

# Use cases

---



**SoftEng**  
<http://softeng.polito.it>

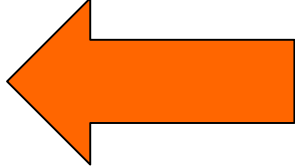
Version 2.7.2 – November 2016

© Maurizio Morisio, Marco Torchiano, 2016



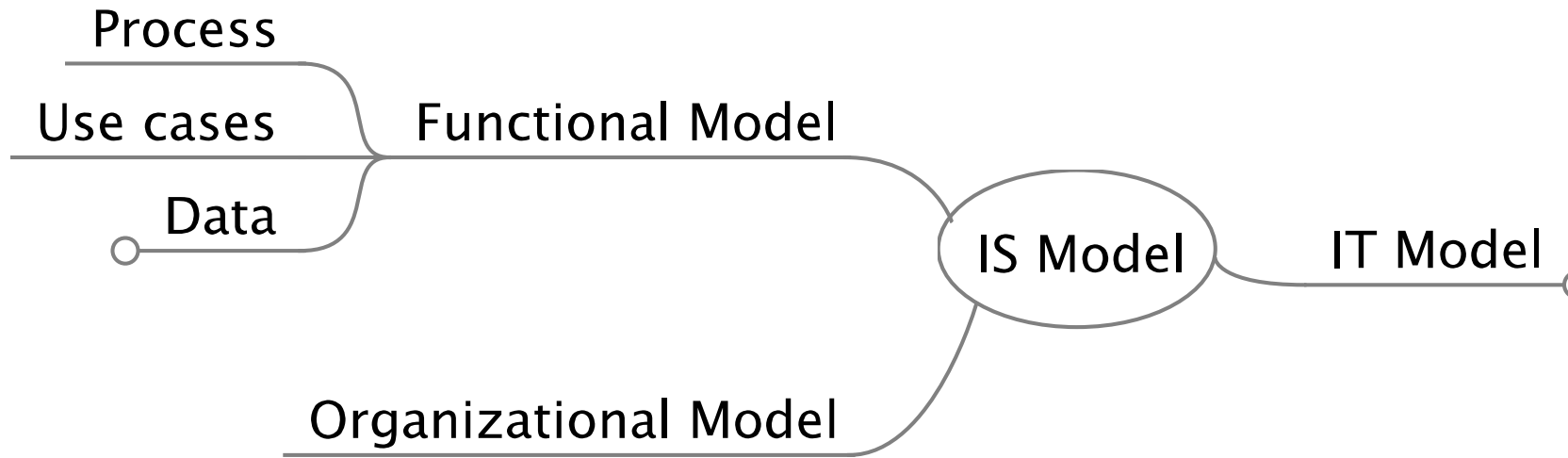
# Requirements Document

---

1. Purpose and scope
2. Glossary
3. **The use cases** 
4. The technology to be used
5. Other various requirements
6. Human backup, legal, political, organizational issues

# Functional model – Use cases

---



# History

---

- Introduced by Ivar Jacobson in 1992
  - ◆ codified a visual modeling technique
- Widespread adoption in the 1990's
  - ◆ initially in the Object-Oriented community

# Use case

---

- Captures a contract between the stakeholders of a system about its behavior.
- Describes the system's behavior under various conditions as it responds to a request
  - ◆ from a stakeholder, the *primary actor*.
- The primary actor initiates an interaction with the system to accomplish some goal.
- The system responds, protecting the interests of all the stakeholders.

# Purpose of Use Cases

---

- Describes business' work process,
- Focus discussion about upcoming software system requirements,
- Describe functional requirements for a system
- Document the design of the system

# Requirements and Use Cases

---

- Use cases can represent the functional (behavioral) requirements
  - ◆ If properly written
- They miss
  - ◆ External interfaces
  - ◆ Data formats
  - ◆ Business rules

