
REQUIREMENTS ANALYSIS, UML & USE CASE DIAGRAMS

REQUIREMENT ANALYSIS

STAGES IN REQUIREMENTS ANALYSIS

1. Domain analysis, requirements elicitation

- Identify stakeholders
- Gather information on domain + requirements
- With users, customers, other stakeholders, literature, other similar systems, ...

2. Evaluation and negotiation

- Identify conflicts, imprecision, omissions, redundancies
- Consult and negotiate with stakeholders to agree resolutions

3. Specification and documentation

- Systematically document requirements as system specification
- In precise, possibly formal, notation
- Agreement between developers and stakeholders on what will be delivered

4. Validation and verification

- Check (formalised) requirements for consistency, completeness and correctness

REQUIREMENTS ANALYSIS TECHNIQUES

- Interviews with stakeholders
- Brainstorming sessions
- Observation of existing processes (ethnography)
- Scenario analysis – model specific scenarios of use of system, e.g., as UML sequence diagrams
- Document mining
- Goal decomposition
- Exploratory prototyping

REQUIREMENTS ENGINEERING

The process of establishing the services that a customer requires from a system and the constraints under which it operates and is developed.

The system requirements are the descriptions of the system services and constraints that are generated during the requirements engineering process

What is a requirement?

- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification
- This is inevitable as requirements may serve a dual function
 - May be the basis for a bid for a contract, therefore must be open to interpretation
 - May be the basis for the contract itself – therefore must be defined in detail
 - Both these statements may be called requirements.

TYPES OF REQUIREMENT

User requirements

- Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

System requirements

- A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

USER AND SYSTEM REQUIREMENTS

1. The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

- 1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
- 1.2 The system shall generate the report for printing after 17.30 on the last working day of the month.
- 1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
- 1.4 If drugs are available in different dose units (e.g. 10mg, 20mg, etc) separate reports shall be created for each dose unit.
- 1.5 Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

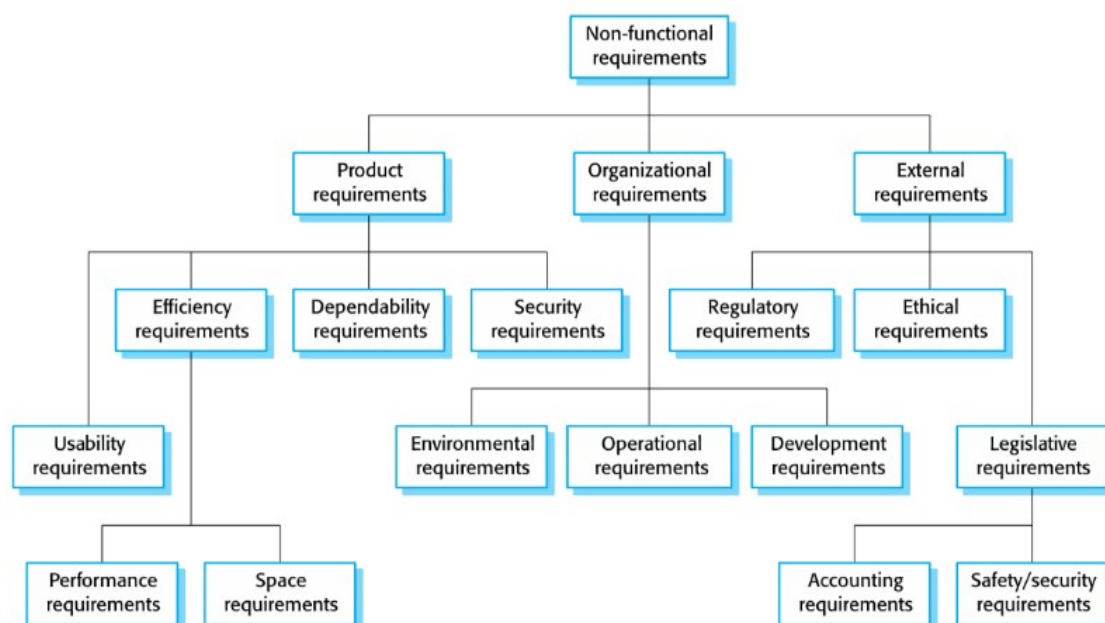
FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

Functional requirements

- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- May state what the system should not do.

