

# Principles of Software Design

## Domain analysis, modeling

Robert Lukočka

lukotka@dcs.fmph.uniba.sk

www.dcs.fmph.uniba.sk/~lukotka

M-255

# System, model view[1]

- System: A set of elements organized to achieve certain objectives form a system. Systems are often divided into subsystems and described by a set of models.
- Model: Model is a simplified, complete, and consistent abstraction of a system, created for better understanding of the system.
- View: A view is a projection of a system's model from a specific perspective.

# What do we model

There are two distinct things to model while creating software

- Domain model - models the domain and the concepts from the domain.
- Model of the SW system we build.

While modeling SW system, modeling with various level of detail is useful

- Classes, most important attributes, relations between classes.
- + most of the methods, however, some aspects are omitted for model simplicity
- Full implementation model, can be used as a template for implementation

# What makes a good model? [2]

- Clearly specified **object of modeling**, that is, it is clear what thing the model describes.
  - an existing artifact
  - physical system
  - a collection of ideas about a system being constructed
- **Specified purpose** and contributes to the realization of that purpose
  - communication between stakeholders
  - verification of specific properties (safety, liveness, timing...)
  - analysis and design space exploration
  - code generation
  - test generation

A model can be descriptive or prescriptive. If a model has to serve several distinct purposes then often it is better to construct multiple models rather than one.

# What makes a good model? [2]

- **Traceable:** each structural element of a model either
  - corresponds to an aspect of the object of modeling,
  - encodes some implicit domain knowledge,
  - encodes some additional assumption. A distinction must always be made between properties of (a component of) a model and assumptions about the behavior of its environment.
- **Truthful:** relevant properties of the model should also carry over to (hold for) the object of modeling.
  - In the construction of models often idealizations or simplifications are necessary.
  - The modeler should always be explicit about such idealizations/simplifications, and have an argument why the properties of the idealized model still say something about the artifact.

# What makes a good model?[2]

- **Simple**
- Extensible and reusable.
  - Ideally, it should be possible to model a whole class of similar systems.
- Interoperability and sharing of semantics.

# Some popular approaches to analysis and design

## Object-Oriented Analysis And Design

- It's a structured method for analyzing, designing a system by applying the object-orientated concepts, and develop a set of graphical system models during the development life cycle of the software.
- Prevalent style of analysis and design (the amount of visualization varies), including if the system is not built using OO programming.

## Domain driven design:

- placing the project's primary focus on the core domain and domain logic;
- basing complex designs on a model of the domain;
- ...

# UML

Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. [4]

- Developed in 1994 by Booch, Jacobson, Rumbaugh at Rational Software
- Managed by Object Management Group, published as a ISO standard
- Most recent revision is 2.5.1 from december 2017.

UML does not have a compiler → older versions are frequently used, adaptation of new features is slow.



# UML - Diagram types

- Use case diagrams
- Activity diagrams
- **Class diagrams**
- Sequence diagrams
- ...

# UML - Class diagram concepts

UML Class, property, operation, Interface

# UML Relationships

- Association, N-ary association, association class,
- Aggregation,
- Composition
- Generalization
- Dependency (e.g., use, call, create, required interface, interface realization).

# UML Relationships

## Examples:

- Association - class has an attribute that refers to the other class
- Aggregation - has a list of references to other class. Many believe, that one should use association instead of aggregation [6] unless you add a specific meaning to it.
- Composition - the class owns instances of the other class, it is responsible for their life cycle.
- Generalization - subclass superclass
- Dependency
  - use, call - gets a reference as a method parameter and does something with it (calls a method)
  - create - e.g. factory pattern
  - required interface - e.g. interface required in constructor or in method call,
  - interface realization - implements interface

# UML - Relationship properties

- naming the relationship (especially association), naming can have a direction to make it more meaningful
- naming the endpoints
- multiplicity
- visibility (not to be confused with the direction of relationship name)
- constraints

