Software Design, Modelling and Analysis in UML

Lecture 1: Introduction

2016-10-18

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Content

An Analogy: Construction Engineering

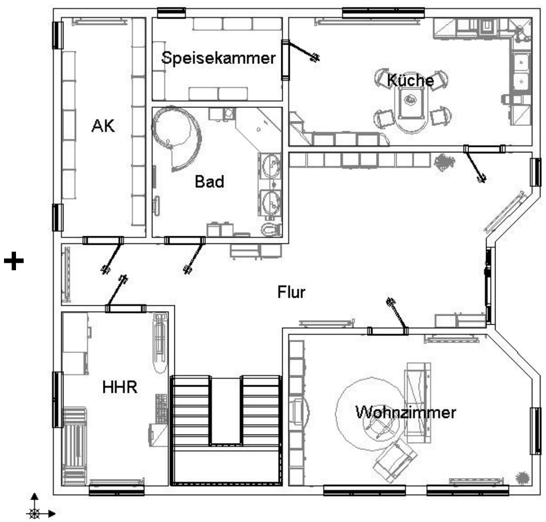
Requirements:

- …
- The bathroom must not have a window.
- …

Specification:

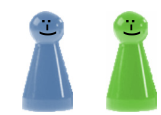
- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …

+



http://wikimedia.org (CC nc-sa 3.0, Ottoklages)

http://wikimedia.org (CC nc-sa 3.0, Bobthebuilder82)



An Analogy: Construction Engineering

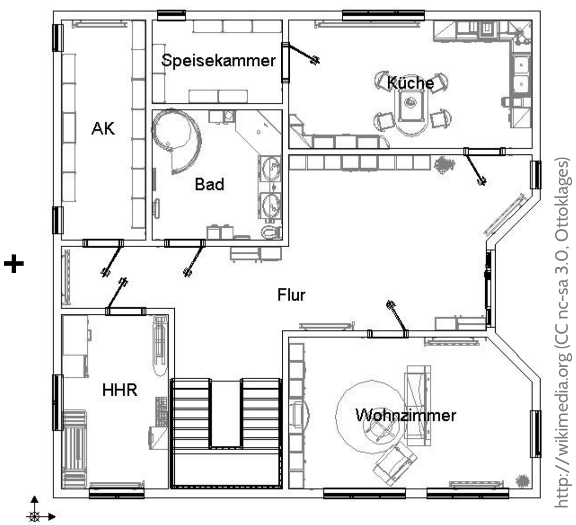
Requirements:

- …
- The bathroom must not have a window.
- …

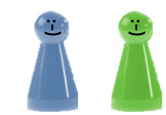
Specification:

- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …

\+



http://wikimedia.org (CC nc-sa 3.0, Bobthebuilder82)



A (semi-)formal **design description** and **specification language** – every construction engineer has pretty much the same understanding of it. (The customer need not understand it: a construction engineer can "translate".)

An Analogy: Construction Engineering

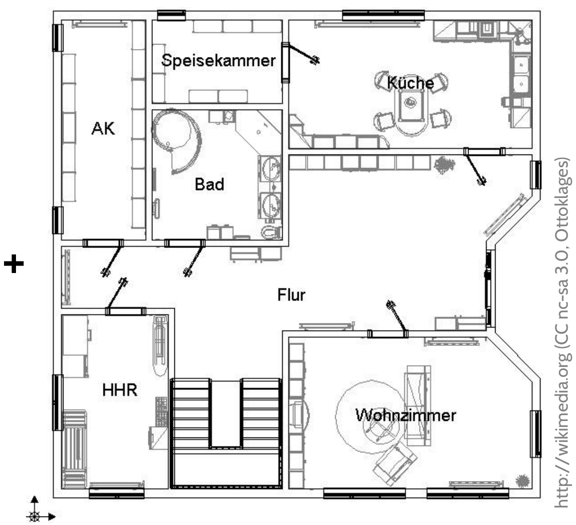
Requirements:

- …
- The bathroom must not have a window.
- …

Specification:

- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …

+



Check whether design satisfies specification – before building the house.

A (semi-)formal **design description** and **specification language** – every construction engineer has pretty much the same understanding of it. (The customer need not understand it: a construction engineer can "translate".)

An Analogy: Construction Engineering

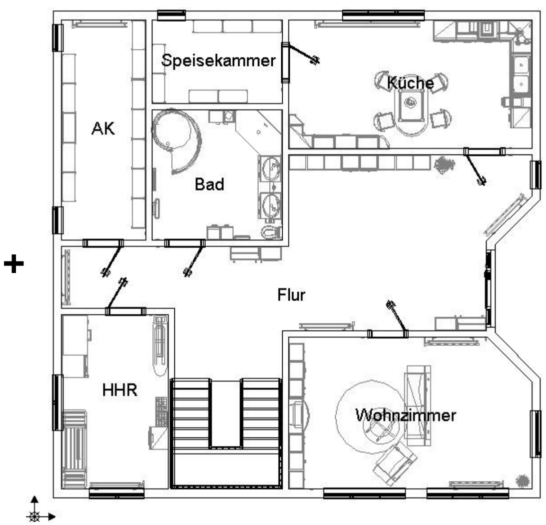
Requirements:

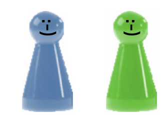
- …
- The bathroom must not have a window.
- …

Specification:

- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …

+





Check whether design satisfies specification – before building the house.

Build house according to the plan.

A (semi-)formal **design description** and **specification language** – every construction engineer has pretty much the same understanding of it. (The customer need not understand it: a construction engineer can "translate".)

Recall: Model

Definition. [Folk] A model is an abstract, formal, mathematical representation or description of structure or behaviour of a (software) system.

Recall: Model

Definition. [Folk] A model is an abstract, formal, mathematical representation or description of structure or behaviour of a (software) system.

Definition. (Glinz, 2008, 425)
A model is a concrete or mental image (Abbild) of something or a concrete or mental archetype (Vorbild) for something.

Three properties are constituent:

(i) the image attribute (Abbildungsmerkmal), i.e. there is an entity (called original) whose image or archetype the model is,

(ii) the reduction attribute (Verkürzungsmerkmal), i.e. only those attributes of the original that are relevant in the modelling context are represented,

(iii) the pragmatic attribute, i.e. the model is built in a specific context for a specific purpose.

Floorplans as Models

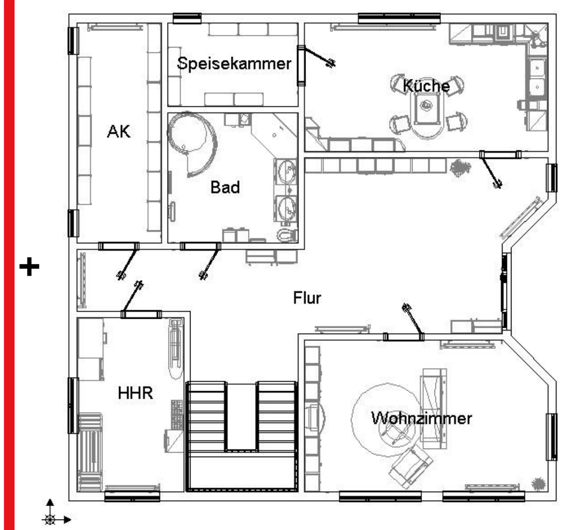
Customer Developer

Requirements:

- …
- The bathroom must not have a window.
- …

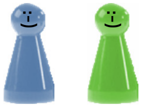
Specification:

- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …



http://wikimedia.org (CC nc-sa 3.0, Ottoklages)

http://wikimedia.org (CC nc-sa 3.0, Bobthebuilder82)



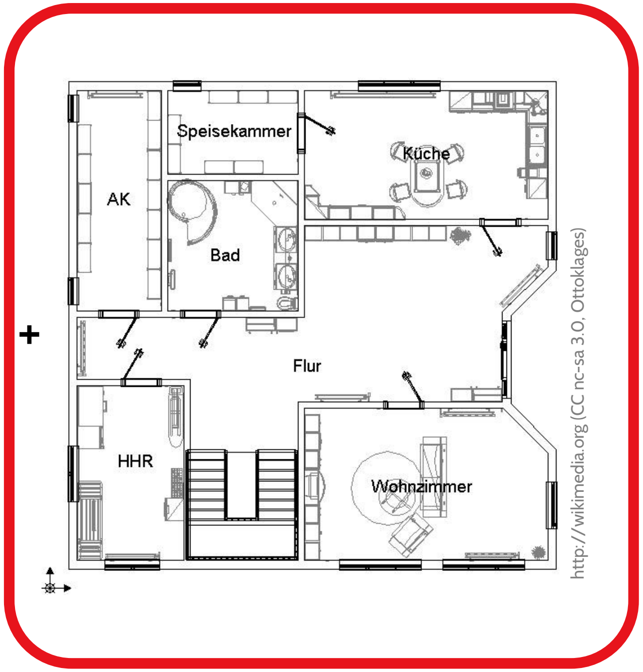
Floorplans as Models

Requirements:

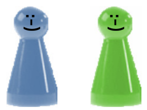
- …
- The bathroom must not have a window.
- …

Specification:

- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …



http://wikimedia.org (CC nc-sa 3.0, Bobthebuilder82)



