3rd Lecture – Identification of computer system requirements

Agenda

- ✓ Requirement engineering
- ✓ Classes of computer system requirements
- ✓ Non-functional requirements
- ✓ Techniques for requirement identification
- ✓ Quality features of the requirements

- ✓ UML use case diagram
- ✓ Use case description using a template

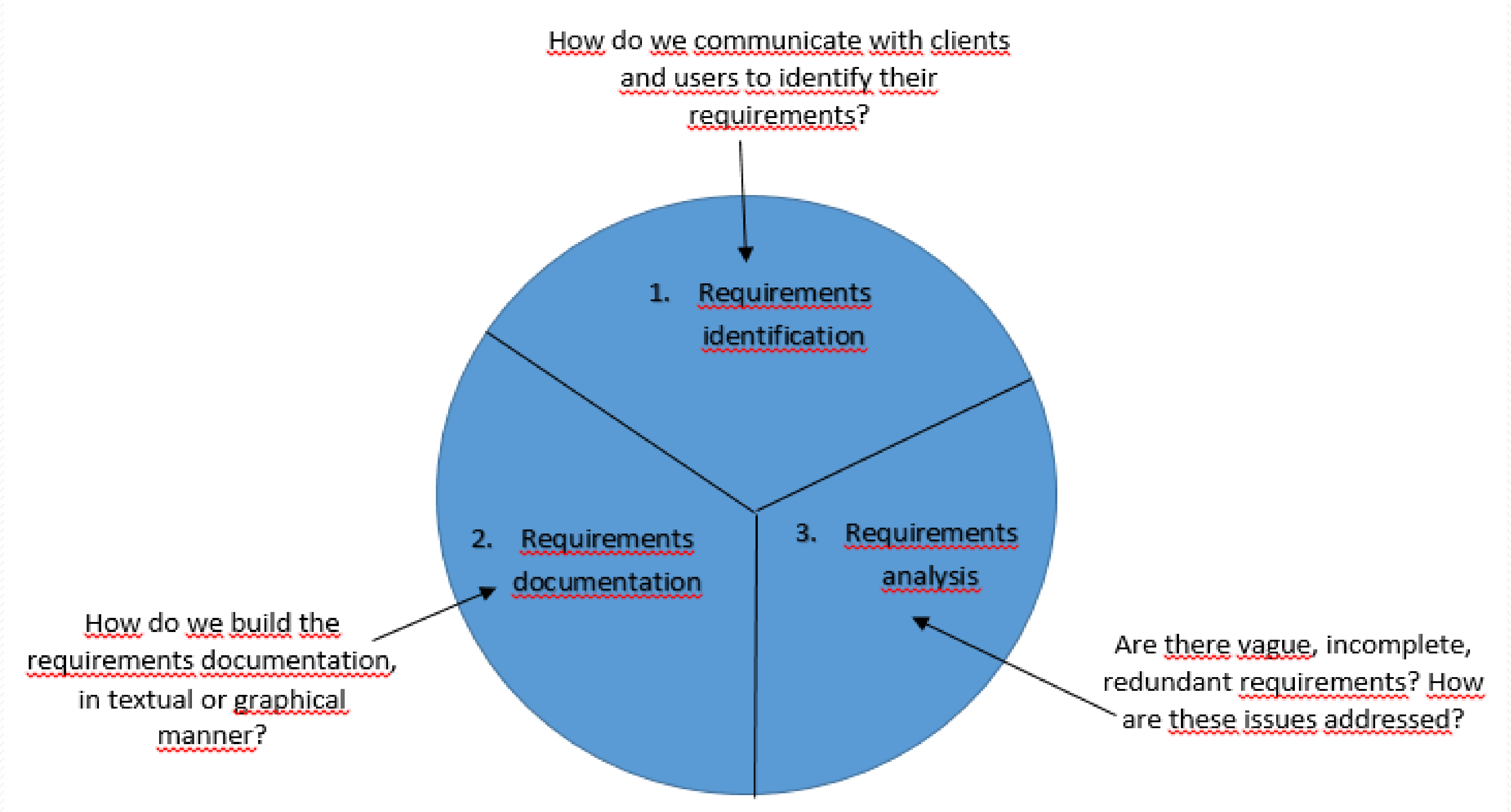
The background

- Poor quality requirements is consistently ranked first in the hierarchy of causes leading to the failure of software projects.
- One explanation would be that development teams allocate too little time to understanding the
  - *real problems of business*,
  - *user needs* or
  - *nature of the environment* in which the system will run.
- Also, developers are trying to provide technical *solutions as quickly as possible*, but starting from an insufficient understanding of the requirements of the problem.

Requirements engineering

- To be sure that a computer system correctly solves a specific problem, you first need to **understand correctly the problem to be solved**. This requires the discovery, understanding, specifying and analyze of the following components:
  - WHAT problem should be solved;
  - WHY the problem needs to be solved;
  - WHO is involved and is going to be responsible with this problem solving.
- These three components constitute the foundation of ***Computer System Requirement Engineering***

Requirements engineering


How do we communicate with clients and users to identify their requirements?
1. Requirements identification
2. Requirements documentation
3. Requirements analysis
How do we build the requirements documentation, in textual or graphical manner?
Are there vague, incomplete, redundant requirements? How are these issues addressed?

Core activities for building a requirement model

Requirements engineering

- Often, difficulties in requirement modeling and analysis come from insufficient understanding of the business logic part of the application.
- *Business logic* is the defining element for the modeling and automation process. It includes both business rules and the business workflow (processes) that describes the way of transferring documents and data from one participant (individual or software system) to another.
- In developing information systems, business logic aims at:
  - *Modeling* real world *business objects* (such as inventories, customers, products);
  - Managing *business object storage solutions* (business objects mapping into the database tables);
  - Describing *how business objects interact with each other*.

Requirements engineering

- In developing information systems, **a requirement** defines an objective that it must be fulfilled, in response to the needs of its users.
- It is part of the general purpose for which the computer system is developed.
- The requirements also dictate the way the system should respond to user interaction.
- Generally, developers should consider the following aspects related to the requirements system :
  - use of a technical language: requirements must always be specified using the **user's language**. The **jargon** of the analyzed field can also be listed and analyzed;
  - relationship with the business goals: each requirement must be clearly linked to business objectives.

Types of information systems requirements

- Requirements engineering process aims to collect, develop, correct and adjust large volumes of specifications which differ in terms of purpose and mode of expression. Differentiations can be made between:
  - *Descriptive requirements;*
  - *Prescriptive requirements.*
- *Descriptive requirements* display properties the system should have, regardless of how it will work. Such properties are generated usually by laws of nature or physical constraints. Examples of descriptive specifications:
  - ❑ Same book can not be borrowed by two subscribers simultaneously.
  - ❑ If a hotel room is under renovation, then it can not be occupied.

