Introduction to Software Engineering

The Planning Phase

Prof. Walter F. Tichy

ქუთაისის
საერთაშორისო
უნივერსიტეტი

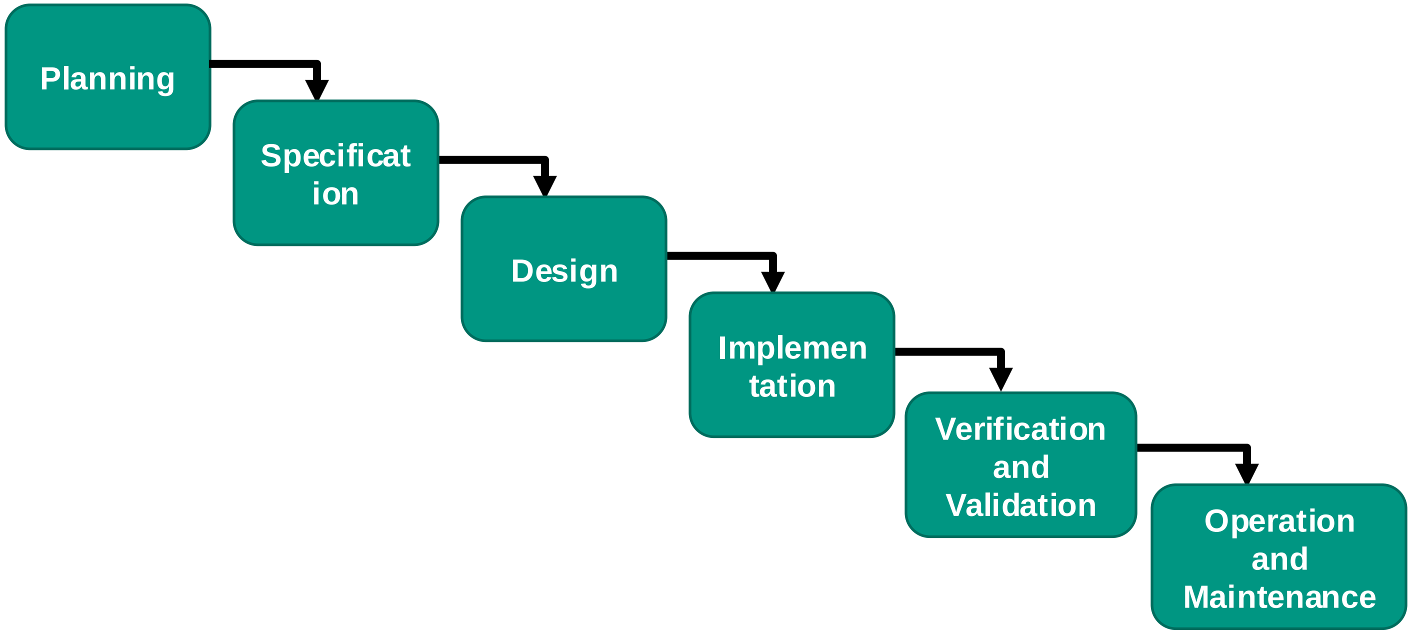
Literature

- We are using Section 2.4.1 **Use Case Diagrams** and Chapter 4 **Requirements Elicitation** from

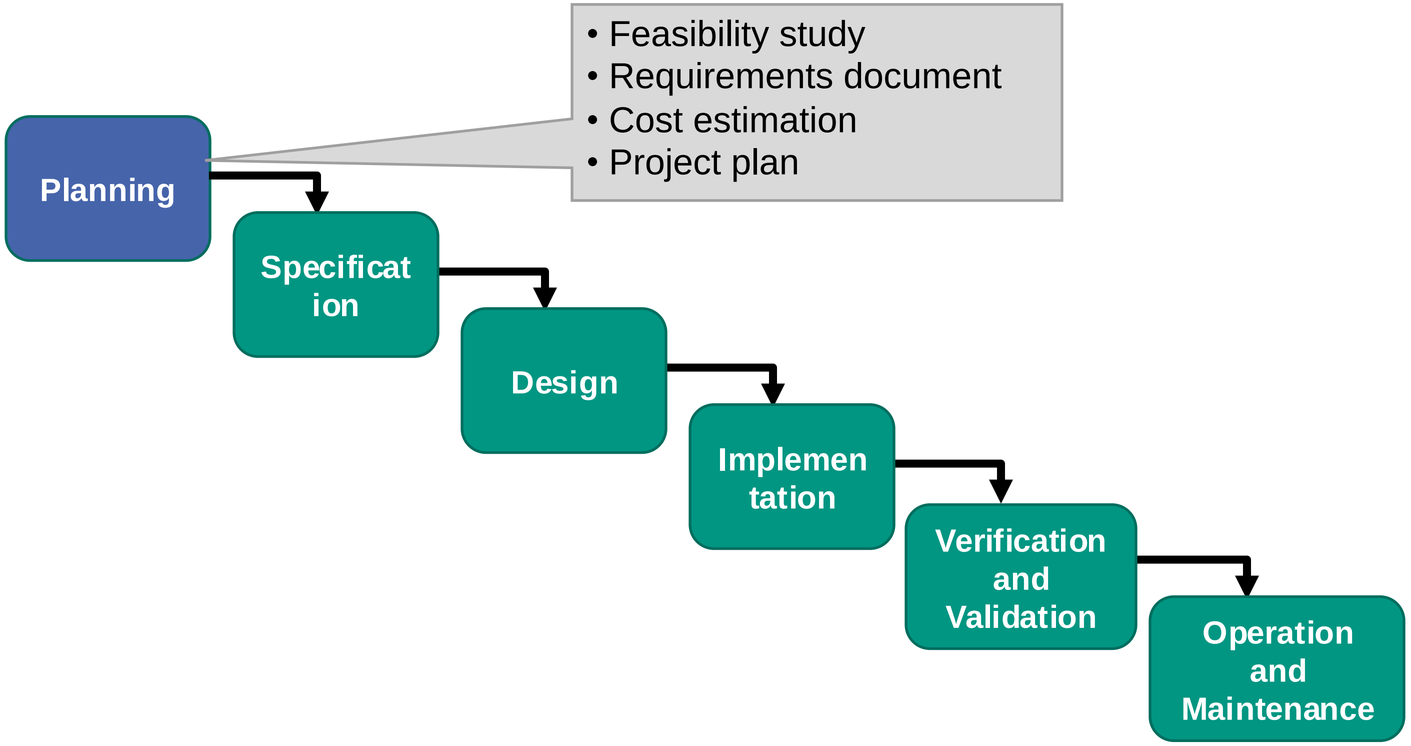
  B. Bruegge, A.H. Dutoit, **Object-Oriented Software Engineering: Using UML, Patterns and Java**, Pearson Prentice Hall.

