LECTURE 4

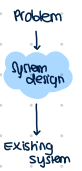
06.05.21

OUTLINE

1) Design

- Analysis: focuses on application domain
- Design: focuses on solution domain → - changing really quickly - cost of hardware is rapidly sinking
  - ↳ Design knowledge = moving target
- Design window: Time in which design decisions have to be made

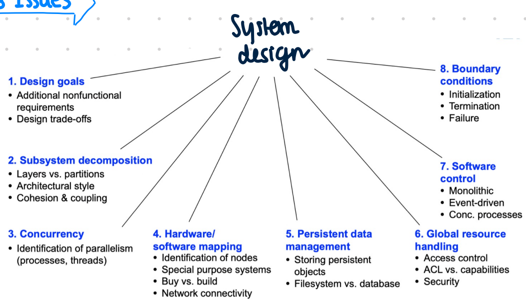
SCOPE OF SYSTEM DESIGN



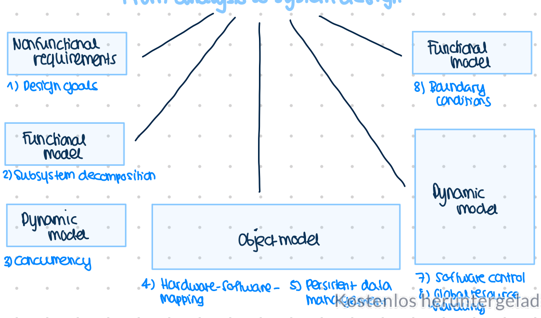
⇒ Bridges gap between a problem and an existing system

HOW:
- use „divide and conquer"
  - → identify design goals
  - → Model the new system as a set of subsystems
  - → Address the major design goals first

8 Issues



From analysis to system design

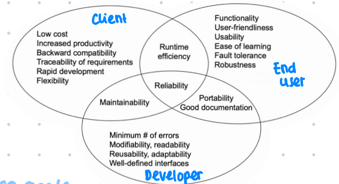


2) Design goals and trade-offs

definition

- govern the system design activities
- As a starter: any non-functional requirement is a design goal
- Additional design goals are identified with respect to → Design methodology, Design metrics, Implementation goals
- Design goals often conflict with each other
  - ↳ TRADE-OFFS

Different types of design goals (example)



Typical design goals TRADE-OFFS

- Functionality vs. usability
- Cost vs. robustness
- Efficiency vs. portability
- Rapid development vs. functionality
- Cost vs. reusability
- Backward compatibility vs. readability

3) Subsystem decomposition

▶ Definition:

- **Subsystem:**
  - ⇒ collection of classes, associations, operations, events that are closely interrelated with each other
- **Service**
  - ⇒ A group of externally visible operations provided by one subsystem (also called subsystem interface)
  - ⇒ The use case in the functional model provide the seeds for services

Subsystem interface

- → Set of fully typed UML operations
- Specifies the interaction and information flow from and to system boundaries, but not inside the subsystem
- Refinement of services, should be well-defined and small
- Subsystem interfaces are defined during object design

Application programming interface (API)

- API = specification of the subsystem interface in a specific programming language
- APIs defined during object design

Coupling and Cohesion

Goal: Reduce system complexity while allowing changes

Cohesion

⇒ measures dependencies between classes within one subsystem

✓ High cohesion: classes have similar tasks → many associations

✗ Low cohesion: lots of miscellaneous and auxiliary classes

Coupling

⇒ Measures dependency between multiple subsystems

✗ High coupling: changes to one subsystem → huge impact

✓ Low coupling: changes to one subsystem will not affect the other ones

Good system design: High Cohesion - Low Coupling

How to archieve high cohesion and low coupling?

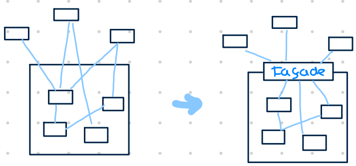
High cohesion	Low coupling
→ operations work on the same attributes → operations implement a common abstraction or service	→ small interfaces → Information hiding → no global data → Interactions mostly within subsystem

Façade design pattern: reduces coupling

↳ provides unified interface for subsystem

- Defines a higher level interface that makes the subsystem easier to use

→ hides spaghetti design



- reduces complexity
- less recompilations
- can be used during testing
- mock objects

Ways to deal with complexity

- Abstraction
- Hierarchy
- Taxonomies
- Decomposition

Decomposition

→ Technique to master complexity ("divide & conquer")

→ 2 major types:

- Functional decomposition
- Object oriented decomposition

FUNCTIONAL D.

- System = decomposed into functions
- Functions can be decomposed into smaller functions

OBJECT ORIENTED D.