Types of information systems requirements

- *Prescriptive requirements* display some properties the system should have, that will be or will be not met by the system, <u>depending on the way the system will work</u>. Exemples of prescriptive specifications:
  - A subscriber cannot borrow more than three book at the same time
  - Goods have to be delivered within the time interval specified by the buyer.

- Distinction between descriptive and prescriptive requirements is very important in the context of requirements engineering, as we can negotiate, change or find alternatives to prescriptive specifications, while for the descriptive specifications these things are not possible.

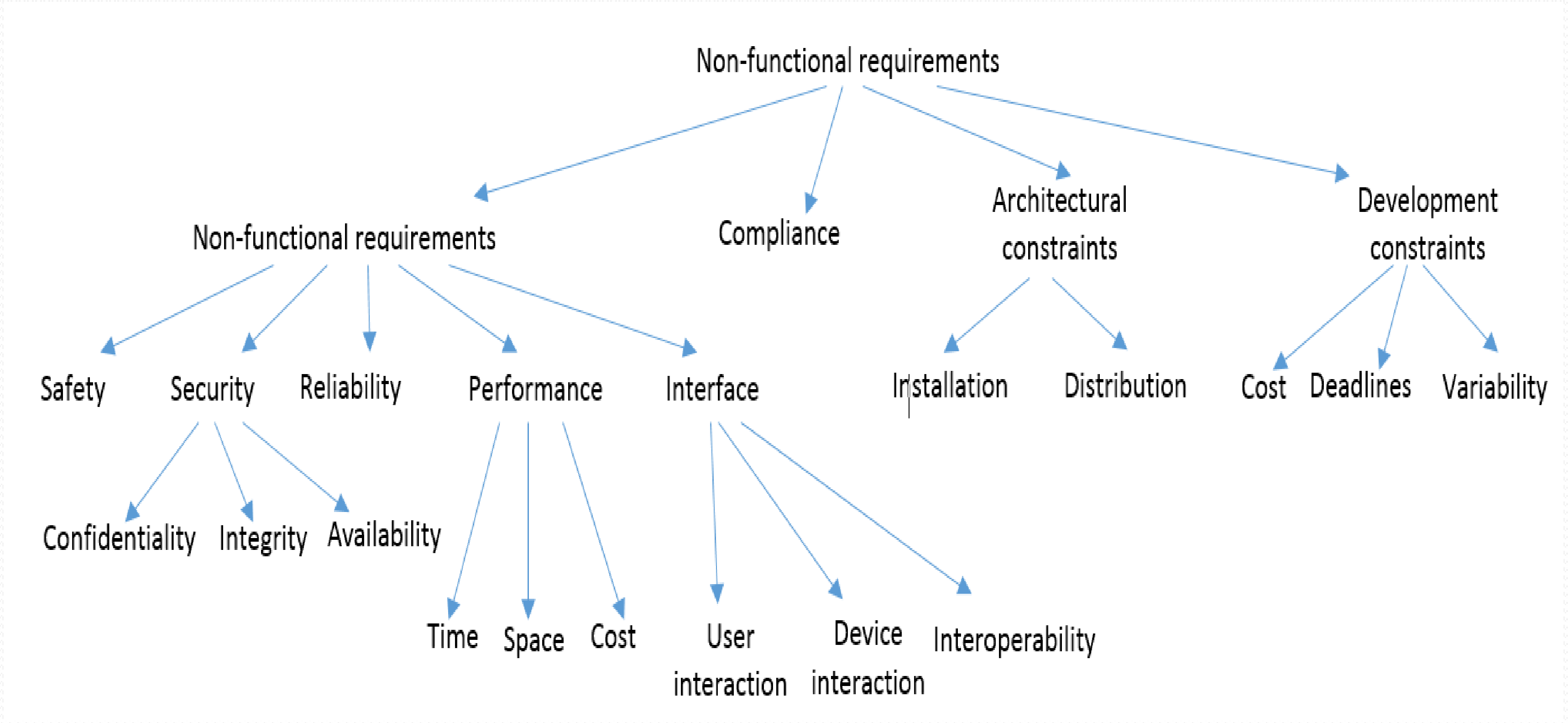
Types of information systems requirements

- Considering their functionalities, requirements can be:
  - *functional;*
  - *non-functional.*
- Functional requirements define the functions of a computer system or of its components, through a set of inputs, behavior and a set of outputs. This category includes:
  - computations,
  - technical details,
  - information regarding data processing and manipulation,
  - functionalities, for example, the way a use case has to be built.
-

Types of information systems requirements

- Non-functional requirements capture criteria that can be used to analyze aspects of the system operation and not its behavior.
- These impose constraints on the functional requirements at design or implementation level (e.g. performance, security or reliability requirements)
- Non-functional requirements are usually referred as ***qualities of the system***, but also as *quality attributes*, *quality objectives, quality features* or *constraints*.

Non-functional requirements



Types of non-functional techniques

Non-functional requirements

A. Quality requirements define issues related to "how well" the system should operate. These include specifications related to:

- **Safety requirements** are quality requirements for preventing occurrence of accidents or environmental degradation.
- **Security requirements** describe system safeguards against undesirable behavior.
  - *Confidentiality requirements* indicate that certain information can not be disclosed to unauthorized parties.
  - *Integrity requirements* indicate that some information can be changed if this was done in a fair and authorized manner.
  - *Availability requirements* indicate that certain requirements and resources can be used by authorized persons when necessary.

Non-functional requirements

A. Quality requirements (continued)

- **Reliability requirements** constraint the computer system to work as expected and desired for long periods of time. Aside from exceptional circumstances, the system must provide services in a fair and robust manner.
- **Performance requirements** put conditions on the system operation manner, for example the maximum time required for executing an operation.
- **Interface requirements** indicate the way the computer system interacts with the environment, the users, and with other systems.

Non-functional requirements

B. **Compliance requirements** impose the necessary conditions so that the system operates in accordance with national laws, international regulations, social norms, political and cultural constraints, standards etc.

C. **Architectural requirements** impose structural constraints on the computer system, so that it can work properly in the environment where it will be implemented. It provides developers with guidance on the suitable type of architecture for the system.

D. **Development requirements** are non-functional requirements that impose the way the computer system should be developed, and not the way the system should meet the functional requirements (time limitations, resource availability).