- **Read it!**

The Waterfall Model -- Overview



The Waterfall Model -- Overview



- Feasibility study
- Requirements document
- Cost estimation
- Project plan

Planning Phase

- The main output of the planning phase is a **requirements document** that describes the system to be developed in **the words of the user(s).**
- The process of determining the requirements is called **requirements elicitation**
- **A feasibility study** determines whether the software development organization is capable of implementing the requirements (capability and capacity)
- **Cost estimation** is necessary to make sure the software can be delivered at a competitive price; the project plan determines the development time and personnel required.

What is a Requirement?

- The IEEE Standard 610.12-1990 defines **requirement** as follows:

  *"A **condition or capability** that must be met or possessed by a system or system component **to satisfy** a contract, standard, specification, or other formally imposed document.*
  The set of all requirements forms the basis for subsequent development of the system or system component".

Planning Phase --- First Steps

- The planning phase begins with the selection of a product or project
  - Customer request
  - Request for competitive bids, for example by the government
  - Research results
  - Predevelopment
  - Trend studies
  - Market analysis
  - An idea! (for a startup or an extension of a product portfolio)

Requirements Elicitation

- Determines the main requirements of the product
- Various techniques available
  - Introspection (think about it yourself!)
  - Questionnaires
  - Interviews with potential users
  - Brainstorming (in groups)
  - Task analysis (studying the work being done)
  - Document analysis (analyzing the documents used in a workflow)
  - Scenarios (a concrete event or sequence of events or actions)
  - Use cases (use case diagrams for that)

Scenarios (1)

- A scenario
  - Describes an event or a sequence of events and actions
  - Describes the use of the (future) system in text from as experienced **by the user** (not the implementer).
  - May contain texts, images, screen sketches, videos, work flows, details about the workplace, the social environment (office, home, restaurant, etc.), as well as restrictions about what is allowed and what resources are available.

Example Scenario (from Dutoit, Bruegge, 2.4.2)

Scenario Name	WarehouseOnFire
Participating actor instances	Bob, Alice: FieldOfficer John: Dispatcher
Flow of events	1. Bob, driving down Main Street in his patrol car, notices smoke coming out of a warehouse. His partner, Alice, activates the „Report Emergency“ function on her laptop. 2. Alice enters the address of the building, a brief description of its location (i.e. northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units given that the area is relatively busy. She confirms her input and waits for an acknowledgement. 3. John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the incident site and sends their estimated arrival time (ETA) to Alice. 4. Alice receives the acknowledgement and the ETA.

Notes to WarehouseOnFire scenario

- This is a specific scenario
  - It describes a single instance of reporting a fire.
  - It does not describe all possible scenarios of reporting fires.
  - Nor does it describe other incidents, such as car accident, drug incident, fugitive, bank robbery, etc.
- Participating actors:
  - Police officers (Bob, Alice) Dispatcher (John)

Scenarios (2)

- Scenarios can later be acted out during integration tests, acceptance test, and installation.
- If scenarios are later used during design, then this is called „scenario-based design“.

Scenarios (3)

- Scenario-based requirements elicitation is iterative, where each scenario is viewed as a work package. For instance, reporting a car accident might be another scenario for the police patrol software.
- Each of those work packages is extended or modified iteratively, if requirements change.
- Scenario-based elicitation is based on concrete descriptions and not on abstract concepts.
- Scenario-based elicitation is informal and without a definitive end point.

