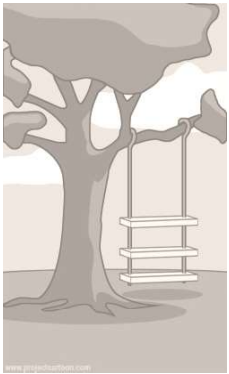
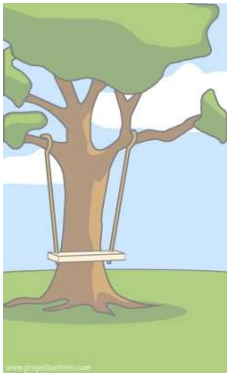
Software Engineering

3. System Modeling | Thomas Thüm | November 3, 2021

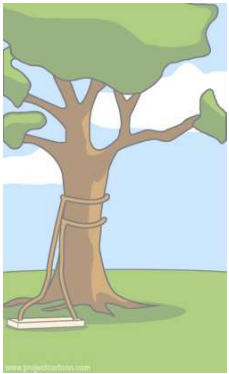
Why System Modeling?



how the customer explained it

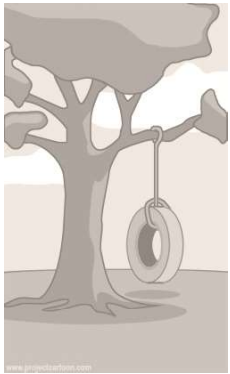


how the project leader understood it



how the programmer implemented it

how the business consultant described it



what the customer really needed

Lecture Overview

1. Why to Model Systems?
2. Modeling Behavior with Activity Diagrams
3. Modeling Behavior with State Machine Diagrams

Lecture Contents

Motivation for Modeling

UML User Guide:

"A successful software organization is one that consistently deploys **quality software** that meets the needs of its users. An organization that can develop such software in a **timely and predictable** fashion, with an **efficient and effective use of resources**, both human and material, is one that has a sustainable business.
[...]
Modeling is a central part of all the activities that lead up to the deployment of good software. We build models to **communicate** the desired structure and behavior of our system. We build models to **visualize and control** the system's architecture. We build models to better **understand** the system we are building, often exposing opportunities for **simplification and reuse**. And we build models to **manage risk**."

UML User Guide:

"We build models of complex systems because we cannot comprehend such a system in its entirety."

Recap: Software Engineering vs Programming

Bjarne Stroustrup (2000):

"The most important single aspect of software development is to be clear about what you are trying to build."

What is System Modeling?

System Modeling [Sommerville]

"**System modeling** is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. [...] Models are used during the requirements engineering process to help derive the **detailed requirements** for a system, during the design process to **describe the system to engineers** implementing the system, and after implementation to **document the system**'s structure and operation."

What is a Model?

UML User Guide:

"A **model** is a simplification of reality."

Sommerville:

"A **model** is an abstract view of a system that deliberately ignores some system details."

Goals of Models [UML User Guide]

- visualize a system as it is (wanted)
- specify the structure or behavior of a system
- template to guide construction of a system
- document the decisions we have made

Sommerville:

"It is important to understand that a system model is **not a complete representation** of system. It purposely leaves out detail to make it **easier to understand**. A model is an abstraction of the system being studied rather than an alternative representation of that system. A representation of a system should maintain all the information about the entity being represented. An abstraction **deliberately simplifies a system** design and picks out the most salient characteristics."

What Language to Use for Modeling?

Towards a Common Language

- Natural language? hard to abstract from details, already used in requirements
- Programming language? unfamiliar to people without programming skills in that language, too early to decide for the programming language
- Textual language? harder to understand
- Graphical language? makes use of our visual abilities, requires common understanding
- Problem: engineers need to be aware of all languages being used
- Solution: use a graphical language independent of company and domain


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION:
THERE ARE
14 COMPETING
STANDARDS.
14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.
YEAH!
SOON:
SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

The Unified Modeling Language

UML [UML Reference Manual]

"The Unified Modeling Language (UML) is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system."

UML User Guide:

"Modeling yields an understanding of a system. No one model is ever sufficient. Rather, you often need multiple models that are connected to one another [...]."