Floorplan **abstracts** from properties,

Floorplan **preserves** properties,

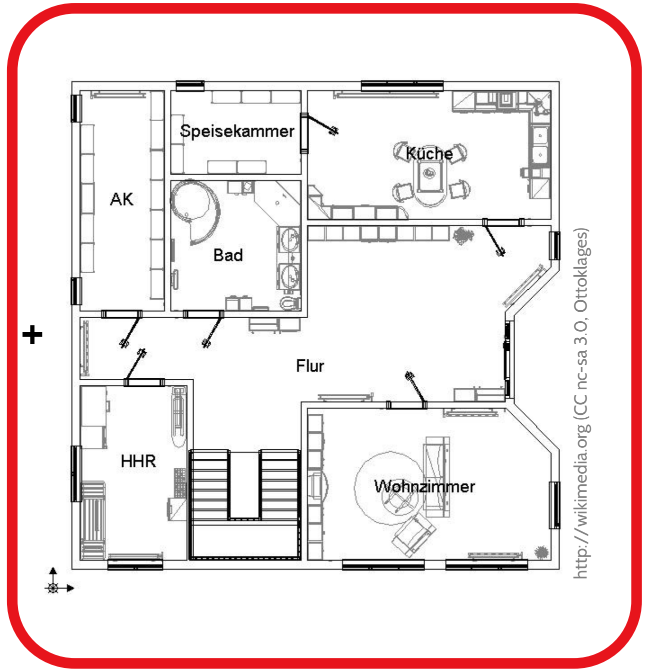
Floorplans as Models

Requirements:

- …
- The bathroom must not have a window.
- …

Specification:

- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …



http://wikimedia.org (CC nc-sa 3.0, Bobthebuilder82)

Floorplan **abstracts** from properties, e.g.,

- kind, number, and placement of bricks,
- subsystem details (e.g., window style),
- water pipes/wiring,
- wall decoration

Floorplan **preserves** properties, e.g.,

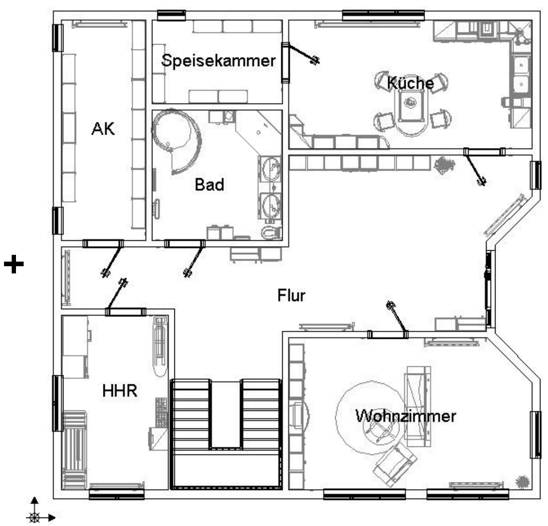
Floorplans as Models

Requirements:

- …
- The bathroom must not have a window.
- …

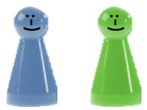
Specification:

- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …



http://wikimedia.org (CC nc-sa 3.0, Ottoklages)

http://wikimedia.org (CC nc-sa 3.0, Bobthebuilder82)



Floorplan **abstracts** from properties, e.g.,

- kind, number, and placement of bricks,
- subsystem details (e.g., window style),
- water pipes/wiring,
- wall decoration

Floorplan **preserves** properties, e.g.,

- house and room extensions (to scale),
- presence/absence of windows and doors,
- placement of subsystems (like windows),
- etc.

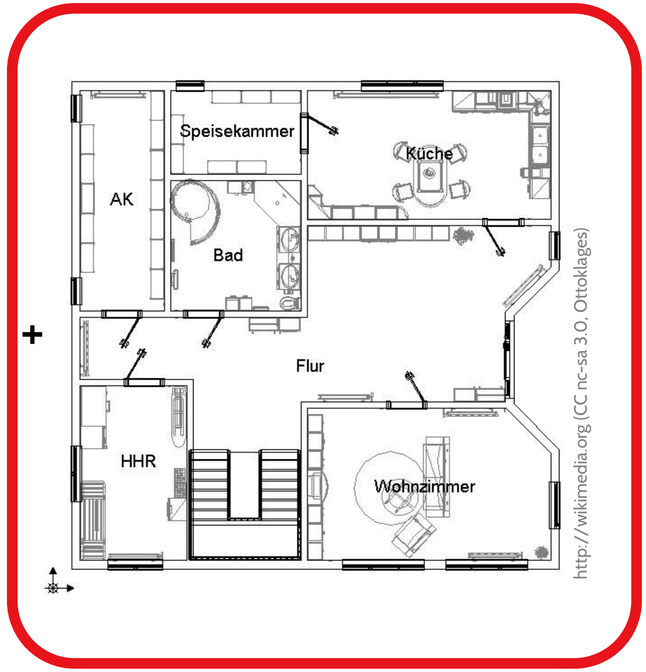
Floorplans as Models

Requirements:

- …
- The bathroom must not have a window.
- …

Specification:

- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …



Floorplan **abstracts** from properties, e.g.,

- kind, number, and placement of bricks,
- subsystem details (e.g., window style),
- water pipes/wiring,
- wall decoration

Floorplan **preserves** properties, e.g.,

- house and room extensions (to scale),
- presence/absence of windows and doors,
- placement of subsystems (like windows),
- etc.

→ construction engineers can **efficiently** work on an **appropriate** level of abstraction, and find design errors **before building** the system (e.g. regarding bathroom windows).

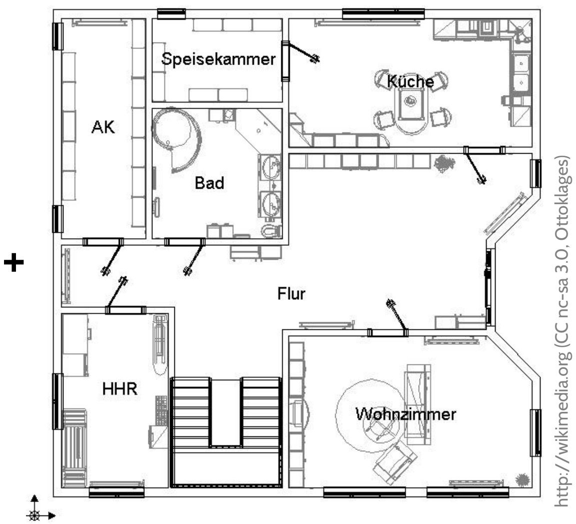
Floorplans as Models

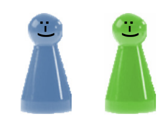
Requirements:

- …
- The bathroom must not have a window.
- …

Specification:

- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …





Build house according to the plan: **pre-image**

Floorplans as Models

Document existing house: **image**.

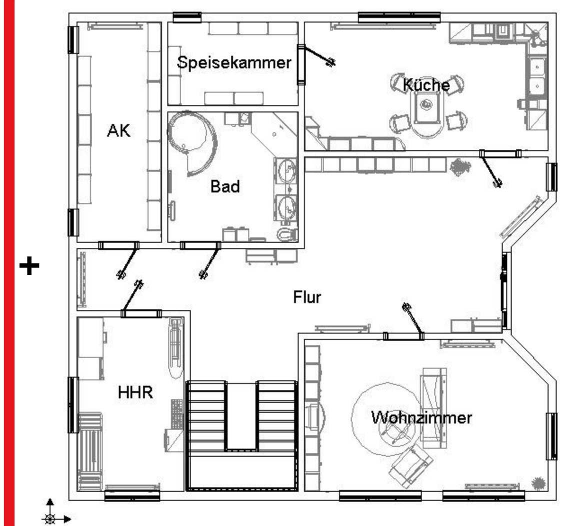
Customer Developer

Requirements:

- …
- The bathroom must not have a window.
- …

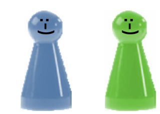
Specification:

- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …



http://wikimedia.org (CC nc-sa 3.0, Ottoklages)

http://wikimedia.org (CC nc-sa 3.0, Bobthebuilder82)



Build house according to the plan: **pre-image**

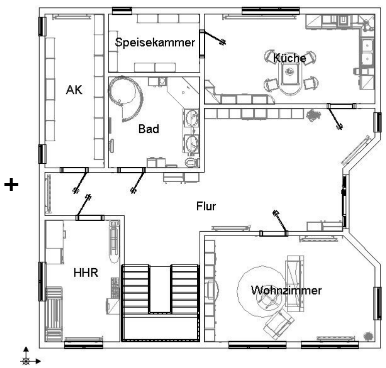
Can We Have the Same for Software?

Construction Engineering:

Requirements:
- ...
- The bathroom must not have a window.
- ...

Specification:
- ...
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- ...

+



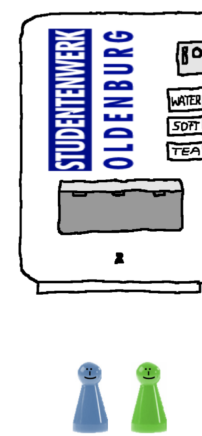
http://wikimedia.org (CC nc-sa 3.0, Ottoklages)

http://wikimedia.org (CC nc-sa 3.0, Bobthebuilder82)

Software Engineering:

?

? + ?