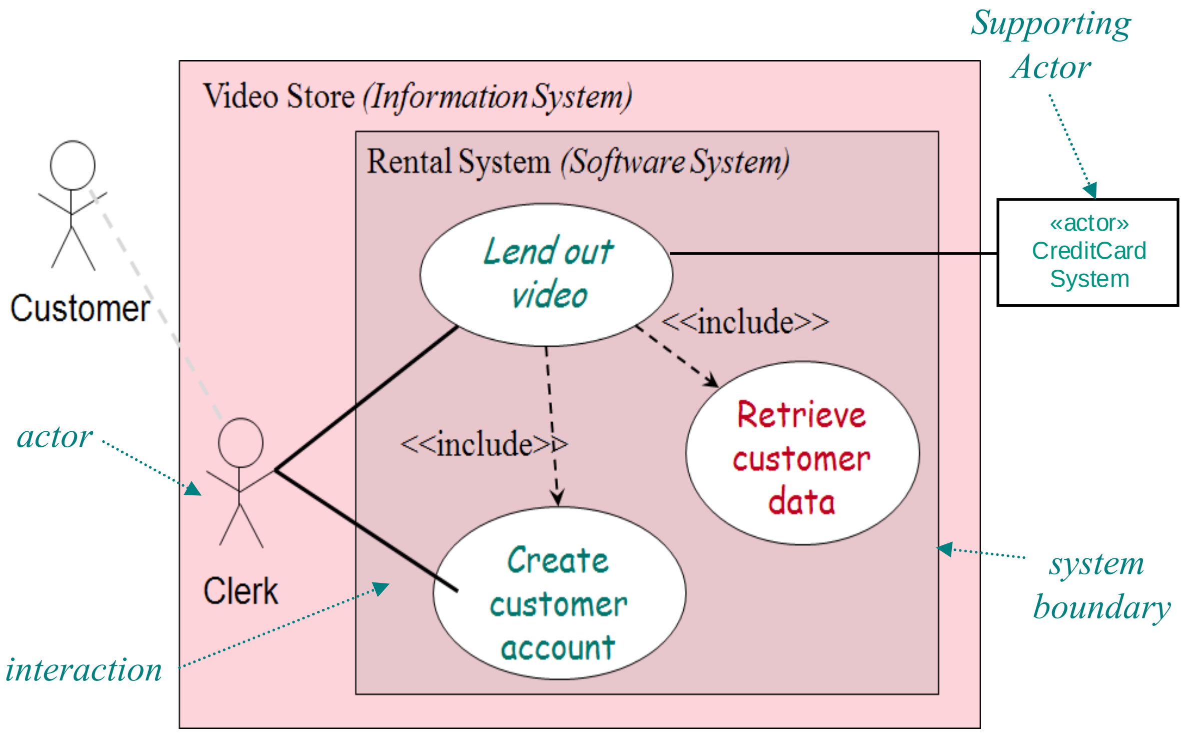
How do you find Scenarios?

- Don’t expect a word-for-word description from a customer, as long as the system does not exist.
  - The customer understands the problem domain, but not necessarily the software and solution domain, can’t imagine how a digital system would work.
- Try to communicate with the customer:
  - Help the customer to formulate scenarios.
  - The customer helps you to understand the requirements.
  - Requirements will change as the scenario is developed.

Tips for finding scenarios

- Ask the customer the following questions:
  - What are the main tasks the system should perform?
  - What data should be gathered, stored, changed, deleted or entered into the system?
  - About which outside changes must the system be informed?
  - About which changes or events must the user be informed by the system?
- Caution: Do not rely on questions and answers alone.
- Use observation, if an older system or workflow already exists (for example, analyze checklists or the forms that are filled out)

UML Use Case Diagrams



<<include>> ~ sub use case "call"

UML Use Case Diagrams

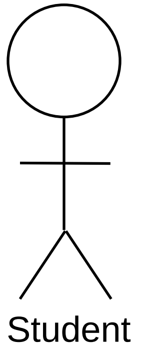
- Use Case Diagrams are employed during the planning phase to represent a user’s possible interactions with the system.
- An **actor** represents the role of a user or an external system that interacts with the planned system.
- A **use case** represents a class of functions offered by the system.
- A **use case model** is the set of **all use cases** that represent the complete functionality of the system.

UML Use Case Diagrams

- An actor represents an external unit that interacts with the system:
  - administrator, end user, environment, external system, attacker…
  - drawn as a stick figure or as a square if it is an external system
- An actor has a unique name plus optionally a description
- Example:
  - **Student**: A studying person
  - **Random number generator**
- Use case diagrams are simplistic, but help clients explain and understand the requirements.
- Good for communication with clients

