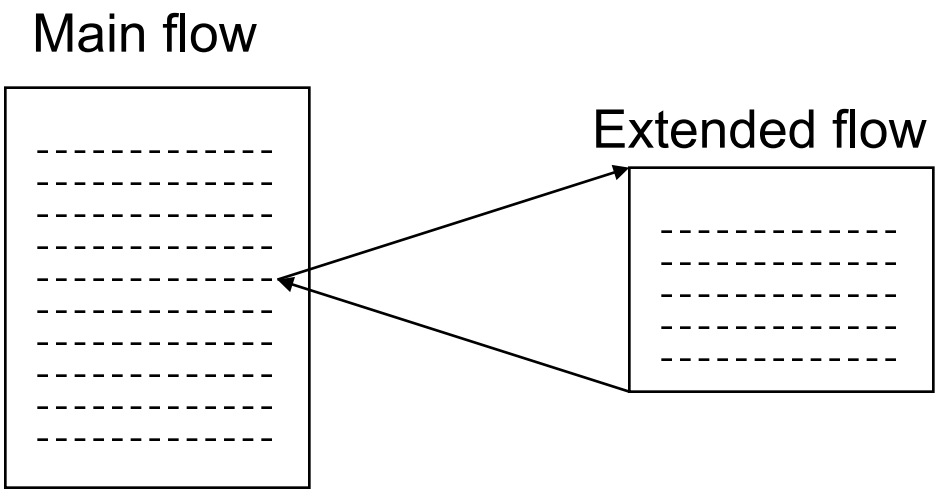
- Use cases A and B includes C:



  - A and B know about C, but not the other way round.
  - C must have at least as high priority as A and B.

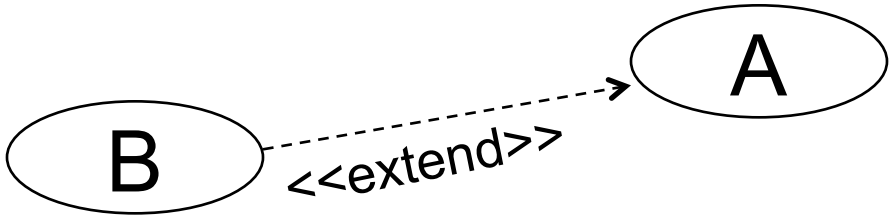
”Extend”

- Sometimes one wants to include extra action steps in a use case. Then ”extend” might be useful:



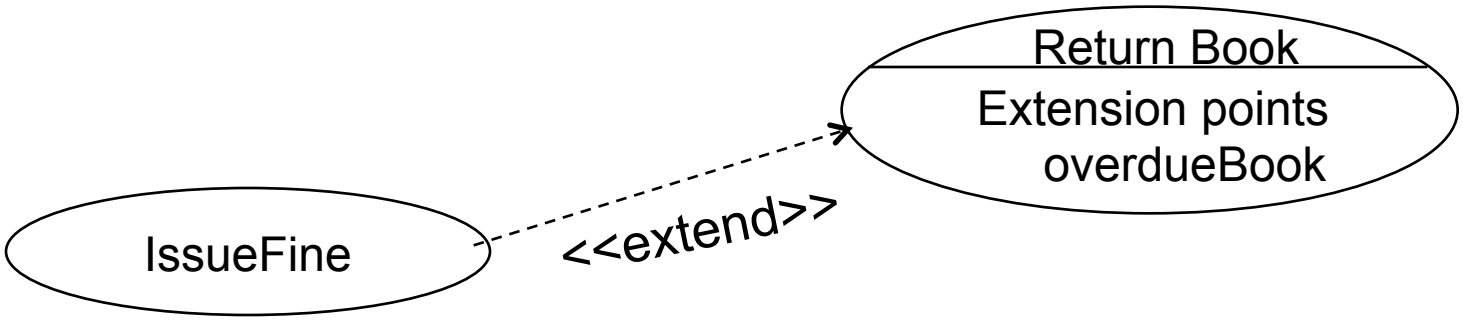
- Important: one has to leave and come back to the main flow at the same place.

UML Syntax



- A is extended with B
- Some condition has to be satisfied for the use case B to be used.
- A has to be a full use case without B.