One Proposal: The Unified Modelling Language (UML)

Construction Engineering:

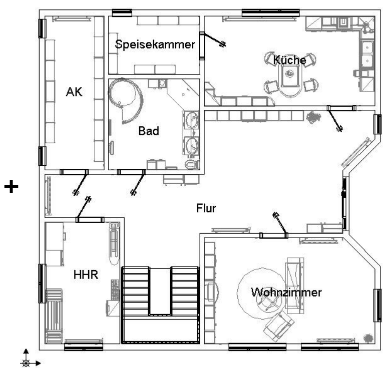
Requirements:

- …
- The bathroom must not have a window.
- …

Specification:

- …
- The outer walls will be built using AAC, the inner walls of sand-lime bricks.
- Steel door frames.
- …

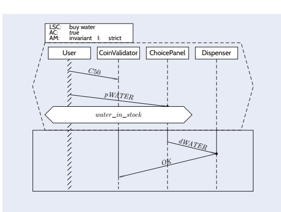
+



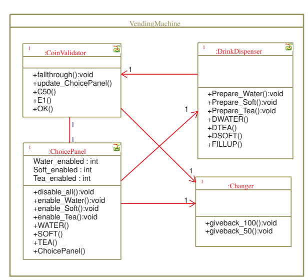
http://wikimedia.org (CC nc-sa 3.0, Ottoklages)

http://wikimedia.org (CC nc-sa 3.0, Bobthebuilder82)

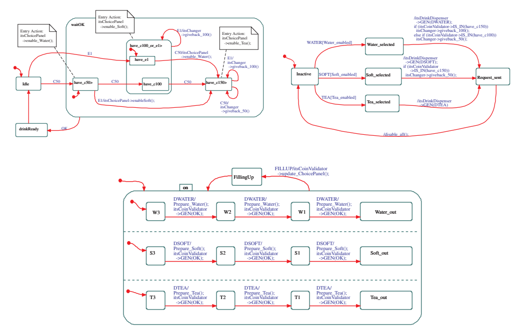
Software Engineering:



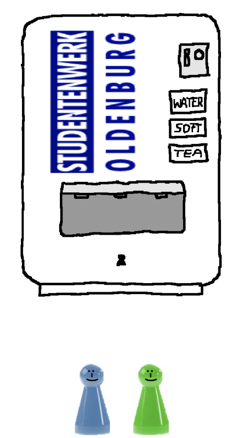
Sequence Diagrams
(behaviour, reflective)



Class Diagrams
(structure)



State Machine Diagrams
(behaviour, constructive)



Goals, Content and Non-Content of the Course

Goal: A Common, Precise Understanding of UML Models

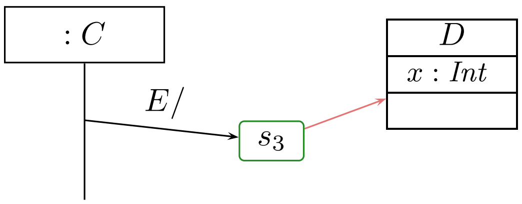
(i) We need to know
how the words of the language **look like**: **Syntax**.

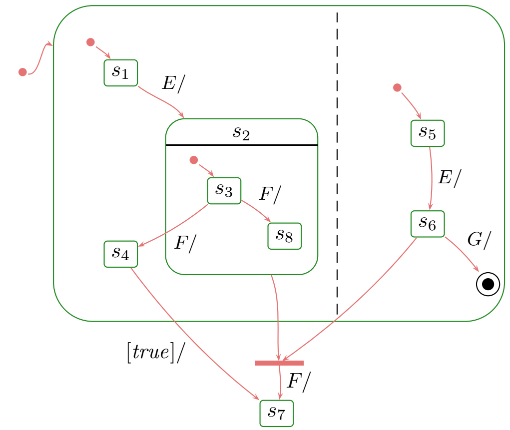
(**UML example**: is this a proper UML state machine diagram?)

Goal: A Common, Precise Understanding of UML Models

(i) We need to know
how the words of the language **look like**: **Syntax**.

(UML example: is this a proper UML state machine diagram?)

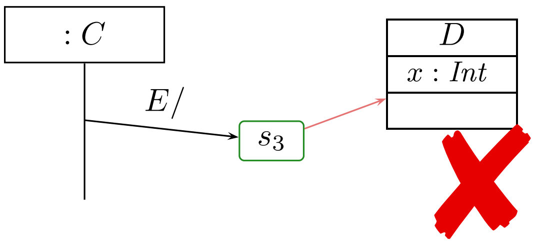


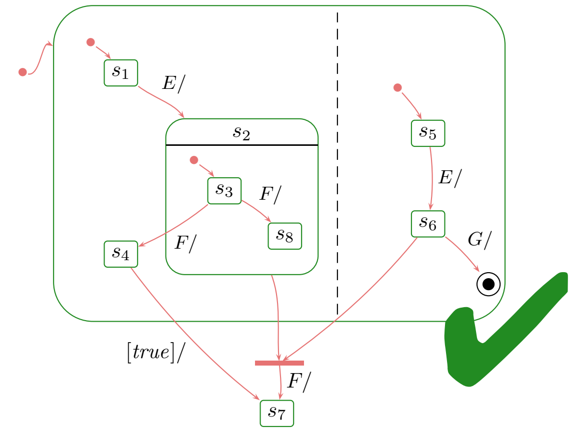


Goal: A Common, Precise Understanding of UML Models

(i) We need to know
how the words of the language **look like**: **Syntax**.

(**UML example**: is this a proper UML state machine diagram?)

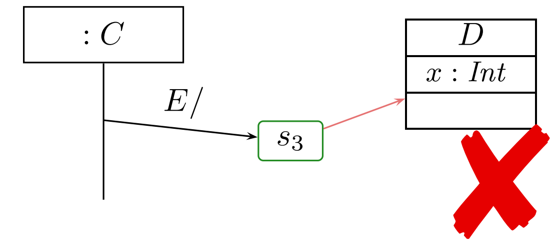




Goal: A Common, Precise Understanding of UML Models

(i) We need to know
how the words of the language **look like**: **Syntax**.

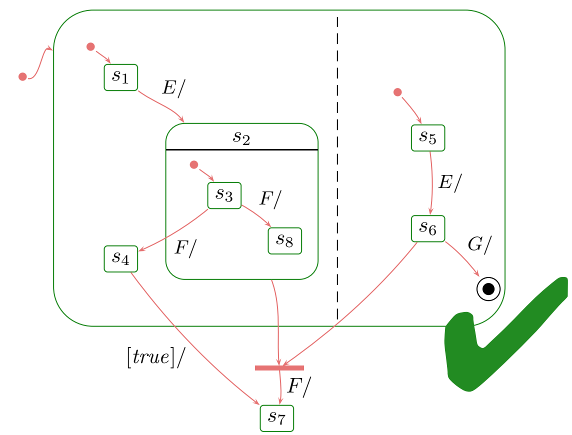
(**UML example**: is this a proper UML state machine diagram?)



(ii) We need to know
what a word of the language **means**: **Semantics**.

$\rightarrow$ Then we can **formally analyse** the model, e.g.,
prove that the design satisfies the requirements,
simulate the model, automatically **generate test cases**,
automatically **generate** equivalent code, etc.

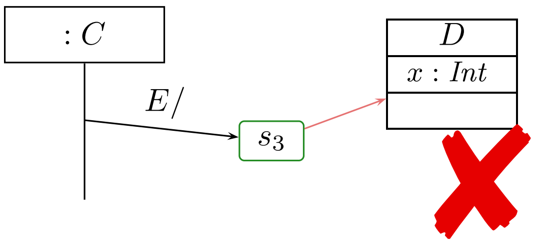
(**UML example**: can sending event E and then G kill the object?)



Goal: A Common, Precise Understanding of UML Models

(i) We need to know
how the words of the language **look like**: **Syntax**.

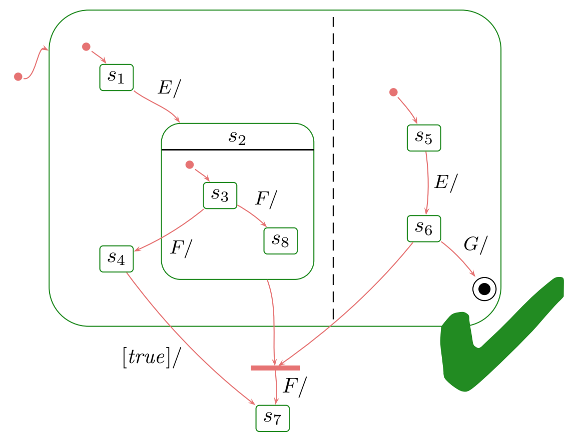
(**UML example**: is this a proper UML state machine diagram?)



(ii) We need to know
what a word of the language **means**: **Semantics**.

$\rightarrow$ Then we can **formally analyse** the model, e.g.,
prove that the design satisfies the requirements,
simulate the model, automatically **generate test cases**,
automatically **generate** equivalent code, etc.

(**UML example**: can sending event E and then G kill the object?)

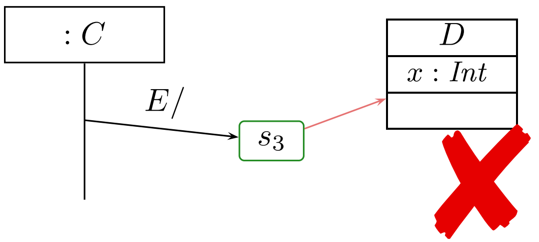


- UML is sometimes (neutrally, or as offence) called **“semi-formal”**

Goal: A Common, Precise Understanding of UML Models

(i) We need to know
how the words of the language **look like**: **Syntax**.