Non-functional requirements

- Constraints on the services or functions offered by the system such as timing constraints on the development process, standards, etc.
- Often apply to the system as a whole rather than individual features or services.



UNIFIED MODELLING LANGUAGE (UML)

Modelling is the designing of software applications before coding.

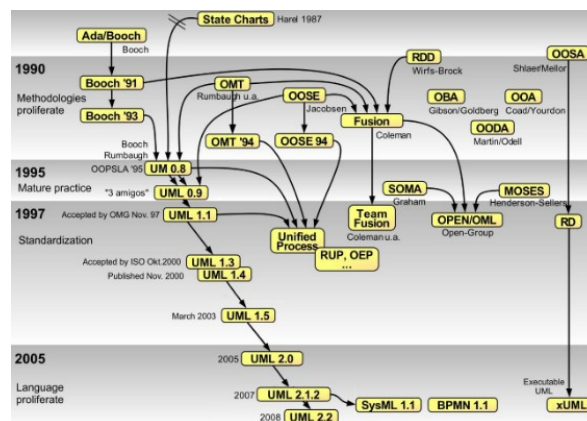
- Higher level of abstraction
- Methodology-independent

HISTORY

UML is the success of object oriented analysis and design methods from 80's and 90's.

It does not include a method though. It is just a language with its metamodel and notation.

The unification was led by Grady Booch, James Rumbaugh and Ivar Jacobson.



WHY UML?

To communicate better

- Natural language is imprecise and gets tangled when describing complex systems/concepts
- Code is usually very detailed
- UML is a way to highlight details
- The art is knowing what to leave out

To learn object orientation

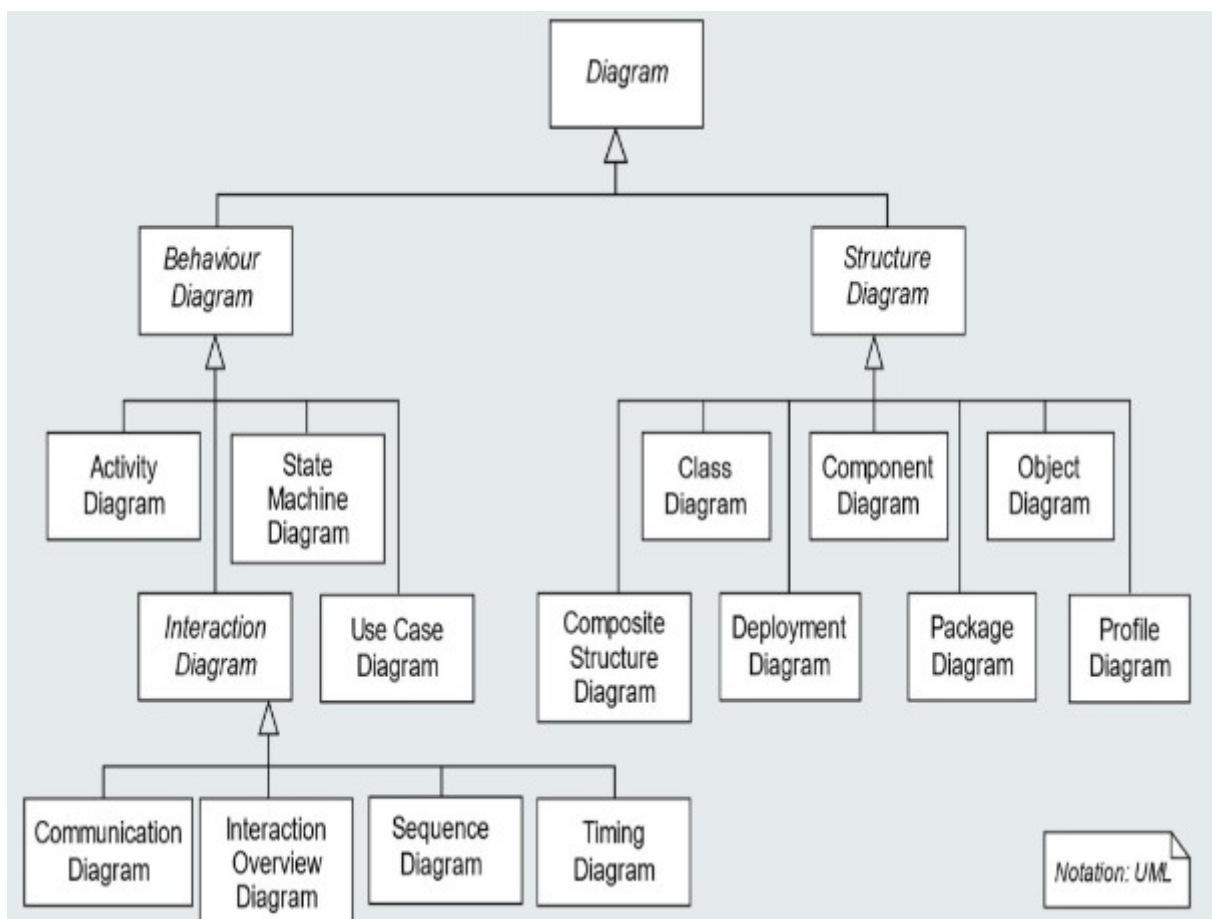
- Interaction diagrams help identify over-centralized designs
- Class diagrams should be drawn from a process perspective and not from a data perspective
- Activity diagrams are relevant when workflows are an important part of a user's world.