# Definition

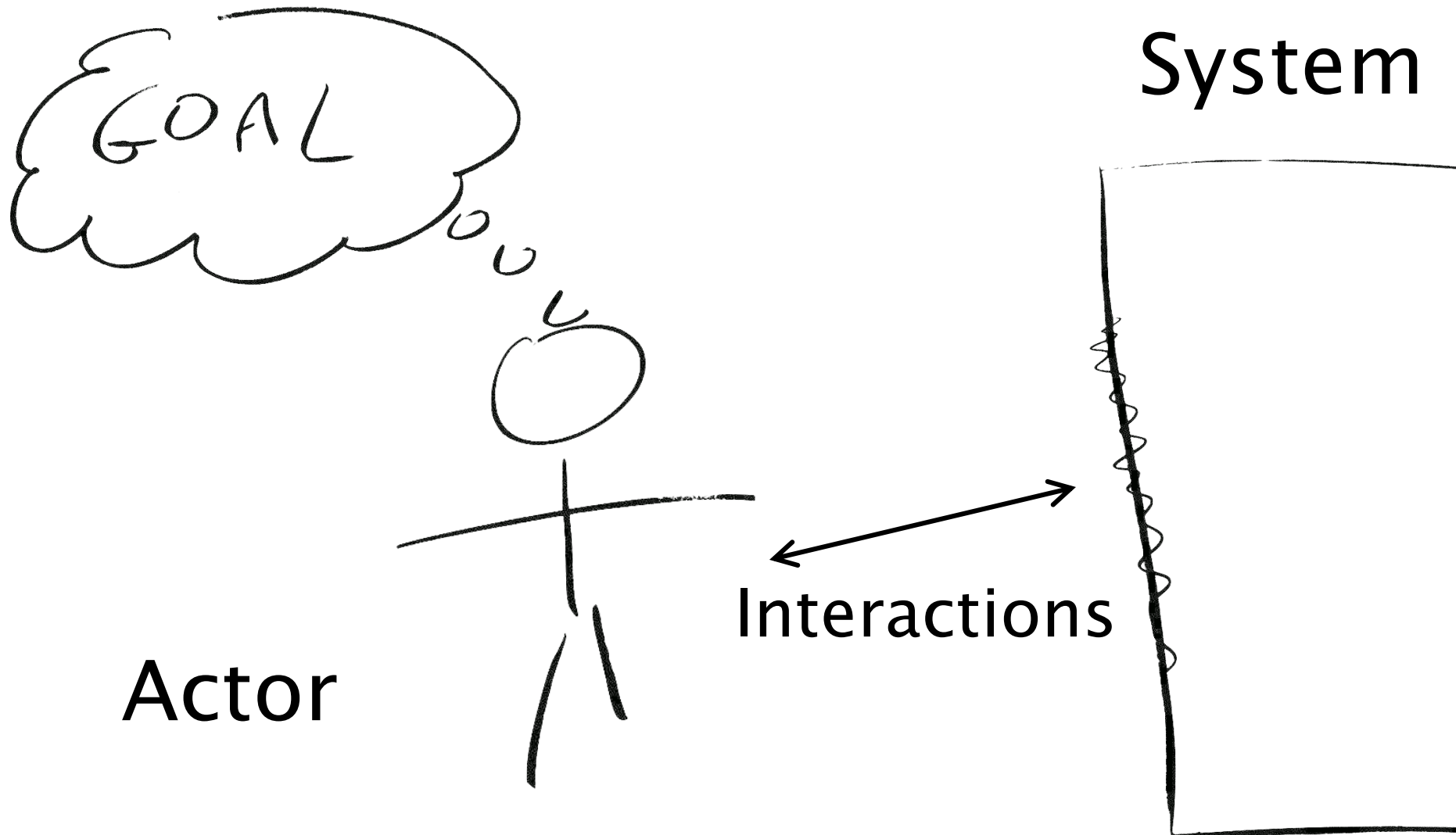
---

A use case describes a **goal-oriented** set of **interactions** between external **actors** and the **system** under consideration.



# Key elements

---



# Key elements

---

- The **actor** involved
  - ◆ type of user that interacts with the system
- The **system** being used
  - ◆ treated as a black-box
- The functional **goal** that the actor achieves using the system
  - ◆ the reason for using the system

# Description

---

- Use cases describe the sequence of interactions between actors and the system
  - ◆ to deliver the service that satisfies the goal
- They also include possible variants of this sequence
  - ◆ either successful or failing

# Actor

---

- Actors are parties **outside the system** that interact with the system.
- An actor may be a class of users, roles users can play, or other systems.
  - ♦ A **primary actor** is one having a goal requiring the assistance of the system.
  - ♦ A **secondary actor** is one from which the system needs assistance.

# Actor

---

- External entities:
  - ◆ Which persons interact with the system (directly or indirectly)?
    - Don't forget maintenance staff!
  - ◆ Will the system need to interact with other systems or existing legacy systems?
  - ◆ Are there any other hardware or software devices that interact with the system?
  - ◆ Are there any reporting interfaces or system administrative interfaces?

# Goals

---

- The use case describes only what is the relationship of the actor to the system
- The goal must be of value to the (primary) actor:
  - ◆ “Enter PIN code” is not
  - ◆ “Withdraw cash” is

# Goal

---

As a *<actor type>*

I want *<to do something>*

So that *<some value is created>*

# Goal

---

bank customer

As a ~~<actor type>~~

to perform a withdrawal

I want ~~<to do something>~~

I get some cash for me

So that ~~<some value is created>~~



# Stakeholders and interests

---

- The system under design operates a contract between stakeholders
  - ◆ Use cases detail the behavioral part
- Use case captures all and only the behaviors related to satisfy the stakeholders' interests
  - ◆ See Guarantees section

# Stakeholders

---

- Someone having a vested interest in the behavior of the system (use case)
  - ◆ E.g. owner, board of directors
  - ◆ Offstage actors
- Actors (users) represent a subset of stakeholders

# Scope

---

- The extent of what we consider to be designed by us
  - ◆ Not already existing
  - ◆ Not someone else's design job
- Defined by
  - ◆ In-out list
  - ◆ Actor-Goal list
  - ◆ Use case briefs

# Scope

---

- Enterprise
  - ◆ Business use cases
- Software System
  - ◆ System use cases
- Software Component

# In-out list

---

- What lies in or out of scope

<b>Topic</b>	<b>In</b>	<b>Out</b>
Invoicing in any form		Out
Producing reports about requests, e.g. by vendor, by part, by person	In	
Merging requests to one PO	In	
Partial deliveries, late deliveries, wrong deliveries	In	
All new system services, software	In	
Any non-software parts of the system		Out
Identification of any pre-existing software that can be used	In	
Requisitions	In	

