Stockholms universitet

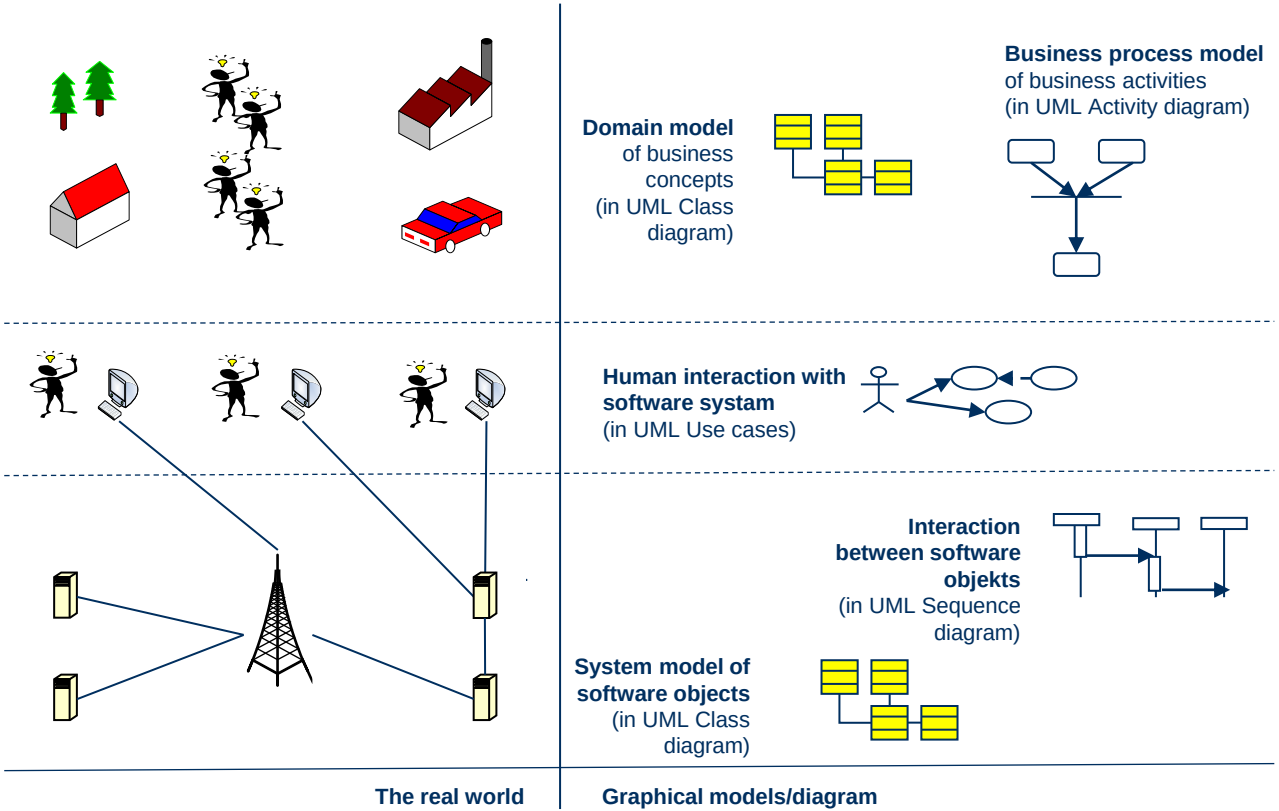
Requirement Modelling with UML Use Case

Erik Perjons

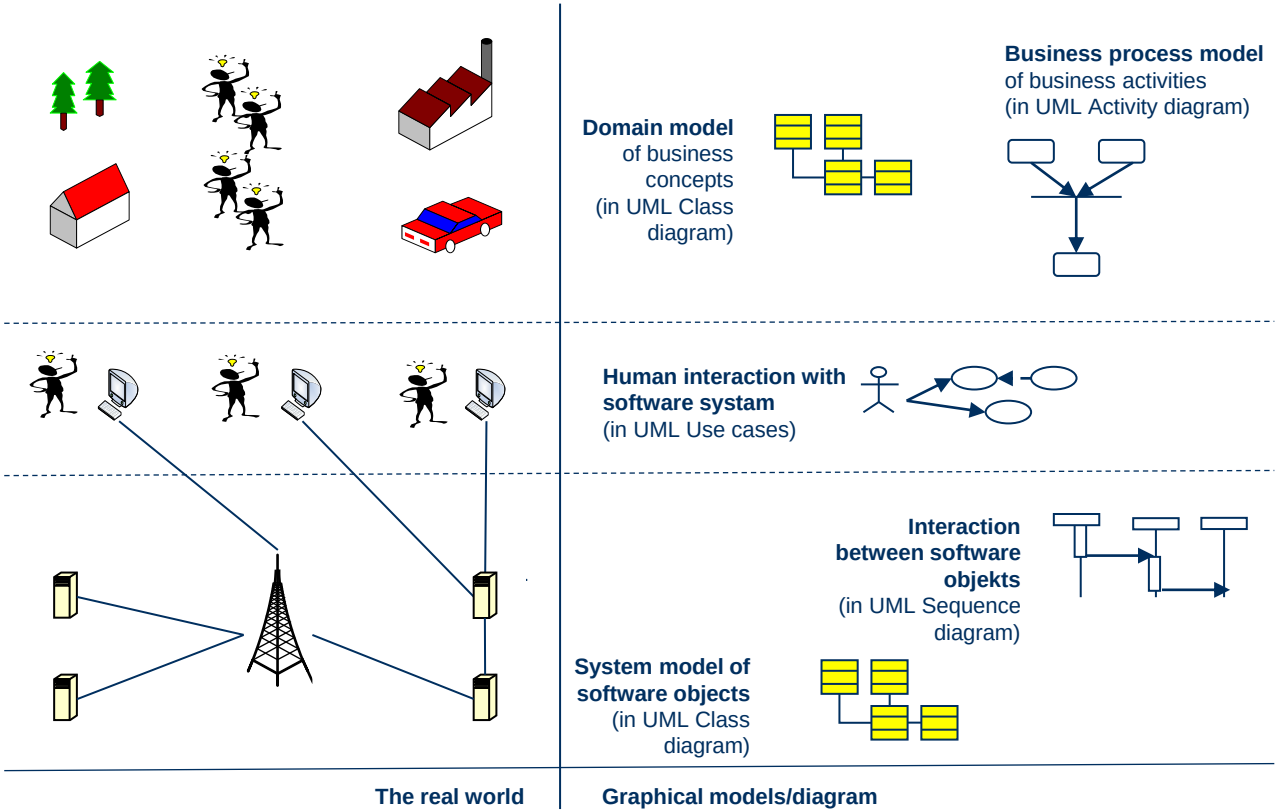
Questions to answer

- What is a requirement?
- What is a requirements specification?
- What is requirements engineering?
- Why are requirements important?
- What types of requirements exist?
- How are requirements captured and documented?
- How can UML use cases support requirements engineering?

