3. Lecture

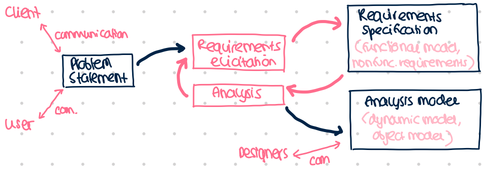
29.04.21

OUTLINE:

1) Requirements elicitation and analysis
2) Dynamic modeling
3) Extension UML with predefined types and stereotypes

Overwiew: requirements engineering

- Requirements elicitation: describes purpose of the system
- Analysis: create a model of the system, which is correct, complete, consistent, verifiable



Requirements engineering

Combination of the two activities: requirements elicitation + analysis
↳ also called „requirements analysis"

Requirements elicitation:
- Def. of the system in terms understood by a customer or user
- Result: requirements specification

Analysis:
- Def. of the system in terms understood by a developer
- Result: analysis model (also: technical specification)

Requirements elicitation

Activities during req.e.:

- Identify actors
- Identify scenarios → very detailed
- Derive use cases → more generalized
- Refine use cases
- Identify relationship among use cases
- Identify nonfunctional requirements (quality aspects of system)

Requirements elicitation is a development activity

→ determine requirements of system specified by customer/user
→ „From the problem statement to requirements specification"
→ Still a very informal process with faults
→ Many softwares fail because of poor requirements elicitation

Requirements specification vs analysis model

(both = models that focus on requirements from user's view)

→ uses natural language

uses (semi-)formal language

Requirements

→ Features that the system must have
→ Constraints that the system must satisfy
→ Describe user's view of the system
→ „what" not „how"!

Functionality, User interaction, Error handeling, Environmental condition (interfaces) ✓

System design, Implementation and development technology ✗

difficulties:

1) How can we identify the purpose of a system?
→ what are requirements/constraints?
2) How can we identify the system boundaries?
→ what is inside, outside the system?

Types of requirements elicitation

Each of these requirements elicitation types should start with a problem statement

- Greenfield engineering
  → developing from scratch
  → Req. from client & endusers
- Re-engineering
  → Re-design or re-implementation of an existing system
  → Req. triggered by new technology
- Interface Engineering
  → Provides services of an existing system in a new environment
  → Req. triggered by technology or new market needs

TYPES OF REQUIREMENTS

- Functionality: what is a software suppose to do? (external interface: interaction with people, hardware, software)

Functional req.

- Quality req.:
  → Usability
  → Reliability
  → Performance
  → Supportability
- Constraints (pseudo req.)

Nonfunctional req.

↳ FURPS acronym for model classifying software attributes

functionality

Includes:
- relationship of outputs to inputs
- response to abnormal situations
- Exact sequence of operations
- validity checks on the inputs

nonfunctional requirements (NFRs)

Criteria for defining NFRs

♡ BOUNDED: when they lack bounded context, NFRs may be irrelevant and lead to significant additional work
♡ INDEPENDENT: should be independent of each other so that they can be evaluated and tested separately
♡ MEASURABLE: NFRs that cannot be measured are too vague and can easily be misunderstood
♡ TESTABLE: must be stated with objective, measurable, testable criteria

USABILITY: The ease with which actors can use system functions

↳ • learnability
• efficiency
• Memorability
• Error handling and robustness
• Satisfaction/user experience

RELIABILITY

→ Robustness: ability of a system to maintain a function
... if: wrong input
... if: changes by environment

→ Safety: protection against unwanted incidents

→ Security: protection against intended incidents

PERFORMANCE

- number of simultaneous users supported
- amount of information handled
...

AVIABILITY

-> ratio of expected uptime of a system tho the sum of excepted uptime and downtime

ADAPTABILITY

-> ability of system to adapt to chanced circumstances

MAINTAINABILITY

-> ease with which a developer can modify the system (bug fixes, new req.)

Constraints: -> pseudo req.

- Implemented req.:
  -> usage of specific tools, programming language, frameworks (not: development technology / methodology)
- Operations req.:
  -> administration and management of the system
- Packaging req:
  -> delivery of system
- Interface req.:
  -> imposed by externally systems
- legal req.:
  -> must comply with law regulations

Model Correctness: model validation vs. model verification

model validation = equivalence check between 2 models, one of them created from the other

model verification = comparison of the model with reality (the client)

TECHNIQUES TO DESCRIBE REQUIREMENTS

Goal: bridging conceptual gap between end user and developers

- Scenario: use of system as series of interactions
  -> very specific: names, numbers, instances
  -> describes single instance of a use case (concrete)
- Use case: set of scenarios
  -> abstraction (generic)
- User story: describe a functional req. from end user perspective

-> informal description of a feature of the system used by actor

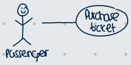
used in:
Req. elicitation
client acceptance test
System development

TYPES OF SCENARIOS:

- AS-IS scenario -> current situation or usage of existing system
  - Re-engineering projects, user describes system
- Visionary scenario -> future system
  - Greenfield, reengineering
- Evaluation scenario -> user task against which the system is to be evaluated
  - Demos & acceptance test
- Training scenario -> step by step instructions that guide a novice user through a system
  - System development

Scenario example (formalized)

1) Name
2) Participating actors
3) Flow of events



1) Name: Purchase ticket ← Instance

2) Participating actors: Joe: Passenger ← Instance

3) Flow of events:

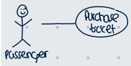
actor step
1. Joe wants to take a trip and selects a single day ticket for Munich zone M-2
   2. Ticket machine displays price of 9.00 €

system step
3. Joe inserts a 20€ bill
   4. Ticket machine returns 11€
   5. –"– prints the ticket
6. Joe takes change and ticket and leaves

