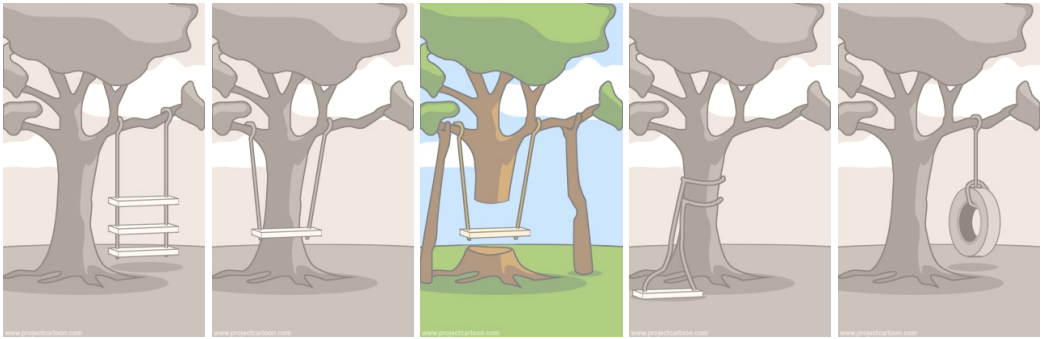
Software Engineering

5. Software Design | Thomas Thüm | November 17, 2021

Why Software Design?



how the customer explained it | how the project leader understood it | how the analyst designed it | how the programmer implemented it | what the customer really needed

Lecture Overview

1. Introduction to Software Design
2. Modeling Structure with Class Diagrams
3. Modeling Interactions with Sequence Diagrams

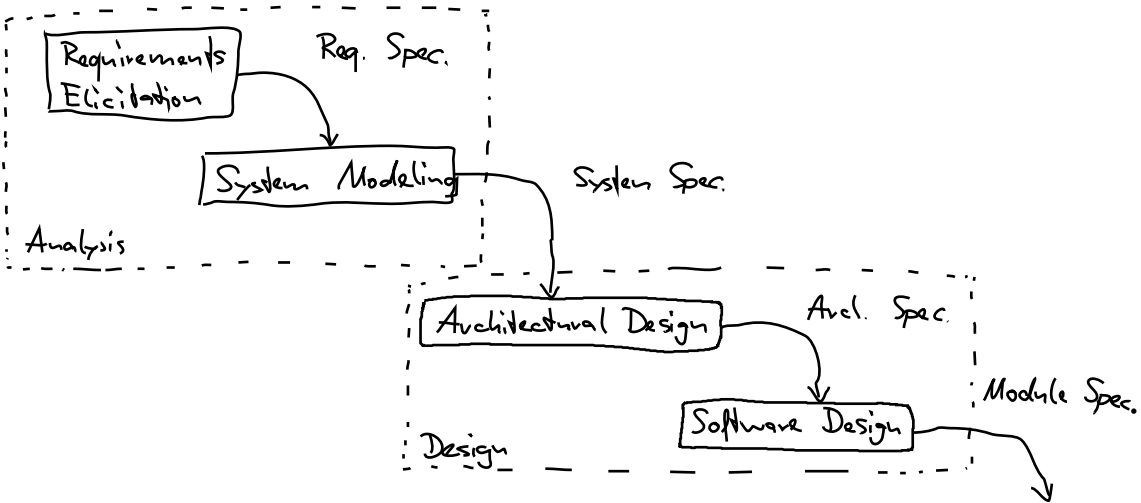
Lecture Contents

1. Introduction to Software Design
   - Analysis and Design
   - Recap: 14 Types of UML Diagrams
   - Lessons Learned
2. Modeling Structure with Class Diagrams
3. Modeling Interactions with Sequence Diagrams