- System is decomposed into classes ("objects")
- Classes can be decomposed into smaller classes

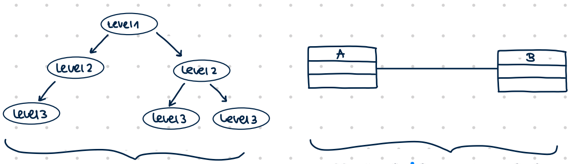
PROBLEM: functionality is spread all over the system

- Source code hard to understand
- ↳ complex and impossible to maintain
- User interface often awkward & non intuitive

→ a maintainer must understand the whole system before making a single change to the system

Model based software engineering approach

1) Focus on functional requirements
2) Find corresponding use cases
3) Identify the participating objects
4) use these participating objects to create the first iteration of the analysis object model



use case model → **object model**

4) Architectural styles

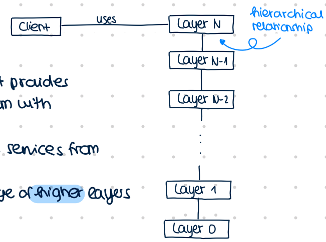
Architectural styles vs. (software) architecture

= a pattern for a subsystem decomposition

= instance of an architectural style

→ subsystem decomposition: identification of subsystems, services, and their relationships to each other

Layered architectural style



- A layer = a subsystem that provides a service to another subsystem with the following restrictions:
  - → a layer only depends on services from lower layers
  - → a layer has no knowledge of higher layers

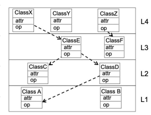
2 major types of hierarchical relationships between layers

- Layer A depends on layer B for its full implementation (= usage dependency in UML)
- Layer A calls layer B (runtime dependency)

Closed architecture (opaque layering)

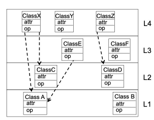
A layered architecture = closed, if each layer can only call operations from the layer directly below („direct addressing")

DESIGN GOALS:
- maintainability
- flexibility
- portability
- → low coupling



Open architecture (transparent layering)

A layered architecture = open, if each layer can call operations from any layer below („indirect addressing")



DESIGN GOALS:
- high performance
- real-time operation support

→ high cohesion

4.1) Layered architecture

3 layered architectural style

= architectural style where an application consists of 3 hierarchically ordered layers

→ often used for the development of web applications

3 tier architecture

= a software where the 3 layers are allocated on 3 separate hardware nodes

NOTE: Layer = Type (e.g. class, subsystem)

Tier = Instance (e.g. hardware node, object)

4 layered architectural style

→ hierarchically ordered layers

⇒ if these layers reside on different HW-nodes ⇒ 4-Tier-architecture

7 layered architectural style

- ISO's OSI Reference Model
  - Open System Interconnection
  - International Standard organization

Application layer: = system you are building

Presentation layer: performs data transformation services

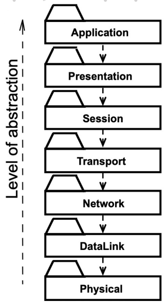
Session layer: responsible for initializing a connection

Transport layer: responsible for transmitting messages

Network layer: ensures transmission & routing

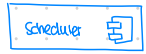
Data Link layer: models frames

Physical layer: represents hardware interface to the network



5) UML Component diagrams

- A UML component is a building block of the system. It is represented as a rectangle with a tabbed rectangle symbol inside



- Components have different lifetimes
  - only at design time: classes, associations
  - only until compilation time: source code, pointers
  - at link time or runtime: linkable libraries, executables, addresses

UML component diagram

→ model top view of system design in terms of components & dependencies

→ also called „software wiring diagrams"

→ use UML interfaces

D. 2 Types of Interfaces:

- a provided interface —○
- a required interface —(

- Dependency - - - - - →
  one component depends on implementation of another component
- Port □
  interaction point between component and environment

4.2) Client-server architechture

- often used in design of database system
  - Client = user application
  - Server = database access and manipulation

Functions performed by:

Client: - Input by user (customized user interface)
- Sanity check on input data

Server: - Centralized data management
- Provision of data integrity & db consistency
- Provision of db security

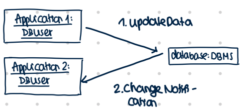
- One or more servers → provide services to client
- Client calls method offered by server
  - server performs service → returns results to client
  - Client knows interface of server (not other way around)
- Response typically immediately
- End users interact only with client

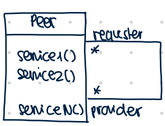
Design goals:

- portability
- Location transparency
- High performance
- Scalability
- Flexibility
- Reliability

PROBLEMS

- C-S-system use a request-response protocol
- Peer-to-peer communication is often needed





Peer to peer architectural style

- Generalization of the client-server style
  → Clients can be servers & servers can be clients
- New abstraction: PEER

I've got a summary of all the Design patterns uploaded (it's a little bit older and called „EIST_Design-Patterns")

Ty for studying with me~ good luck :)