# Actor-goal list

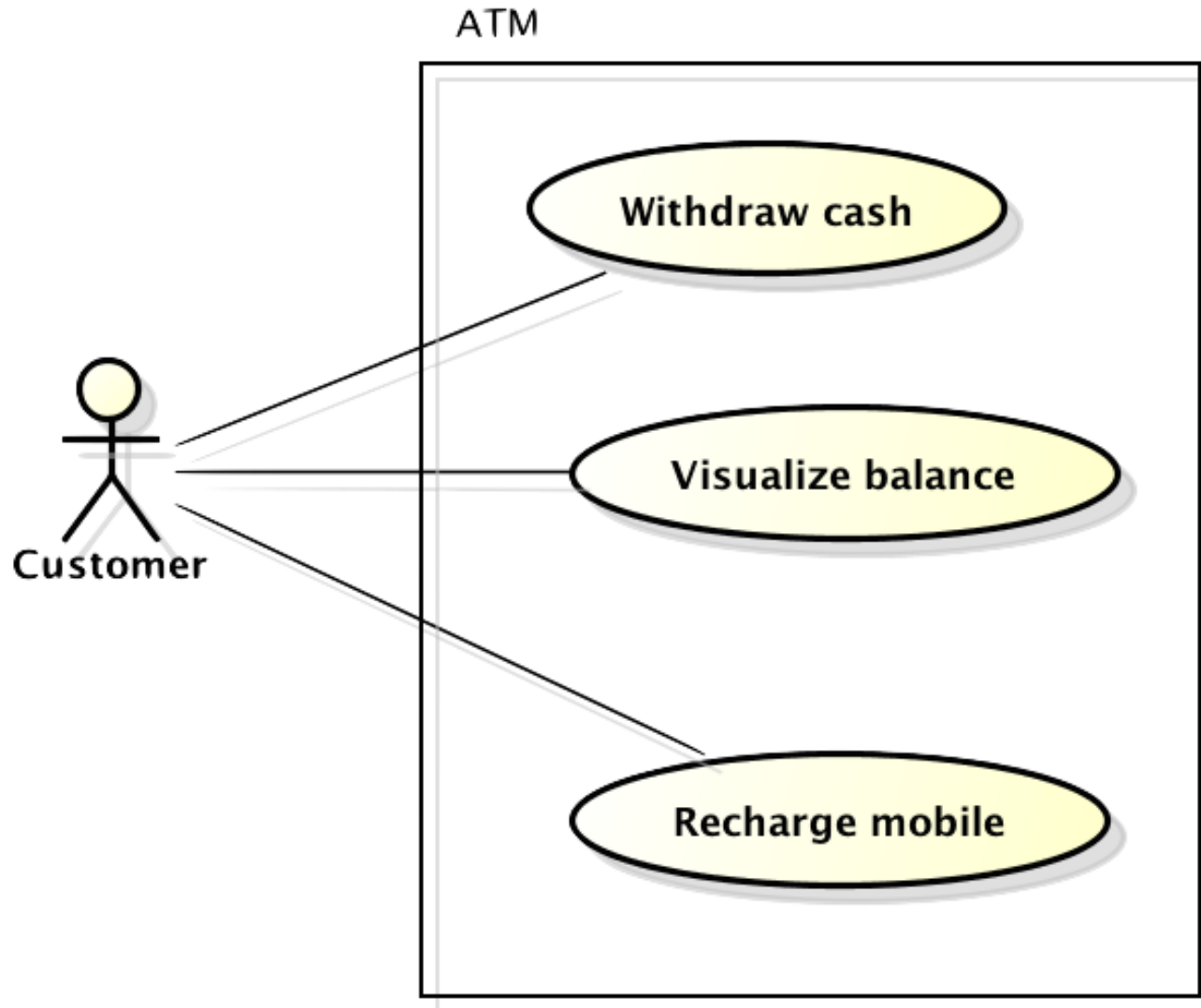
---

- Actors and corresponding goals + Priority
  - ◆ See UML Use Case Diagrams

Actor	Task-level Goal	Priority
Any	Check on requests	1
Authorizer	Change authorizations	2
Buyer	Change vendor contacts	3
Requestor	Initiate an request	1
"	Change a request	1
"	Cancel a request	4
"	Mark request delivered	4
"	Refuse delivered goods	4
Approver	Complete request for submission	2
Buyer	Complete request for ordering	1
"	Initiate PO with vendor	1
"	Alert of non-delivery	4
Authorizer	Validate Approver's signature	3
Receiver	Register delivery	1

# Actor-goals as UC Diagram

---



# Use case briefs

---

- Summary of the use case
- Consisting of 2–6 sentences
- Clearly state
  - ◆ What is going on
  - ◆ Most significant activities



# Use case briefs

Actor	Goal	Brief
Production Staff	Modify the administrative area lattice	Production staff add admin area metadata (admin hierarchy, currency, language code, street types, etc.) to reference database and contact info for source data is catalogued.
Production Staff	Prepare digital cartographic source data	Production staff convert external digital data to a standard format, validate and correct it in preparation for merging with an operational database. The data is catalogued and stored in a digital source library
Production & Field Staff	Commit updated transactions of a shared checkout to an operational database	Staff apply accumulated update transactions to an operational db. Non-conflicting transactions committed to operational dn. Application context synchronized with op. db. Committed transactions cleared from application context. Leaves op. db consistent with conflicting transactions available for manual/interactive resolution.

# Precision

---

- How much detail is provided
  - ◆ Brief
    - few sentences summarizing the use case
  - ◆ Casual
    - few paragraphs of text elaborating the use case in the form of a summary or story
  - ◆ Detailed
    - formal document based on a long template with fields for various sections

---

# USE CASE DIAGRAMS

# Use case diagrams

---

- Use cases can be (collectively) represented using UML use-case diagrams
- Often the “brief” level of detail can be useful in this context
- Note:
  - ◆ Use case diagrams report only the goal (in summary) they lack the detailed interactions (a.k.a. the narrative)

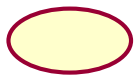
# Elements of use case diagrams

---



Actor

- Someone (user) or something (external system, hardware) that
  - ◆ Exchanges information with the system
  - ◆ Supplies input to the system, or receives output from the system

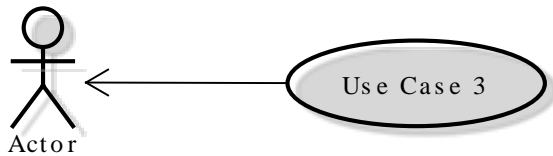
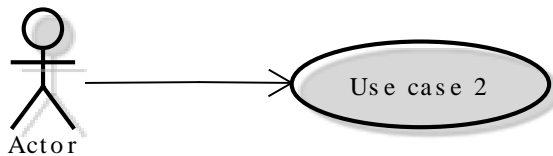
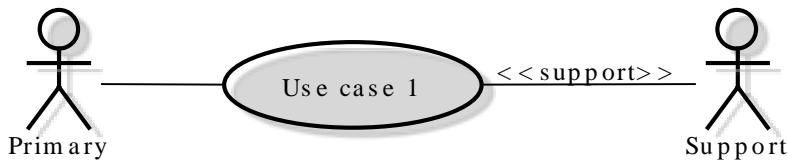