WHAT CAN YOU MODEL WITH UML?

UML 2.0 defines fourteen types of diagrams, divided into three categories:

Seven diagram types represent static application structure; three represent general types of behaviour, and four represent different aspects of interactions:

- **Structure diagrams** include the Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, Deployment Diagram and Profile diagram.
- **Behaviour diagrams** include the Use Case Diagram (used by some methodologies during requirements gathering); activity diagram, and State Machine Diagram.
- **Interaction Diagrams** all derived from the more general Behaviour Diagram, include the Sequence Diagram, Communication Diagram, Timing Diagram and
- Interaction Overview Diagram



DIAGRAMS

1. Use case (relation of actors to system functions)
2. Class (static class structure)
3. Object (same as class – only using class instances – i.e. objects)
4. State (states of objects in a particular class)
5. Sequence (Object message passing structure)
6. Collaboration (same as sequence but also shows context – i.e. objects and their relationships)
7. Activity (sequential flow of activities, i.e. action states)
8. Component (code structure)
9. Deployment (mapping of software to hardware)

ANOTHER CLASSIFICATION

User view

- Use case diagrams

Structural view

- Class diagrams
- Object diagrams

Behavioural view

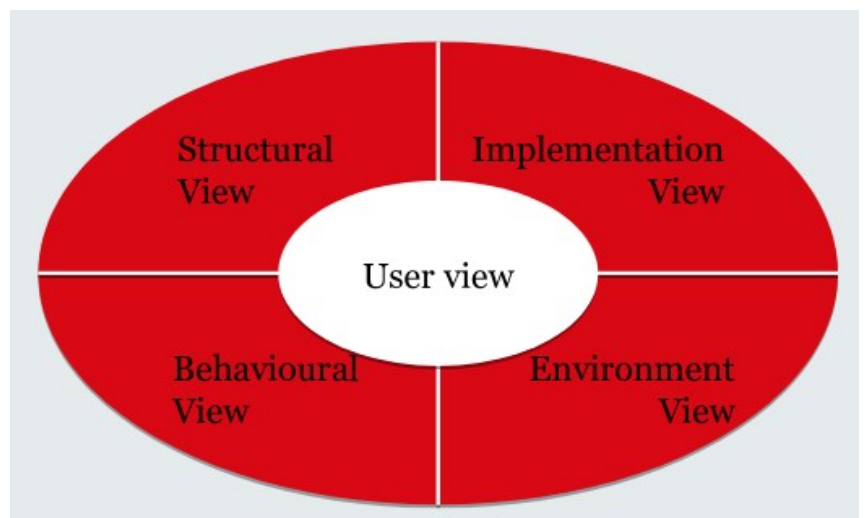
- Sequence diagrams
- Collaboration diagrams
- State diagrams
- Activity diagrams