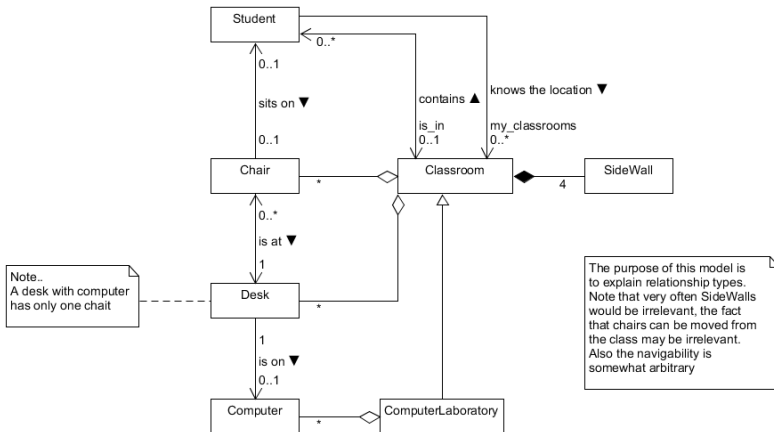
# UML - other important features

- Comments
- Constraints
- Stereotypes

# UML - how to draw a class diagram

- Minimize crossings (if you need many crossings, your model is too complex and probably lacks some abstraction)
- Don't repeat yourself
  - If a class A is associated with class B and e.g. the end of the relationship at B has the name c, you **should not** class A should not have attribute c of type B.
  - If the name of the attribute end is b (or bs i in case to many multiplicity) the naming is unnecessary.
- The diagram should have a subject of modeling. This includes time scope. Some relationships are static within the scope, some are created and destroyed. Your model should have means to do this.
- The model should be able to perform use cases within its scope.

# UML - Relationship examples





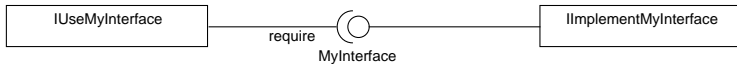
# UML - Relationship examples

Note the importance of “knows the location” relation. Without it we would not be able to make any reasonable scenario in the model.

- Scenario preconditions: Students knows the room, the room contains chairs, desks and computers and they are correctly arranged.
- Step 1: Student enters the room
- Step 2: Student finds a chair.
- Step 3: Student sits in a chair.
- Step 4: Student leaves the chair.
- Step 5: Student leaves the room.

The model can be improved in several ways. But you can do stuff in it. There is an implementation of a simulation attached to this model.

# UML - Various interface notations



# Domain model : Restaurant

- Aim: describe how the use case "Customer gets a dish" is performed.
- Note that it is possible for the customer to sit at a table and order a dish and so on.
- Note that dealing with how tables are assigned to the staff is outside the scope, thus we can assume that the associations between the waiters and the tables are static and need not to be created within the model.

# How to make a good model

How to make a model describing complex world simple and truthful? Abstraction

- Abstract from things that are not relevant to the problem.
- Question relevancy of apparently obvious classes and relationships.
- Should a concept be a subclass or is it just an attribute?
- Develop concepts that capture common characteristic of various elements in the model.

# How to make a good model : Examples

Abstraction of types:

- Instead of separate classes, one can just have type properties
- We can control which values are allowed for which property using a relation between properties and values

# How to make a good model : Examples

## Abstraction of attributes

- Instead of attributes one can have just a list of attributes
- We can control which attributes are allowed for a given type using a relation between type and properties

# How to make a good model : Examples

## Abstraction of relationships








- Instead of many relationships we can make a relationship class that generalizes all of them.
- This can be used not only to generalize relations between two classes
- We can associate relationship types with its allowed operands.

# Resources I

- UML Class and Object Diagrams Overview
- Martin Fowler - Aggregation vs composition



# References I

-  [Tutorialspoint: OOAD - UML Analysis Model](#)
-  [Frits Vaandrager - What is a Good Model?](#)
-  [Omar ElGabry : Object-oriented analysis and design](#)
-  [Wikipedia - UML](#)
-  [R: Červenka: Analysis patterns](#)
-  [Martin Fowler - Aggregation vs composition](#)
-  [B. D. McLaughlin, G. Pollice, D. West : Head First Object-Oriented Analysis and Design, O'Reilly Media, Inc., 2007.](#)