Optional Description

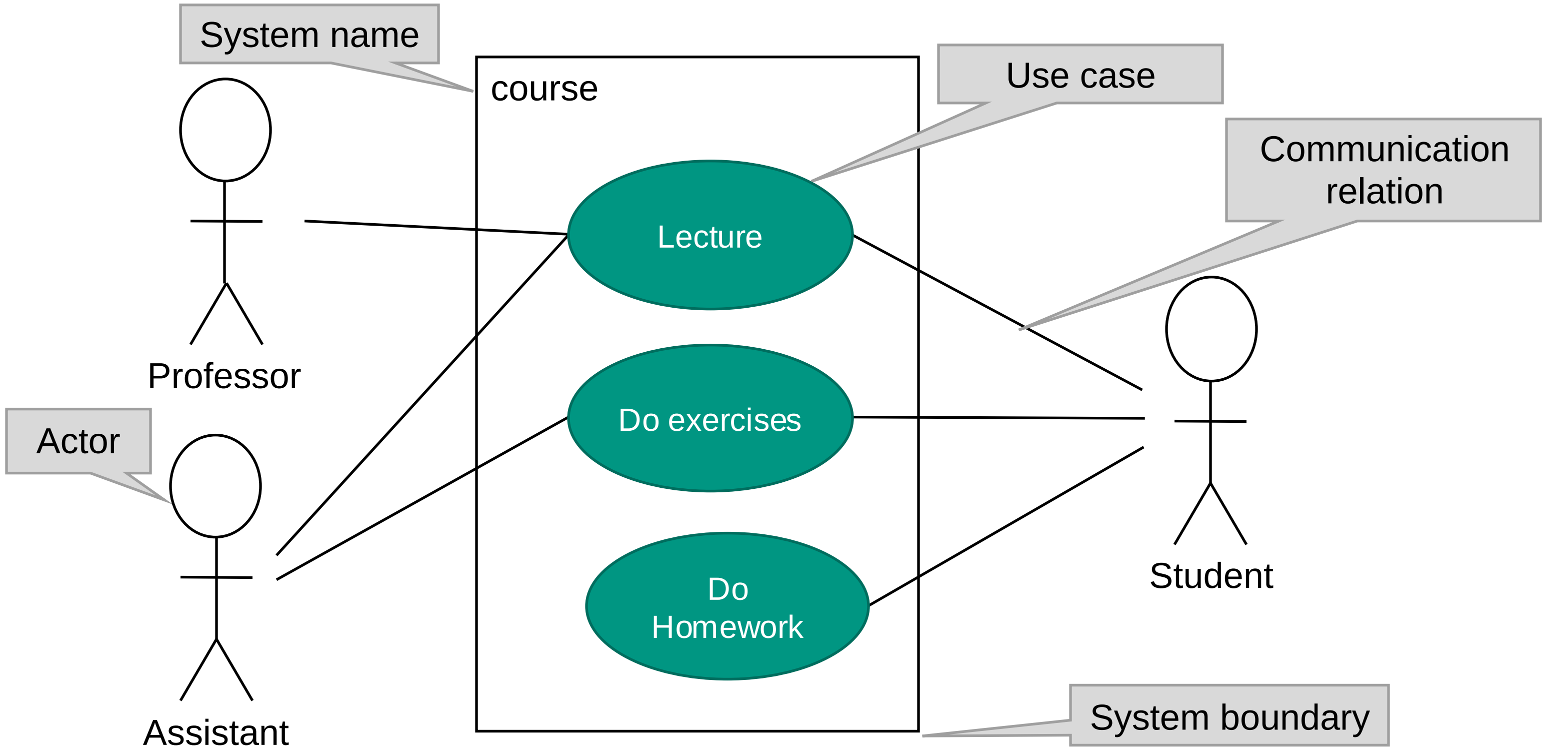
Name



UML Use Case Diagram

- Use cases can be **supplemented with textual descriptions**. These should concentrate on the flow of interactions between actors and the system.
- Die textual description of a use case consists of 6 parts:
  1. Unique name
  2. Participating actors
  3. Entry conditions (conditions that must be satisfied before use case starts)
  4. Flow of events (sequence of interactions of the use case, numbered; common cases and exceptional cases described separately for clarity)
  5. Exit conditions (conditions satisfied after completion of the use case)
  6. Exceptions
  7. Special requirements (e.g., response time, reliability, etc.)
- Use cases are often accompanied by other diagrams as well (discussed later)

UML Use Case Diagrams

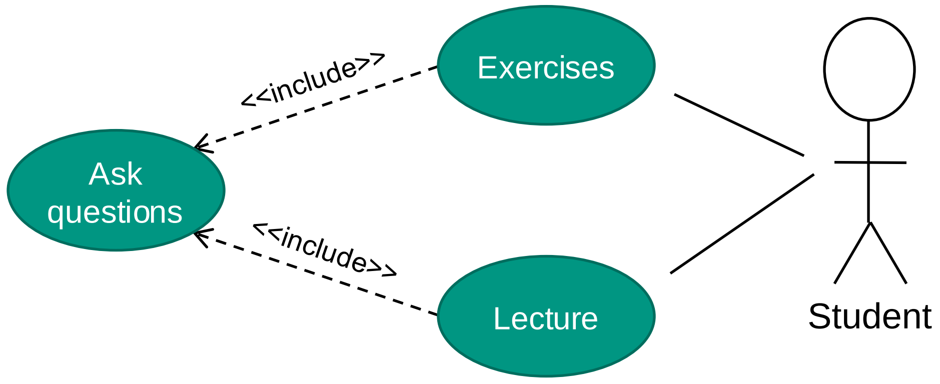


Relations among use cases

- Use cases can be related according to the following relations:
  - **<<include>> relation:** represents an „is called by“ relationship. The „included“ use case should represent functionality that is called by several others, thereby reducing replication.
  - **<<extend>> relation**: for exceptional behavior, or rare conditions of a use case. The extending use case is called from the extended case under special conditions that are provided as entry conditions in the extending use case. Separating exceptional behavior from common behavior enables us to write better focused use cases.