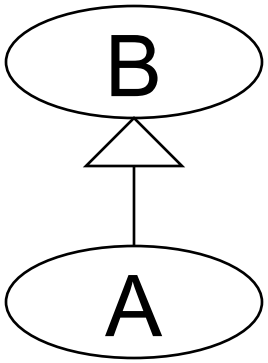
extend



- In the use case IssueFine's flow:
- …
- …
- Extension point:overdueBook
- …

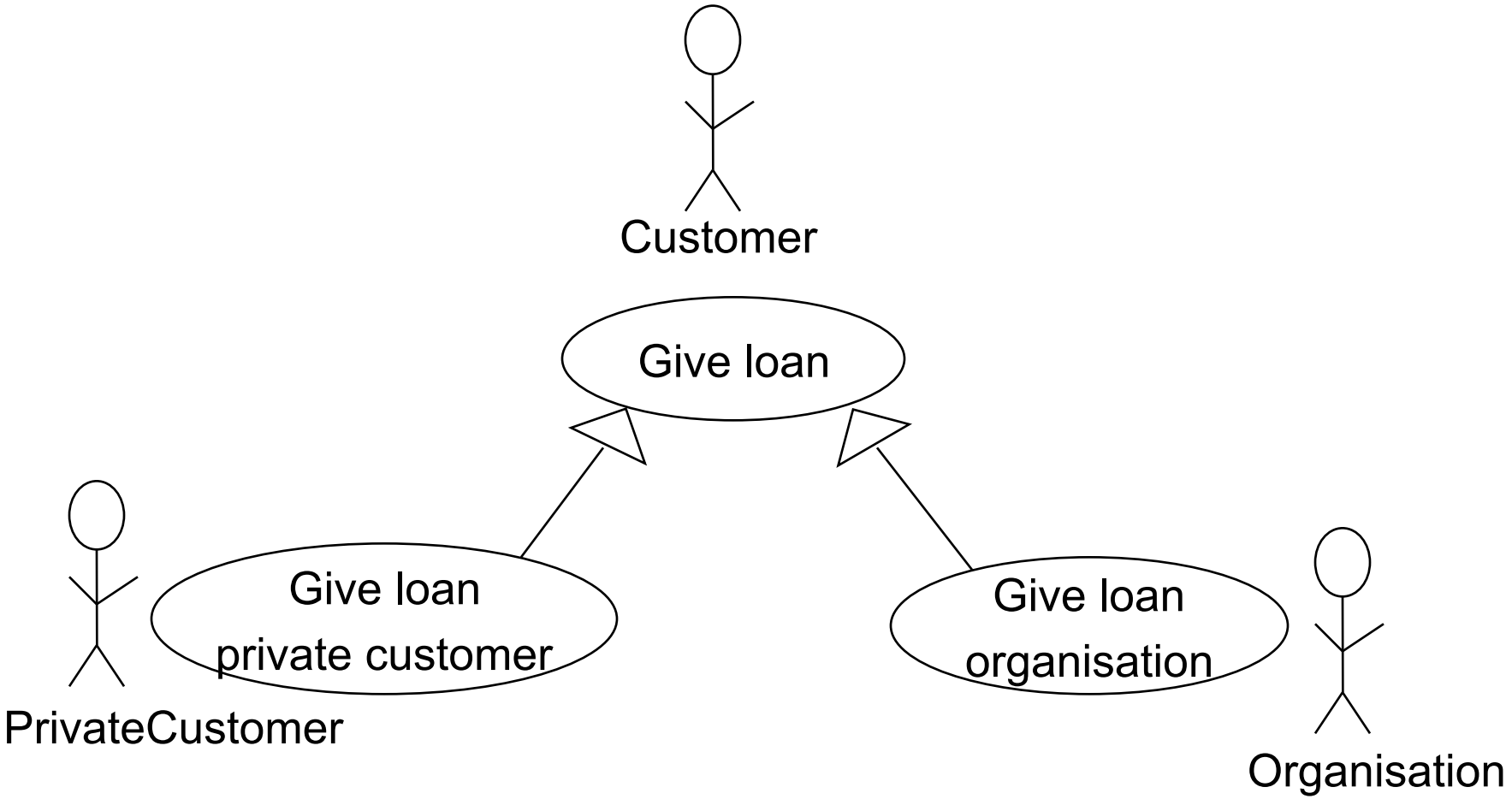
Inherit relation

- It might happen that several use cases have action steps which are similar. In this case one can use abstract use cases:



- A inherits from B.