Techniques for requirement identification

1. Interviews

- Interviewing the beneficiary is the most popular method for requirement identification. Its success depends on the involvement of all stakeholders, including managers, stockholders, employees, etc. that will further be called **general users.**
- Normally, an interview will focus on activities of the user within the organization, the way the system implementation will influence his work and the issues that he has identified in the actual processes that take place in the organization. The interview can reveal new requirements that were not originally identified as part of the project objectives, but also conflicting requirements.

Techniques for requirement identification

1. Interviews - continued

- The identification of conflicting requirements may be confusing, as it does not seem normal that distinct persons working in the same company have different visions on the same analyzed aspect.

The analyst may ask himself the following pertinent question : *"How can a business be competitive and profitable if there is no consensus, at least within it, on how it works?"*

A possible answer would be: *"...the detail level necessary for building a computer system is higher that the detail level necessary for successfully running a business."*

Techniques for requirement identification

2. Join Requirement Planning Sessions

- Join Requirement Planning - JRP can be equivalent with conducting interviews of all users (or at least a substantial part thereof) at the same time and in the same room.
- All people that shape the direction of system development are brought together in one place to provide details about what the system should do.
- There should be present:
  - a mediator to moderate these sessions, but also
  - a person to document user proposals, using, for example, a video projector and a visual modeling software.

Techniques for requirement identification

3. Use cases

- Use cases describe interactions between users and computer system and show **what** users should do with the system by identifying the **important functions**.
- A use case specifies the sequence of interactions between system and external actors (such as persons, hardware devices or other software systems), including also **versions or extensions** of the main behavior of the system.
- Use cases can be one of the first forms of representation of the requirements of a computer system. For this reason, they are suitable for the early stages of the software development process. Analysts together with beneficiaries of the system will have to **examine and validate** every proposed use case.

Techniques for requirement identification

4. Social observation and analysis

- Observation based methods imply the existence of an observer that watches the system users and records information on the work they perform.
- Social observations can by direct observations, involving observer presence during activity performance or indirect observations, when user activity is watched through other means, such as a video recording.
- This method offers the advantage of facilitating the collection of quality data and it is useful for analyzing real processes and activities. This way, **errors or omissions** coming from the beneficiary description of the workflow **can be avoided**.

Techniques for requirement identification

5. Prototyping

- **A prototype** is useful for **the end user** who will understand better what he wants or what is expected from the product. It is also useful for **developers** to test some techniques, developed algorithms, interfaces etc.
- There are two approaches:
  - **Testing/trial prototype** – it must be **fast**, if **not perfect**, it can be used to validate the interface, to customize the architecture to include requirements as well as possible, or to validate specific algorithms;
  - **Evolutive prototype** – it develops the final product in order to be able to consider **all the quality features** of the final software product; Generally this prototype may <u>not be fast</u>, but can improve the model by ensuring **high quality of the software**.

Use case UML diagram

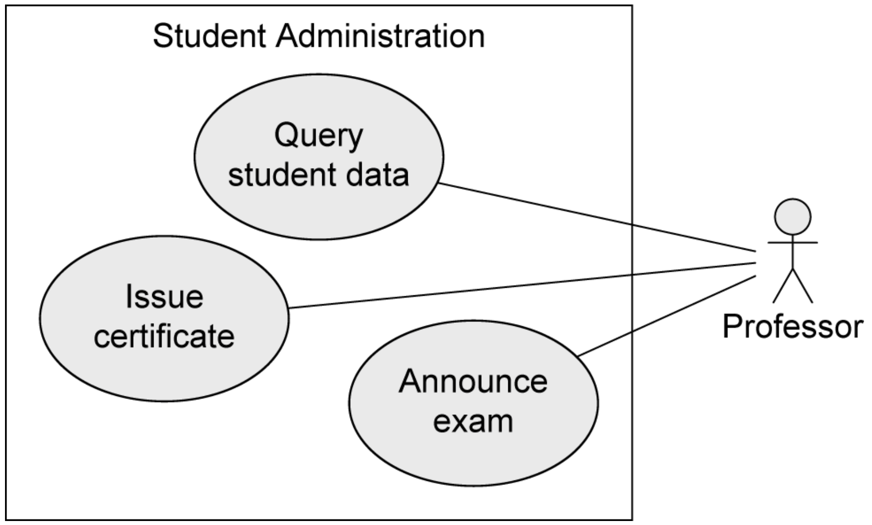
- Its role is to represent in graphical format the functionalities that have to be met by the computer system in the final stage.
- The model developed by use case diagrams and documents that briefly describe each use case is called REQUIREMENTS MODEL.
- Use case diagrams are made up of actors and use cases, on one side, and the relationships between them, on the other side.

Introduction

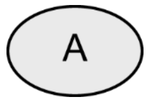
- The use case is a fundamental concept of many object-oriented development methods.
- Use case diagrams express the expectations of the customers/stakeholders
  - essential for a detailed design
- The use case diagram is used during the entire analysis and design process.
- We can use a use case diagram to answer the following questions:
  - What is being described? (The system.)
  - Who interacts with the system? (The actors.)
  - What can the actors do? (The use cases.)

Example: Student Administration System

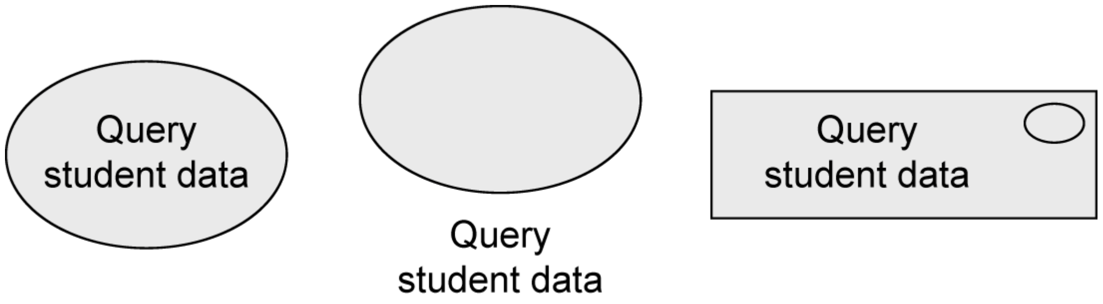
- **System**
  (what is being described?)
  - Student administration system
- **Actors**
  (who interacts with the system?)
  - Professor
- **Use cases**
  (what can the actors do?)
  - Query student data
  - Issue certificate
  - Announce exam