Different Kinds of UML Diagrams

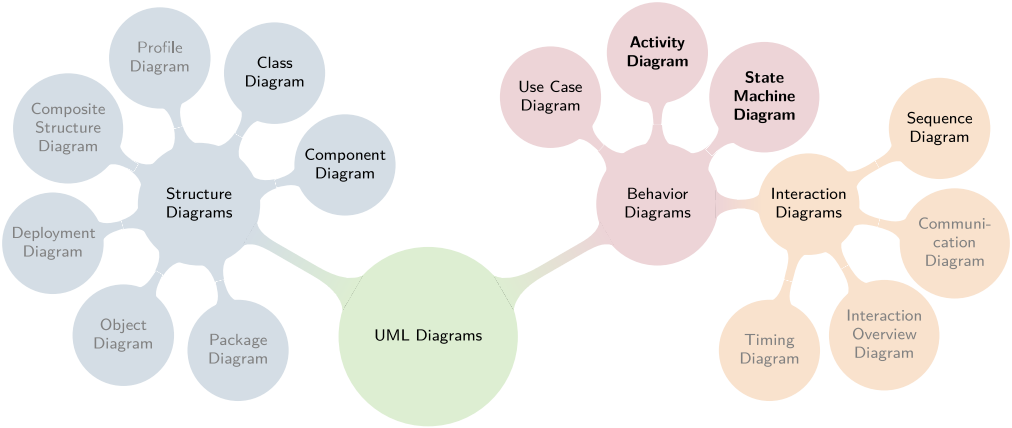
Structure Diagrams (Strukturdiagramme)

"**Structure diagrams** show the static structure of the objects in a system. That is, they depict those elements in a specification that are irrespective of time. The elements in a structure diagram represent the meaningful concepts of an application, and may include abstract, real-world and implementation concepts." [UML 2.5.1]

Behavior Diagrams (Verhaltensdiagramme)

"**Behavior diagrams** show the dynamic behavior of the objects in a system, including their methods, collaborations, activities, and state histories. The dynamic behavior of a system can be described as a series of changes to the system over time." [UML 2.5.1]

14 Types of UML Diagrams [UML 2.5.1]



Six most important UML diagrams* discussed in this course

*John Erickson and Keng Siau. 2007. Theoretical and practical complexity of modeling methods. Commun. ACM 50, 8 (August 2007), 46–51.

Why to Model Systems?

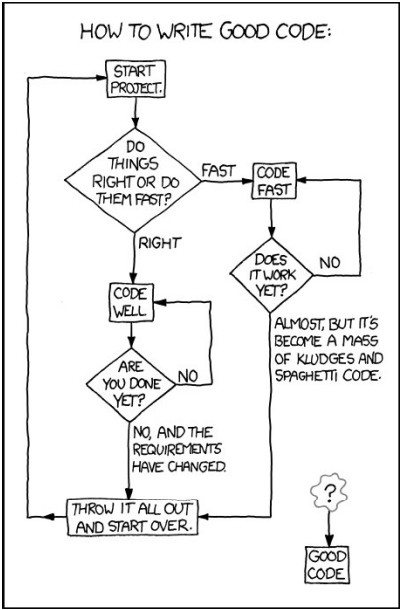
Lessons Learned

- What is the motivation for system modeling?
- What are models and what is UML?
- Which (kinds of) UML diagrams exist?
- Further Reading: UML User Guide, Chapter 1 — great introduction to modeling

Practice

- See Moodle
- Practice abstraction by explaining a room that you know well to a good friend. Use 1 minute (not more!) to create sketch of the room and upload that to Moodle.
- Look at another sketch submitted and create an answer in Moodle with a list of ten things that the sketch abstracts from (i.e., what details are not shown in the visualization).

Lecture Contents


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
RIGHT
DOES IT WORK YET?
NO
CODE WELL.
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE.

Activity Diagrams

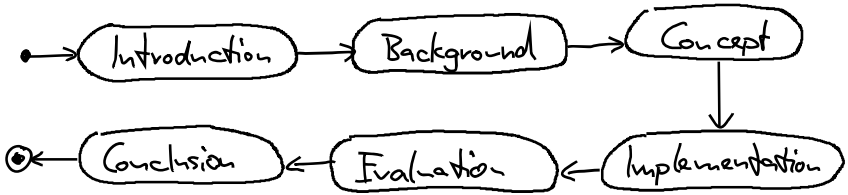
Activity Diagram (Aktivitätsdiagramm)

An **activity diagram** is a diagram visualizing activities and their order of execution. An activity diagram contains **activities** (rounded box) that are connected by means of **flows** (solid arrows). The execution begins at the **initialization** (filled circle) and ends with the **completion** node (bull's eye). (Aktivität, Fluss, Startzustand, Endzustand)

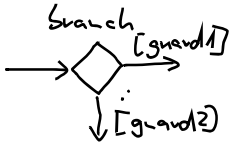
Rules for Activity Diagrams

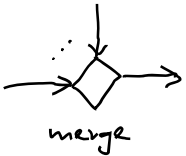
- exactly one initialization/completion node
- at least one activity
- every activity has one incoming and one outgoing flow
- every activity is reachable from initialization
- completion is reachable from every activity