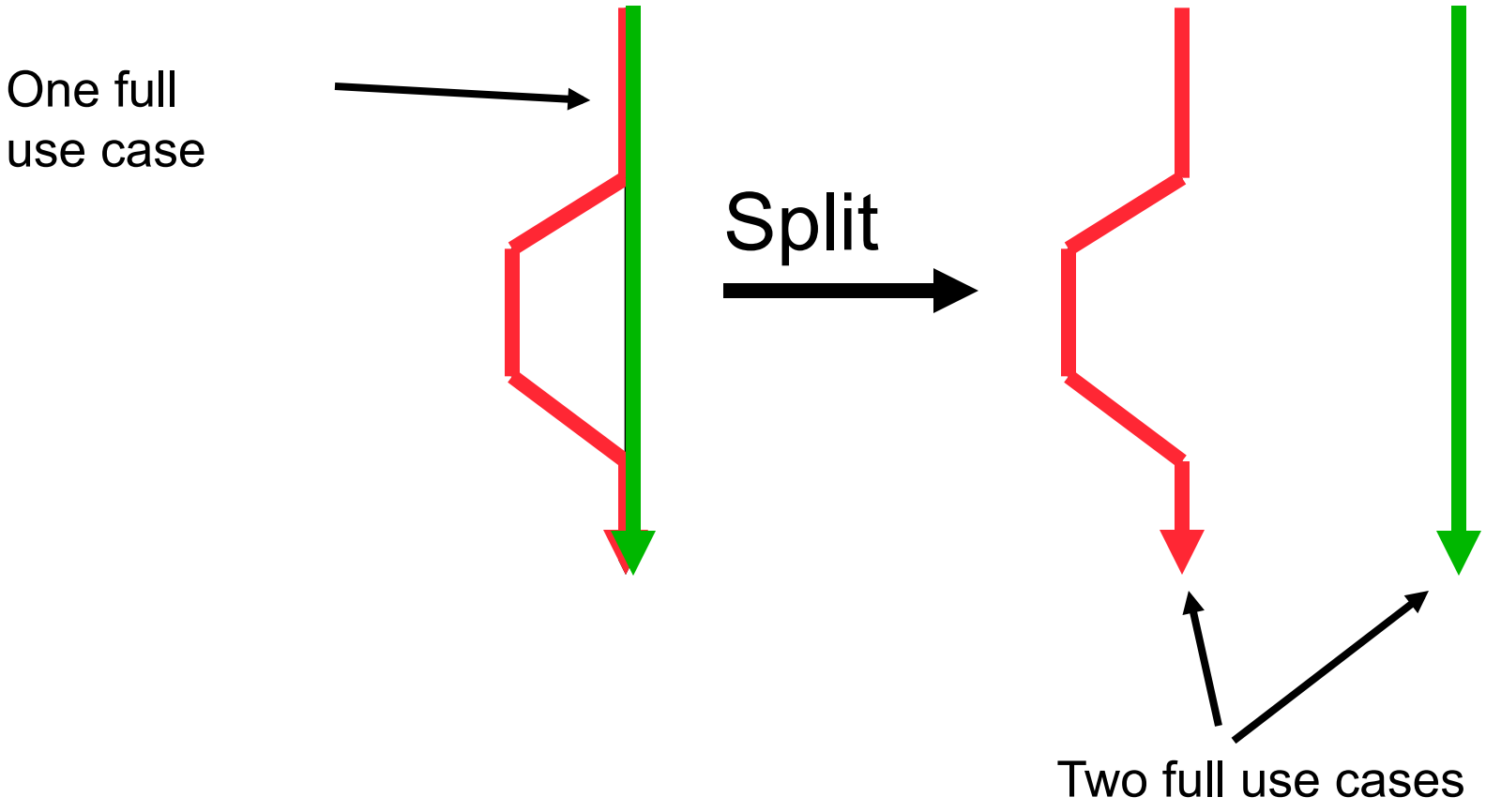
Example


Customer
Give loan
Give loan
private customer
Give loan
organisation
PrivateCustomer
Organisation