Louis Srygley

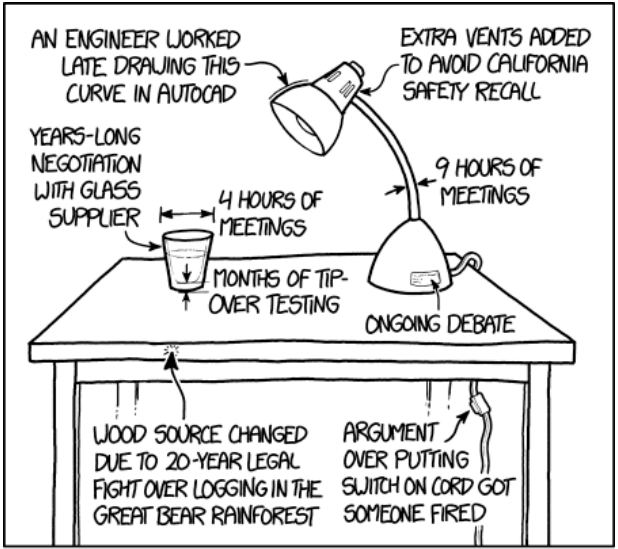
"Without requirements or design, programming is the art of adding bugs to an empty text file."

Analysis and Design



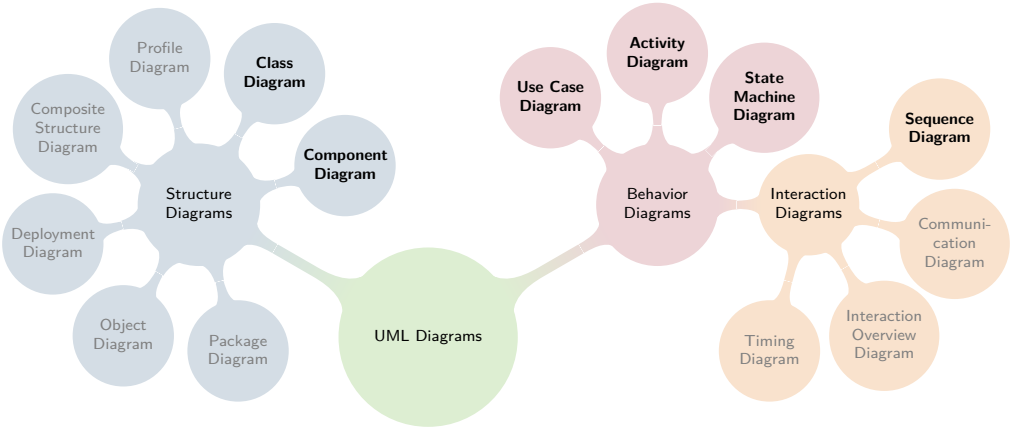
Alistair Cockburn

"If it's your decision, it's design; if not, it's a requirement."



SOMETIMES I GET OVERWHELMED THINKING ABOUT THE AMOUNT OF WORK THAT WENT INTO THE ORDINARY OBJECTS AROUND ME.

Recap: 14 Types of UML Diagrams [UML 2.5.1]



Introduction to Software Design

Lessons Learned

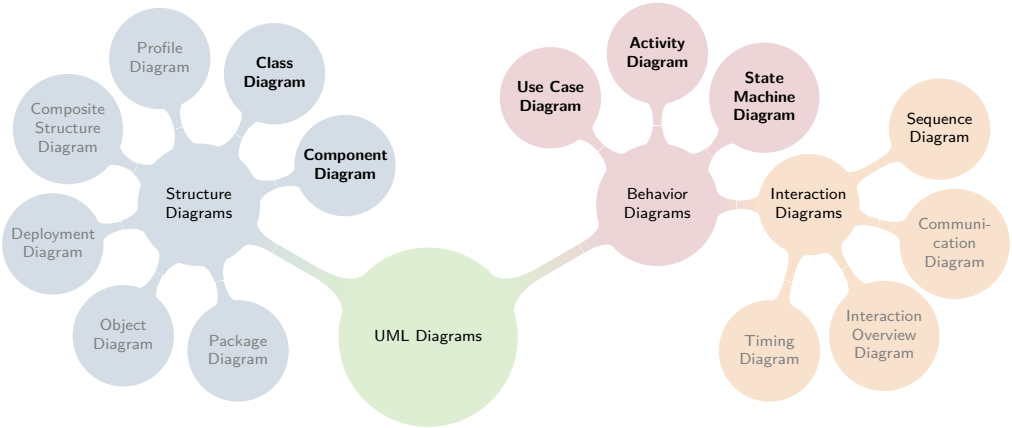
- What is the role of software design?
- How is it connected to requirements and architecture?
- Further Reading: Sommerville, Chapter 7.0–7.1 (p. 196–209)