Use Case

- A functional unit (functionality) part of the system

# Relations – Participation

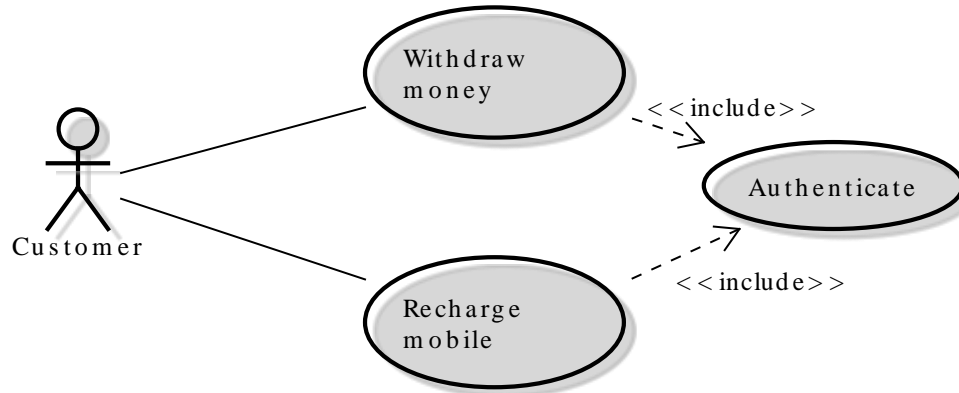
---



- Participation models:
  - ◆ Which actors participate in a use case
  - ◆ Whether an actor is just supporting
  - ◆ Where execution starts

# Relations – Inclusion

---

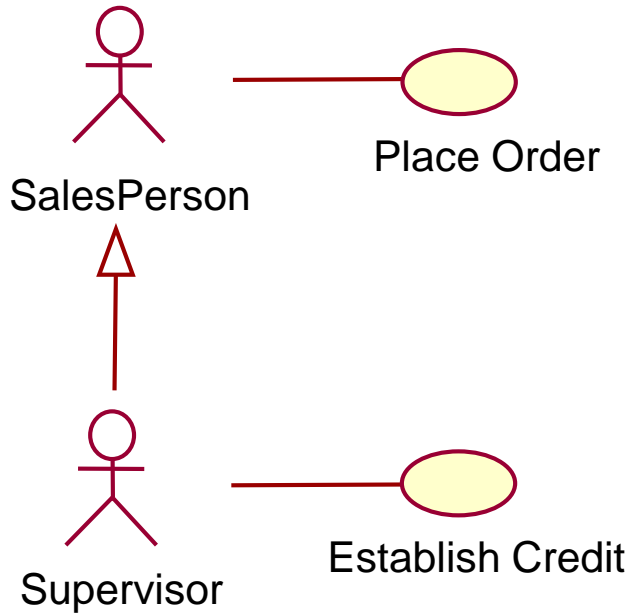


- Inclusion models

- ◆ Models that functionality A is used in the context of functionality B (one is a phase of the other)
- ◆ It is like a sub-function call

# Relations – Extension

---

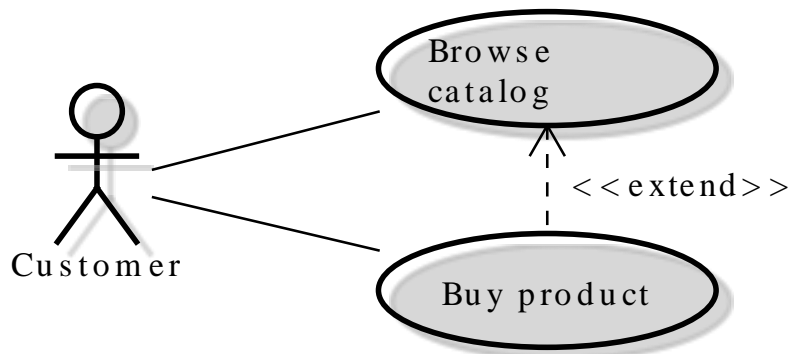


- A generalization from an actor B to an actor A indicates that an instance of B can interact with the same use-case as an instance of A