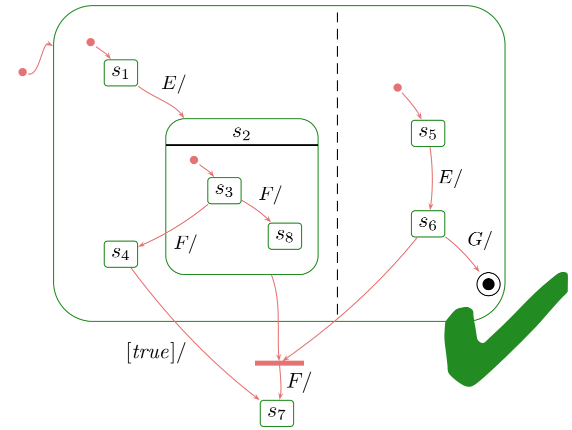
(**UML example**: is this a proper UML state machine diagram?)



(ii) We need to know
what a word of the language **means**: **Semantics**.

→ Then we can **formally analyse** the model, e.g.,
prove that the design satisfies the requirements,
simulate the model, automatically **generate test cases**,
automatically **generate** equivalent code, etc.

(**UML example**: can sending event E and then G kill the object?)

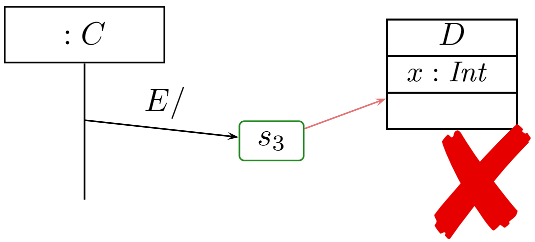


- UML is sometimes (neutrally, or as offence) called **“semi-formal”**:
  the UML standard OMG (2011a,b) is strong on (i), but weak(er) on (ii).
  (“the diagram is self-explanatory”, “everybody understands the diagram” – No.)

Goal: A Common, Precise Understanding of UML Models

(i) We need to know
how the words of the language **look like**: **Syntax**.

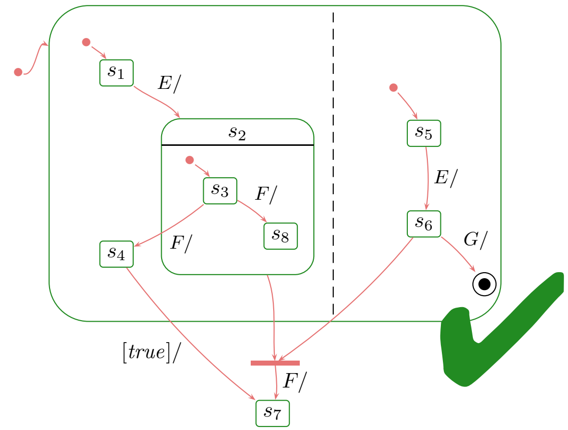
(**UML example**: is this a proper UML state machine diagram?)



(ii) We need to know
what a word of the language **means**: **Semantics**.

→ Then we can **formally analyse** the model, e.g.,
prove that the design satisfies the requirements,
simulate the model, automatically **generate test cases**,
automatically **generate** equivalent code, etc.

(**UML example**: can sending event E and then G kill the object?)



- UML is sometimes (neutrally, or as offence) called **"semi-formal"**:
  the UML standard OMG (2011a,b) is strong on (i), but weak(er) on (ii).
  ("the diagram is self-explanatory", "everybody understands the diagram" – No.)

- **In the lecture**: **study** the (!) **syntax**, **define** one (!) **semantics**.

Date: August 2011

OMG Unified Modeling Language™ (OMG UML), Infrastructure

Version 2.4.1

OMG Document Number: formal/2011-08-05
Standard document URL: http://www.omg.org/spec/UML/2.4.1/Infrastructure
Associated Normative Machine-Readable Files:
http://www.omg.org/spec/UML/20110701/Infrastucture.xmi
http://www.omg.org/spec/UML/20110701/L0.xmi
http://www.omg.org/spec/UML/20110701/LM.xmi
http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi

Version 2.4.1 supersedes formal/2010-05-04.

(230 pages)

Date: August 2011

OMG Unified Modeling Language™ (OMG UML), Superstructure

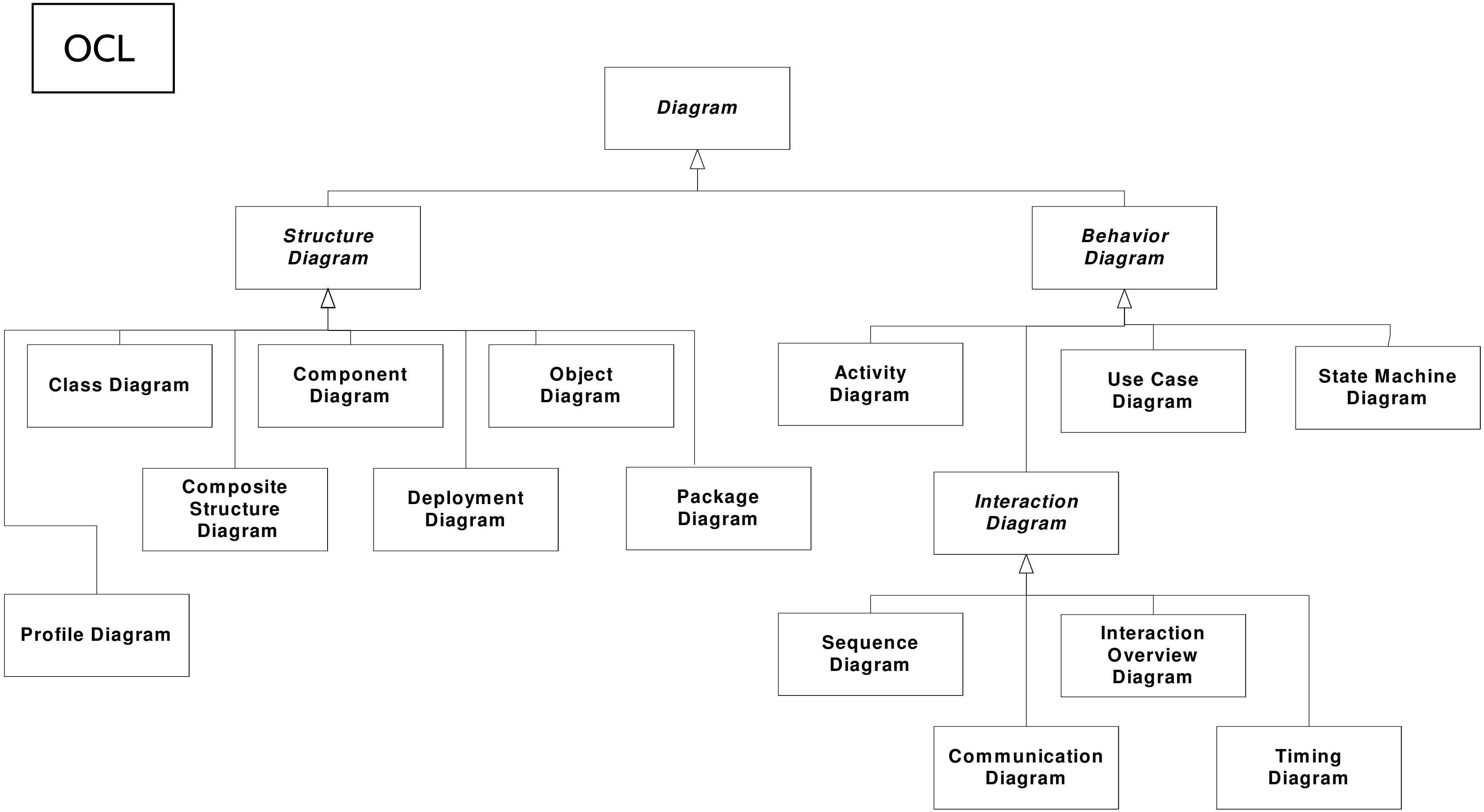
Version 2.4.1

OMG Document Number: formal/2011-08-06
Standard document URL: http://www.omg.org/spec/UML/2.4.1/Superstructure
Associated Normative Machine-Readable Files*:
http://www.omg.org/spec/UML/20110701/Infrastucture.xmi
http://www.omg.org/spec/UML/20110701/Superstructure.xmi
http://www.omg.org/spec/UML/20110701/L0.xmi
http://www.omg.org/spec/UML/20110701/L1.xmi
http://www.omg.org/spec/UML/20110701/L2.xmi
http://www.omg.org/spec/UML/20110701/L3.xmi
http://www.omg.org/spec/UML/20110701/LM.xmi
http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi
http://www.omg.org/spec/UML/20110701/UML.xmi
http://www.omg.org/spec/UML/20110701/StandardProfileL2.xmi
http://www.omg.org/spec/UML/20110701/StandardProfileL3.xmi