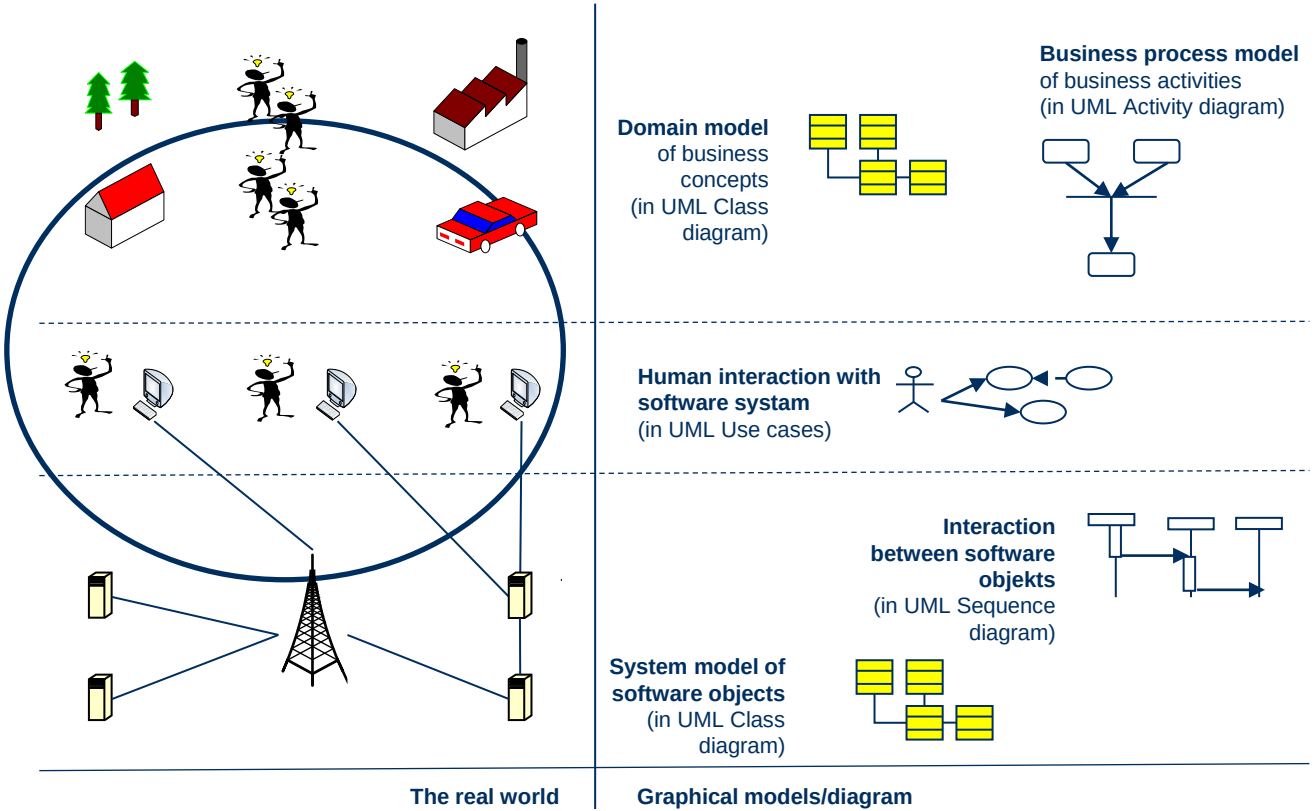
Real World and Models



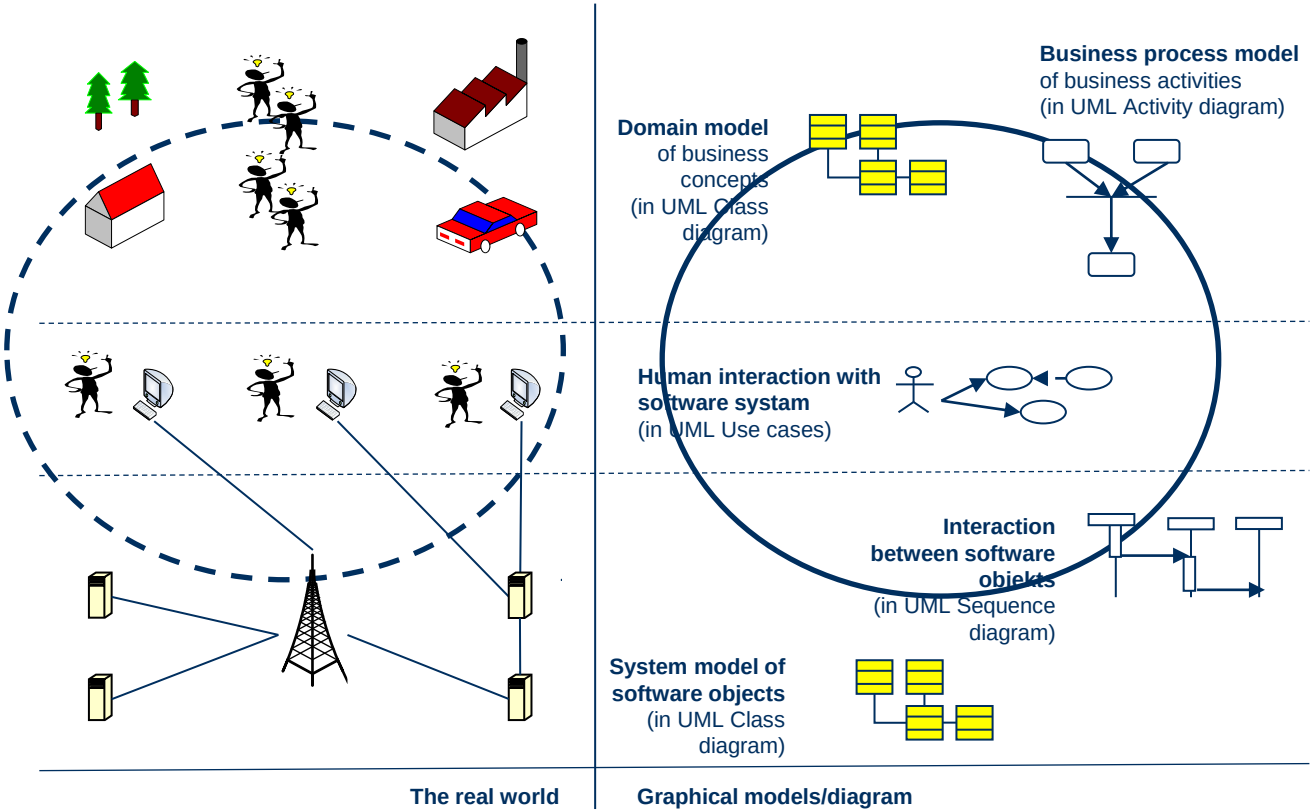
Real World and Models



Requirement Engineering (in Real World)



Requirement Engineering Models



Problems, Needs, and Requirements

- Requirements have their foundation in problems and needs.
- A **problem** can be seen as something undesirable — or you can also say — a problem is a gap between the current state and a more desirable state. A problem describes something that does not work, works poorly, or is inefficient. It focus on something negative in the current situation (as-is).
- **Examples of problems:**
  - Patients have to wait a long time in the phone queue to book appointments.
  - The electronic health record system is slow, causing staff to lose valuable time interacting with patients.

Problems, Needs, and Requirements

- A **need**, on the other hand, is the desired state that must be achieved in order to eliminate the problem gap. A need expresses a wish or expectation for the future, (while the problem focus on something negative in the current situation). Moreover needs should be described without pointing to a specific solution.
- **Examples of needs:**
  - Patients need to be able to book appointments quickly and easily
  - Staff need quick and easy access to patient information in order to work efficiently and provide high-quality care

Problems, Needs, and Requirements

- A **requirement** describes what a solution should do to meet a need, without prescribing too much in detail how this should be done. This latter is important because overly detailed requirements may lock the design too early, limit creativity, and prevent alternative details of a solution from being explored.
- **Examples of requirements**
  - The system shall allow patients to book appointments online through a self-service portal.
  - The health record system should support quick access to patient information by providing simplified navigation, clear search functions, and a minimal number of clicks to reach key data.

Problems, Needs, and Requirements

Stockholms universitet

- A **requirement** can be defined as a desirable property, feature, attribute, quality, or capacity of a solution, in our case an IT system. However, a requirement could also concern a process, a method, or something else, but in this presentation we focus on IT system requirements.

Summary of the differences: Problems, Needs, and Requirements

- A **problem** is often formulated as something negative in the current situation (*as-is*). It answers the question: *"What is not working today?"*
- A **need** is formulated as something desirable or as a possibility, without directly referring to the problem itself. In other words, it expresses a wish or expectation for the future (*to-be*). It answers the question: *"What should the user be able to do or experience instead?"*
- A **requirement** is a concrete statement of what the solution (e.g. the IT system) must deliver in order to meet the need. A requirement always relates to the solution (e.g. the IT system). It specifies what the solution must do, often in terms of functionality or quality. It answers the question: *"What shall the solution provide or enable, in practice?"*

Functional and Non-Functional Requirements

- Requirements are usually categorized into two main types: functional and non-functional.