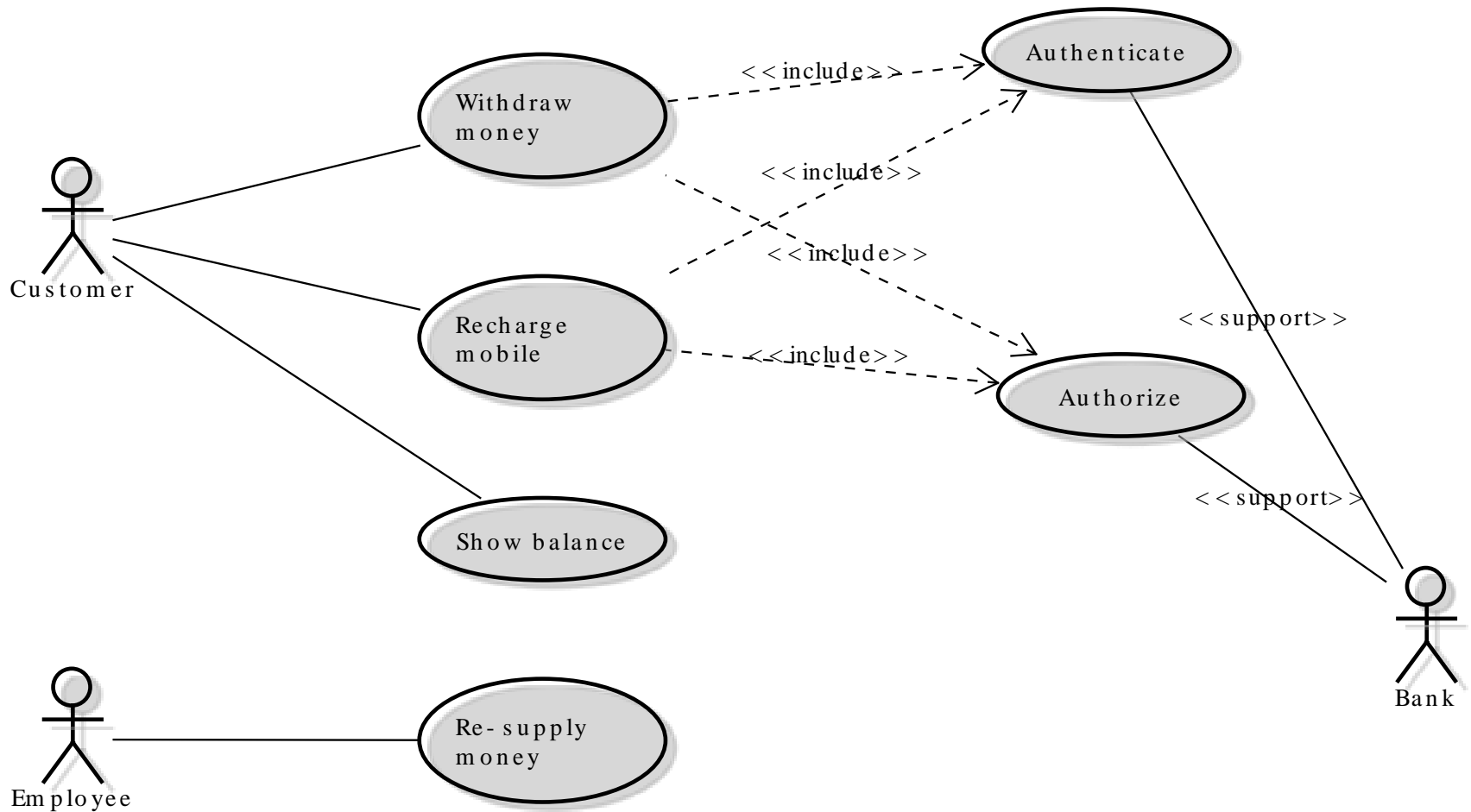
# Relations – Extension

---



- A UC extend another UC when it represent the natural (though not mandatory) prosecution of the extended UC

# Use case example



# Granularity

---

- **Summary level** is the 50,000 feet perspective,
- **User-goal level** is the sea-level perspective,
- **Subfunction** is the underwater perspective.

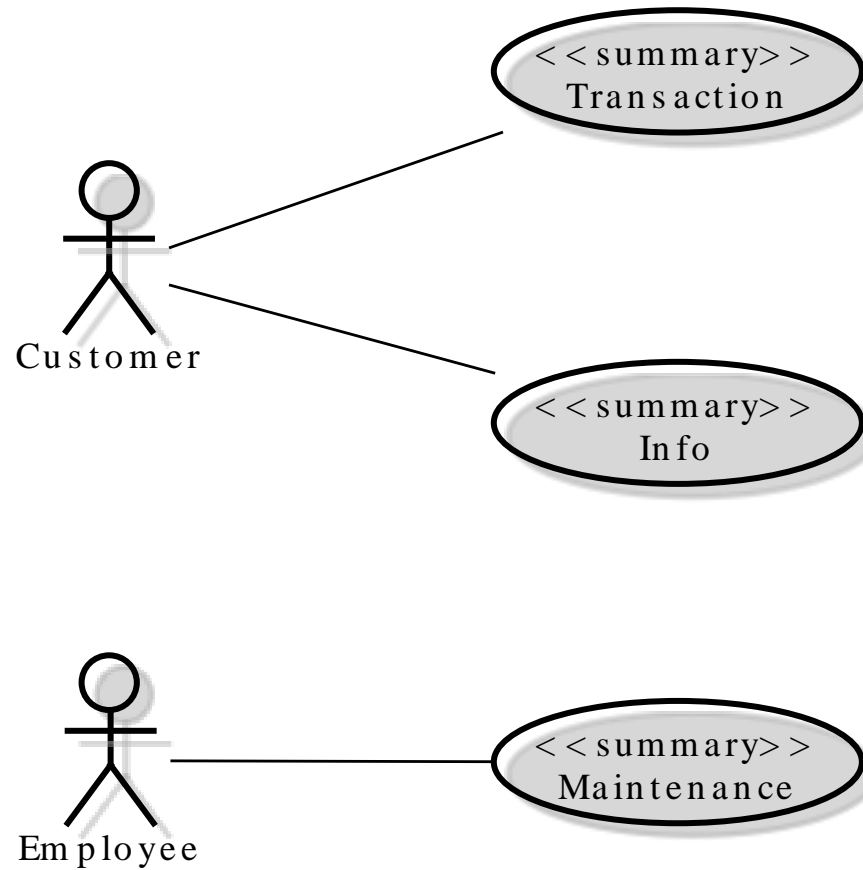
# Summary level

---

- Summary level use cases:
  - ◆ are large grain use cases that encompass multiple lower-level use cases; they provide the context (lifecycle) for those lower-level use cases.
  - ◆ they can act as a table of contents for user goal level use cases.