Example of Sequential Activities



Branching and Merging in Activity Diagrams





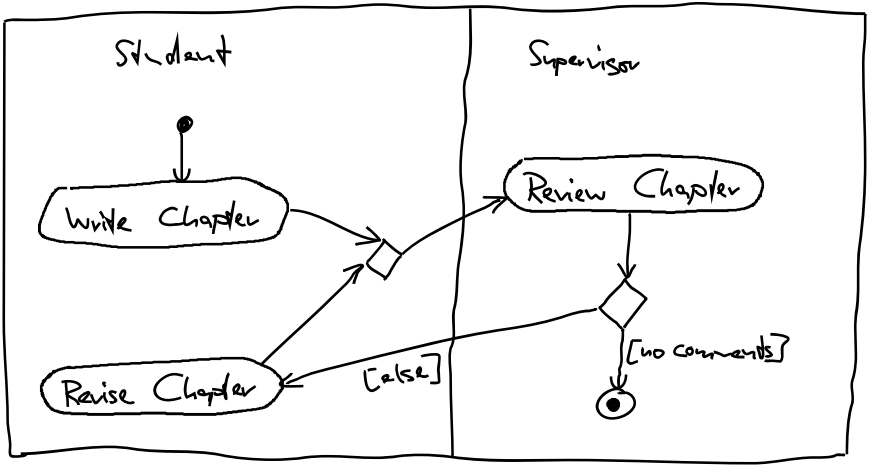
Branching and Merging [UML User Guide]

Motivation: model control flow that depends on certain conditions (i.e., actions that may happen)
Branching: A **branch** has exactly one incoming and two or more outgoing flows. Each outgoing flow has a Boolean expression called **guard**, which is evaluated on entering the branch. (Verzweigung)
Merging: A **merge** has two or more incoming and exactly one outgoing flow. (Zusammenführung)

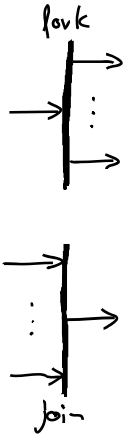
Further Rules for Activity Diagrams

- guards on outgoing flows should not overlap (flow of control is unambiguous)
- guards should cover all possibilities (flow of control does not freeze)
- keyword **else** possible for one guard (sonst)

Example of Conditional Activities



Forking and Joining in Activity Diagrams



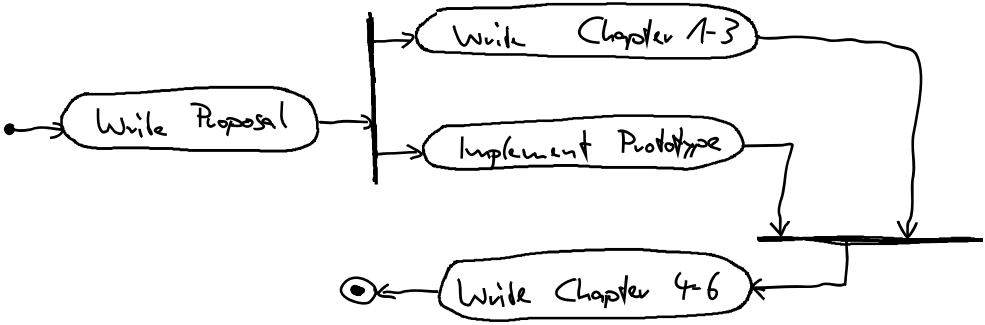
Forking and Joining [UML User Guide]

Motivation: model concurrent control flows (i.e., activities that run in parallel)
Forking: A **fork** (thick horizontal or vertical line) has exactly one incoming and two or more outgoing flows. (Gabelung)
Joining: A **join** (thick horizontal or vertical line) has two or more incoming and exactly one outgoing flow. (Vereinigung)

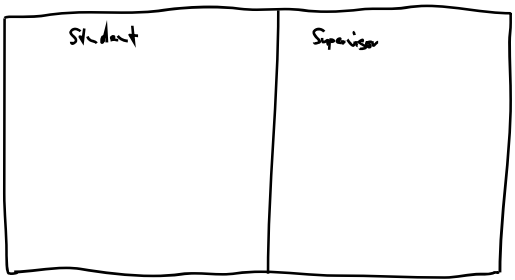
Further Rules for Activity Diagrams

- branched paths must be merged eventually (letztendlich)
- forked paths must be joined eventually
- only outgoing edges of branch nodes have guards

Example of Concurrent Activities



Swimlanes in Activity Diagrams



Swimlanes [UML User Guide]

Motivation: group activities according to responsibilities
Swimlane: An activity diagram may have no or at least two swimlanes. A **swimlane** (rectangle) represents a high-level responsibility activities within an activity diagram.
(Verantwortlichkeitsbereiche)

Further Rules for Activity Diagrams

- each swimlane has a name unique within its diagram
- every activity belongs to exactly one swimlane
- only flows may cross swimlanes

Modeling Behavior with Activity Diagrams

Lessons Learned

- What can be modeled with activity diagrams?
- What are branching and merging (used for)?
- What are forking and joining (used for)?
- What can be modeled with swimlanes?
- Further Reading: UML User Guide, Chapter 20

Practice

- See Moodle
- Draw a simple activity diagram in the context of a contract tracing app and submit it in Moodle
- Inspect one other diagram and check whether any rules are violated. Document those as an answer.