- **Functional requirements** specify the functions the system should perform — essentially what the system should do. A good guideline is that a functional requirement requires the user of the system to act using the IT system, such as clicking, selecting, or writing something in the user interface using, for example, a mouse or keyboard.
- **Examples of functional requirements:**
  - The system shall be able to register an order.
  - The system shall be able to register a new customer.
  - The system shall be able to find a placed order in the system.

Functional and Non-Functional Requirements

- Requirements are usually categorized into two main types: functional and non-functional.
- **Functional requirements** specify the functions the system should perform — essentially what the system should do. A good guideline is that a functional requirement requires the user of the system to act using the IT system, such as clicking, selecting, or writing something in the user interface using, for example, a mouse or keyboard.
- **Examples of functional requirements:**
  - The system shall be able to register an order.
  - The system shall be able to register a new customer.
  - The system shall be able to find a placed order in the system.

Functional and Non-Functional Requirements

- **Non-functional requirements** specify how the system should perform these functions specified by the functional requirements. Non-functional requirements are about the overall usability, security, and performance of the system.
- **Examples of non-functional requirements:**
  - The system shall be easy to use.
  - The system shall support secure communication.
  - The system shall be able to handle 100 orders in parallel.
  - The system shall be able to integrate with Microsoft platform systems.

How to Formulate Requirements

Stockholms universitet

When writing requirements, it is important that they are:

- **Clear** – easy to understand.
- **Unambiguous** – only one possible interpretation.
- **Verifiable** – it should be possible to test if the requirement is fulfilled.
- **Consistent** – not conflicting with other requirements

In requirements engineering, we often distinguish between:

- **"Shall" requirements –** mandatory requirements, must be fulfilled.
- **"Should" requirements –** desirable requirements, nice-to-have, but not strictly mandatory.

How to Formulate Requirements

Stockholms universitet

Recommended form for formulate requirements:

1. Subject (the system or component): “The system …” or “The module …”
2. Modal verb (shall / should): “shall” for mandatory, “should” for recommended.
3. Action / property: description of what the system should/shall do

Examples:

- The system should provide a search history to simplify repeated searches (desirable (should) functional requirement)
- The system shall respond to search queries within two seconds (mandatory (shall) non-functional requirement)

Requirement specification

A **requirements specification** is the document where you actually specify the requirements of the system.

The core of this specification is the **list of functional and non-functional requirements**, but often it also **includes the context** in which these requirements apply: the problems to be addressed by the solution, the needs, the as-is and to-be processes that the IT system aims to support, the main concepts used in the organization, and the existing IT infrastructure, which the IT system needs to be part of.