Implementation view:

- Component diagrams

Environment view:

- Deployment diagrams



USING DIAGRAMS

Pros:

- Aesthetic
- Descriptive
- Simple
- Understandable
- Universal
- Formalise-able / Standardise-able

Cons:

- Not inherent knowledge
- Easily cluttered
- Require some training
- Not necessarily revealing
- Must be liked to be accepted and used
- Effort to draw

USE CASE DIAGRAMS

Use case diagrams represent the stakeholders/users view of a system

- it is important to note that requirements are always evolving and changing

They can answer questions such as

- Which system is being described?
- Who are the stakeholders who interact with the system (actors)?
- What features do the system offer? (What can actors do?)

Use case diagram elements:

- Actors
- use cases
- Associations
- Include relationships
- Extend relationships
- System boundary
- etc.

EXAMPLE: STUDENT ADMINISTRATION SYSTEM

System (What is being described)?

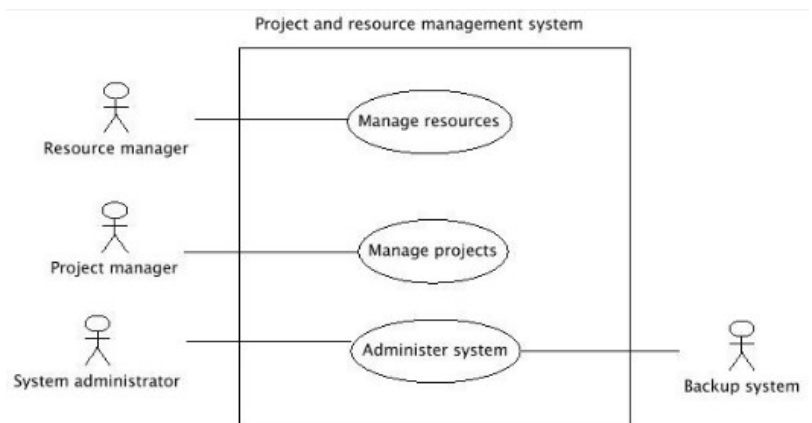
- Project and resource management system

Actors (Who interacts with the system?)

- Resource manager
- Project manager
- System administrator
- Backup system

Use cases (What can the actors do?)

- Manage resources
- Manage projects
- Administer system



USE CASES

These describe 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.

ACTORS

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/systems 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

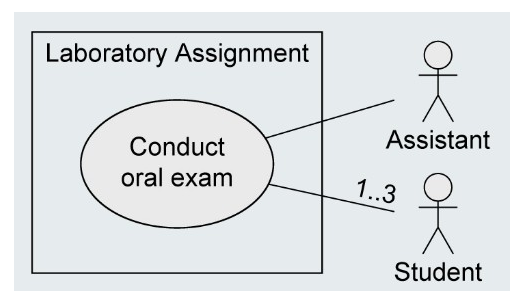
Usually user data is also administered within the system. This data is modelled within the system in the form of objects and classes.

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, studentNumber, ...).

Actors are connected with use cases via solid lines (associations).

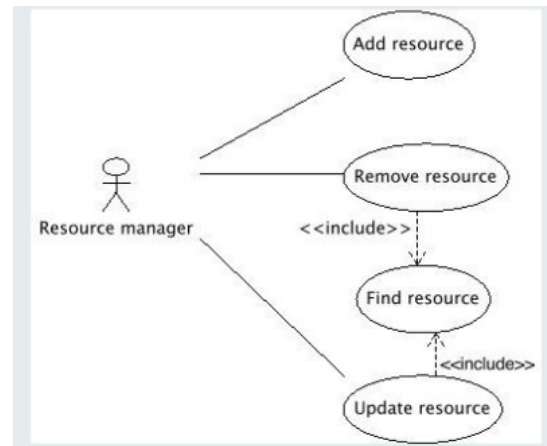
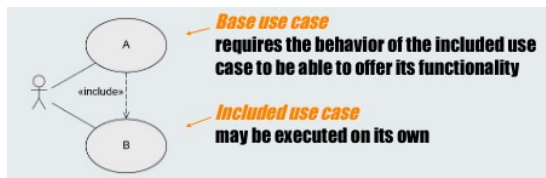
- Every actor must communicate with at least one use case.
- An association is always binary.
- Multiplicities may be specified



