Lecture 11: Architecture & Design

Contents of the Block “Design”

(i) Introduction and Vocabulary
(ii) Principles of Design
   a) modularity
   b) separation of concerns
   c) information hiding and data encapsulation
   d) abstract data types, object orientation
(iii) Software Modelling
   a) views and viewpoints, the 4+1 view
   b) model-driven/based software engineering
   c) Unified Modelling Language (UML)
   d) modelling structure
      1. (simplified) class diagrams
      2. (simplified) object diagrams
      3. (simplified) object constraint logic (OCL)
   e) modelling behaviour
      1. communicating finite automata
      2. Uppaal query language
      3. basic state-machines
      4. an outlook on hierarchical state-machines
(iv) Design Patterns

Goals and Relevance of Design

(i) structuring the system into manageable units (yields software architecture).
(ii) concretising the approach to realise the required software.
(iii) hierarchical structuring into a manageable number of units at each hierarchy level.

The structure of something is the set of relations between its parts.

Oversimplified process model:
principles of (architectural) design

— the degree to which a system or computer program is composed of components which try to cover different task areas, tend to be unnecessarily complex, and their relationships to each other and to the environment, and the principles guiding its design and evolution.

— the fundamental organization of a system embodied in its architecture, components, interfaces, and other characteristics of a system or component.

— the result of the process in (1).

— the process of defining the architecture, components, interfaces, and other characteristics of a system or component.

— one of the parts that make up a system. A component may be hardware or software and may be subdivided into other components. A component can be a module or a set of software units and their relations, if they to communicate and record a system's architecture. An architectural description conveys a set of views each of which depicts the system by describing its function or set of functions.
Information Hiding and Data Encapsulation

(i) Information hiding and data encapsulation not enforced,

(ii) information hiding and data encapsulation by modules,

(iii) information hiding and data encapsulation not enforced,

With information hiding and data encapsulation, components can be validated in isolation.

Components may use of this component.

Advantages:

- What is hidden is information which other components need not know from using information about the module that is not in the modules interface specification.
- The "need to know principle"—A software development technique in which each modules interfaces grouping for one component what other
- Information hiding come at the price making explicit which data and operations of which other component.
- It is more efficient to read a component's data directly than calling an operation to provide the intermediate results is needed in a different context.

Information hiding and data encapsulation by modules may be enforced by hiding encapsulated data in a component which offers operations to access (read, write, etc.) the data.

Do not access data (variables, files, etc.) directly where needed, but encapsulate the data in a component which offers operations to access (read, write, etc.) the data.

Examples:

- Example: Collecting Names
- Example: Game Object
- Example: Data Type Modules
- Example: Functional Modules
- Example: Data Object Modules
- Example: Classification of Modules
- Example: Data Encapsulation

Math occurrence seldom, one example is Java's class java.util.List

Implementations: Plain

Operations:

\begin{verbatim}
#include <algorithm>

// example:

std::vector<int> names;

// insert example:

names.insert(names.begin() + i, n);

// remove example:

names.erase(names.begin() + i);

// bounds example:

if (names.begin() > names.end()) { std::cout << "Names bounds are invalid."; }

// insert example:

names.insert(names.begin() + i, n);

// for one component what other component what other

\end{verbatim}
Software Modelling

Data Encapsulation / Information Hiding

Object Oriented

AVOID PROBLEM

/*
 * Avoid the problem of exposing the internal structure of objects to external code
 * by using data encapsulation.
 */

#include <string>

class Names {

private:
    std::vector<std::string> names;

public:
   Names() ;
    ~Names() ;
    void insert ( std::string n ) ;
    void remove ( std::string n ) ;
};

void main ()
{
    Names names ;
    names.insert ( "Mayer" ) ;
    names.insert ( "Naumann" ) ;
    names.insert ( "Wernersen" ) ;
    names.insert ( "Neumann" ) ;
    names.insert ( "Schultz" ) ;
    names.insert ( "Mayer" ) ;
    names.insert ( "Wernersen" ) ;

    for ( int i = 0 ; i < 2 ; i++ )
    {
        std::string name = "Name " + std::to_string ( i ) ;
        names.insert ( name ) ;
    }

    int count = 0 ;
    for ( int i = 0 ; i < 2 ; i++ )
    {
        count += names.count ( "Name " + std::to_string ( i ) ) ;
    }

    std::cout << "Count: " << count ;
}

// Tell Them What You've Told Them

// Example: Data Encapsulation + Data Exposition / Information Hiding

// Example: Data Encapsulation + Data Exposition / Information Hiding
**Views and Viewpoints**

A **view** — A representation of a whole system from the perspective of a related set of concerns.

IEEE 1471 (2000)

A **viewpoint** — A specification of the conventions for constructing and using a view. A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis.

IEEE 1471 (2000)

A **perspective** is determined by concerns and information needs:
- **team leader**, e.g., needs to know which team is working on what component,
- **operator**, e.g., needs to know which component is running on which host,
- **developer**, e.g., needs to know interfaces of other components.
- etc.