Requirement Engineering

Stockholms universitet

Requirements engineering (RE) can be defined as the systematic process of eliciting, analyzing, documenting, validating, and managing the requirements of a system.

Key Activities in Requirements Engineering:

Elicitation – discovering and gathering requirements from stakeholders (for example using interviews, workshops, and observations).

Analysis & Negotiation – checking the requirements for conflicts, prioritizing the requirements, and refining them so they are clear, consistent, and feasible.

Specification – documenting requirements in a clear, structured, and testable way (including, for example, requirement lists, graphical models, and use cases).

Validation – ensuring that the requirements reflect the real needs of stakeholders and that they are correct and complete.

Management – tracking requirements over time, handling changes, and maintaining traceability

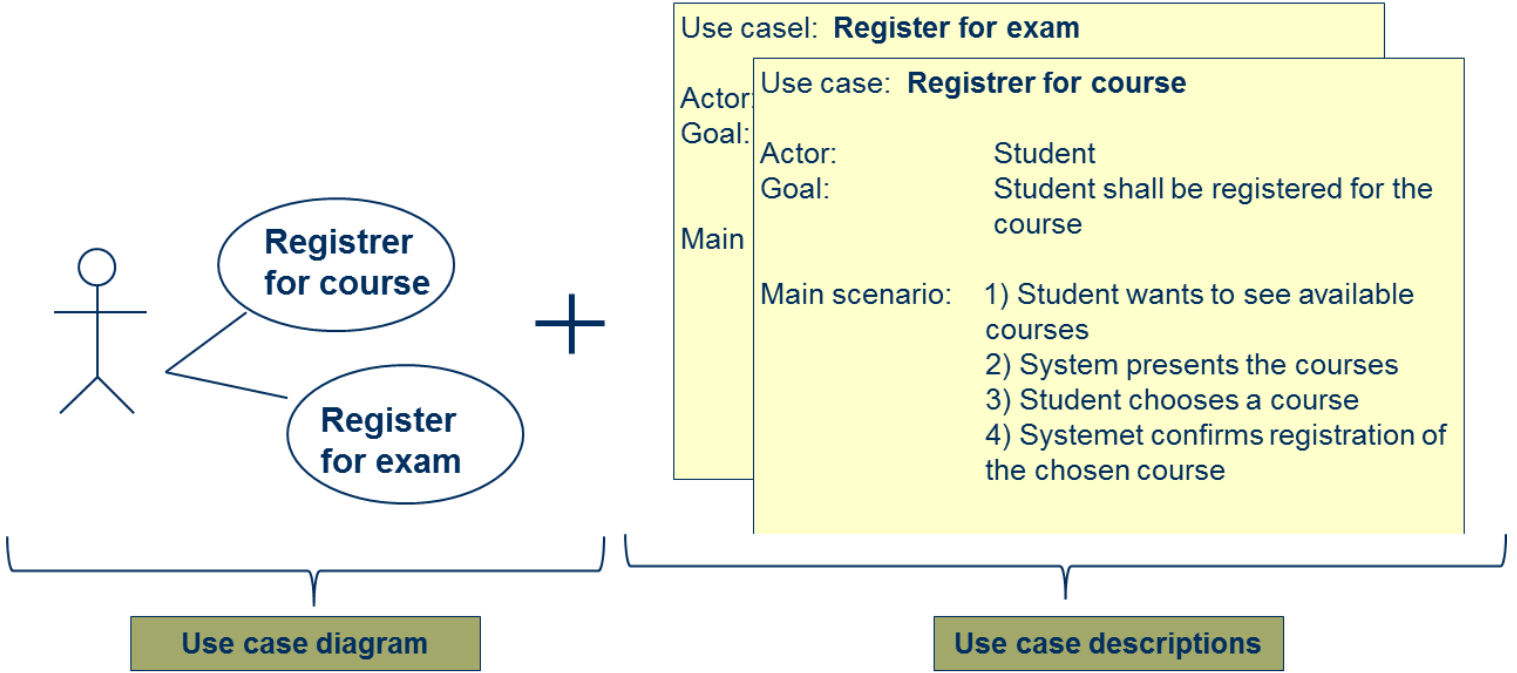
Why do we need RE?

Stockholms universitet

- Research has shown that a **major reason for failure in system development is shortcomings in requirements engineering** — very often because no or only a limited set of requirements are gathered, and not all stakeholders have been involved before development starts. This can result in a system that no user actually wants.
- Another problem is that **users often do not know what requirements they want** — it is difficult to identify requirements before the users have worked with the system in a real-world setting.
- Requirements are **central in system development**, and they usually drive the development process.

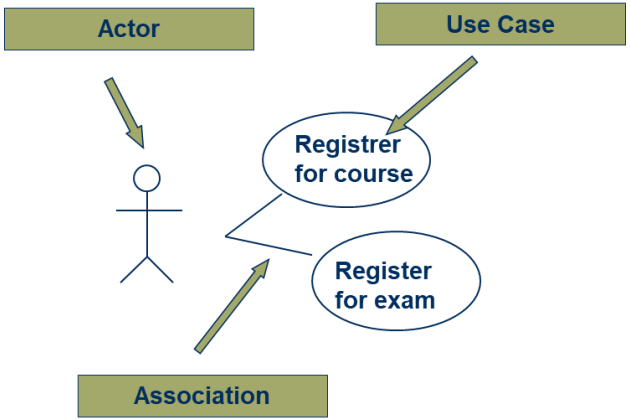
Use Case Model

- **Use case model –** consists of **Use case diagram** and **Use case descriptions**