Use Case

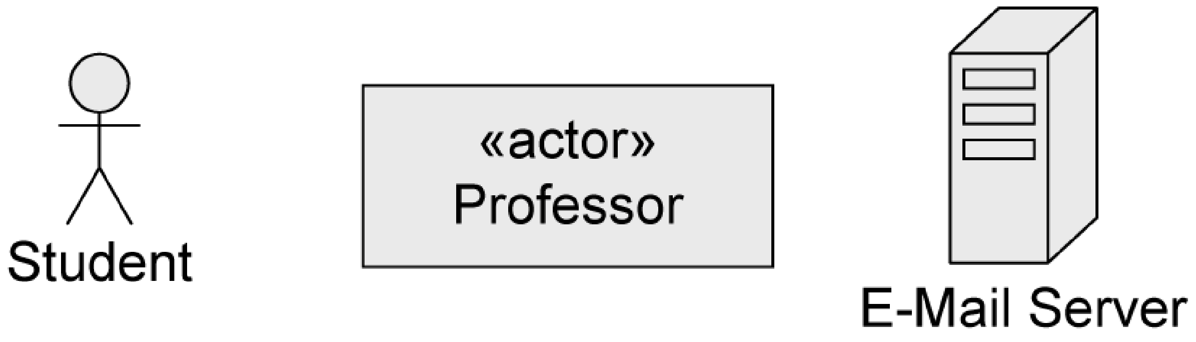


- Describes functionality expected from the system under development.
- Provides tangible benefit for one or more actors that communicate with this use case.
- Derived from collected customer wishes.
- Set of all use cases describes the functionality that a system shall provide.
  - Documents the functionality that a system offers.
- Alternative notations:



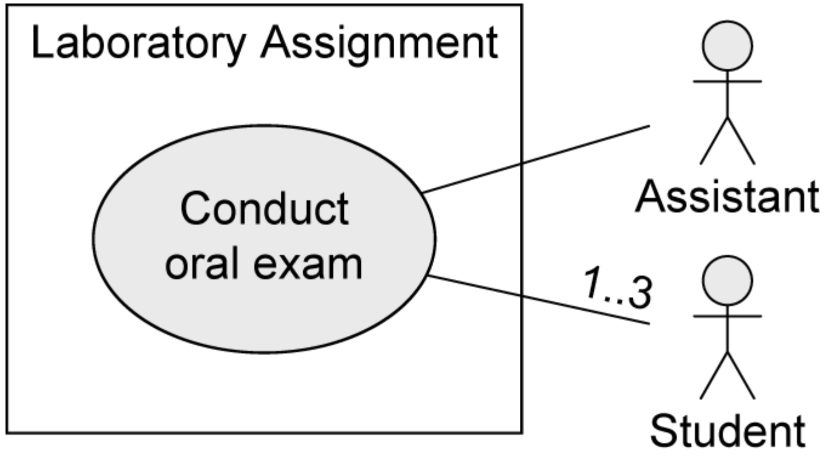
Actor (1/3)

- Actors interact with the system …
  - by **using** use cases,
    i.e., the actors initiate the execution of use cases.
  - by **being used** by use cases,
    i.e., the actors provide functionality for the execution of use cases.
- Actors represent roles that users adopt.
  - Specific users can adopt and set aside multiple roles simultaneously.
- Actors are not part of the system, i.e., they are outside of the system boundaries.
- Alternative notations:



Actor (2/3)

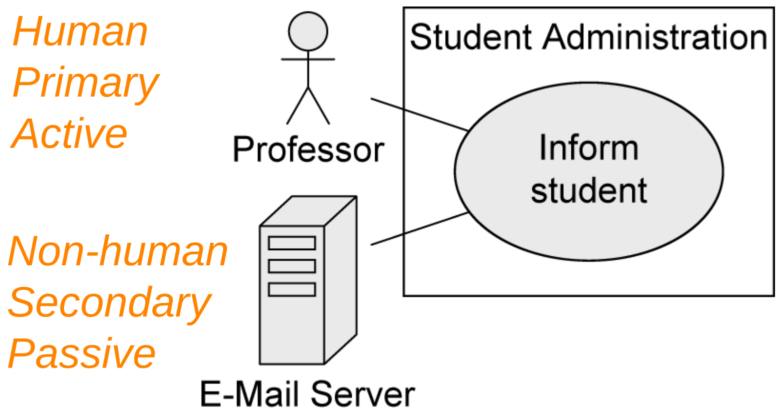
- Usually user data is also administered within the system. This data is modeled within the system in the form of objects and classes.
- Example: actor **`Assistant`**
  - The actor **`Assistant`** interacts with the system **`Laboratory Assignment`** by using it.
  - The class **`Assistant`** describes objects representing user data (e.g., name, ssNr, …).

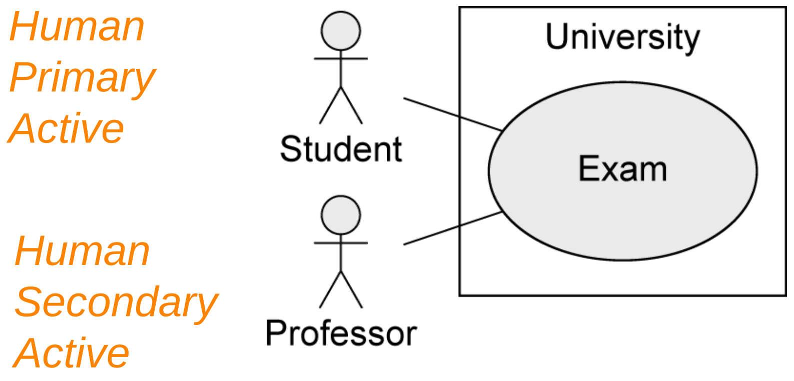


Actor (3/3)

- **Human**
  - E.g., **`Student`**, **`Professor`**
- **Non-human**
  - E.g., **`E-Mail Server`**
- **Primary**: has the main benefit of the execution of the use case
- **Secondary**: receives no direct benefit
- **Active**: initiates the execution of the use case
- **Passive**: provides functionality for the execution of the use case