# Summary level

---



# User Goal Level

---

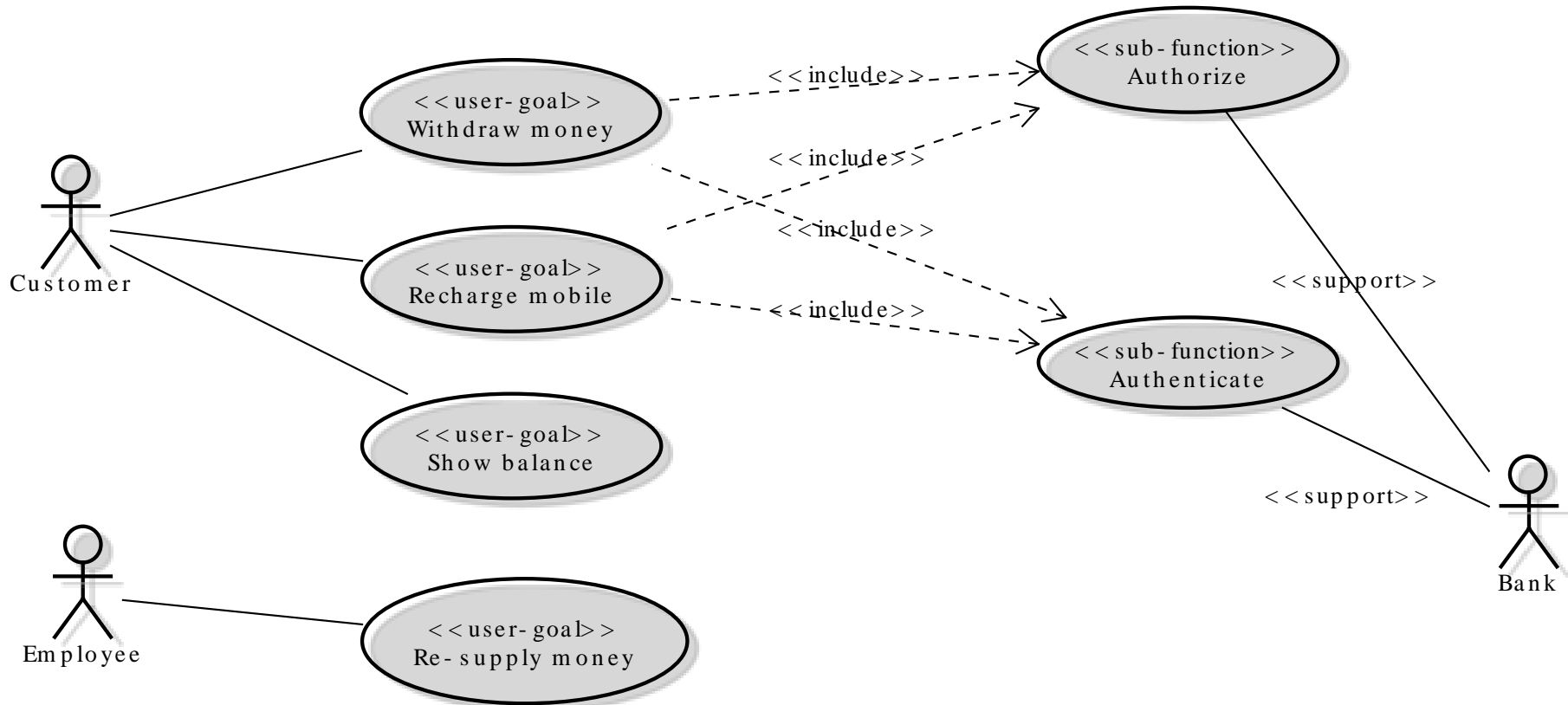
- User-goal level use cases:
  - ◆ describe the goal that a primary actor or user has in trying to get work done or in using the system.
  - ◆ *are usually done by one person, in one place, at one time; the (primary) actor can normally go away happy as soon as this goal is completed.*

# Subfunction level

---

- Subfunction level use cases
  - ◆ provide “execution support” for user–goal level use cases; they are low–level and need to be justified, either for reasons of reuse or necessary detail

# User-goal and sub-function





# Right level

---

- Too-much levels
  - ◆ Very summary, cloud level
  - ◆ Sub-sub-functions (black)
- Tendency to be too specific
  - ◆ What does the primary actor really want?
  - ◆ Why is this actor doing this?

# Exercise

---

*“Jenny is standing in front of her bank's ATM. It is dark. She has entered her PIN, and is looking for the 'Enter' button.”*

- Name a summary, a user and a sub-function goal for Jenny.

# Answers

---

- ◆ Very Summary: Take some special one out for dinner.
- Summary: Use the ATM
- User goal: Get money from the ATM
- Sub-function: Enter PIN
  - ◆ Sub-Sub-function: Find the Enter button.

---

# USE CASE NARRATIVE

# Template

---

- SWEED template

- ◆ One column (no table)
- ◆ Sequenced: Numbered steps (Dewey decimal numbering system) and extensions to main scenario use alphabetic letters to differentiate from main steps

– Based on Cockburn “fully dressed” template

[http://diwww.epfl.ch/researchlgl/research/use\\_cases/RE-A2-case-studies/index.html](http://diwww.epfl.ch/researchlgl/research/use_cases/RE-A2-case-studies/index.html)

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.196.7782&rep=rep1&type=pdf>

# Template

---

- **Use Case:**
  - ◆ Name of the use case. This is the goal stated by a short active verb phrase.
- **Scope:**
  - ◆ The scope of the “system” being considered (black-/white- box and enterprise/system/component).
- **Level:**
  - ◆ Summary, User-goal, or Sub-function
- **Intention in Context:**
  - ◆ A statement of the primary actors intention and the context within which it is performed.

# Template

---

- **Primary Actor:**
  - ◆ The primary actor of the use case.
- **\*Support Actors:**
  - ◆ The optional secondary actors.
- **\*Stakeholders' Interests:**
  - ◆ The list of stakeholders and their key interests in the use case.
- **\*Precondition:**
  - ◆ What we can assume about the current state of the system and environment.

# Template

---

- **\*Minimum Guarantees:**
  - ◆ How the interests of the stakeholders are protected in all circumstances.
- **\*Success Guarantees:**
  - ◆ The state of the system and environment if the goal of the primary actor succeeds.
- **\*Trigger:**
  - ◆ What event starts the use case.



# Template

---

- **Main Success Scenario:**

- ◆ `<step_number> "." <action_description>`  
The numbered steps of the scenario, from trigger to goal delivery, followed by any clean-up.
- ◆ Conditions and alternatives are shown in the extension part.