An Early Proposal: The 4+1 View

(Kruchten, 1995)

- **Logical View** — How is the system under development integrated into (or seen by) its environment; with which other systems (including users) does it interact.
- **Process View** — How and when are components instantiated and how do they work together at runtime.
- **Deployment View** — How are component instances mapped onto infrastructure and hardware units.

Possibly: assignment of development, test, etc. onto teams.

**Purpose of architecture:** support functionality; functionality is not part of the architecture.?!
A Brief History of the Unified Modelling Language (UML)

- 1992, Jacobson et al. (Object-Oriented software engineering (OMS): driven by framework)
- 1993, Booch (Booch Method and Notation)
- 1990, Rumbaugh et al. (Object-Oriented modelling: software engineering (MOSE))
- 1990, Rumbaugh et al. (Object-Modeling Technique (OMT))
- 1990, Jacobson et al. (OMT)

- Inflation of notations and methods, most prominent:
  - Early 1990's
  - Mid 1990's
  - Software Crisis

- Software/Process and Physical View

- Ideas: learn from engineering disciplines to handle growing complexity.
- Boxes/lines and finite automata are used to visualise software
- For example, a simple smartphone App, process and physical view may be trivial or determined by framework. For more complex systems, data and physical views are essential.

- The process aspect of a system is to describe which part of the overall software is running on which ECU.
- The physical aspect describes which function is used when? Event triggered, time triggered, continuous, etc.
- Large number of electronic control units (ECUs) spread all over the car,
  - Reflective: more on the software side,
  - Constructive: more on the physical side,
  - Assertive: both.

- UML Syntax

- Notations
  - Use case diagrams
  - Activity diagrams
  - Class diagrams
  - Component diagrams
  - Sequence diagrams
  - Interaction diagrams
  - Statecharts

- UML Models
  - System models

- UML Semantics
  - Refinement of models
  - Interpretation of models

- UML Tools
  - Use cases
  - Activity diagrams
  - Class diagrams
  - Component diagrams
  - Sequence diagrams
  - Interaction diagrams
  - Statecharts

- UML Applications
  - Software engineering (SE)
  - Software architecture (SA)
  - System engineering (SE)
  - System architecture (SA)
  - System design (SD)
  - System implementation (SI)

- UML Related
  - Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams

- UML Bibliography
  - Statecharts
  - StateMate
  - Harel et al.

- UML References
  - ISO/IEC 14761 (UML)
  - OMG Model Driven Architecture (MDA)
  - OMG Model Driven Development (MDD)

- UML Development
  - UML 1.1
  - UML 2.0
  - UML 3.0

- UML Tools
  - Rational Rose
  - IBM Rational Rose
  - Microsoft Visio
  - Adobe Illustrator

- UML Case Studies
  - Automotive industry
  - Aerospace industry
  - Telecommunications industry
  - Financial services industry

- UML Best Practices
  - Model-driven development (MDD)
  - Model-driven architecture (MDA)
  - Model-driven engineering (MDE)
  - Model-driven design (MDD)
  - Model-driven implementation (MDI)

- UML History
  - 1990: First edition of the UML standard
  - 1997: Second edition of the UML standard
  - 2005: Third edition of the UML standard

- UML Evolution
  - Initial ideas:
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)

- UML Future
  - Future trends:
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)

- UML in Practice
  - UML in industry
    - Automotive industry
    - Aerospace industry
    - Telecommunications industry
    - Financial services industry
  - UML in academia
    - Computer science
    - Information technology
    - Engineering

- UML in Research
  - UML-related research
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)
  - UML-related conferences
    - Model Integrated Development (MID)
    - Model Driven Architecture (MDA)
    - Model Driven Engineering (MDE)
    - Model Driven Implementation (MDI)

- UML in Education
  - UML in education
    - Computer science
    - Information technology
    - Engineering
  - UML in training
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Standards
  - UML in standards
    - ISO/IEC 14761 (UML)
    - OMG Model Driven Architecture (MDA)
    - OMG Model Driven Development (MDD)
    - OMG Model Driven Engineering (MDE)
    - OMG Model Driven Implementation (MDI)

- UML in Legislation
  - UML in legislation
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Policy
  - UML in policy
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Practice
  - UML in practice
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Research
  - UML in research
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Education
  - UML in education
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Standards
  - UML in standards
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Legislation
  - UML in legislation
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Policy
  - UML in policy
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Practice
  - UML in practice
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Research
  - UML in research
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Education
  - UML in education
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Standards
  - UML in standards
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Legislation
  - UML in legislation
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)

- UML in Policy
  - UML in policy
    - Model-driven design (MDD)
    - Model-driven architecture (MDA)
    - Model-driven engineering (MDE)
    - Model-driven implementation (MDI)
Software Architecture Foundations, Theory, and Practice

Object-Oriented Modeling and Design

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules.


Softwaretechnik: Methodisches Programmieren im Großen

Software Engineering

IEEE Software

IEEE Recommended Practice for Architectural Description of Software-Intensive Systems

IEEE Standard Glossary of Software Engineering Terminology


Object-oriented Analysis and Design with Applications