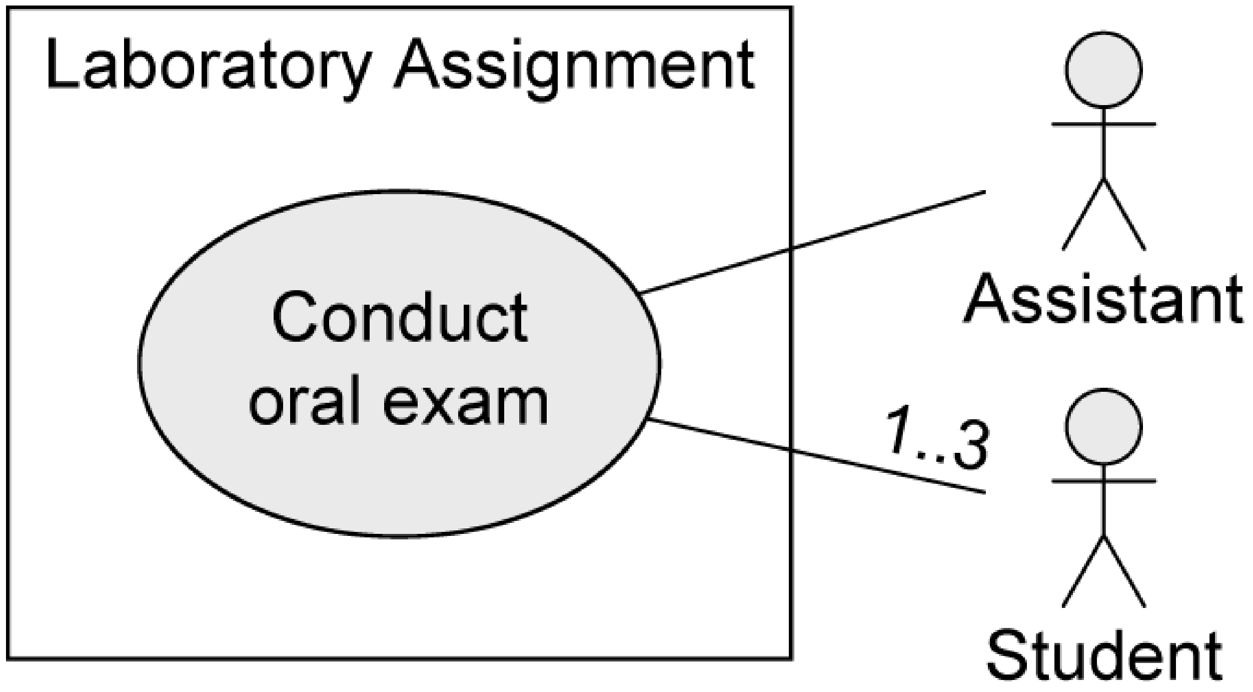
- Example:





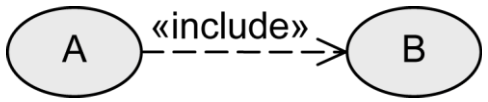
Relationships between Use Cases and Actors

- Actors are connected with use cases via solid lines (*associations*).
- Every actor must communicate with at least one use case.
- An association is always binary.
- Multiplicities may be specified.

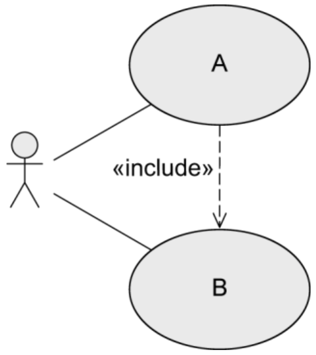


Relationships between Use Cases

«include» - Relationship



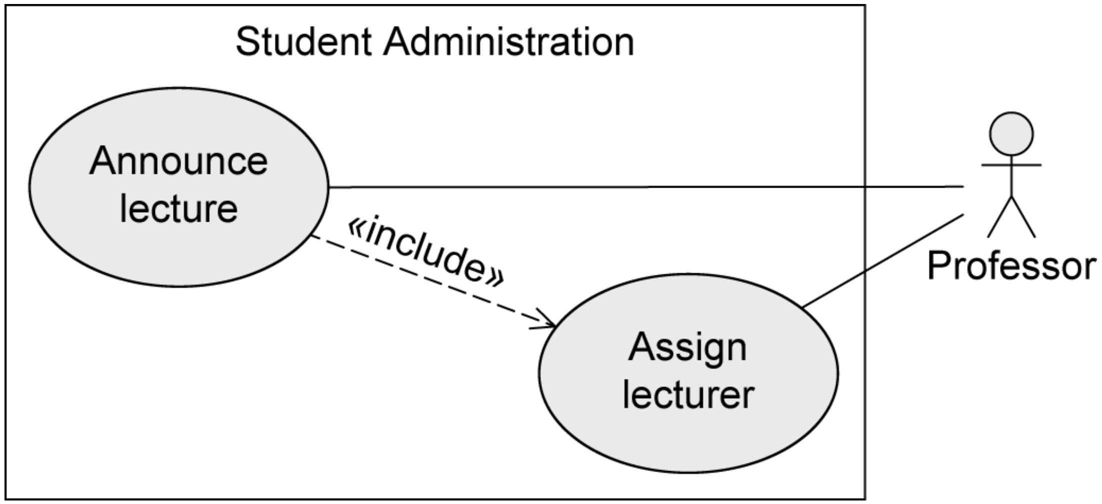
- The behavior of one use case (included use case) is integrated in the behavior of another use case (base use case)



Base use case
requires the behavior of the included use case to be able to offer its functionality

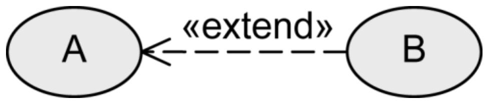
Included use case
may be executed on its own

- Example:

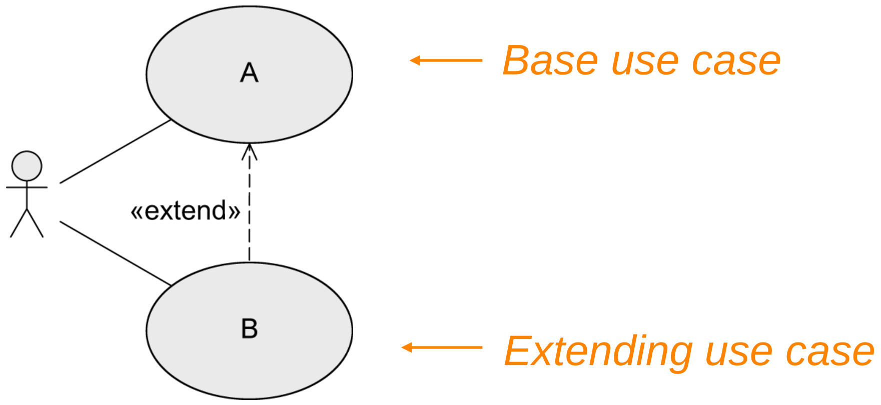


Relationships between Use Cases

«extend» - Relationship



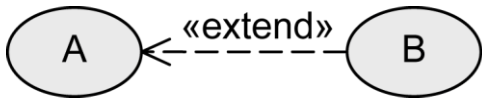
- The behavior of one use case (extending use case) may be integrated in the behavior of another use case (base use case) but does not have to.
- Both use cases may also be executed independently of each other.