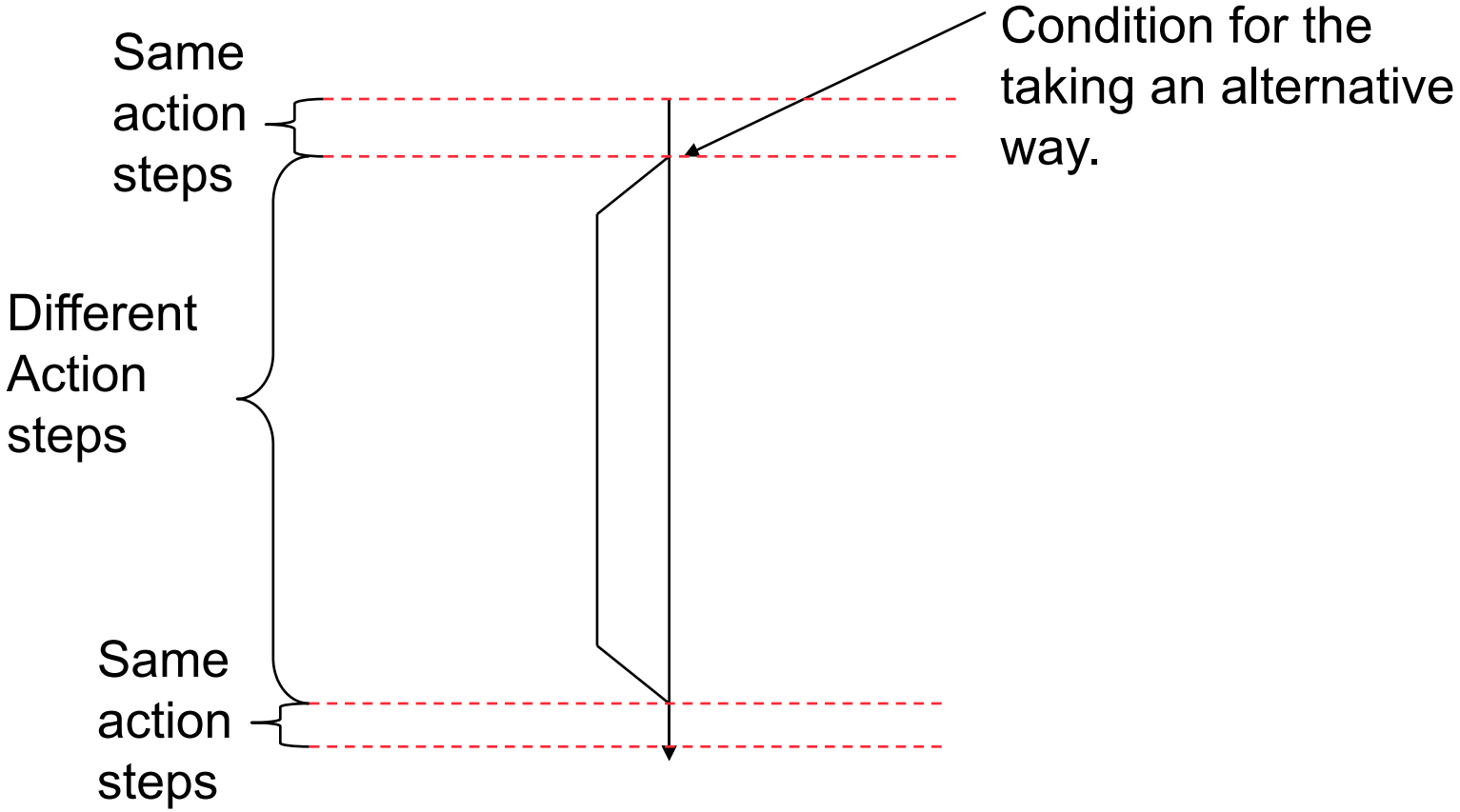
Uses

- Whether to use ”include”, ”extend”, and inherit is discussed a lot.
- Most important reason for using these features: they can improve readability

Split



Vertical split



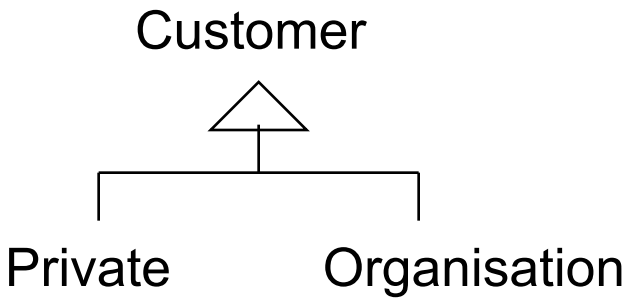
When to split

- If it is not important to show the condition
- If main flow and alternative flow do not have many steps in common
- If the two flows are complete use cases in themselves

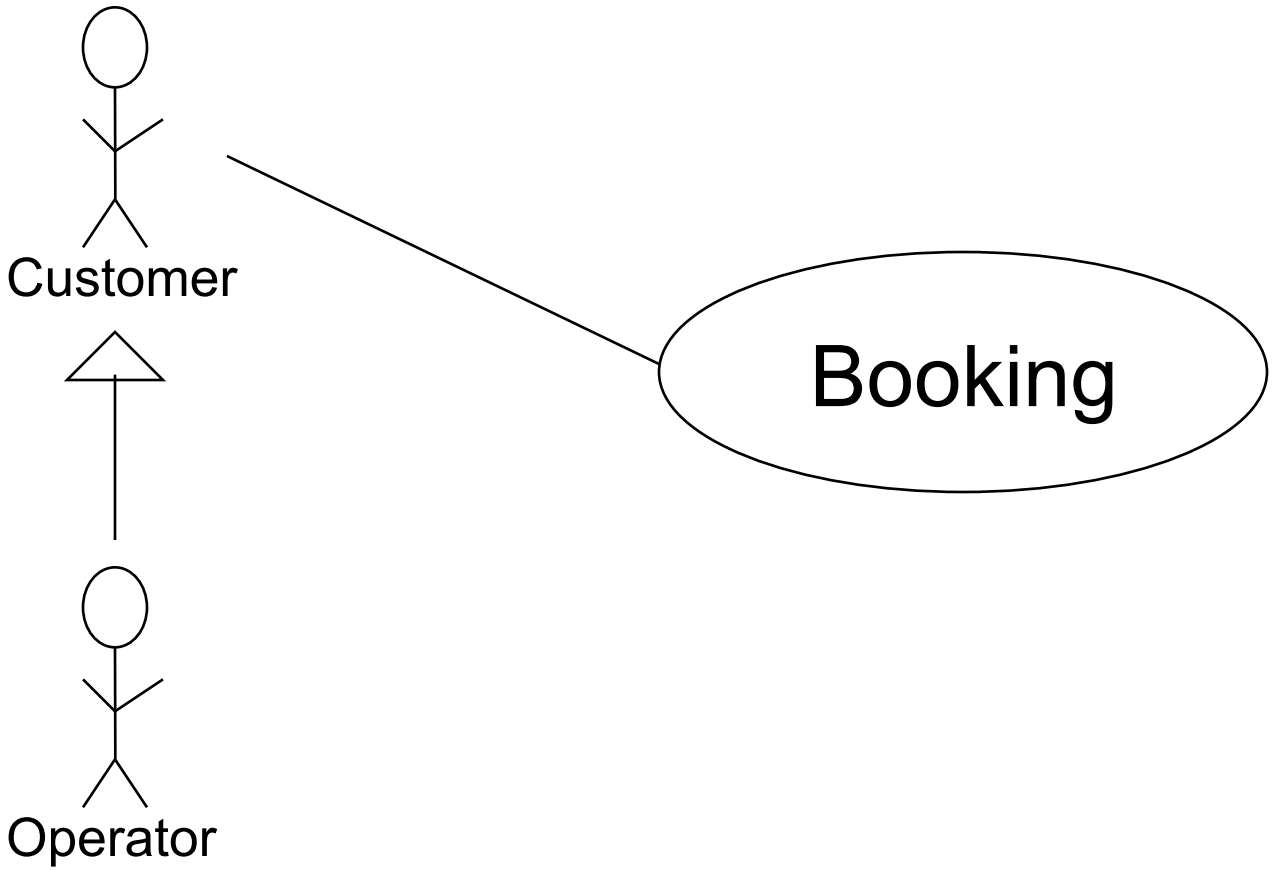
- Sometimes one might want to combine use cases as well.