Practice

- See Moodle
- In Moodle there is a cloze (Lückentext) on object-oriented programming.
- Pick exactly one sentence and submit the completed version. If all sentences are filled, think of a further sentence about object orientation that has wholes, too.

Lecture Contents

Recap: 14 Types of UML Diagrams [UML 2.5.1]



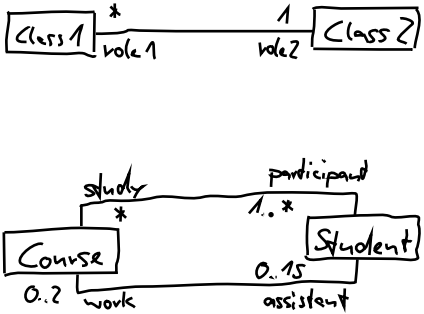
Class Diagrams (Klassendiagramme)

Class Diagram [UML User Guide]

"A **class** is a description of a set of objects that share the same attributes, operations, relationships, and semantics. [...] An **association** is a structural relationship that specifies that objects of one thing are connected to objects of another. [...] When a class participates in an association, it has a specific role that it plays in that relationship; a **role** is just the face the class at the far end of the association presents to the class at the near end of the association. [...] When you state a **multiplicity** at the far end of an association, you are specifying that, for each object of the class at the near end, how many objects at the near end may exist."

Example Multiplicities (default=*)

0..* (=*) or 1..* or 0..1 or 1..1 (=1) ... 2..5 ...



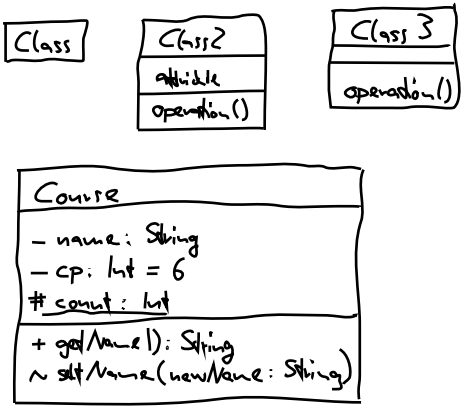
Attributes and Operations of Classes

Attributes and Operations [UML User Guide]

"An **attribute** is a named property of a class that describes a range of values that instances of the property may hold. [...] An **operation** is the implementation of a service that can be requested from any object of the class to affect behavior. In other words, an operation is an abstraction of something you can do to an object that is shared by all objects of that class." **Static** attributes and operations exist only once for each class and are underlined (opposed to **instance** ones).

Visibility Modifiers [UML User Guide]

−: private is available only in this class
+: public is available from each class
#: protected is available from each subclass
~: package is available in classes of same package



Completeness of Attributes and Operations

UML User Guide:

"When drawing a class, you don't have to show every attribute and every operation at once. In fact, in most cases, you can't (there are too many of them to put in one figure) and you probably shouldn't (only a subset of these attributes and operations are likely to be relevant to a specific view). For these reasons, you can elide a class, meaning that you can choose to show only some or none of a class's attributes and operations."