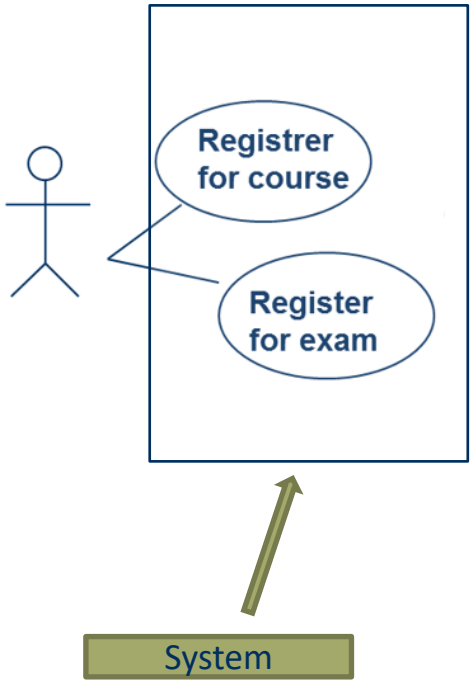
UML Use Case Diagram

- A use case diagram shows the **actors**, the **use cases,** and the **associations** between actors and use cases
- **An actor** is the user of the system. More precisely, an actor is a role played by someone or something, like in theater when an actor plays a certain role. But an actor could also be a software system.
- **A use case** is represented as an ellipse with a phrase, for example “Register for course” or “Register for exam.”
- **Associations** show which actors can perform which use cases



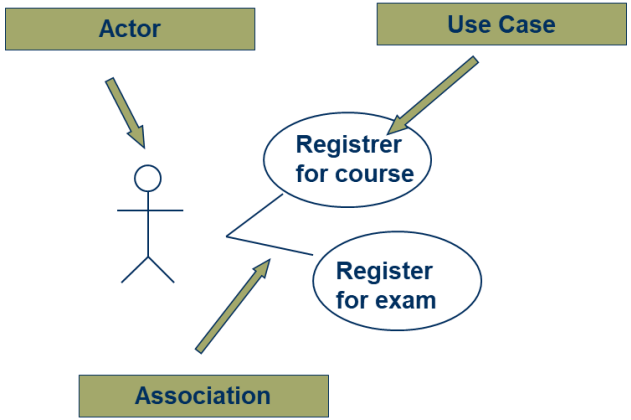
UML Use Case Diagram

- A use case diagram shows the **actors**, the **use cases,** and the **associations** between actors and use cases
- **An actor** is the user of the system. More precisely, an actor is a role played by someone or something, like in theater when an actor plays a certain role. But an actor could also be a software system.
- **A use case** is represented as an ellipse with a phrase, for example “Register for course” or “Register for exam.”
- **Associations** show which actors can perform which use cases



Use Case Description

- A use case description presents the interaction between the actor and the system. It includes the use case name, the actor, the goal, and the main scenario (step-by-step interaction).



Use case: **Registrer for course**

Actor: Student
Goal: Student shall be registered for the course

Main scenario:
1) Student wants to see available courses
2) System presents the cources
3) Student chooses a course
4) Systemet confirms registration

Use Case Diagram and Functional Requiremets

- **Use cases are not the same thing as functional requirements**
- Use cases show the interactions between the actor and the system that lead to a valuable result for the actor. They describe the functionality of the system from the user's perspective.
- **Use cases can**, therefore, **be seen as a technique to capture and structure functional requirements.**
- Use cases do not capture non-functional requirements.

Use Cases and Functional Requiremets

Example: Use Case: Book an Appointment

- **Actor:** Patient
- **Goal:** Schedule a medical appointment.
- **Main Scenario:**
  The patient selects a date and time.
  The system checks availability.
  The system confirms the booking.

Derived functional requirements from the use case:

- **High-level requirement (based on the use case name):**
  The system shall make it possible for the patient to schedule an appointment.
- **Detailed requirements (based on the steps in the scenario of the use case description):**
  The system shall allow the patient to select a preferred date and time for an appointment.
  The system shall check the availability of doctors for the selected time slot.
  The system shall confirm the booking and provide a confirmation number.

Use Case Description– Main scenario

- The main scenario should be divided into numbered steps.
- Each step must be a simple, concise statement of what is communicated between actor and system.
- The acting party (user or system) must be stated first in the step.
- Should not include XOR-splits; use a single scenario. Alternatives and exceptions go in Extension scenarios (see next page)

Use Case: Book Repair Appointment
Actor: Customer
Goal: A repair time is booked

Main scenario:

1) 1) The customer requests available time slots
2) 2) The system returns available time slots
3) 3) The customer selects one of the available time slots
4) 4) The system confirms the selected time slot

Extension scenarios:

2a) No time slots are available
1. The system informs the customer that no time slots are available
2. The system asks customer to select another function or leave system
3.The customer chooses to leave the system

2b) The system cannot access available time slots
1. The system asks the customer to try again or select other function
2. The customer tries again
3. The scenario continues from step 2 in the Main scenario

Use Case Description– Extension scenarios

- In addition to the main scenario (a single normal flow), a **use case description can also contain extension scenarios** (exceptions or alternative flow).
- These describe what happens when something goes wrong (exception) or when the actor chooses another path (alternative flow), see use case description (to the right).

Use Case: Book Repair Appointment
Actor: Customer
Goal: A repair time is booked

Main scenario:

1) 1) The customer requests available time slots
2) 2) The system returns available time slots
3) 3) The customer selects one of the available time slots
4) 4) The system confirms the selected time slot

Extension scenarios:

2a) No time slots are available
1. The system informs the customer that no time slots are available
2. The system asks customer to select another function or leave system
3.The customer chooses to leave the system

2b) The system cannot access available time slots
1. The system asks the customer to try again or select other function
2. The customer tries again
3. The scenario continues from step 2 in the Main scenario

Use Case Description – Extension scenarios

Use Case: Book Repair Appointment
Actor: Customer
Goal: A repair time is booked

Main scenario

1) *1) The customer requests available time slots*
2) 2) *The system returns available time slots*
3) *3) The customer selects one of the available time slots*
4) 4) *The system confirms the selected time slot*

Extension scenarios:

2a) No time slots are available (condition)
1. The system informs the customer that no time slots are available
2. The system asks customer to select another function or leave system
3.The customer chooses to leave the system

2b) The system cannot access available time slots (condition)
1. The system asks the customer to try again or select another function
2. The customer tries again to requests available time slots
3. The scenario continues from step 2 in the Main scenario

Condition for the first extension in step 2 of the main scenario (called 2a). That is, if the condition is true, the first extension is performed