Actor Specialisation

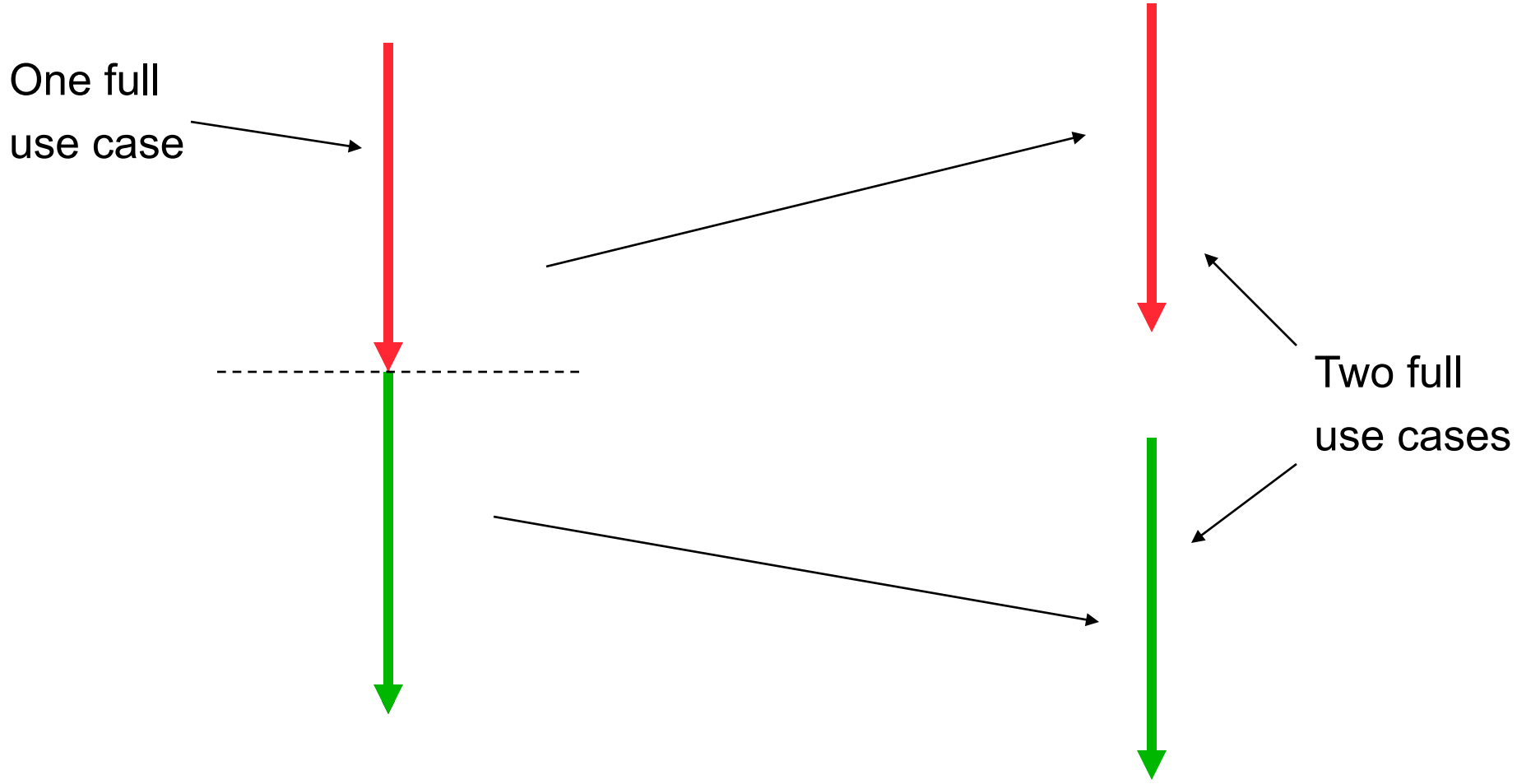
- Vertical split can lead to more specialised actors, for example:



Example of Actor Inheritance



Horizontal Split



Business Use Cases

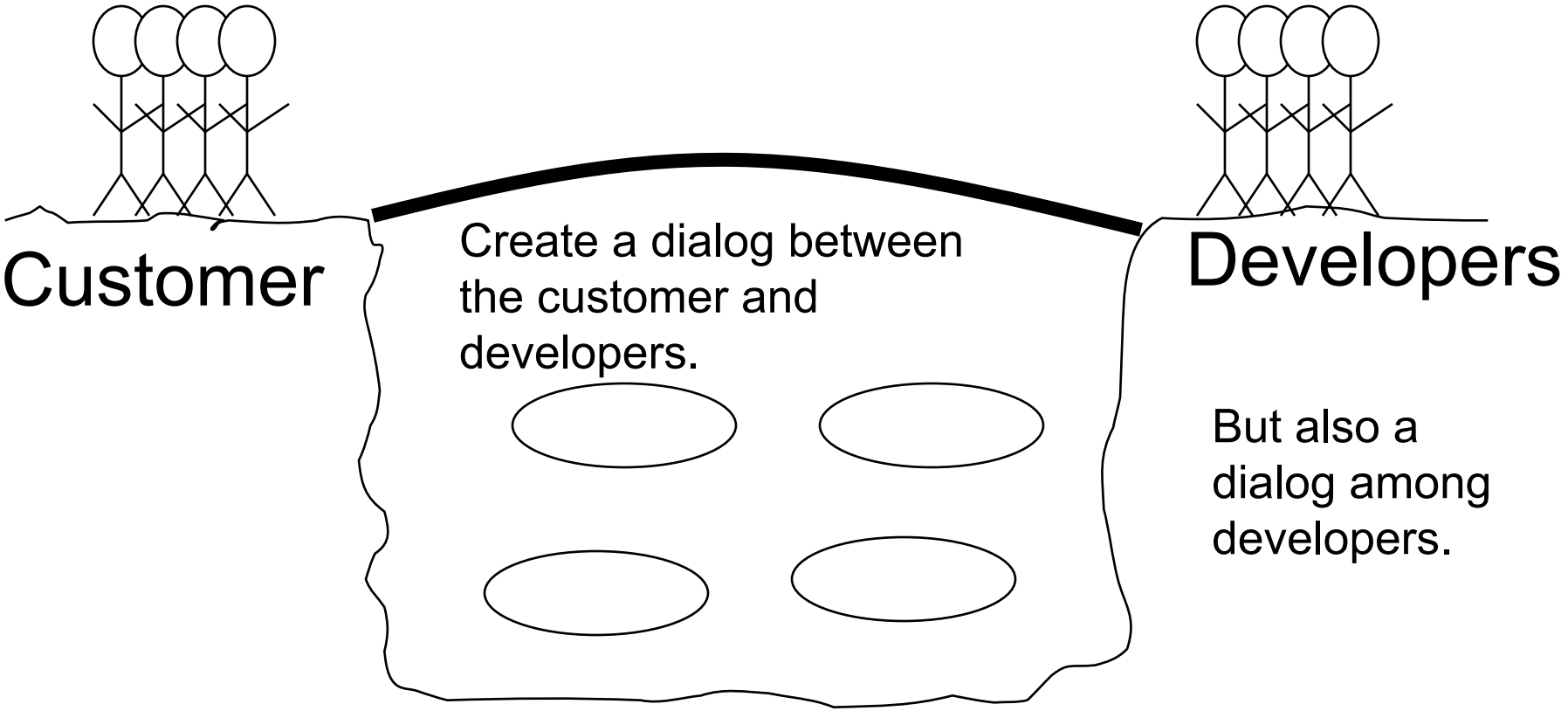
- Describe how a business works. Might describe human behaviour as well.
- Might contain system use cases.

Applications of Use Cases

Problem

- Why using Use Cases?
- Because:
  - Describing the behavior of the system
  - Communicating with the customer
  - Catching functional requirements on the system
  - Obtaining the user interface
  - Driving the development process, deciding what should be done in each iteration
  - Obtaining tests for the system

Communication



Requirement analysis

- Often these kinds of requirements have to be identified (FURPS+):
  - Functionality
  - Usability
  - Reliability
  - Performance
  - Supportability
  - ”+” represents further requirements

Example: ATM

1. ATM saves information about withdrawals
2. Can be given a code
3. Gives customer amount X of money if customer has at least X on the account.
4. Can be given a card
5. Can return a card when withdrawal is finished or when transaction is cancelled.
6. Can make transactions between accounts
7. Can insert money into the account
8. The amount of money inserted should be added to the account
9. Reduce the account by the amount withdrawn.
10. Can choose an amount.
11. Can choose to withdraw.
12. Check amount on account
13. Can obtain a receipt.
14. Can stop the process of withdrawal.
15. Can give code up to three times.
16. If wrong code three times then the ATM keeps the card.
17. …

Problem

- What is the problem with the list of requirements on the previous slide?
- Problems:
  - How to priorities requirements
  - How to group requirements
  - Often imprecise
  - Hard to obtain an overview
  - Is it a complete list of requirements?
  - Many!!! Can be several thousands.

Different kinds of functional requirements

- Business requirements:
  - A customer shall be able to book a taxi via telephone
- System requirements:
  - The system should estimate the time until a taxi arrives

- Use cases will help to separate these two types of requirements, since we write use cases only for describing system behavior.
- We will obtain system requirements from use cases.