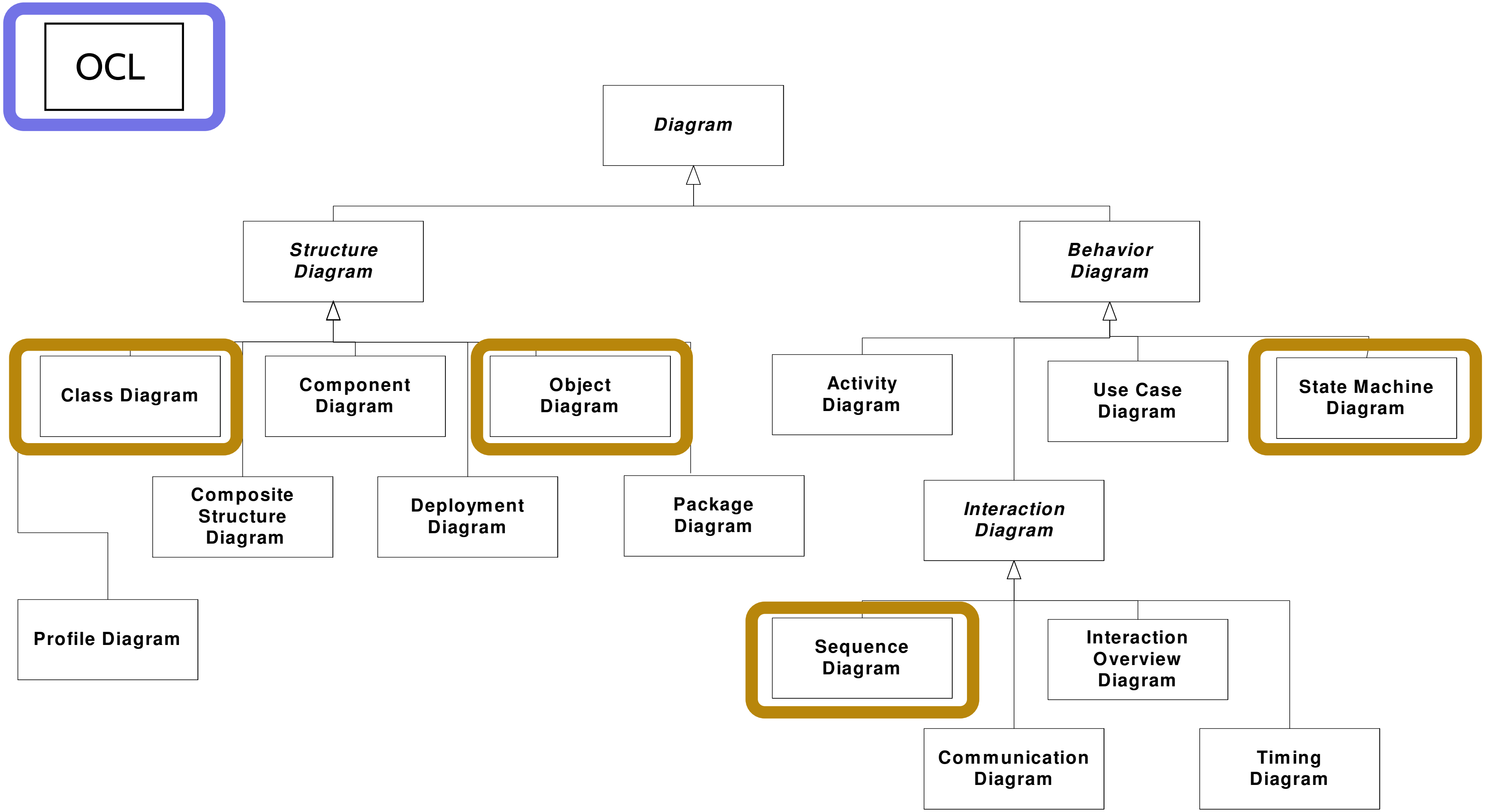
Version 2.4.1 supersedes formal/2010-05-06.

(748 pages)

UML Diagrams (OMG, 2011b, 694)

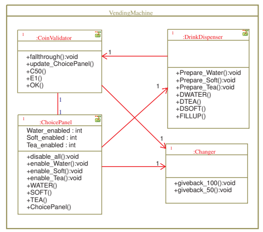


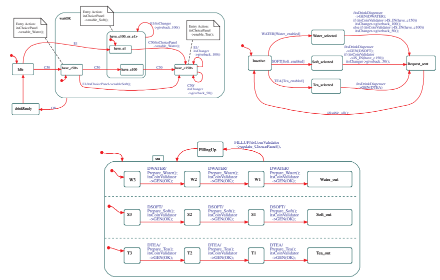
UML Diagrams (OMG, 2011b, 694)

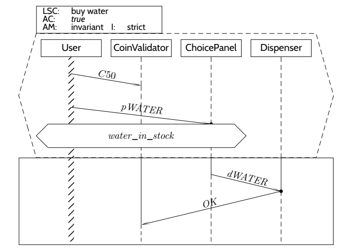


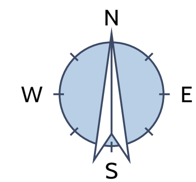
Dobing and Parsons (2006)

Course Map









UML

$\mathcal{CD}, \mathcal{SM}$ $\qquad \varphi \in$ OCL $\qquad \mathcal{CD}, \mathcal{SD}$

Model

$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr), SM$ $\qquad expr$ $\qquad \mathscr{S}, SD$

$M = (\Sigma_{\mathscr{S}}^{\mathscr{D}}, A_{\mathscr{S}}, \to_{SM})$ $\qquad B = (Q_{SD}, q_0, A_{\mathscr{S}}, \to_{SD}, F_{SD})$

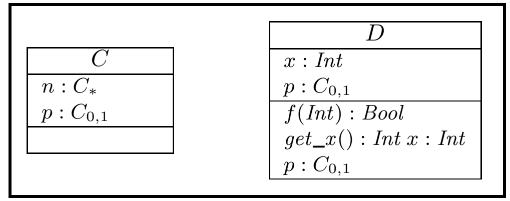
$\pi = (\sigma_0, \varepsilon_0) \xrightarrow[u_0]{(cons_0, Snd_0)} (\sigma_1, \varepsilon_1) \cdots$ $\qquad w_\pi = ((\sigma_i, cons_i, Snd_i))_{i \in \mathbb{N}}$

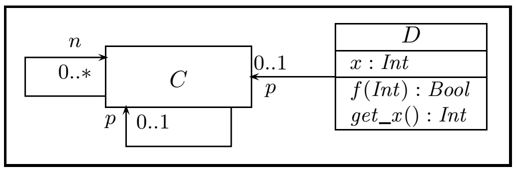
Instances

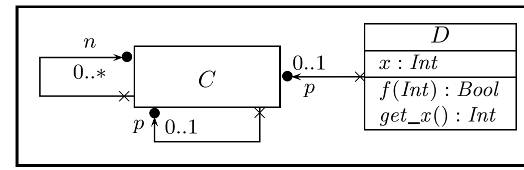
$G = (N, E, f)$ $\qquad$ Mathematics

$\mathcal{OD}$ $\qquad$ UML

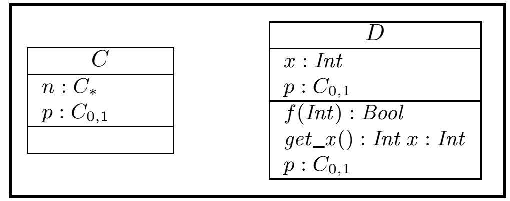
Outlook: Concrete vs. Abstract Syntax

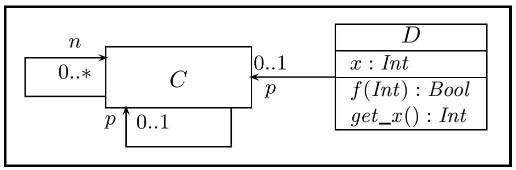




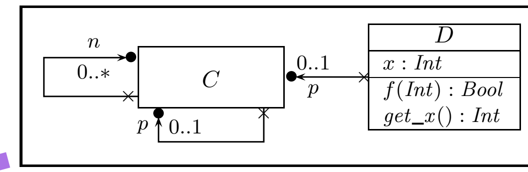


Outlook: Concrete vs. Abstract Syntax

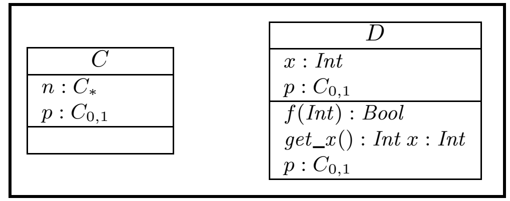


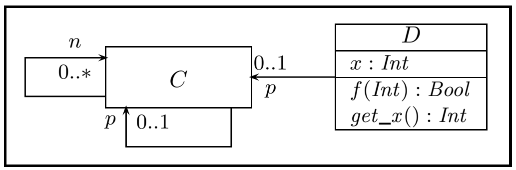


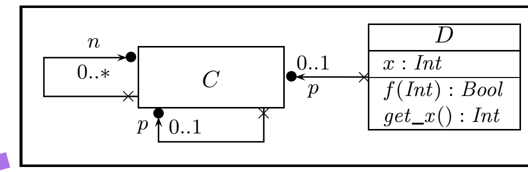
$$
\begin{aligned}
\mathscr{S}_0 = (&\{Int, Bool\},\\
&\{C, D\},\\
&\{x : Int, p : C_{0,1}, n : C_*\},\\
&\{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},\\
&\{f : Int \to Bool, get\_x : Int\},\\
&\{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\})
\end{aligned}
$$