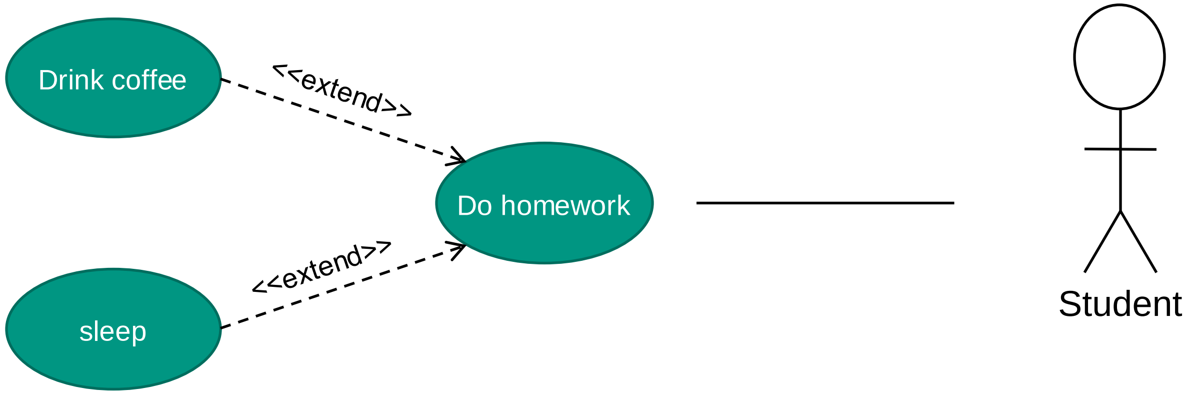
The <<*include*>>-Relation

- An <<include>>-relation identifies a use case that is used in more than one other use case. It is similar to the „calls“ relation.
- The <<include>>-relation enables us to factor out reusable cases.
- The arrow of the <<include>>-relation points to the use case being called.



The <<*extend*>>-Relation

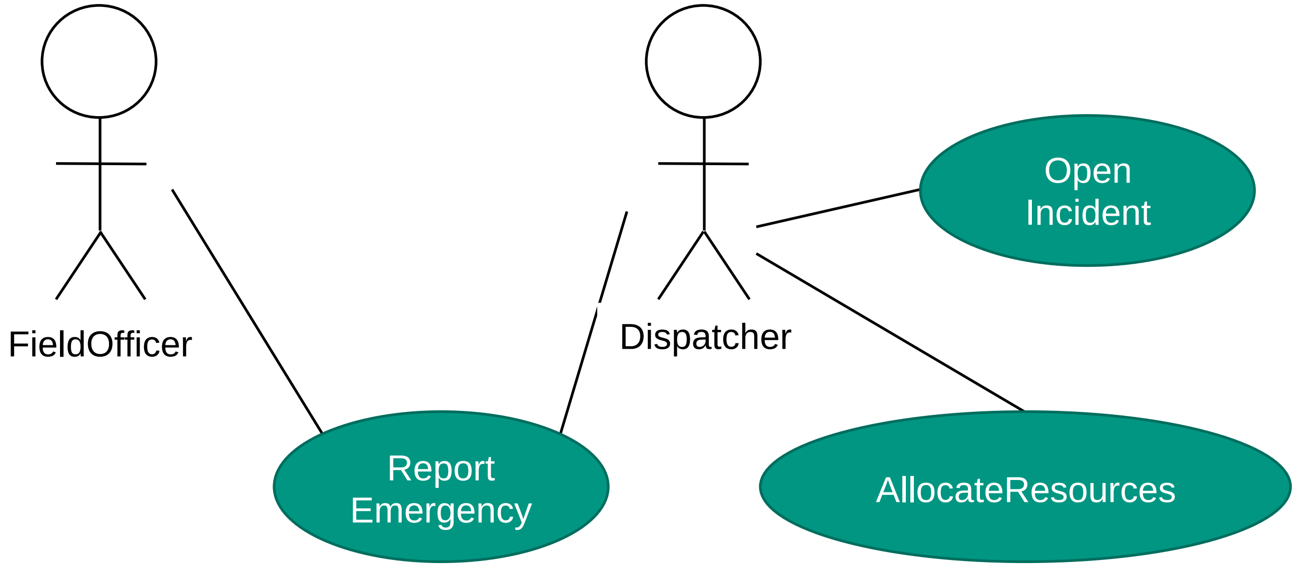
- Exceptional or rare event flows are pulled out of the normal even flow to provide a better overview.
- The arrow of the <<*extend*>>-relation points to the extended use case (a kind of subclass relation)
- Exceptional use cases can extend several other use cases.