Refining scenario into a (more general) use-case

Textual use case description: example

1) Name
2) Participating actors
3) Flow of events
4) Entry conditions
5) Exit conditions
6) Special requirements



No! instances

1) Name: Purchase ticket

2) Participating actors: Passenger

3) Flow of events:
1. Passengers select number of zones
   2. Ticket machine displays amount due
3. Passenger inserts at least the amount due
   4. Ticket machine returns change
   5. Ticket machine issues ticket

4) Entry conditions:
- Passenger stands in front of the Ticket machine
- Passenger has sufficient money

5) Exit conditions:
- Passenger has a ticket

6) Special req:
- Ticket machine is connected to a power source

Requirement quality criteria (validation)

- Correctness
- Clarity
- Completeness
- Consistency
- Realism
- Traceability

NOT Part of requirements:

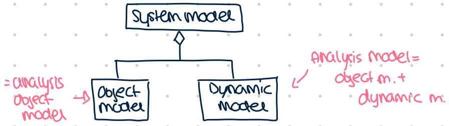
- description of system structure
- development methodology
- development environment
- specific implementation language
- –"– technology

no constraints of a client!

ANALYSIS:

analysis concepts:

- Analysis model: object model and dynamic model of a system to be developed.
- Entity, boundary, control objects: object divisible into 3 categories describing their use inside the system
- Generalization & Specialization: hierarchies; inheritance, abstraction

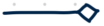


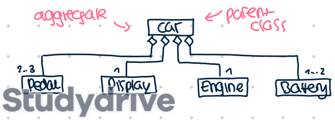
Object model:

-> defines structure of system by identifying object, attributes, methods, associations

- Inheritance
- Aggregation
- Composition
- Dependency
- Unidirectional association
- Bidirectional –"–

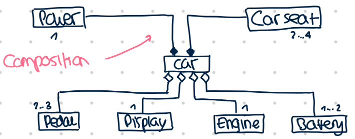
Aggregation: „part of"



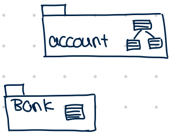


Composition: Special form of aggregation →◆

→ components do not exist without the aggregate
(aggregation prefered over composition tho)



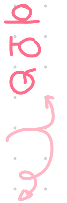
UML Package Notation:



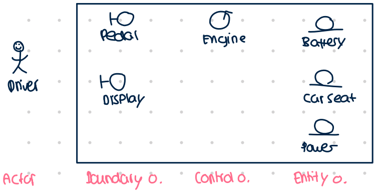
- → increase readability
- → organizes classes into subsystems
- → Decomposition into subsystems

Different types of objects:

- ♡ Entity objects: persistent info tracked by system
- ♡ Boundary objects: interaction between user ↔ system
- ♡ Control objects: control tasks to be performed by system



Example: STEREOTYPE GRAPHIC



Pros and Cons: Stereotype graphics

+ advantages

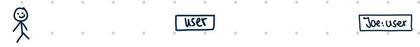
- → UML diagrams easier to understand
- → increase readability even to clients that are not trained in UML

- disadvantages

- → if unfamilier with the graphics ↳ harder to understand
- → additional icons add to burden of learning UML

IMPORTANT DISTINCTION:

actor vs. class vs. object



- actor: → any entity outside the system, interacting with it
- class: → a concept from application- or solution domain → classes part of system model
- object: → A specific instance of a class

Why all these models?

Functional model: describes functionality of the system ← using use cases and scenarios

Object model: describes structure of the system ← using classes, attributes, operations & associations

Dynamic model: describes dynamic behavior of the system

DYNAMIC MODELLING: → describes behavior of objects in system

- State chart diagrams: states of 1 class
- Activity diagrams: workflows within use cases
- Communication diagrams: interaction between multiple classes/objects

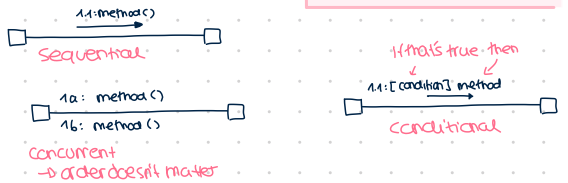
UML communication diagrams

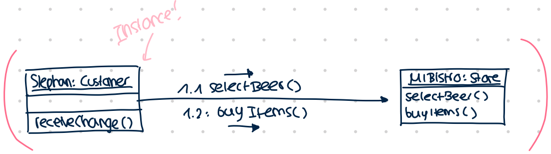
- → interaction as flow of messages (i.e. call of methods)
- → Describes static structure + dynamic behavior of the system
- → Reuse layout of classes & associations in class diagram
- → Messages are labeled with numbers (order)

3 different types of messages

1) sequential m. x.x : method ()
2) conditional m. x.x : [cond] method ()
3) Concurrent m. x.a : method

Examples:





Recipe:
class diagram →to communication diagram

1. take all steps from the event flow of a use case
2. Instantiate the participating objects
3. Number messages from each of the steps in the event flow
4. Is there a corresponding method?
   → NO: add a public method
5. Draw the message from sender to receiver

Identification of classes and operation from dynamic model

- → application domain → talking/observing end user
- → General world knowledge
- → Textual analysis of event flow in use case (Abbott)

Generalization and Specialization

Generalization

- → identifies abstract concepts from lower-level ones
- → identifies common features → to create abstract concept

„from low level to high level"
„from subclass to superclass"

Specialization

- → Identifies specialized concepts from higer-level ones
- → more specific concepts from higer-level one

„from high level to low level"
„from superclass to subclass"