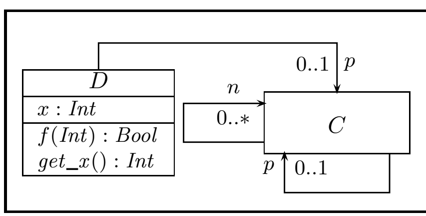
Outlook: Concrete vs. Abstract Syntax



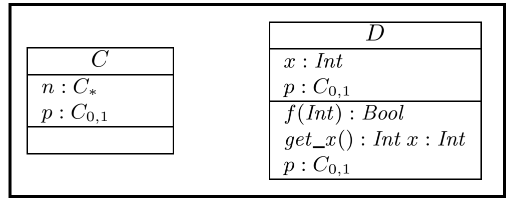


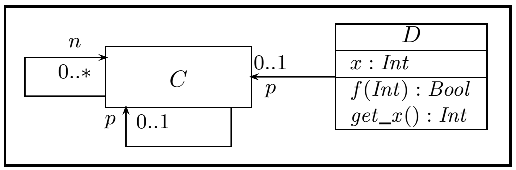


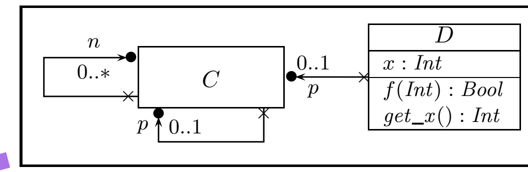
$$\begin{aligned}\mathscr{S}_0 = (&\{Int, Bool\},\\ &\{C, D\},\\ &\{x : Int, p : C_{0,1}, n : C_*\},\\ &\{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},\\ &\{f : Int \to Bool, get\_x : Int\},\\ &\{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\})\end{aligned}$$



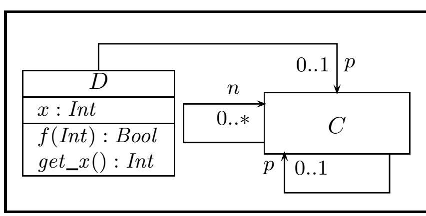
Outlook: Concrete vs. Abstract Syntax

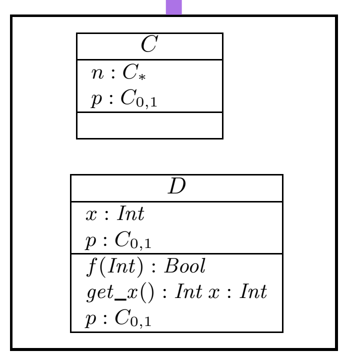




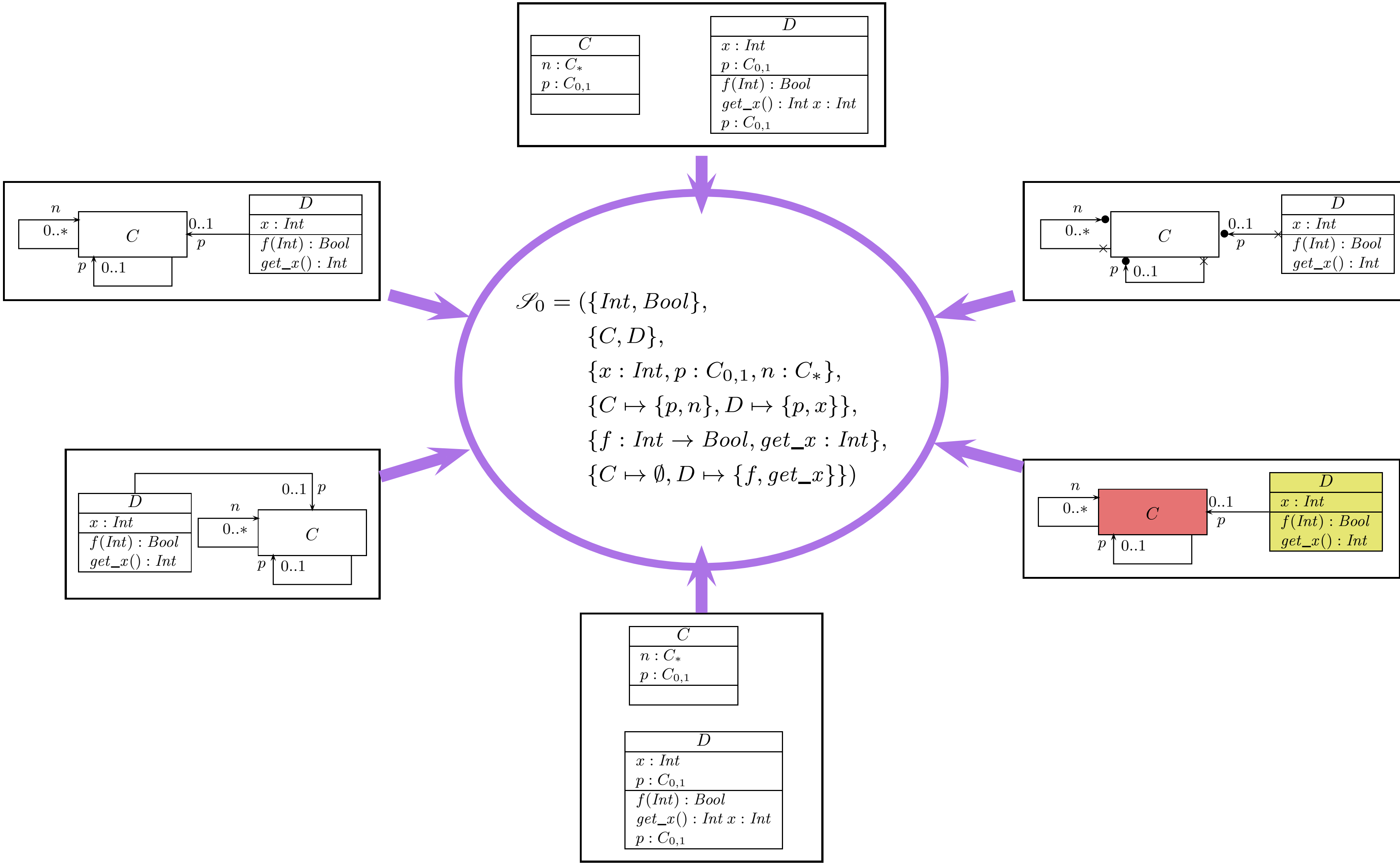


$$
\begin{aligned}
\mathscr{S}_0 = (&\{Int, Bool\},\\
&\{C, D\},\\
&\{x : Int, p : C_{0,1}, n : C_*\},\\
&\{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},\\
&\{f : Int \to Bool, get\_x : Int\},\\
&\{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\})
\end{aligned}
$$





Outlook: Concrete vs. Abstract Syntax



$$\begin{aligned}\mathscr{S}_0 = (&\{Int, Bool\},\\ &\{C, D\},\\ &\{x : Int, p : C_{0,1}, n : C_*\},\\ &\{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},\\ &\{f : Int \to Bool, get\_x : Int\},\\ &\{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\})\end{aligned}$$

Visualisation of Implementation

- The class diagram syntax can be used to **visualise code**:
  provide rules which map (parts of) the code to class diagram elements.

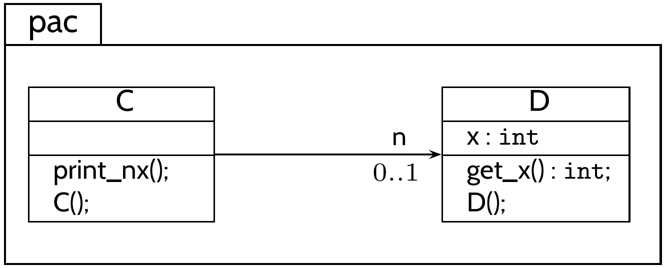
```
package pac;

import pac.D;

public class C {

  public D n;

  public void print_nx() {
    System.out.printf(
      "%i\n", n.get_x() ); };

  public C() {};
}
```

```
package pac;

import pac.C;

public class D {

  private int x;

  public int get_x()
    { return x; };

  public D() {};
}
```

- The class diagram syntax can be used to **visualise code**:
  provide rules which map (parts of) the code to class diagram elements.

```java
package pac;

import pac.D;

public class C {

  public D n;

  public void print_nx() {
    System.out.printf(
      "%i\n", n.get_x() ); };

  public C() {};
}
```

```java
package pac;

import pac.C;

public class D {

  private int x;

  public int get_x()
    { return x; };

  public D() {};
}
```



Visualisation of Implementation: (Useless) Example

- open favourite IDE,
- open favourite **project**,
- press "**generate class diagram**"
- **wait**…

Visualisation of Implementation: (Useless) Example

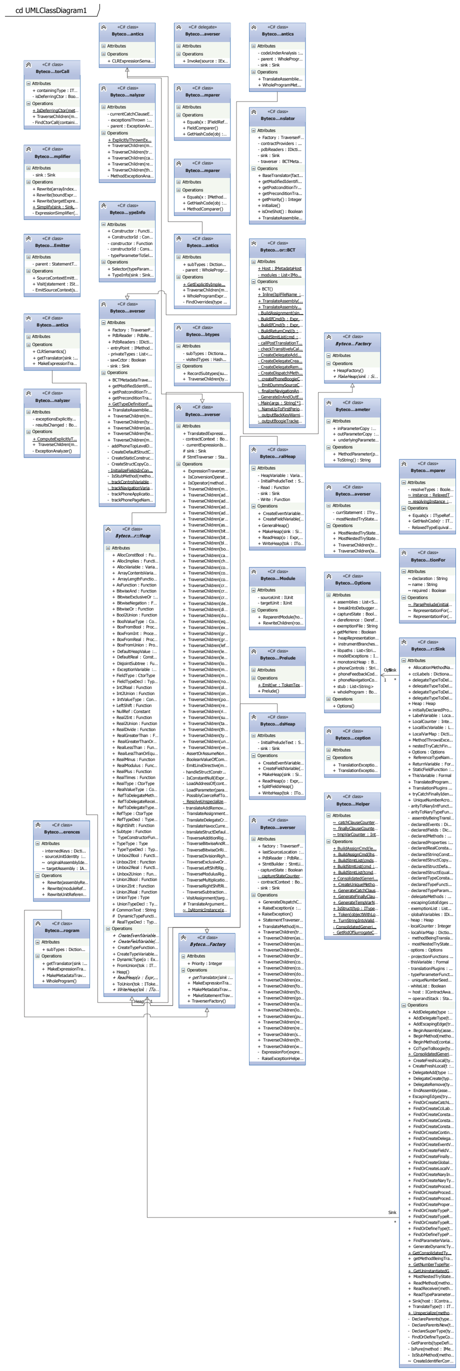
- open favourite IDE,
- open favourite **project**,
- press “**generate class diagram**”
- **wait**. . . **wait**. . .