RELATIONSHIPS BETWEEN USE CASES

<<include>> - Relationship

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

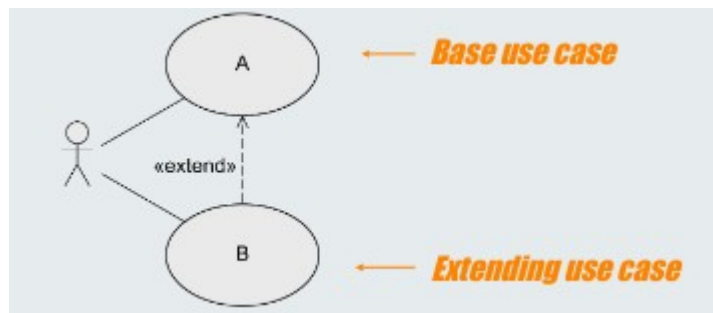


<<extend>> - Relationship

The behaviour of one use case (extending use case) may be integrated in the behaviour of another use case (base use case) but does not have to.

Both use cases may also be executed independently of each other.

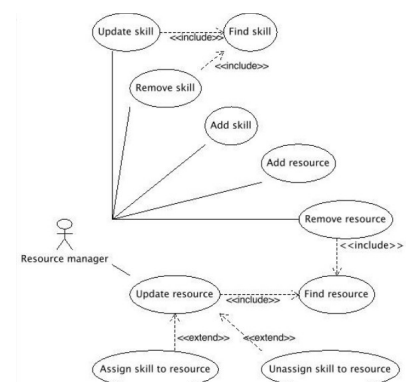
- A decides if B is executed
- Extension points define at which point the behaviour is integrated
- Conditions define under which circumstances the behaviour is integrated



<<extend>> - Relationship: Extension Points

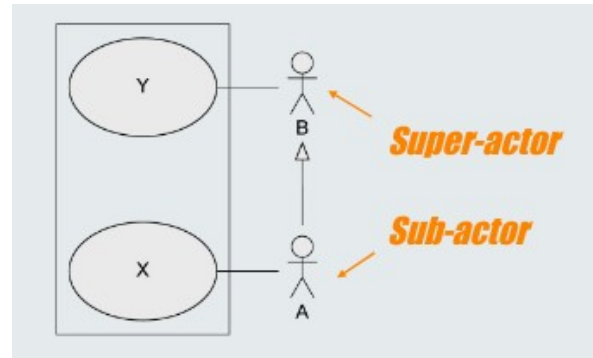


- Extension points are written directly within the use case.
- Specification of multiple extension points is possible. e.g.:

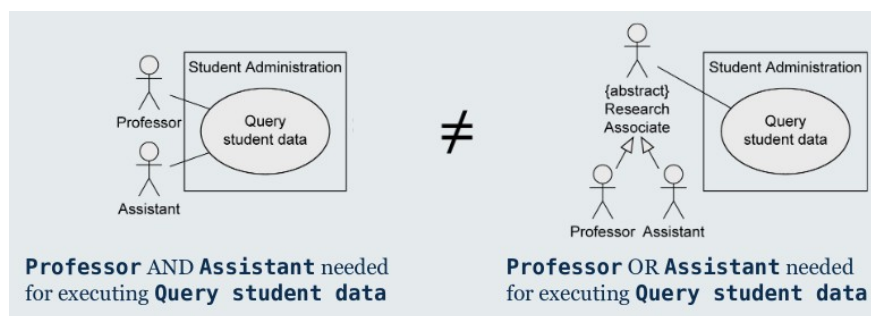


RELATIONSHIPS BETWEEN ACTORS

- 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.



e.g.:



DESCRIPTION OF USE CASES

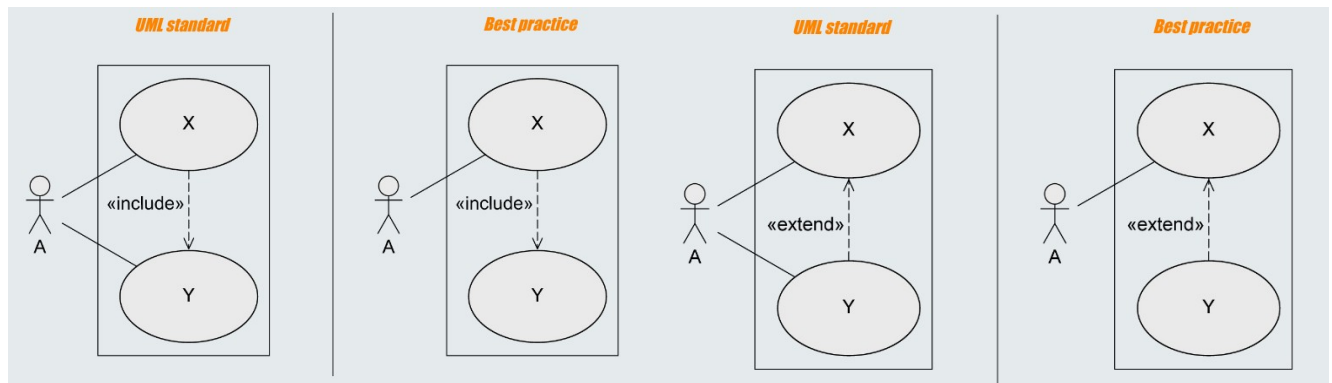
Structured approach

- Name
- Short description
- Precondition
- Postcondition
- Error situations
- 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

- 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.

BEST PRACTICES

<<include>> and <<extend>>



Actors

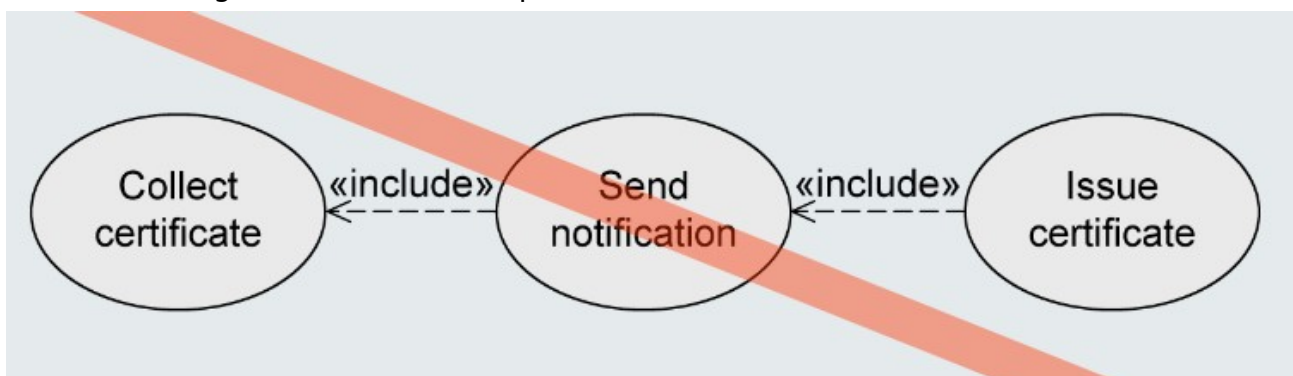
- Who uses the main use cases?
- Who needs support for their daily work?
- Who is responsible for system administration?
- What are the external devices/(software) systems with which the system must communicate?
- Who is interested in the results of the system?