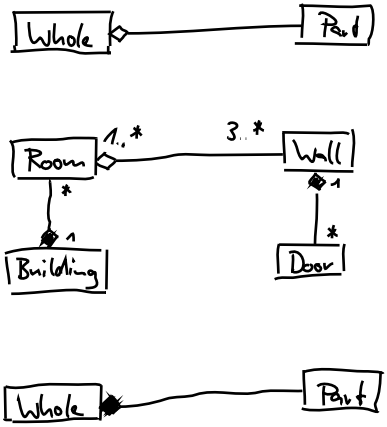
Aggregation and Composition of Classes

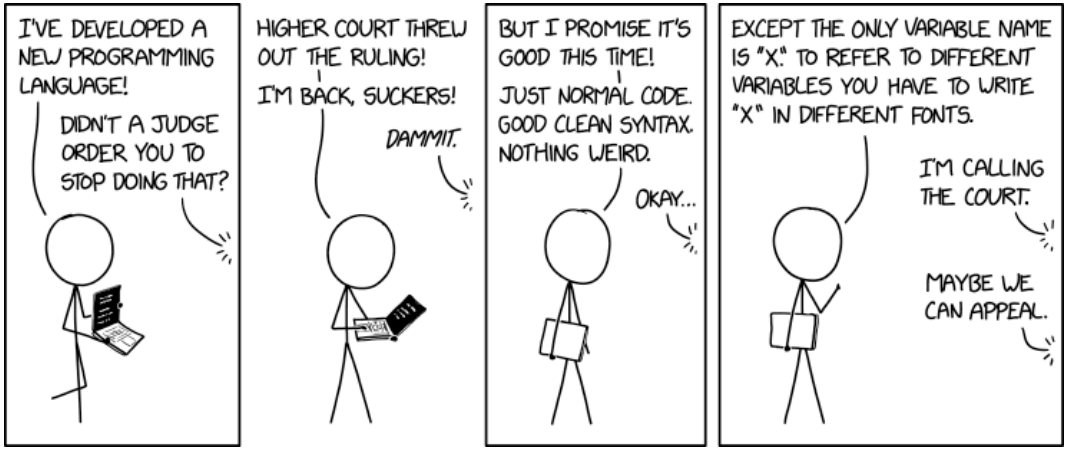
Aggregation [adapted from UML User Guide]

Aggregation is a special association in which one class represents a larger thing (the whole), which consists of smaller things (the parts) (i.e., has-a relationship). In contrast, a plain association between two classes represents a structural relationship between peers, meaning that both classes are conceptually at the same level.

Composition [UML User Guide]

"**Composition** is a form of aggregation, with strong ownership and coincident lifetime as part of the whole. Parts with non-fixed multiplicity may be created after the composite itself, but once created they live and die with it. Such parts can also be explicitly removed before the death of the composite."




I'VE DEVELOPED A NEW PROGRAMMING LANGUAGE!
DIDN'T A JUDGE ORDER YOU TO STOP DOING THAT?
HIGHER COURT THREW OUT THE RULING!
I'M BACK, SUCKERS!
DAMMIT.
BUT I PROMISE IT'S GOOD THIS TIME!
JUST NORMAL CODE. GOOD CLEAN SYNTAX. NOTHING WEIRD.
OKAY...
EXCEPT THE ONLY VARIABLE NAME IS "X." TO REFER TO DIFFERENT VARIABLES YOU HAVE TO WRITE "X" IN DIFFERENT FONTS.
I'M CALLING THE COURT.
MAYBE WE CAN APPEAL.

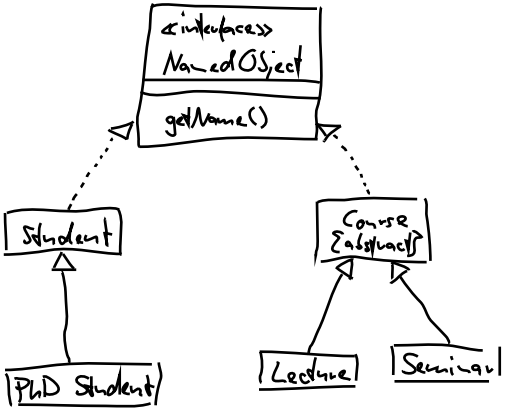
Inheritance Relationships

Generalization and Abstract Classes

"A **generalization** is a relationship between a general kind of thing (called the superclass or parent) and a more specific kind of thing (called the subclass or child). Generalization is sometimes called an is-a-kind-of relationship: one thing is-a-kind-of a more general thing." If a superclass cannot be instantiated it is an **abstract class** (also denoted with *italic name*). (Generalisierung, abstrakte Klasse) [UML User Guide]