Visualisation of Implementation: (Useless) Example

- open favourite IDE,
- open favourite **project**,
- press "**generate class diagram**"
- **wait**. . . **wait**. . . **wait**. . .

Visualisation of Implementation: (Useless) Example

- open favourite IDE,
- open favourite **project**,
- press "**generate class diagram**"
- **wait**...**wait**...**wait**...



- ca. 35 classes,
- ca. 5,000 LOC C#

Table of Contents

Table of Non-Contents

Everything else, including

- **Development Process**
  UML is only the language for artefacts. **But**: we'll discuss exemplarily, where in an abstract development process which means could be used.

- **How to come up with a good design**
  UML is only the language to write down designs.
  But: we'll have a couple of examples.

- **Artefact Management**
  Versioning, Traceability, Propagation of Changes.

- **Every little bit and piece of UML**
  Boring. Instead we learn how to read the standard.

- **Object Oriented Programming**
  Interestingly, inheritance is one of the last lectures.

Content

A Brief History of UML

A Brief History of the Unified Modelling Language (UML)

- Boxes/lines and finite automata are used to visualise software **for ages**.

A Brief History of the Unified Modelling Language (UML)

- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's**, **Software Crisis**™

  Idea: learn from engineering disciplines to handle growing complexity.

  Modelling languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**

A Brief History of the Unified Modelling Language (UML)

- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's**, **Software Crisis**™

  Idea: learn from engineering disciplines to handle growing complexity.

  Modelling languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**
- Mid **1980**'s: **Statecharts** (Harel, 1987), **StateMate**™ (Harel et al., 1990)

A Brief History of the Unified Modelling Language (UML)

- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's**, **Software Crisis**™

  Idea: learn from engineering disciplines to handle growing complexity.

  Modelling languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**
- Mid **1980**'s: **Statecharts** (Harel, 1987), **StateMate**™ (Harel et al., 1990)
- Early **1990's**, advent of **Object-Oriented**-Analysis/Design/Programming
  – Inflation of notations and methods, most prominent:

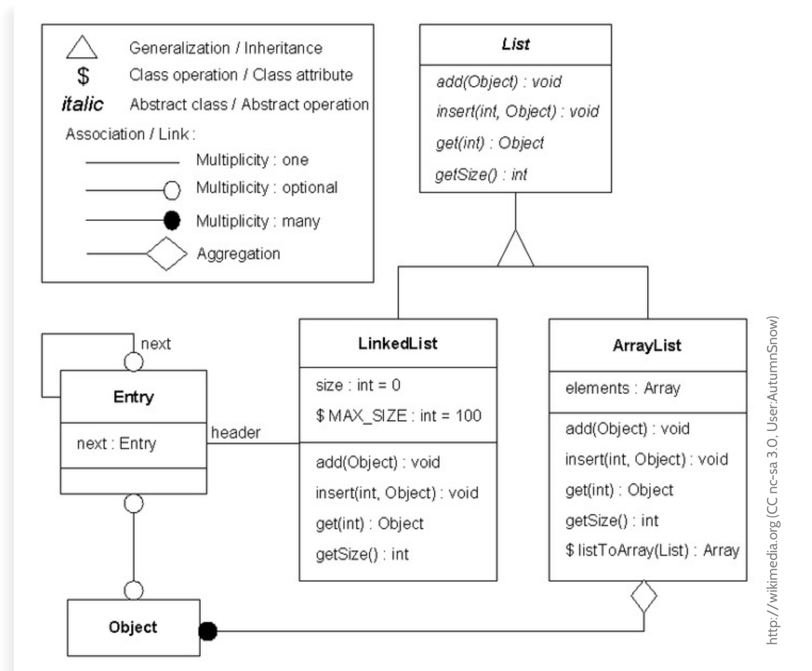
A Brief History of the Unified Modelling Language (UML)

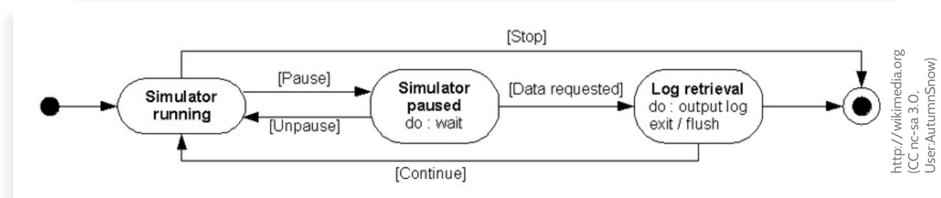
- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's**, **Software Crisis**™

  Idea: learn from engineering disciplines to handle growing complexity.

  Modelling languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**

- Mid **1980**'s: **Statecharts** (Harel, 1987), **StateMate**™ (Harel
- Early **1990's**, advent of **Object-Oriented**-Analysis/Desigr
  – Inflation of notations and methods, most prominent:
  - **Object-Modeling Technique** (OMT)
    (Rumbaugh et al., 1990)





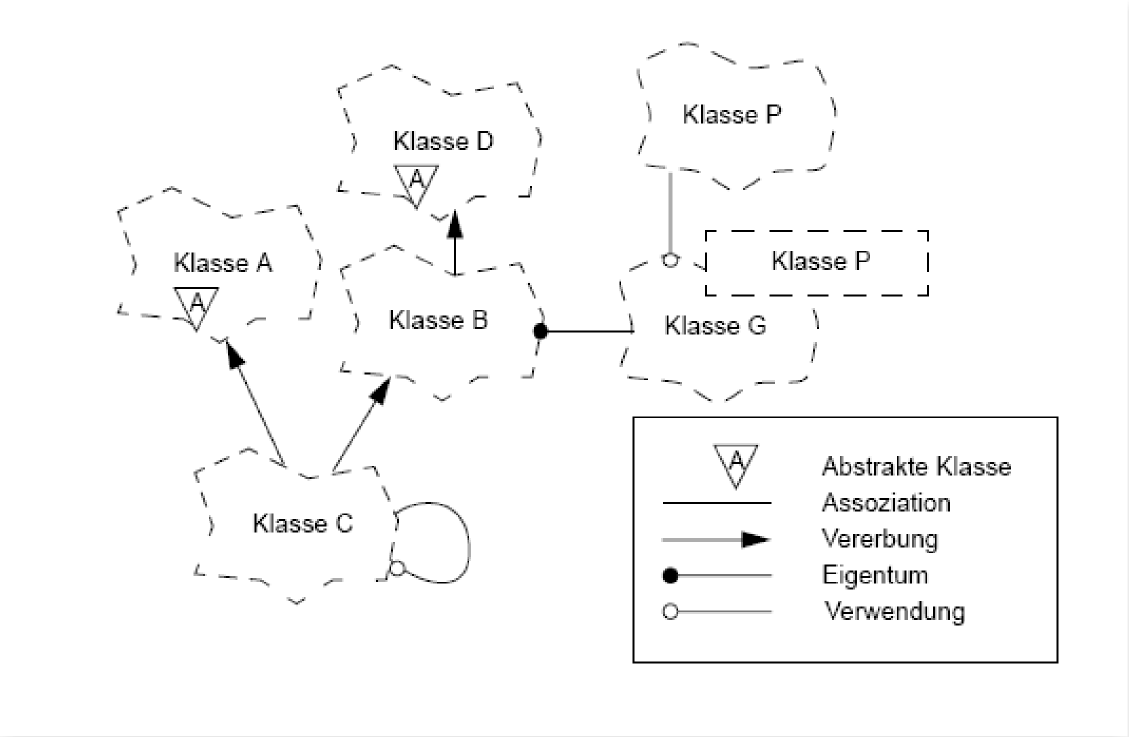
A Brief History of the Unified Modelling Language (UML)

- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's**, **Software Crisis**™

  Idea: learn from engineering disciplines to handle growing complexity.