Grouping Requirements

- Requirements can be grouped in several ways
- One way: Use cases
- For example, Withdraw Money relates to the requirements:
  - R1, R2, R3, …
- Implementing a requirement might not make a system more useful; implementing a use case does!
- Use cases tackle the problem of making requirements readable, understandable, and to choose priorities

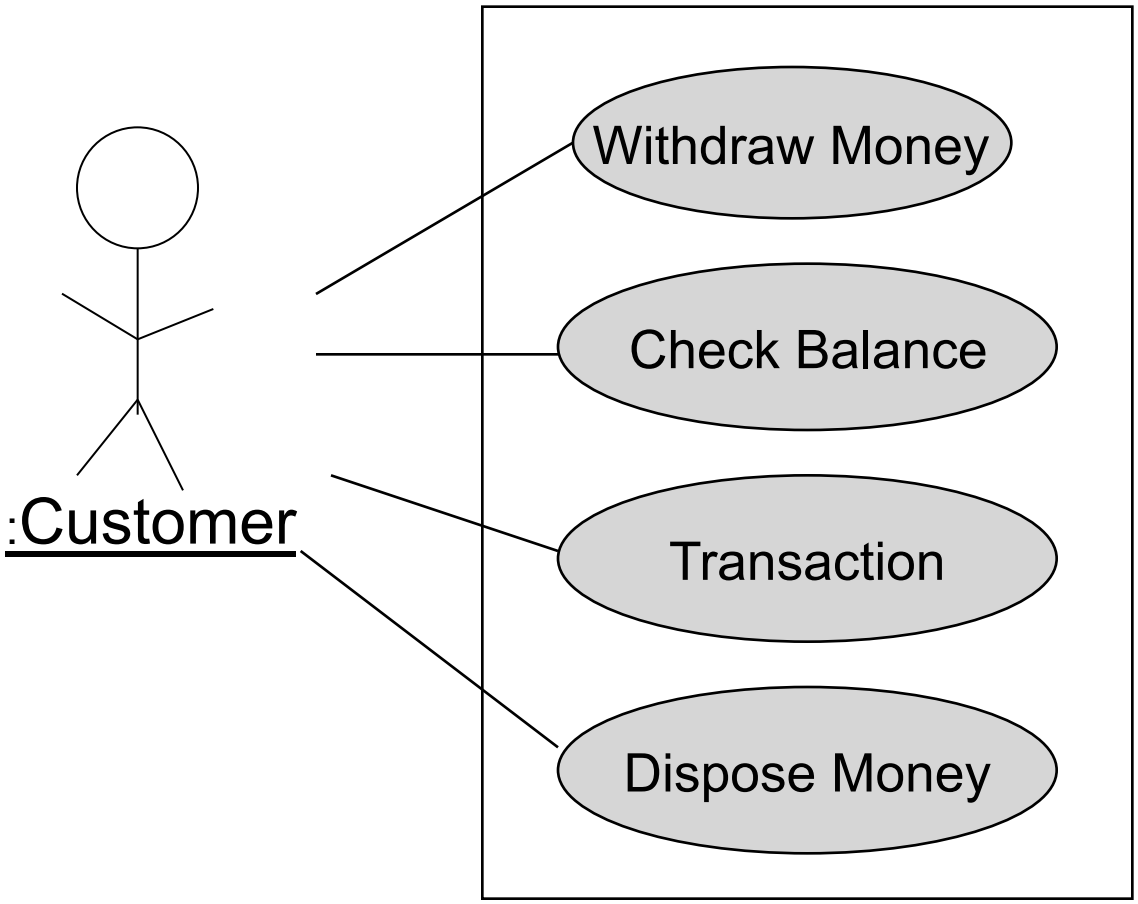
Functional requirements

- Use cases capture most functional requirements.
- But: Some functionality can be ”hidden” in several/all use cases
  - For instance: Logging occurring events

Dealing with Requirements

- Different ways of dealing with functional requirements:
  - Only having a requirement list
  - Only having use cases
  - A combination of both

Example: ATM


Withdraw Money
Check Balance
Transaction
Dispose Money
:Customer

Use Case/Requirement Matrix

	Withdraw Money	Check Balance	Transaction	Dispose Money
R1	X			
R2	X	X	X	X
R3	X			
R4	X	X	X	X
R5	X			

. . .

Non-functional requirements

- Non-functional requirements are hard to handle by use cases, but sometimes one can relate them to use cases.
- Further documents (apart from use cases) are needed

Example (1)