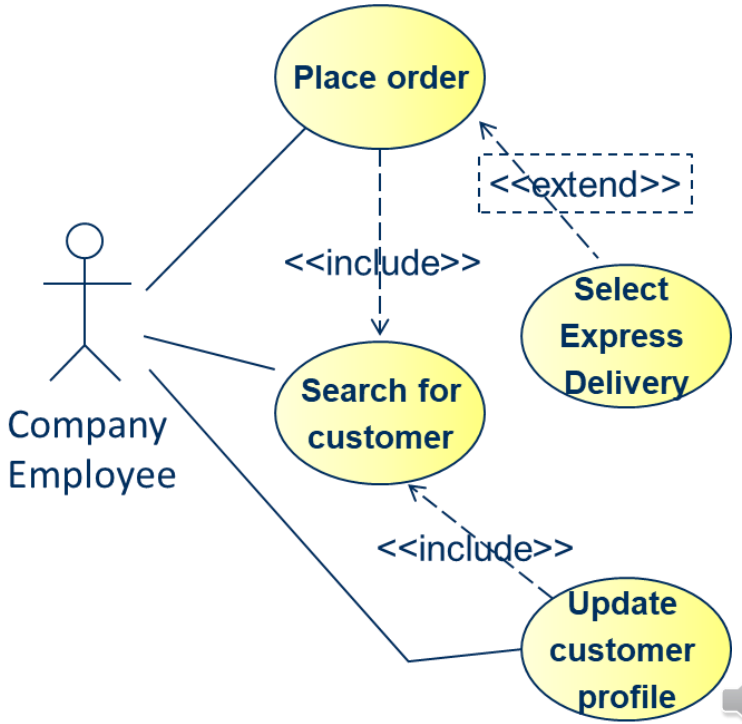
Step-by-step extension scenario

Condition for the second extension in step 2 (called 2b). That is, if the condition are true, the second extension is performed

Step-by-step extension scenario

Use Case Diagram - Include

- **Include use case** - can be used when several use cases share common behavior in parts of the use case scenarios.
- It is often common parts of scenarios, such as “Search for customer,” that can be extracted into a separate use case, and the arrow indicates where the include use case is needed.



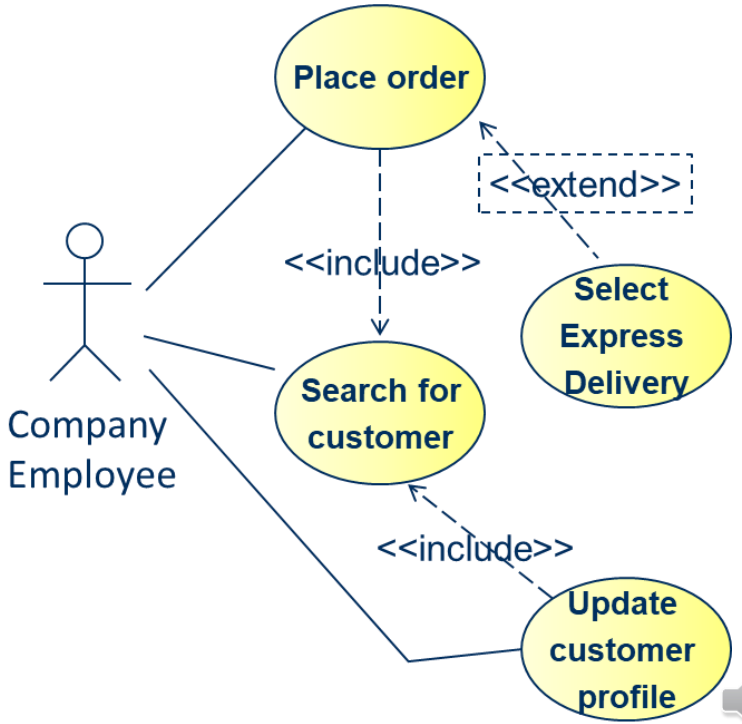
Use Case Diagram - Include

More precisely regarding include use cases:

- The included use case represents mandatory behavior.
- The base use case depends on the included use case.
- The included use case is always executed as part of the base use case.

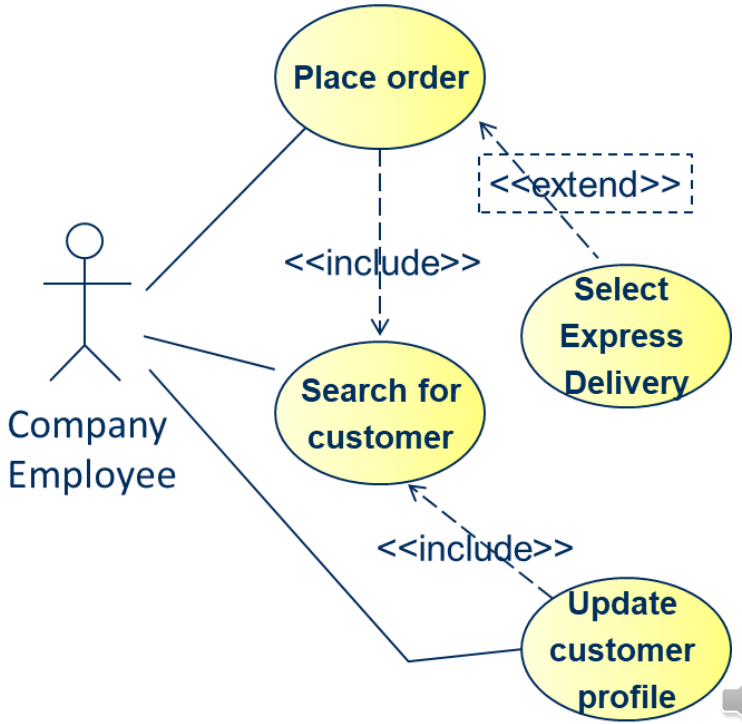
Why use include use cases?

- To reuse common parts of scenarios.
- To improve clarity and consistency.
- To reduce maintenance effort when requirements change.



Use Case Diagram - Extend

- **Extend use case** - can be used when a use case sometimes has additional optional behavior.
- The extend use case adds steps at a specific extension point in the base use case.