Use Case in Text Form

- **Name:**
  - Do homework
- **Participating actor:**
  - Student
- **Entry condition**
  - Student receives assignment
  - Student is healthy
- **Exit condition:**
  - Student hands in solution
- **Flow of events**
  1. Student fetches problem statement
  2. Student reads problem statement
  3. Student solves problem and types up solution
  4. Student prints solution
  5. Student hands in printed solution
- **Exceptional conditions**
  - none
- **Special Requirements**
  - None

Use Case Model for “Report Emergency“



Short Questions

- True or false?
  - Scenarios and use cases with textual description contain essentially the same information.
  - Scenarios can be deleted after the end of the planning phase because they are documented by use cases
  - The scenario-based design can only be employed if one has a customer (discussions are not possible otherwise)

How to find use cases

- Choose a limited, vertical section of the system (for example, reporting a fire)
  - Discuss this in detail with the user, to determine the preferred method of interaction with the system
- Choose a horizontal section (for example, several scenarios) to define a larger area of the system (for example, reporting any incident)
  - Discuss this area with the clients
- Use prototypes or mock-ups to define the visual interface.
- Determine what the user does
  - Observation (good)
  - Interviews (often fuzzy answers)

Use case from scenario

- For example, **search for use cases in scenarios** that cover all instances of reporting a fire. („Report incident“ in the first paragraph of the scenarios is a candidate for a use case)
- **Document** these **use cases** as precisely as possible:
  - Participating actors
  - Entry conditions
  - Event flow
  - Exit conditions
  - Exceptional conditions
  - Special requirements

Example use case „Report Incident“

- Name: ReportIncident
- Participating actors
  - Field officer (Bob and Alice in the scenario)
  - Dispatcher (John in this scenario)
- Exceptional conditions
  - Field officer and dispatcher are informed immediately, if connection between terminal and central office is lost.
- Flow of events: see next slide
- Quality requirements:
  - Report of field officer is acknowledged within 30s.The answer of the dispatcher arrives at the field officer no more than 30s after dispatcher sends it.

Example use case „Report Incident“

1. The field officer activates the function „Report Incident“ on the terminal. The system reacts by displaying a form.
2. The field officer fills out the form by indicating the incident level, the incident type, the address, and a short description of the situation. The field officer also describes a reaction to the incident.
3. The dispatcher creates an incident in the database by calling the use case „Open Incident“. He chooses a reaction and acknowledges the report.
4. The field officer receives the acknowledgement and chooses the reaction.

Example use case „Request Resources“

- Name: RequestResources
- Actors:
  - Officer-in-charge: the person responsible for the operation
  - Resource handler: responsible for request and release of resources managed by the system
  - Dispatcher: enters, updates, closes, and deletes incidents in the system.
  - Field officer: reports incidents.
- Entry condition:
  - The resource handler has selected an available resource
- Event flow:
  - The resource handler chooses an incident
  - The resource is assigned to the incident
- Exit condition:
  - The use case finishes when the resource is assigned
  - The assigned resource is not available for other incidents.
- Exceptional conditions:
  - The officer-in-charge is responsible for the use of the resources

Writing use cases

1. Give it a name
   („ReportIncident“)
2. Find the actors:
   Generalize concrete names („Bob“) to type of actor („fieldOfficer“)
3. Find the event flow:
   Document it in natural language
4. Determine entry and exit conditions, exceptional conditions
5. Add quality requirements, if any

Exercise

Write a use case for a police patrol to report an incident of assassination (think of the assassination attempts on Donald Trump)

Problems during requirement elicitation

- Shallow domain knowledge
  - Distributed over many sources
    - Rarely explicitly recorded
  - The various sources may contradict each other
    - Affected persons may have different goals
    - Affected persons may understand problem differently
- Tacit knowledge
  - It is difficult to explain things that one uses routinely without thinking.
- Limited observability
  - Organizational blindness
  - Presence of an observer changes behavior
- Bias
  - Affected persons must not/dare not say certain things
    - Political climate, organizational factors
  - Affected persons don't want to say what is necessary, hold back information
    - Afraid of being optimized away
    - Affected persons try to influence observer for their own goals (hidden agenda)

Example of an imprecise requirement

- During an experiment, a laser beam was sent from earth to a mirror on the space shuttle Discovery.
- Experimenters expected that the beam would be reflected to the top of a mountain 10,021 feet high.
- The experimenter entered the altitude as „10023“.
- The laser beam never hit the top of the mountain.
  What was the problem?
- The computer interpreted the altitude as given in miles.
- So the laser beam went out into space to a mountain 10023 **miles** high (higher than the diameter of earth: 12750 km.)

Example for an unintended function

- From the news: In London, a subway train left the station without driver.
- What happened?
    - A passenger door was stuck and would not close.
    - Driver left the train to close the door:
        - He left the driver’s door open.
        - He relied on the specification that the train does not start with a door open.
    - Driver closed the door with a kick, and the train departed without him.
        - The driver´s door was not included in the control program.