Identifying use cases

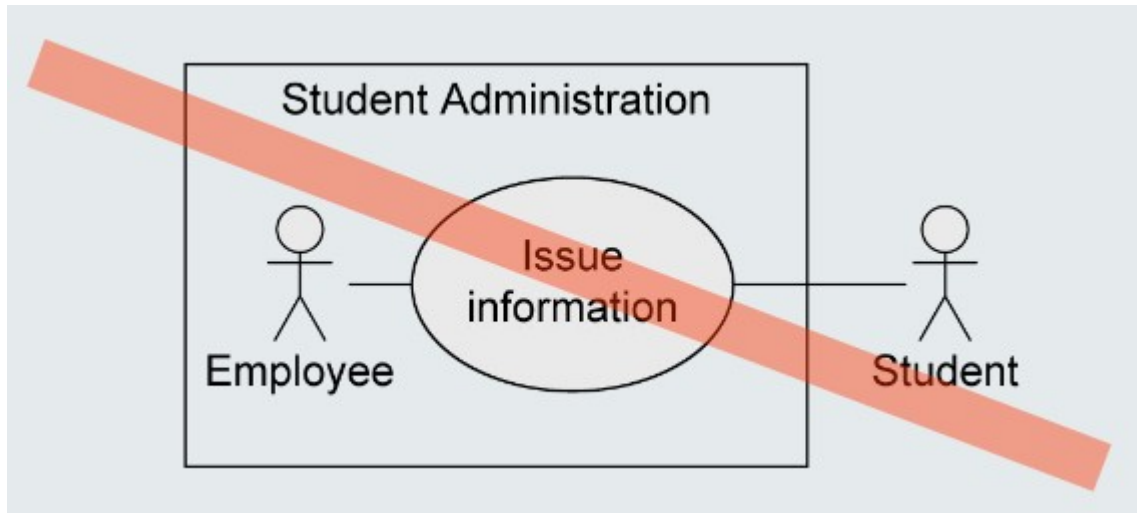
- What are the main tasks that an actor must perform?
- Does an actor want to query or even modify information contained in the system?
- Does an actor want to inform the system about changes in other systems?
- Should an actor be informed about unexpected events within the system?

Typical errors

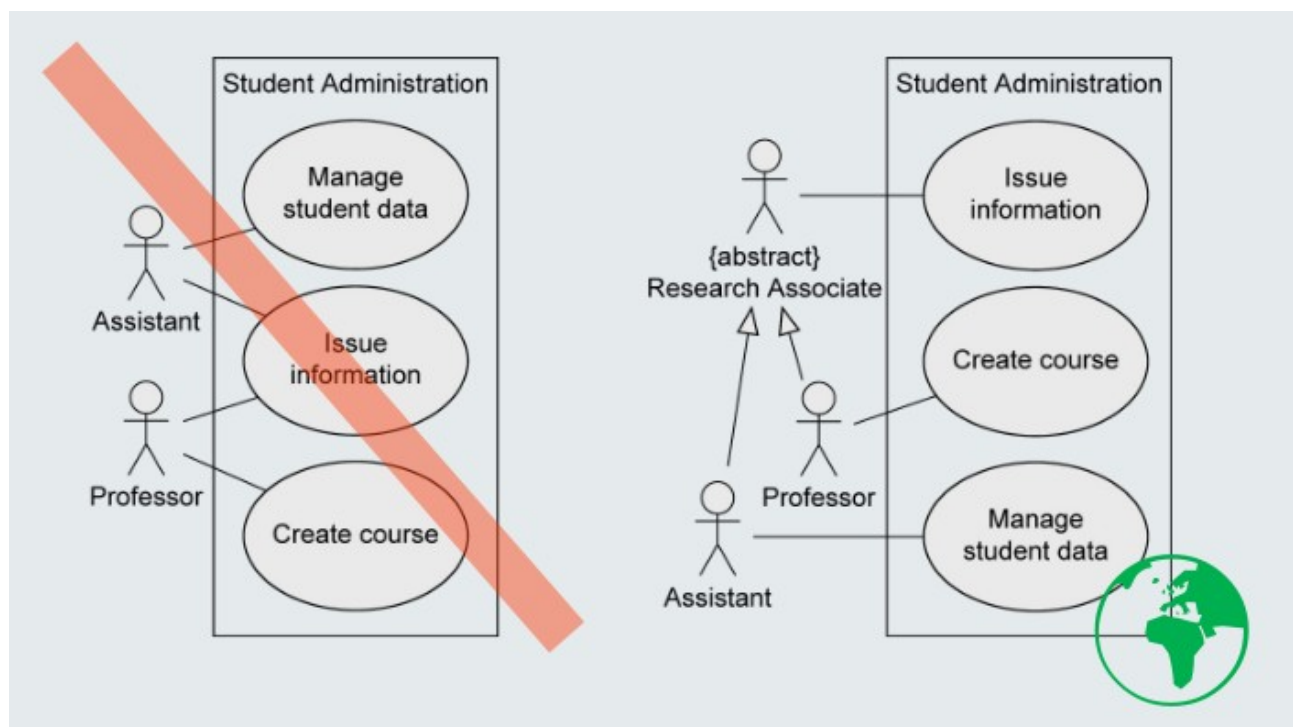
- Case diagrams do not model processes/work flows!