- A decides if B is executed.
- Extension points define at which point the behavior is integrated.
- Conditions define under which circumstances the behavior is integrated.

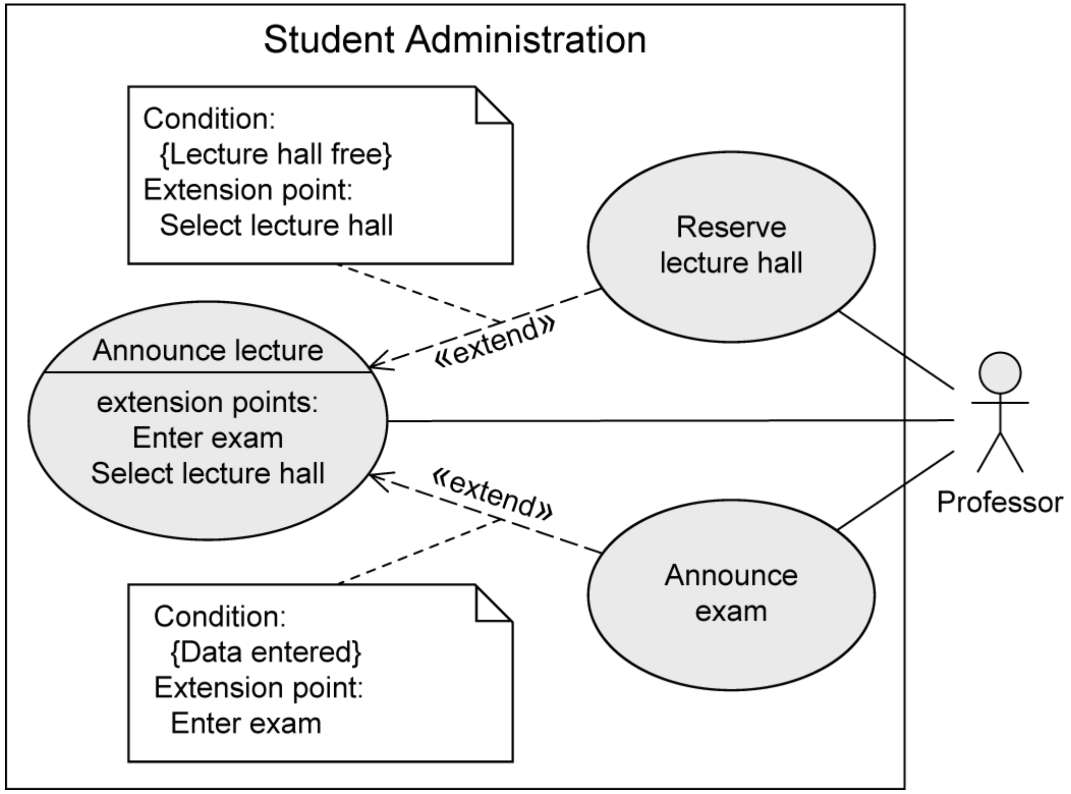
Relationships between Use Cases

«extend» - Relationship: Extension Points



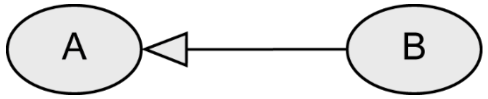
- Extension points are written directly within the use case.
- Specification of multiple extension points is possible.

- Example:

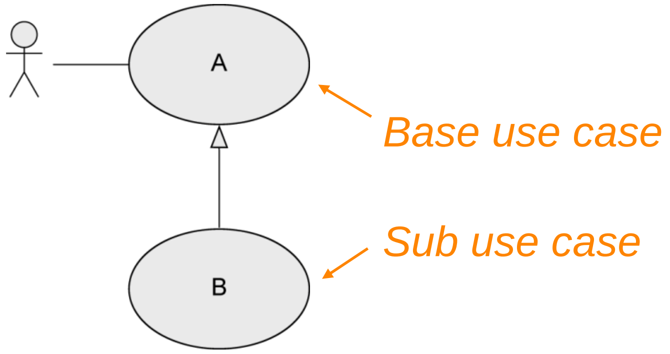


Relationships between Use Cases

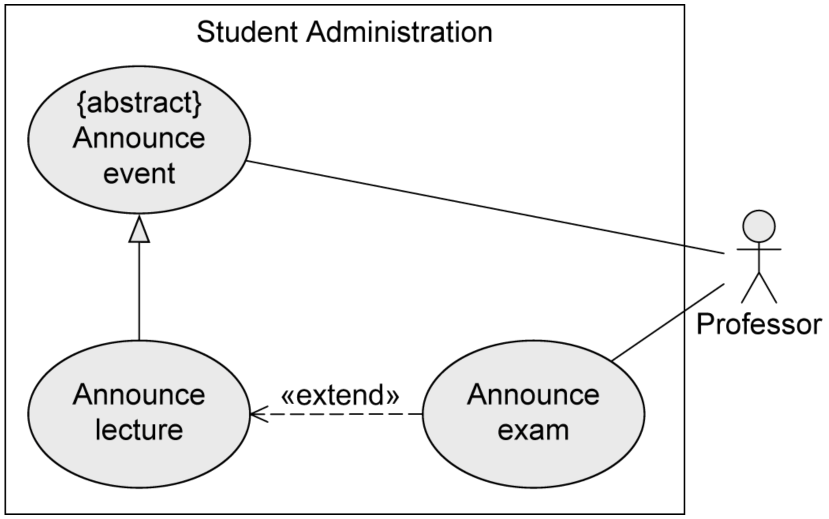
Generalization of Use Cases



- Use case **A** generalizes use case **B**.
- **B** inherits the behavior of **A** and may either extend or overwrite it.
- **B** also inherits all relationships from **A**.
- **B** adopts the basic functionality of **A** but decides itself what part of A is executed or changed.
- **A** may be labeled **{abstract}**
  - Cannot be executed directly
  - Only **B** is executable

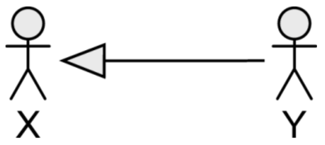


- Example:

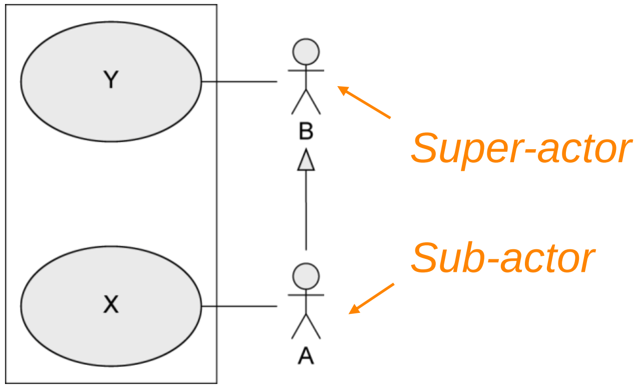


Relationships between Actors

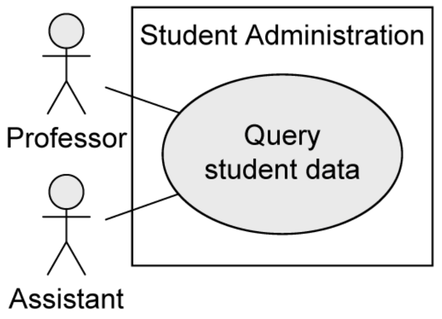
Generalization of Actors



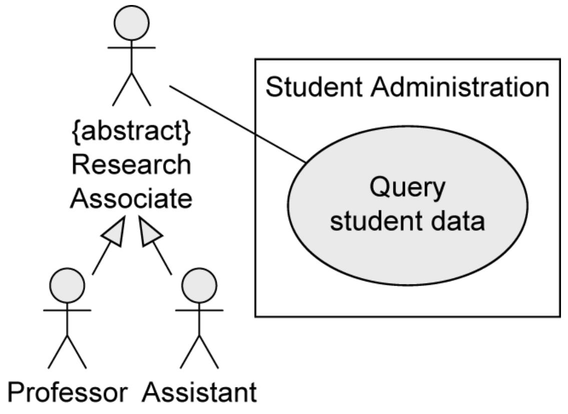
- Actor **`A`** inherits from actor **`B`**.
- **`A`** can communicate with **`X`** and **`Y`**.
- **`B`** can only communicate with **`Y`**.
- *Multiple inheritance* is permitted.
- *Abstract* actors are possible.



- Example:



≠



`Professor` AND **`Assistant`** needed for executing **`Query student data`**

`Professor` OR **`Assistant`** needed for executing **`Query student data`**

Description of Use Cases

- Structured approach
  - Name
  - Short description
  - Precondition: prerequisite for successful execution
  - Postcondition: system state after successful execution
  - Error situations: errors relevant to the problem domain
  - System state on the occurrence of an error
  - Actors that communicate with the use case
  - Trigger: events which initiate/start the use case
  - Standard process: individual steps to be taken
  - Alternative processes: deviations from the standard process

  [A. Cockburn: Writing Effective Use Cases, Addison Wesley, 2000]

Description of Use Cases - Example

- Name: **`Reserve lecture hall`**
- Short description: An employee reserves a lecture hall at the university for an event.
- Precondition: The employee is authorized to reserve lecture halls.
- Postcondition: A lecture hall is reserved.
- Error situations: There is no free lecture hall.
- System state in the event of an error: The employee has not reserved a lecture hall.
- Actors: **`Employee`**
- Trigger: Employee requires a lecture hall.
- Standard process: (1) Employee logs in to the system.
  (2) Employee selects the lecture hall.
  (3) Employee selects the date.
  (4) System confirms that the lecture hall is free.
  (5) Employee confirms the reservation.
- Alternative processes: (4’) Lecture hall is not free.
  (5’) System proposes an alternative lecture hall.
  (6’) Employee selects alternative lecture hall and confirms the reservation.

Use case template

Use case element	Description
Code	It is a unique identifier of the use case.
State	The state of completion is, for example, draft, completed or approved
Scope	The system (subsystem) under design
Name	The use case name, short and suggestive. Typically the name is of the format *action* + *object*
Primary actor	The beneficiary actor that initiates the use case and follows a specific purpose
Description	Brief, free-text presentation of the use case
Preconditions	Any state the system must be in or conditions that must be met before the use case is started.
Postconditions	Any state the system must be in or conditions that must be met after the use case is completed successfully.
Trigger	An event or a sequence of events that initiate the use case.
Main course	The most common, successful path of interactions between the user and the system.
Alternate courses	Most significant alternatives and exceptions that may occur in the flow of events
Relationships	What are the relationships with other use cases (include or extend types)
Frequency of use	How often it is this functionality expected to be used
Business rules	What rules govern the use case; What privileges should actors have

Importance of use case diagrams

- All processes that have to be carried on by the system are found in the form of a use case. Processes are then described textually or by o sequence of steps. For graphical modeling of scenarios, **activity diagram** can be used.
- Once the system behavior has been so depicted, the use cases are further analyzed to identify how objects interact to model this behavior. **Sequence diagrams** and **communication diagrams** are used for modeling object interactions.
- Use case diagram is also used for testing weather the system is consistent with the initial requirements. It takes up all use cases to see if the system meets customer requirements.

Exemple of scenario

The project goal is to develop a software application for the **management of a hotel business unit**. In order to check in, a **customer** can request to reserve one or more rooms by e-mail or telephone. For this, he provides the **receptionist** with information on the period of accommodation and type of rooms required. Customers will get **discounts** if they reserve at least 3 rooms or if the period of accommodation exceeds 5 days. The receptionist checks availability and notifies the client of this and the estimated cost of accommodation. If there are no rooms available as requested, the receptionist can provide alternatives to the customer. The client may request a discount (additional or not) and the receptionist will decide the feasibility discount, assisted mandatory by the hotel manager. If the client agrees with the proposed price, they proceed to the reservation. For new customers, the receptionist asks **identification data**, which he introduces in the application.

Once at the hotel and if he has made a prior booking, the customer will provide his identification and / or booking number and the **check in** is finalized. If there is no reservation, the availability for the required period will be checked. When there is such a room, **accommodation** is made. At the end of the stay, the receptionist prepares **a list of all the services** used by the customer and their price. The list must be validated by the customer, then the **final invoice** is drawn up. The invoice can be **paid** partially or fully by bank transfer, cash or using a credit card. Also, before leaving the hotel, the customer is asked to complete **a form to evaluate** the services provided by the hotel premises

General requirements for hotel accommodation activity

The project aims at developing a computer system for a hotel that would provide support for its activities.

The **core activity** of the hotel is providing **accommodation, catering services and other services** of commercial agents affiliated to the hotel.