Types of Requirements

- *Functional requirements:*
  - Describe the behavior or reaction of the system to data entered or events received, or the interaction between system and environment, independent of the implementation. („A field officer shall be able to request resources“)
- *Quality requirements* a.k.a. *non-functional requirements:*
  - Describe the quality of how a function is performed
    „Response time shall be below 1 sec.“
- *Constraints:*
  - Restrictions imposed by client, law, or environment
    „The implementation shall be in Java.“

Functional vs. non-functional requirements

Functional requirements

- describe user tasks that the system must support
- are described as actions
  - „Notify customers“
  - „Create a new table“

Non-functional or quality requirements

- describe properties of the system or the domain
- are formulated as constraints or assertions
  - „Every user input shall be recognized within 1 sec.“
  - „A system crash must not lead to loss of data“

Types of non-functional requirements

Quality requirements

- *Usability (ease of use)*
- *Reliability (frequency of failure, non-response, downtime)*
- *Fault tolerance (behavior if the system is used erroneously)*
- *Security*
- *Performance*
  - Response time (time between request and reply)
  - Throughput (number of requests handled per time unit)
- *Scalability* (increase number of users, data, performance)
- *Availability* (max downtime per time period)
- *Maintainability*
  - *Adaptability (to new HW, external systems, OS)*
  - *Extendibility (new functions)*
  - Understandability
  - *Portability (port to different HW, OS, country (language, laws)*
- …

Constraints

- Implementation (e.g. prog. language)
- Interfaces (used or implemented)
- Operational environment (PC, mobile phone, office, car, airplane, etc.)
- Scope of delivery (including source code, tests, documentation?)
- Legal obligations
  - Licenses
  - Certificates
  - Data security
- …

Definition of some quality requirments

- *Usability*
  - Ease of use of the system’s functions
  - The concept usability is often used imprecisely
  - Usability must be measurable. Otherwise, it is mere marketing.
    - Example: „Number of interactions to place an order on the internet“
    - Example: „Number of erroneous inputs per hour“
    - Usability often requires experiments to measure.
- *Fault tolerance* or *robustness*
  - The ability of the system to continue functioning, when
    - the user enters incorrect data, nonsensical choices
    - failures occur
    - Operational conditions are not maintained
      - „operating temperature between -10°C bis +50°C.“
      - „maximum number of requests is 2000/s.“
- *Availability*
  - Ratio of uptime to a time period
  - Example: „Per week, the system shall be unavailable for not more than 5 sec.“

More examples

- “Users shall be able to watch games without prior registration or knowledge” → Usability
- “The system shall support 10 games simultaneously.” → Performance
- “The manager shall be able to add a game without having to change those that are already available” → Usability

Quick Questions…

- True or False?
  - Functional requirements always come form the client.
  - The customer is not allowed to request non-functional requirements.
  - Client may prescribe implementation language and software tools
- How do you measure
  - Response time,
  - Throughput,
  - Usability?

What is *not* part of the requirements?

- System architecture, algorithms, implementation details
- Development tools
- Development environment
- Usually not prescribed: programming language, reusability

- Obviously, none of the above points should be prescribed by the client. The developers should be able to choose according to their abilities, familiarity, and experience with prog. languages and other tools. (Exceptions possible.)

Important properties of requirements

- Correctness
    - The requirements reflect the wishes of the customers correctly.
- Completeness
    - All situations, in which the system can be used, are described, including errors and faulty operation.
- Consistency
    - No requirements contradict each other (important if several people work on them)
- Clarity (Understandability)
    - The requirements can be understood by all concerned in the same way.
- Unambiguity
    - The requirements are formulated in unambigous ways.

The 4 Cs of requirements: correct, complete, consistent, clear

More desirable properties

- Feasibility
  - Requirements can be realized and delivered.
- Confirmability
  - Requirements can be tested
- Traceability
  - It should be possible to associate every requirement with a set of functions that realize it. Traceability is established during development with links

Requirements should be validated, i.e., they should be reviewed whether they fulfill these properties. This is a first step in quality control.

Problems can occur, though, when requirements change during development: Inconsistencies can creep in and must be eliminated with re-validation.

Requirements for requirements management

- Functional requirements
    - Store requirements in a configuration management system (GIT, etc.) to manage the version history.
    - Allow multi-user access to the documents
    - Generate a complete requirements document automatically from various parts
    - Support traceability of requirements

Tools for Requirements Management (background; not asked in exam)

- DOORS (Rational)
  - Tool for requirements management for distributed teams (client-server architecture)
- RequisitePro (IBM/Rational)
  - Integrated in Microsoft Word

- Recent research: analyze texts for weaknesses, for instance passive voice or nominalizations (using nouns in place of verbs)
  - example: “After registration, customer is sent to home page…” who registers, how?
  - „Transport of the pallets...“ Who transports? How? From where to where?

Types of Developments

- Greenfield development
  - Development begins from zero. There is no existing system that could be extended or built upon. Requirements come form customers.
  - Is triggered by customer requests
- Re-Engineering
  - Redesign and/or re-implementation of an existing system using new technologies, such as a new programming language, client/server architecture, etc., to make it more maintainable.
  - Is triggered by new technology or new requirements
- Interface development
  - Make existing services available in new environment, with a graphical user interface or natural language interface (Alexa, Siri, ChatGPT, etc.)
  - Is triggered by new technologies or market demand

Template for requirements documents

1. Goal of the development
2. Operating environment
3. Functional requirements
4. Product data
5. Non-functional requirements
6. System models
   a) Scenarios
   b) Use cases
7. Glossary (definitions of terms used)

Example: Seminarmanagement

1. Goal of the development

- The company Teachware shall be enabled to manage its seminar offerings with computer support.