- **\*Extensions:**

- ◆ `<step_altered> <condition> ":"  
<action_description> or <sub-use_case>`

# Main Success Scenario: Steps

---

- An interaction between two entities
  - ♦ Customer enters address
- A validation step to protect an interest of a stakeholder
  - ♦ System validate PIN code
- An internal change to satisfy a stakeholder
  - ♦ System deducts amount from balance

# Actions steps

---

- Use a simple grammar
  - ◆ *<Subject> <verb> <direct object>*  
*<prepositional phrase>*
- Show clearly “who has the ball”
- Written from a bird’s eye pov
- Shows the process progressing
- Shows intent not movement
- Contains a reasonable set of actions
- Doesn’t “check whether”, “validates”

# Action steps idioms

---

- Timing
  - ◆ “At any time...” or
  - ◆ “As soon as..”
- Trigger interaction with other system
  - ◆ “User has System A kick System B”
- Repeat
  - ◆ “Do steps x–y until condition”
- Parallel
  - ◆ “Steps x–y can happen in any order”

# Exercise

---

*Mary, taking her two daughters to the day care on the way to work, drives up to the ATM, runs her card across the card reader, enters her PIN code, selects FAST CASH, and enters \$35 as the amount. The ATM issues a \$20 and three \$5 bills, plus a receipt showing her account balance after the \$35 is debited. The ATM resets its screens after each transaction with FAST CASH, so that Mary can drive away and not worry that the next driver will have access to her account.*

- Write the main success scenario for the task-level use case "Withdraw money using FASTCASH option"

# Answer

---

1. Customer runs ATM card through the card reader.
2. ATM reads the bank id and account number from the card, validates them with the main computer.
3. Customer enters PIN.
4. ATM validates PIN.
5. Customer selects FASTCASH and withdrawal amount, a multiple of \$5.
6. ATM
  - a. notifies main banking system of customer account, amount being withdrawn, and receives back acknowledgement plus the new balance.
  - b. delivers the cash, card and a receipt showing the new balance.
  - c. logs the transaction.



# Extensions

---

- Brainstorm and include every possibility
  - ◆ Alternate success path
  - ◆ Primary actor behaves incorrectly
  - ◆ Inaction by the primary actor
  - ◆ Negative outcome of validation
  - ◆ Internal failure (expectable)
  - ◆ Unexpected or abnormal failure
  - ◆ Critical performance failure



# Extensions

---

- The condition expresses what was detected
  - Not what happened
    - “PIN entry time-out” **OK**
    - “Customers forgets PIN” **Wrong**
- The system must be able to detect the condition
- Merge equivalent conditions

# Template

---

- Steps can be nested. Dewey numbers are then used, e.g. 3a.1
- An extension always refers to a step of the Main Success Scenario.
- An extension step takes place in addition to the respective main step, notation: 2 ||,
- or as an alternative, notation: 2a.
- An extension might correspond to regular behavior, exceptional behavior that is recoverable, or unrecoverable erroneous behavior.

# Exercise

---

- Brainstorm and list the things that could go wrong during the operation of an ATM.

# Answer

---

- Card reader broken or card scratched
- Card for an ineligible bank
- Incorrect PIN
- Customer does not enter PIN in time
- ATM is down
- Host computer is down, or network is down
- Insufficient money in account
- Customer does not enter amount in time
- Not a multiple of \$5
- Amount requested is too large
- Network or host goes down during transaction
- Insufficient cash in dispenser
- Cash gets jammed during dispensing
- Receipt paper runs out, or gets jammed
- Customer does not take the money from the dispenser

# Precision stages

---

## 1. Actors and Goals

- ◆ Prioritized list of actors and their goals

## 2. Main success scenario

- ◆ Trigger and main success scenario
- ◆ Check meeting stakeholders interests

## 3. Failure conditions

- ◆ List of all possible failures

## 4. Failure handling

- ◆ How to respond to each failure
- ◆ Potentially revealing

# Applying use cases

---

- Define the scope
  - ◆ What's in and what's out
- Identify your actors:
  - ◆ who will be using the system?
- Identify their goals:
  - ◆ what will they be using the system to do?
- Write a use case brief
  - ◆ 2–6 sentences
- Detail the steps in main success scenario
  - ◆ 3–9 steps
- Identify failure conditions
  - ◆ Which conditions the system can detect?
- Define failure handling

# Warnings

---

- Use case alone are not the requirements
- Use cases should be based on some form of conceptual model or glossary
- Use cases should not include details about the user interface

# Typical errors

---

- Undefined or inconsistent system boundary
- Take System's viewpoint instead of Actor's
- Inconsistent Actor names
- Use cases refer to too many actors
  - ◆ Spider's web

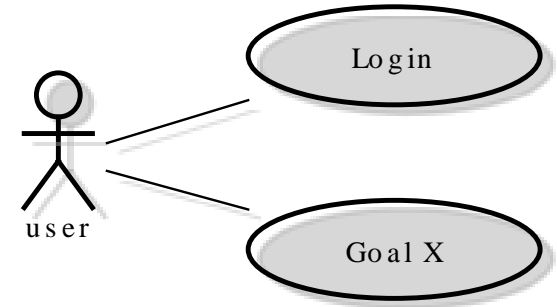
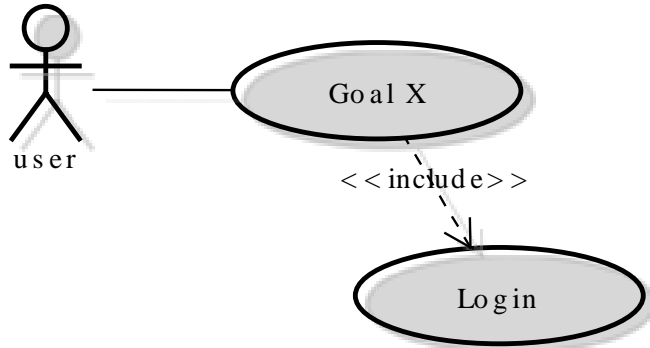


# Typical errors

---

- Too long specifications
- Confused specifications
  - ◆ Use of conditional logic
  - ◆ Attempt to describe algorithms
- Actor non fully entitled
- Customer not understanding
- Never ending use cases
- Dialog descriptions
  - ◆ Long, over-constrained, brittle

# Inclusion vs. Precondition



Use case: Goal X

...

Main success scenario:

...