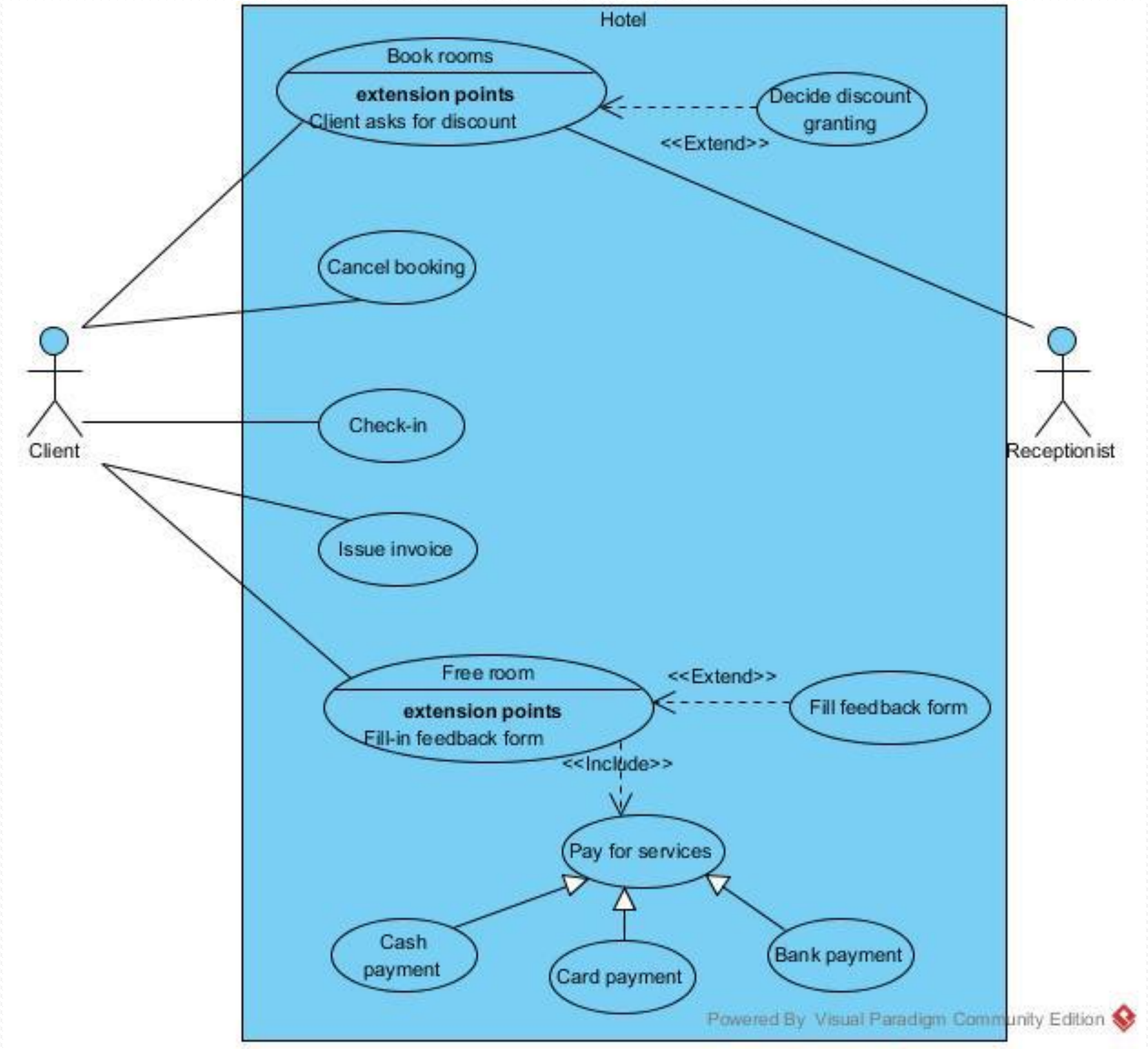
The IT system should achieve: booking rooms and customer registration, management of room occupancy, and other services used by customers.

The arrangements for booking, customer registration and corresponding payments derive from the marketing and management policy of the hotel.

The **basic requirements** that should be accomplished are:

- Reducing the time to fill in the forms for reservation and registration by automating the process of updating availability and allocation of rooms;
- Monitoring staff involved in bookings, registrations, and various changes or cancellations.
- Reducing time spent making lists of rooms to be cleaned, lists of arrivals, lists of room status;
- Increased promptness in responding to customer about the availability of the accommodation requested for a specific period;
- Creating a secure and effective system that protects against overbooking.

Use case diagram - example



Use case element	Description
Code	CU01
State	Draft
Scope	Manage hotel accommodation activity
Name	Book rooms
Primary actor	Client
Description	It involves booking for one or more rooms
Preconditions	The receptionist is logged in the system
Postconditions	The booking was accomplished and the client receives booking confirmation.
Trigger	Client asks for booking one ore more rooms by email or by phone.
Main course	1. The client provides the receptionist with information on the period of accommodation and type of rooms required 2. The receptionist checks room availability. 3. The receptionist notifies the client that there are available rooms. [A alternate course: There are not available rooms according to client requirements] 4. The receptionist informs the client of the estimated costs. [Extention point: CU07 Decide discount granting] 5. The client confirms booking period and agrees estimated cost. [B alternate course: the client doesn't agree the booking conditions] 6. The receptionist asks the customer identification data. [C alternate course: customer data are already in the system] 7. The receptionist enters customer data in the system. 8. The receptionist makes room booking . 9. The client receives the confirmation of booking.
Alternate courses	A: 1. The receptionist provides booking alternatives to the client. 2. The client selects an alternative and the scenario ends. B: 1. the client does not confirm booking and the scenario ends. C: 1. Go to step 8.
Relationships	Extended by CU07 Decide discount granting
Frequency of use	Very often
Business rules	When the client asks for discount, the receptionist can grant it only if manager agrees with it .

Exemples of non-functional requirements

- **Quality requirements**
  1. **Safety**
     - The system should stop working if the outside temperature drops below 4 degrees C.
     - The system should stop working in case of fire.
     - The system should stop working in case of obvious attacks.
  2. **Security**
     - Data access **permissions** can only be changed by the **system administrator .**
     - All system data must be **copied** every 24 hours and backups must be stored in a safe place that is not in the same building as the system.
     - All external communications between data server and client must be **encrypted**.
  3. **Reliability**
     - The system must have an availability of 999/1000 or 99%. This is a requirement of reliability which implies that out of every 1,000 service requests, 999 must be met.
  4. **Performance**
     - The system will process at least X transactions per second.

Exemples of non-functional requirements

- **Compliance requirements**
  - Legal and regulatory requirements: all changes to user data must be registered and stored at least 6 years.
  - Licensing requirements:
    - “Update order” process will be licensed for 300 concurrent users.
    - PostalCode_Address file will be licensed for a year, starting each April.
- **Architectural requirements**
  - Data persistence will be ensured by a relational database.
  - This database will be Oracle 12g.
  - The system will run 24 hours a day, 7 days a week..