Interfaces and Realization

An **interface** is a special class that only has static non-private attributes, abstract non-private operations, cannot be instantiated, and not be in a generalization relationship. Classes may **realize** an interface. (Schnittstelle, Realisierung)



Rules and Hints for Class Diagrams

Rules for Class Diagrams

- Class and attribute names are short nouns or noun phrases
- Operation names are verbs or short phrases starting with verbs
- Use camel case, whereas the first letter of attributes and operations is not capitalized
- A class has an arbitrary number of attributes/operations, any subset thereof shown in a diagram
- No cycles in inheritance relationships
- Abstract operations only live in abstract classes
- Interfaces cannot have private members (attributes/operations)
- Each class can only be a **part** of one composition

Hints on Inheritance [UML User Guide]

"An object of the child class may be used for a variable or parameter typed by the parent, but not the reverse. In other words, generalization means that the child is substitutable for a declaration of the parent. A child inherits the properties of its parents, especially their attributes and operations. Often—but not always—the child has attributes and operations in addition to those found in its parents. An implementation of an operation in a child overrides an implementation of the same operation of the parent; this is known as polymorphism. To be the same, two operations must have the same signature (same name and parameters)."

Modeling Structure with Class Diagrams

Lessons Learned

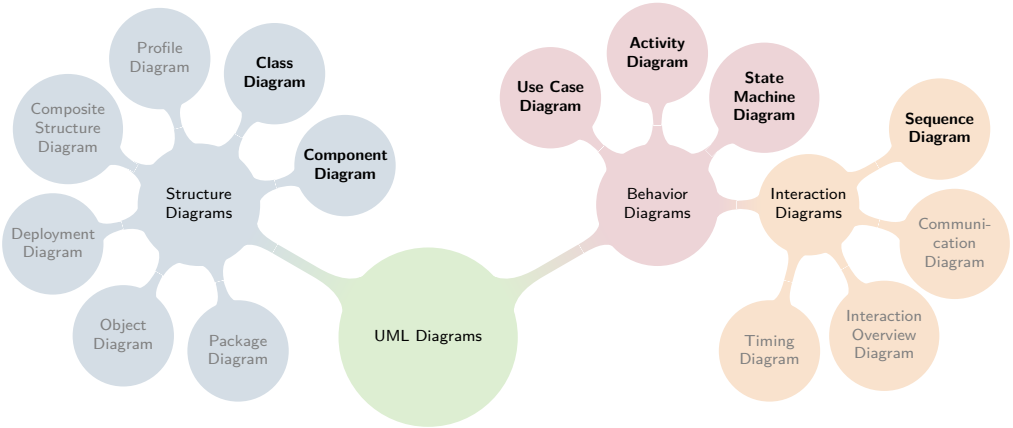
- What are class diagrams?
- Notation and semantics of association, role, multiplicity, (static) attribute, (static/abstract) operation, visibility modifiers, aggregation, composition, generalization, abstract class, interface, realization
- Further Reading: UML User Guide, Chapter 4–5 and 9–10

Practice

- See Moodle
- Sketch a class diagram with 3–5 classes for a software of your choice. However, add at least one error violating the rules of class diagrams (design errors do not count and are ignored for this exercise).
- Upload your diagram to Moodle and find one error in one other diagram (graphical corrections with red color).

Lecture Contents

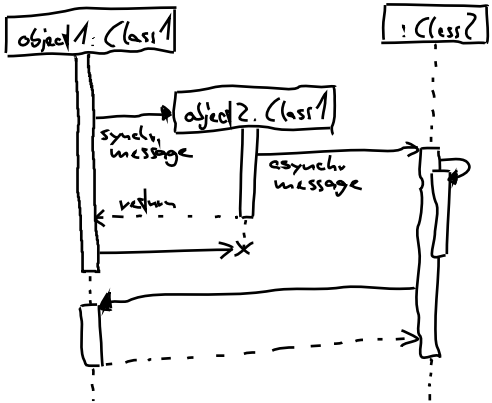
Recap: 14 Types of UML Diagrams [UML 2.5.1]