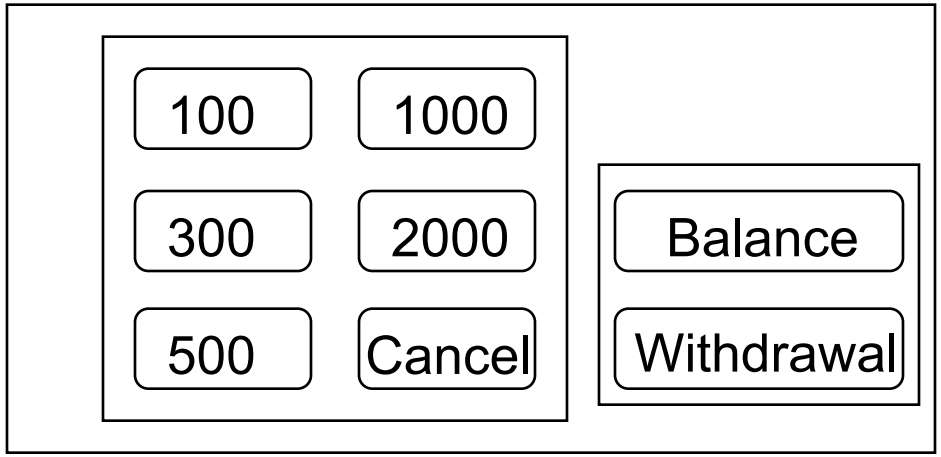
- Usability
  - ATM should be usable for colour blind persons
- Reliability
  - Frequency of failure
    - At most one failure per year (or per 10 sec)
  - Restart after an error
    - When restarting, account balance should be checked against bank to ensure right value (in case of unfinished transactions)

Example (2)

- Supportability
  - ATM system should be adaptable to
    - Different currencies
    - Different languages
    - Different bank computer systems
    - Different card types

User interface

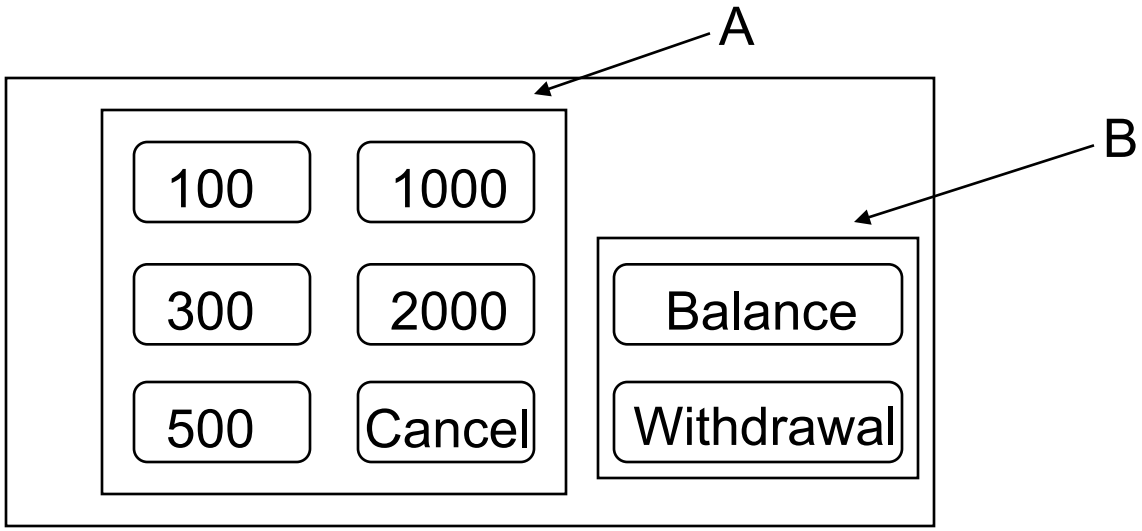
- Usually an ”user interface expert” will derive the user interface from use cases
- For instance:



- … and make a description of the interface

Connecting user interface and use cases

- For instance: ”Customer chooses amount in window A”



- Dangerous: Such use cases are very fragile concerning changes in user interface

Problem

- Should one consider user interface details when writing use cases?
- Should use cases contain information about the interface?
- Should one make the user interface before or after the use cases?

Interface first or last?

- Most people agree that use cases should be written before user interface is designed
- Exception: Interface can be given, no changes are possible
- (Some people even recommend designing the user interface first)
- Interface is important, because customers might get new ideas by looking at it (less abstract than use cases, easier to understand, things become more concrete and more obvious)

Essential Use Cases

- A use case which abstracts from user interface, implementation details etc.
- Avoids premature design decisions of how to develop the system, such as the look of user interface, whether to use a database etc.

Summary

- We have considered
- How to write brief use cases
- How to write complete use cases
  - How to write main flow
  - How to write alternative flows
  - How to write post-conditions
  - …

Appendix

Role play

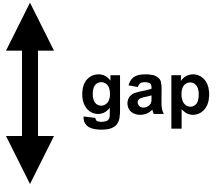
- To illustrate one flow through a use case, one can use a concrete case.
- One can play the interaction between the system and the actor.
- One person plays the system and for each actor there is a person playing the actor.

Problem

- One plays the ATM
- One plays the customer which wants to take out money
- Write down the lines for each role, and then play it.

Goal

- In books
  - Often the goal are the use cases themselves.



- In industry
  - The goal is the complete running system.

Gap between Analysis and Design

- Often the case:
- Analysis
  - Use cases

Often different people

- Design
  - Realization of use cases

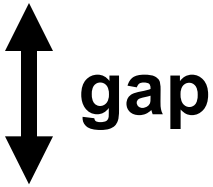
Trend

- Use cases are good in grouping requirements

- Why not use them for everything?

Gap between use cases and code

- Often the case:
  - Use cases (Might not be modified after code produced)



  - Code (Being modified)