1. User performs Login

...

Use case: Goal X

...

Precondition:

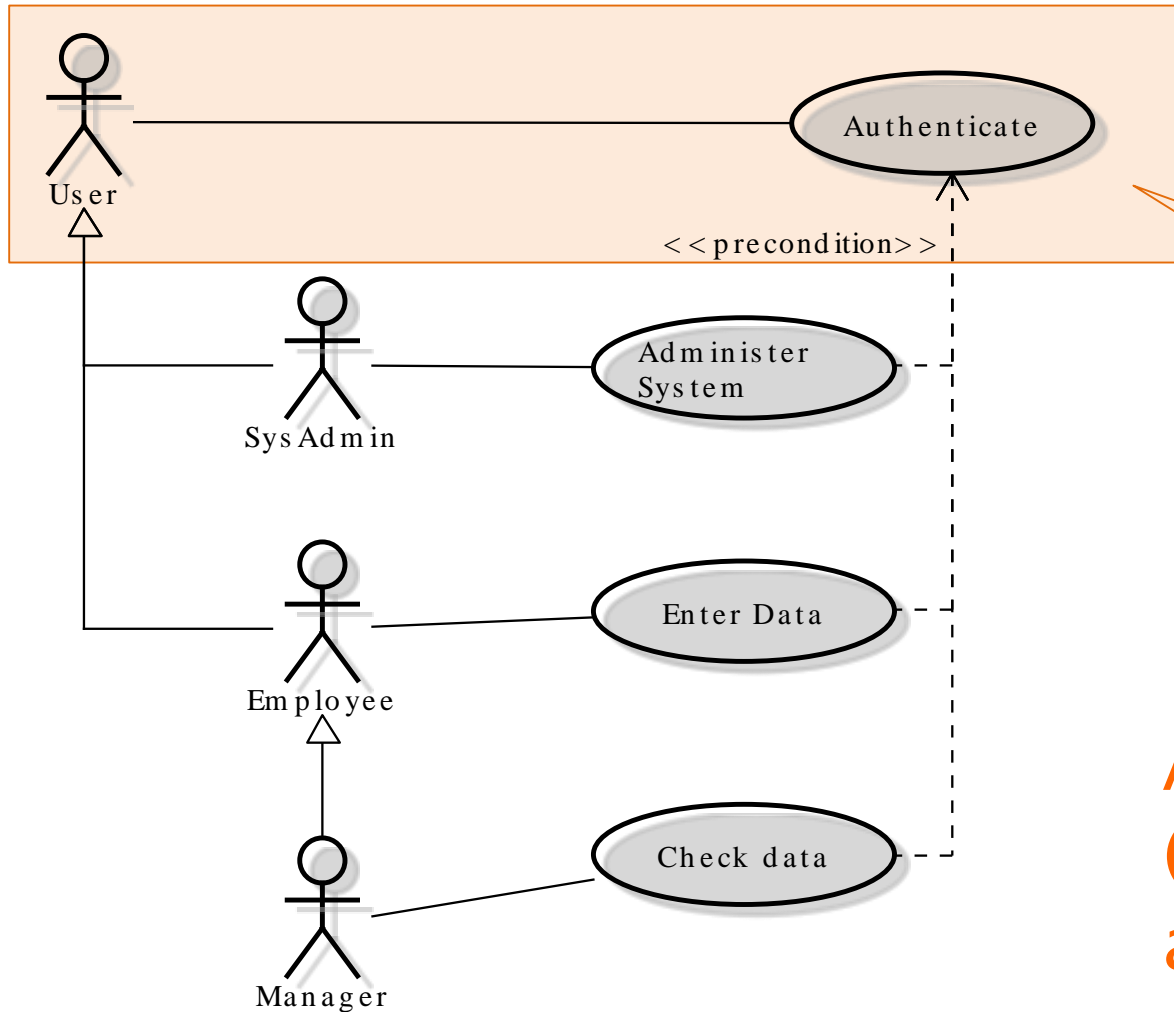
- User performed Login

...

Use case: Login

...

# Authentication



This part should be repeated for every UCD

Authentication (login) will be always implicit

---

# SUMMARY

# Pros

---

- Use case modeling is generally regarded as an excellent technique for capturing the functional requirements of a system.
- Use cases discourage premature design
- Use cases are traceable.
- Use cases can serve as the basis for the estimating, scheduling, and validating effort.

# Pros

---

- Use cases are reusable within a project.
  - ◆ The use case can evolve at each iteration from a method of capturing requirements, to development guidelines to programmers, to a test case and finally into user documentation.
- Use case alternative paths capture additional behaviour that can improve system robustness.

# Pros

---

- Use cases are useful for scoping.
  - ◆ Use cases make it easy to take a staged delivery approach to projects; they can be relatively easily added and removed from a software project as priorities change.
- Use cases have proved to be easily understandable by business users, and so have proven an excellent bridge between software developers and end users.

# Pros

---

- Use case specifications don't use a special language. They can be written in a variety of styles to suit the particular needs of the project.
- Use cases are concerned with the interactions between the user and the system.
  - ◆ UI designers can get involved in the development process either before or in parallel with software developers.



# Pros

---

- Use cases put requirements in context, they are clearly described in relationship to business tasks.
- Use case diagrams help stakeholders to understand the nature and scope of the business area or the system under development.

# Pros

---

- Test Cases (System, User Acceptance and Functional) can be directly derived from the use cases
- Use cases are critical for the effective execution of Performance Engineering

# Cons

---

- Use case flows are not well suited to easily capturing
  - ◆ non–interaction based requirements of a system (such as algorithm or mathematical requirements) or
  - ◆ non–functional requirements (such as platform, performance, timing, or safety–critical aspects)
- Use cases templates do not automatically ensure clarity. Clarity depends on the skill of the writer(s).

# References

---

- A.Cockburn, “Writing Effective Use Cases”. Addison–Wesley, 2000.
- M.Fowler, “UML Distilled” IV Edition. Addison–Wesley, 2010.