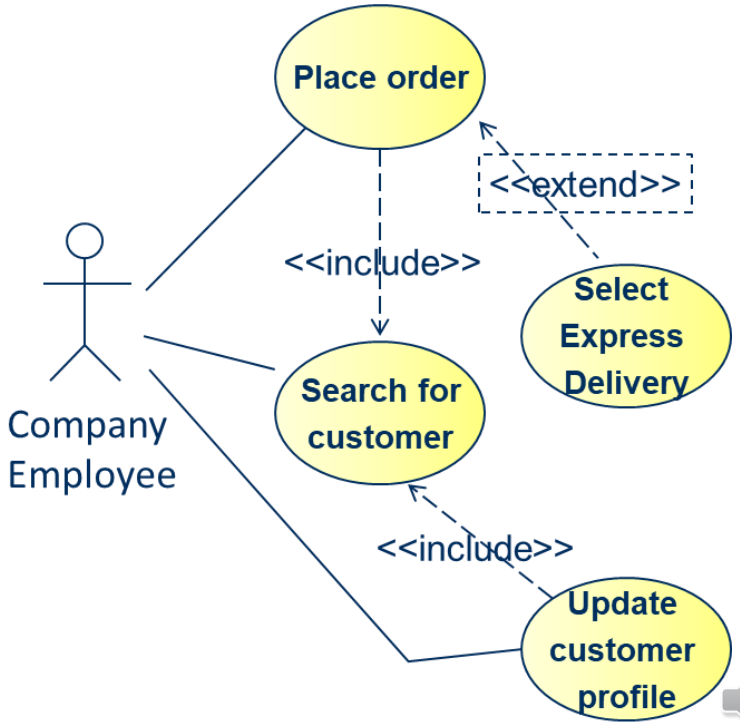
Use Case Diagram - Extend

More precisely regarding extend use cases:

- An extend relationship is used when a base use case can be extended with optional or conditional behavior.
- The base use case represents the main goal or main scenario.
- The extend use case adds behavior only under certain conditions and is attached at a specific extension point in the base use case.
- The base use case does not depend on the extension — it can run fully on its own

Why use extend use cases?

- To avoid overloading a use case with too many conditions and exceptions.
- To keep the base use case simple and focused on the "happy path."



Use Case Model – Include & Extend

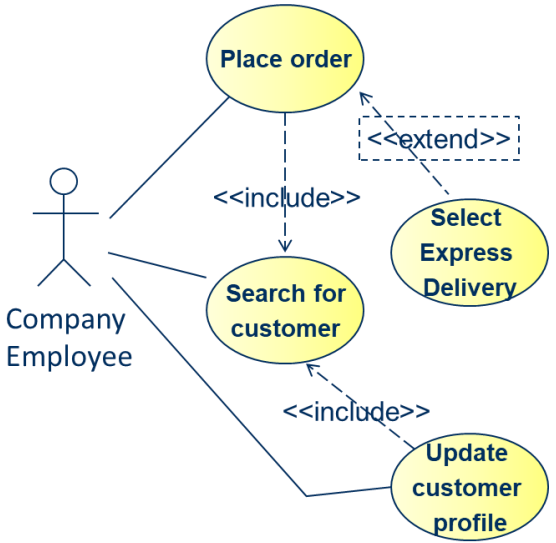
Stockholms universitet

Use Case: Place Order

Actor: Company Employee
Goal: To register a correct customer order in the system so that the products can be delivered to the customer.
Precondition: The customer has contacted the company (phone, e-mail, in-store, etc.) with an order request.

Main scenario:
1. Company Employee chooses to create a new order in the system.
2. Company Employee performs Search for Customer use case (include).
3. The system displays the customer's information.
4. Company Employee adds products and services to the order
5. The system calculates the total price.
6. Company Employee confirms the order with the customer.
7. The system registers the order and generates an order confirmation.

Extension scenarios:
4a) Customer requests express delivery.
1. Company employee carries out Select Express Delivery use case (extend).
2. Go to step 5 in the main scenario.



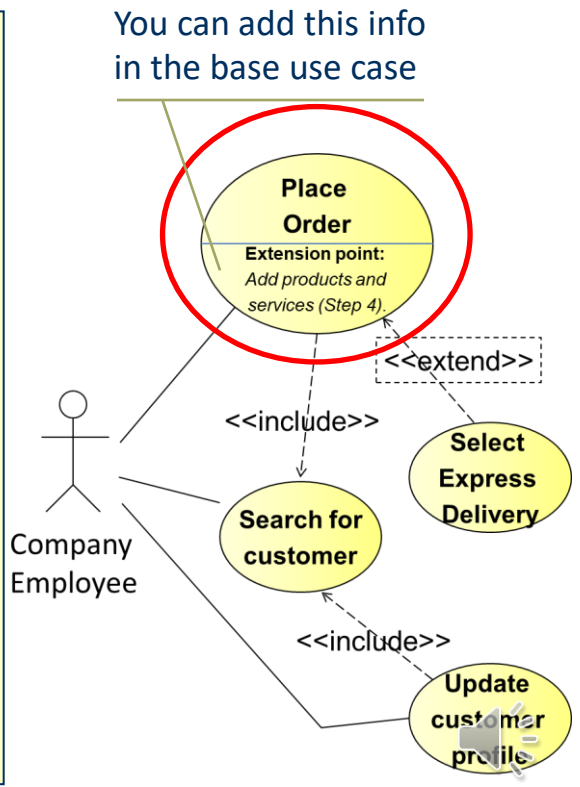
Use Case Model – Include & Extend

Use Case: Place Order

Actor: Company Employee
Goal: To register a correct customer order in the system so that the products can be delivered to the customer.
Precondition: The customer has contacted the company (phone, e-mail, in-store, etc.) with an order request.