Lecture Contents

1. Why to Model Systems?
2. Modeling Behavior with Activity Diagrams
3. Modeling Behavior with State Machine Diagrams
   - Activity and State Machine Diagrams
   - State Machine Diagrams
   - Hierarchical State Machine Diagrams
   - Lessons Learned

Activity and State Machine Diagrams

UML User Guide:

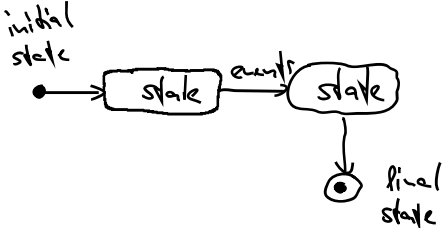
We can visualize the dynamics of execution in two ways: by emphasizing the flow of control from activity to activity (**activity diagrams**) or by emphasizing the potential states and transitions among those states (**state machine diagrams**).

State Machine Diagrams

State Machine Diagram (Zustandsdiagramm)

A **state machine diagram** specifies the sequences of states the (a part of) the system goes through during its lifetime in response to events, together with its responses to those events. Every **state** (rectangle with rounded corners) is characterized by a condition or situation. An **event** is an occurrence of a stimulus that can trigger a state transition. A **transition** (solid arrow) is a relationship between two states. (Zustand, Ereignis, Zustandsübergang)

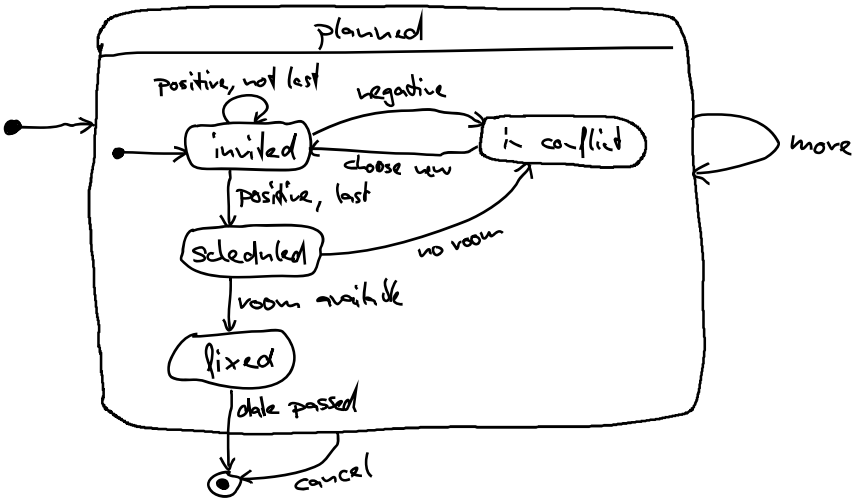
[adapted from UML User Guide]



Rules for State Machine Diagrams

there is a single **initial state** (filled circle) and a single **final state** (bull's eye) (Start- und Zielzustand) — see exception below

Example of a State Machine Diagram



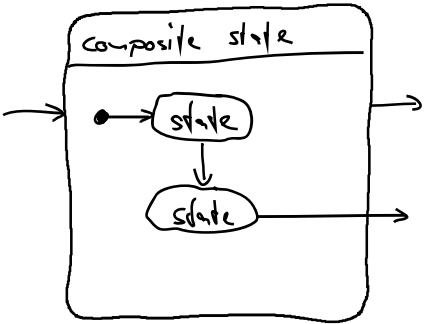
Hierarchical State Machine Diagrams

Simple and Composite States [UML User Guide]

Motivation: avoid duplicated transitions, improve overview in complex state machine diagrams
Simple State: "A **simple state** is a state that has no substructure." (einfacher Zustand)
Composite State: "A state that has substates (i.e., nested states) is called a **composite state**." (komplexer Zustand)

Rules for State Machine Diagrams

- every composite state has its own single **initial state** (Startzustand)
- substates may be nested to any level



Modeling Behavior with State Machine Diagrams

Lessons Learned

- What can be modeled with state machine diagrams?
- What is the advantage of hierarchical state machines?
- Further Reading: UML User Guide, Chapter 22

Practice

- See Moodle
- Draw a simple state machine diagram with a single mistake and submit it in Moodle.
- Find the mistake in one other diagram, correct it with red ink, and submit in to Moodle.