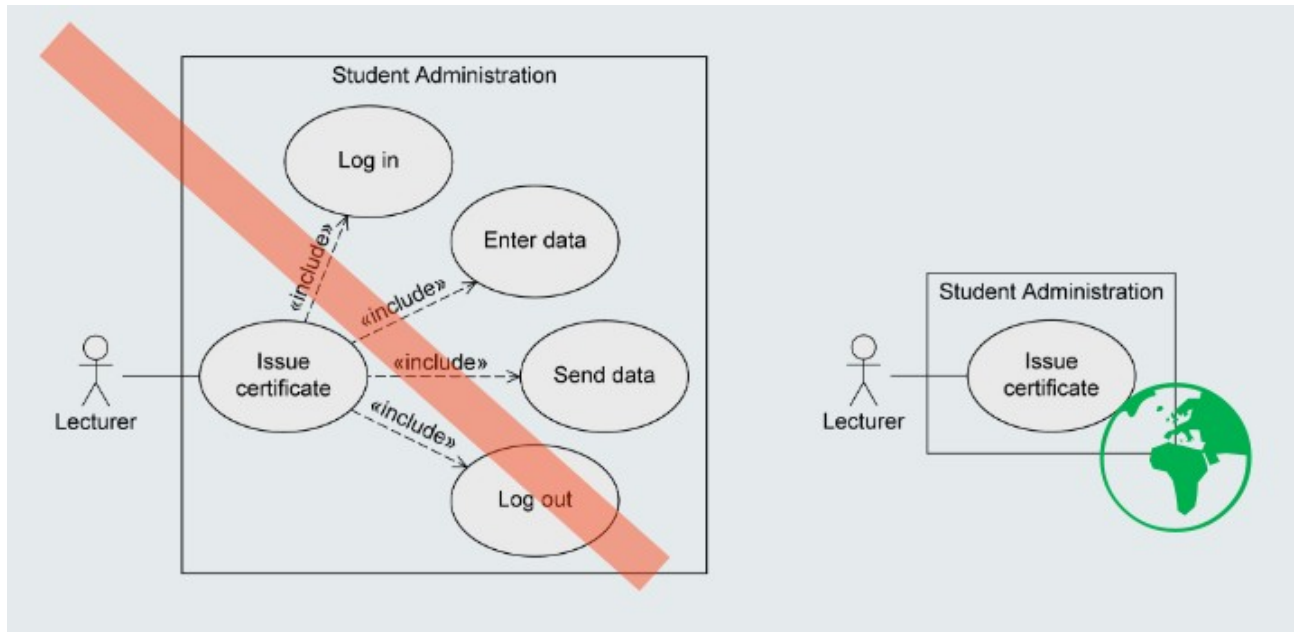
- Actors are not part of the system, hence they are positioned outside the system boundaries!



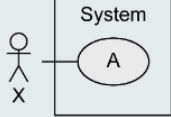



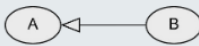

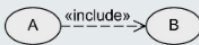
- Use case `Issue information` needs EITHER one actor `Assistant` OR one actor `Professor` for execution.



- The various steps are part of the use cases, not separate use cases themselves
→ NO functional decomposition



NOTATION ELEMENTS

Name	Notation	Description
System		Boundaries between the system and the users of the system
Use case		Unit of functionality of the system
Actor		Role of the users of the system
Association		Relationship between use cases and actors
Generalization		Inheritance relationship between actors or use cases
Extend relationship		B extends A: optional use of use case B by use case A
Include relationship		A includes B: required use of use case B by use case A