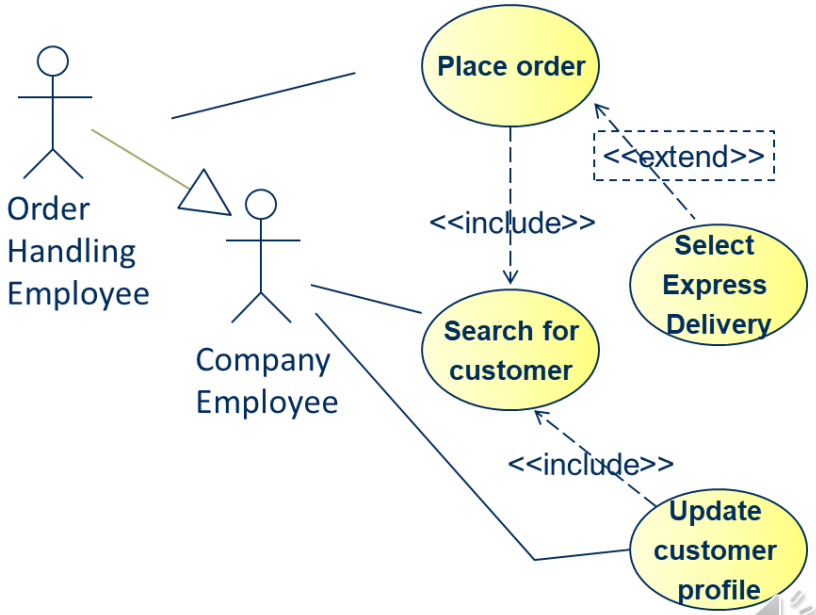
Main scenario:
1. Company Employee chooses to create a new order in the system.
2. Company Employee performs Search for Customer use case (include).
3. The system displays the customer's information.
4. Company Employee adds products and services to the order
5. The system calculates the total price.
6. Company Employee confirms the order with the customer.
7. The system registers the order and generates an order confirmation.

Extension scenarios:
4a) Customer requests express delivery.
1. Company employee carries out Select Express Delivery use case (extend).
2. The system updates the order with express delivery cost and details.



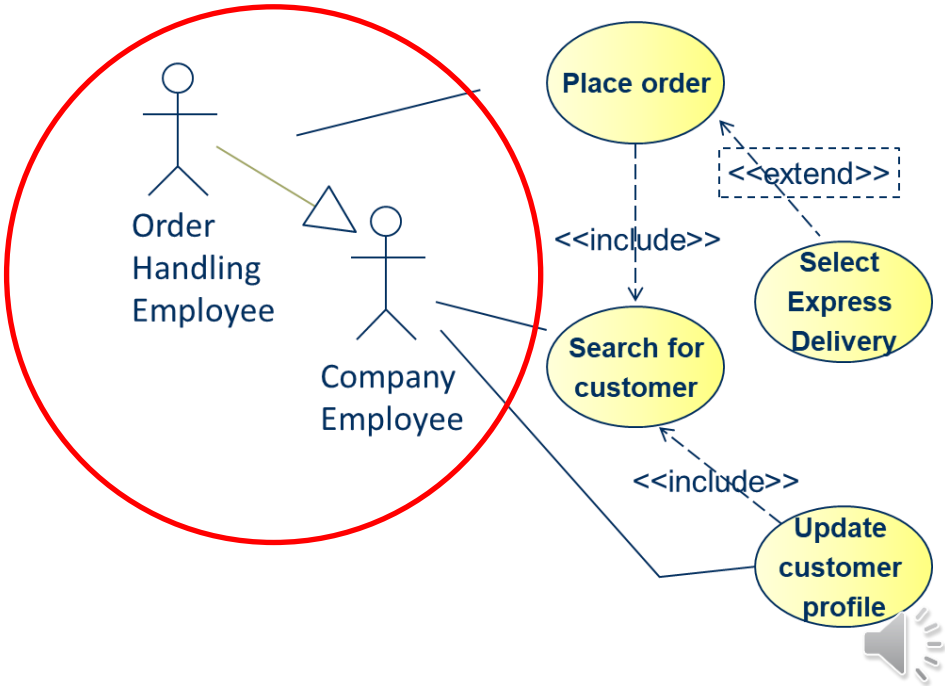
Use Case Diagram – Actor generalization/specialization

- Actors can be generalized or specialized.
- A specialized actor inherits the behavior of its general actor but may also have additional interactions.



Use Case Diagram – Actor generalization/specialization

- Actors can be generalized or specialized.
- A specialized actor inherits the behavior of its general actor but may also have additional interactions.



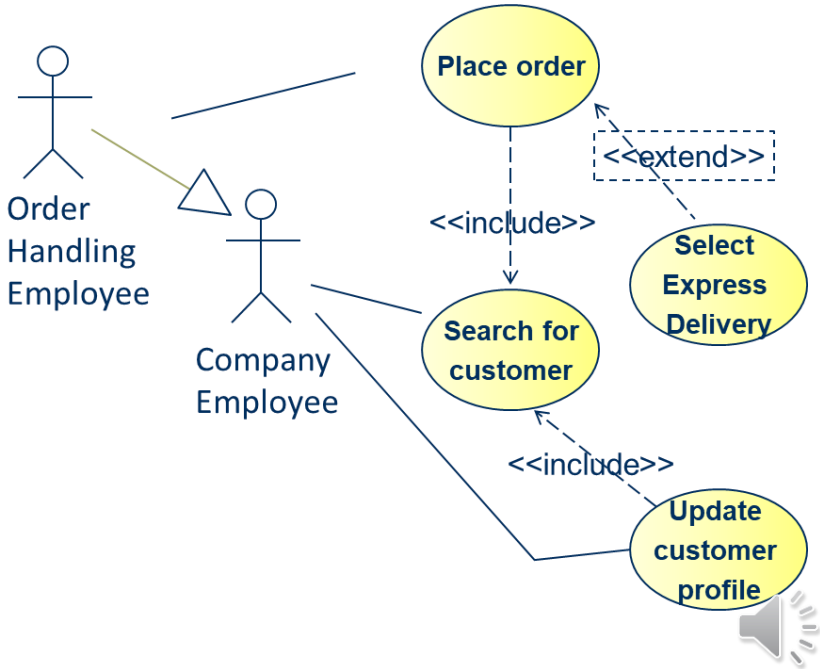
Use Case Diagram – Actor generalization/specialization

More precisely regarding actor generalization/specialization:

- A specialized actor inherits the interactions of its general actor.
- A specialized actor may add new use cases that the general actor cannot perform.
- Generalization reduces duplication in diagrams by grouping common behavior.

Why use actor generalization/specialization?

- To capture both common and specific interactions.
- To simplify diagrams when there are many actors.
- To make organizational roles and responsibilities clearer.



Questions to answer

- What is a requirement?
- What is a requirements specification?
- What is requirements engineering?
- Why are requirements important?
- What types of requirements exist?
- How are requirements captured and documented?
- How can UML use cases support requirements engineering?