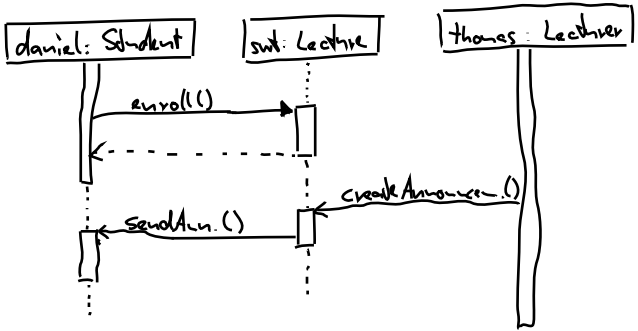
Sequence Diagrams (Sequenzdiagramme)

Sequence Diagram [UML Reference Manual]

"A sequence diagram displays an interaction as a two-dimensional chart. The **vertical dimension** is the time axis; time proceeds down the page. The **horizontal dimension** shows the roles that represent individual objects in the collaboration. Each role is represented by a vertical column containing a head symbol and a vertical line – a **lifeline**. During the time an object exists, it is shown by a dashed line. During the time an execution specification of a procedure on the **object is active**, the lifeline is drawn as a double line. [...] A **message** is shown as an arrow from the lifeline of one object to that of another. The arrows are arranged in time sequence down the diagram. An **asynchronous message** is shown with a stick arrowhead." (Lebenslinie, Aktivierungsbalken, (a)synchrone Nachricht)



Example of a Sequence Diagram

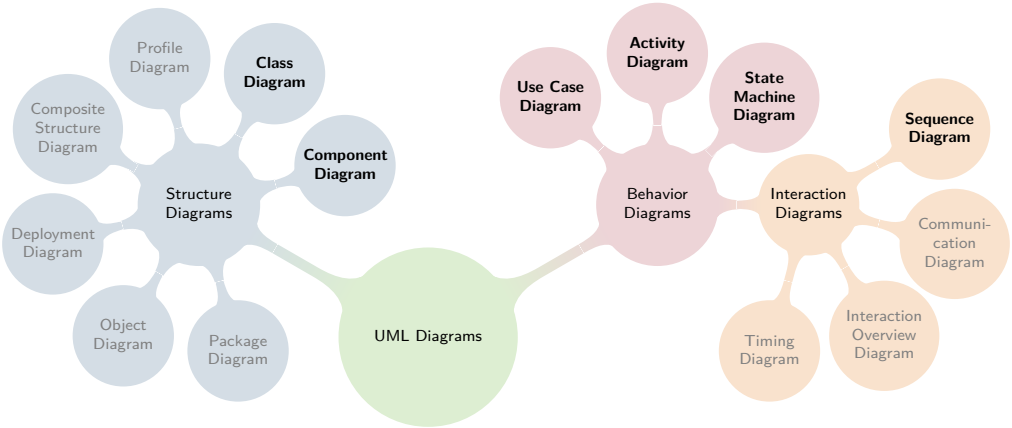


Rules for Sequence Diagrams

Rules for Sequence Diagrams

- Activity of objects begins at the role or with a message
- Only active objects can send messages
- Every synchronous message has its own return (in this lecture, optional in UML)
- Every return has its own synchronous message
- Activities can be stacked to arbitrary, but finite depth
- A destroyed object is dead forever
- Ordering of messages for each single lifetime matters
- Distance of messages in the diagram does not imply timing constraints

Recap: 14 Types of UML Diagrams [UML 2.5.1]



Modeling Interactions with Sequence Diagrams

Lessons Learned

- What are sequence diagrams?
- Notation and semantics of roles, lifelines, (stacked) activity, (a)synchronous message, return
- Further Reading: UML Reference Manual Chapter 9 and UML User Guide Chapter 19

Practice

- See Moodle
- Sketch a sequence diagram with 2–4 roles for a messenger app.
- Upload your diagram to Moodle and correct other solutions if you find errors.