Version	Author	QS	Date	Status	Comment
2.1	Balzert		3/2020	accepted	
2.2	Balzert		10/2020	accepted	/F115/

Vgl. „Lehrbuch der Softwaretechnik“, Balzert, 2000

Example: Seminarmanagement

2. Operating environment

- The product supports the management of customers and seminars of company Teachware.
- Target user group: employees of Teachware.
- Platform: PC with current version of MS Windows

Example: Seminarmanagement

Requirements have been shortened due to lack of space

3. Functional requirements

/FR10/
Entry, change, and deletion of customer data (participants, prospective clients)

/FR20/
Notification of customers (booking, cancellation, change notification, invoice, advertisement),

/FR30/
Entry, change, and deletion of seminar events and seminar types

/FR40/
Entry, change, deletion of lecturers and assignments of lecturers to seminar events and seminar types.

Example: Seminarmanagement

/FR50/
Entry, change, and deletion of seminar bookings

/FR60/
Generation of invoices

/FR70/
Generation of various lists (participant list, revenue list, list of certificates),

/FR80/
Queries of the following sort:

When is seminar X scheduled next? Which employees of company Y have participated in seminar X?

Example: Seminarmanagement

4. Product data

/PD10/
Store relevant data about customers

/PD20/
If a customer is employed by a company, store relevant data about the company

/PD30/
Store relevant data about seminar instances, seminar types, and lecturers

/PD40/
If a customer books a seminar instance, store the coresponding booking data

Example: Seminarmanagement

5. Non-functional requirements

/NF10/

The function /FR80/ must execute in 15 sec or less, all others must react within 2 sec.

/NF20/

The system must manage at most 50.000 participants and maximally 10.000 seminars.

/NF30/

Reliability: at most two crashes per week.

/NF40/

Availability: during regular business hours, not more than one hour per month unavailable.

/NF50/

Usability: after training of 4h, users make no more than 2 mistakes per day.

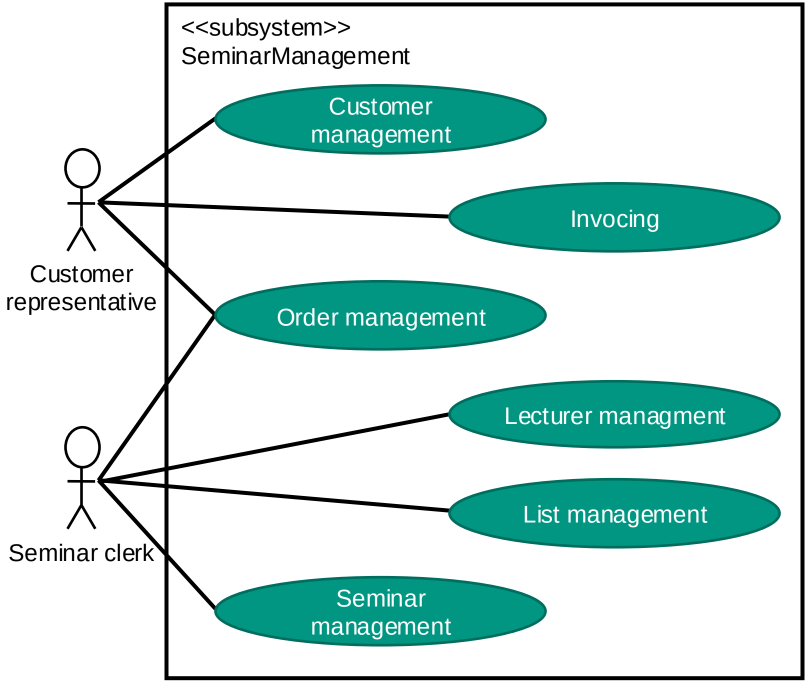
/NF60/

Portability: software contains not more than 0.1% platform-specific instructions

Example: Seminarmanagement

6. System models – Use case: „Seminar management“

- Actors:
  - Customer representative
  - Seminar clerk
- Use cases:
  - Customer management
  - Invoicing
  - ...
- (Textual description missing)



Example: Seminarmanagement

7. Glossary

- Lecturer

  Teaches one or more seminar types

- Customer

  (Paying) participant in one or more seminar events

- Seminar type

  Type of a seminar for a particular topic (e.g., „Watercolors for Beginners“)

- Seminar event

  Actually occurring seminar (e.g., „Watercolors for Beginners“, summer 2022“)

- Etc.

Feasibility Study

- Finally, scrutinize the feasibility of the project
  - Review technical feasibility
    - Software realizability
    - Availability of development computers and target computers
  - Review alternative solutions
    - Example: Purchase and adaptation of standardized software vs. individual development
  - Review availability of personnel
    - Are qualified developers available; new developers to be hired?
  - Review risks

Feasibility study continued

- Reviw economic feasibility
  - Cost and time estimates
  - Estimate profitability.
- Review legal considerations
  - Data security
  - Certification
  - Applicable standards
  - Licenses