  Modelling languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**

- Mid **1980**'s: **Statecharts** (Harel, 1987), **StateMate**™ (Harel et al., 1990)
- Early **1990's**, advent of **Object-Oriented**-Analysis/Design/Programming
  – Inflation of notations and methods, mo
  - **Object-Modeling Technique** (OMT) (Rumbaugh et al., 1990)
  - **Booch Method and Notation** (Booch, 1993)



http://wikimedia.org (Public domain, Johannes Fasolt)

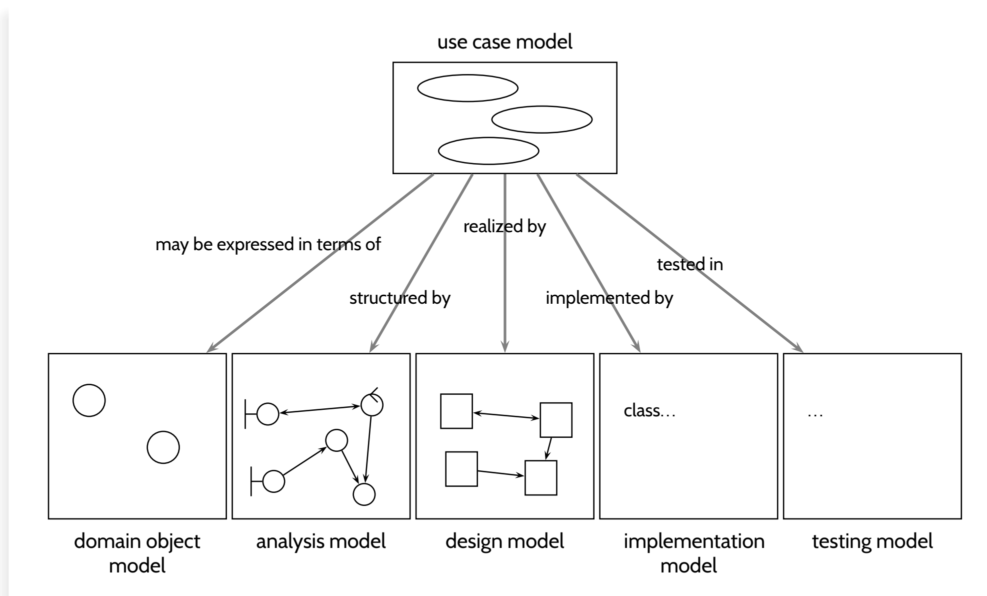
A Brief History of the Unified Modelling Language (UML)

- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's**, **Software Crisis**™

  Idea: learn from engineering disciplines to handle growing complexity.

  Modelling languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**
- Mid **1980**'s: **Statecharts** (Harel, 1987), **StateMate**™ (Harel et al., 1990)
- Early **1990's**, advent of **Object-Oriented**-Analysis/Design/Programming
  – Inflation of notations and methods, most prominent:
  - **Object-Modeling Technique** (OMT) (Rumbaugh et al., 1990)
  - **Booch Method and Notation** (Booch, 1993)
  - **Object-Oriented Software Engineering** (OOSE) (Jacobson et al., 1992)

  Each "persuasion" selling books, tools, seminars…



A Brief History of the Unified Modelling Language (UML)

- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's**, **Software Crisis**™

  Idea: learn from engineering disciplines to handle growing complexity.

  Modelling languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**

- Mid **1980**'s: **Statecharts** (Harel, 1987), **StateMate**™ (Harel et al., 1990)
- Early **1990's**, advent of **Object-Oriented**-Analysis/Design/Programming
  – Inflation of notations and methods, most prominent:

  - **Object-Modeling Technique** (OMT)
    (Rumbaugh et al., 1990)
  - **Booch Method and Notation**
    (Booch, 1993)
  - **Object-Oriented Software Engineering** (OOSE)
    (Jacobson et al., 1992)

  Each "persuasion" selling books, tools, seminars...

- Late **1990's**: joint effort of "the three amigos" yielded **UML 0.x** and **1.x**

  The standards are published by **Object Management Group** (OMG), "*international, open membership, not-for-profit **computer industry** consortium*". Much criticised for lack of formality.

A Brief History of the Unified Modelling Language (UML)

- Boxes/lines and finite automata are used to visualise software **for ages**.
- **1970's**, **Software Crisis**™

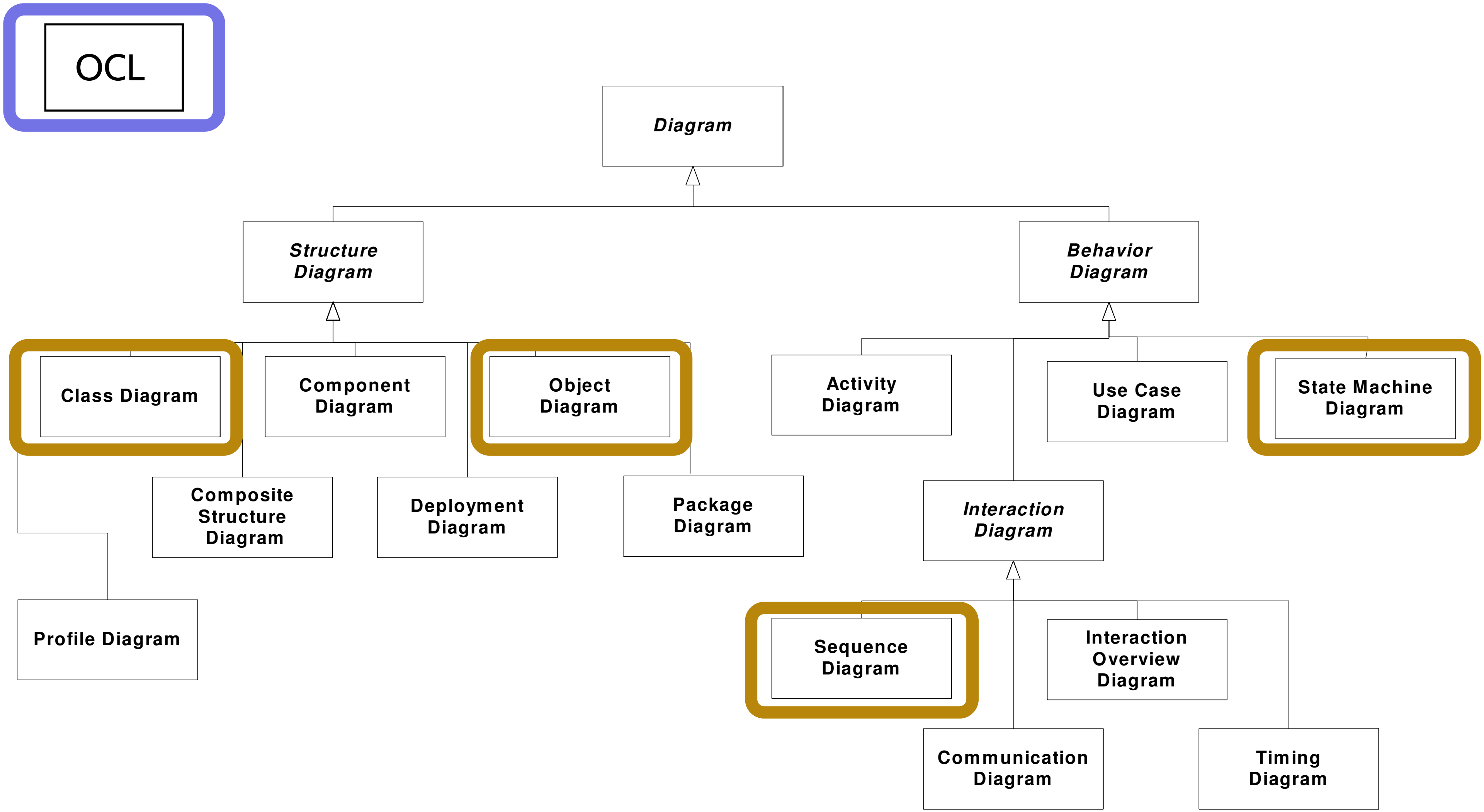
  Idea: learn from engineering disciplines to handle growing complexity.

  Modelling languages: **Flowcharts, Nassi-Shneiderman, Entity-Relation Diagrams**
- Mid **1980**'s: **Statecharts** (Harel, 1987), **StateMate**™ (Harel et al., 1990)
- Early **1990's**, advent of **Object-Oriented**-Analysis/Design/Programming
  – Inflation of notations and methods, most prominent:
  - **Object-Modeling Technique** (OMT)
    (Rumbaugh et al., 1990)
  - **Booch Method and Notation**
    (Booch, 1993)
  - **Object-Oriented Software Engineering** (OOSE)
    (Jacobson et al., 1992)

  Each "persuasion" selling books, tools, seminars...
- Late **1990's**: joint effort of "the three amigos" yielded **UML 0.x** and **1.x**

  The standards are published by **Object Management Group** (OMG), "*international, open membership, not-for-profit **computer industry** consortium*". Much criticised for lack of formality.
- Since **2005**: **UML 2.x**, split into infra- and superstructure documents.

Recall: UML Diagrams (OMG, 2011b, 694)

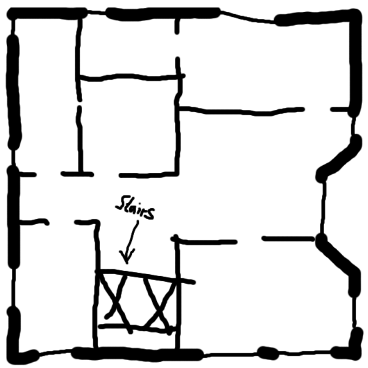


Dobing and Parsons (2006)

UML Modes

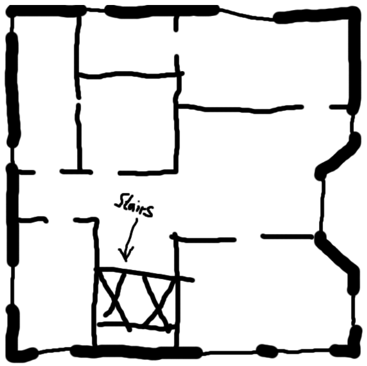
Floorplan and UML Modes!

Sketch:

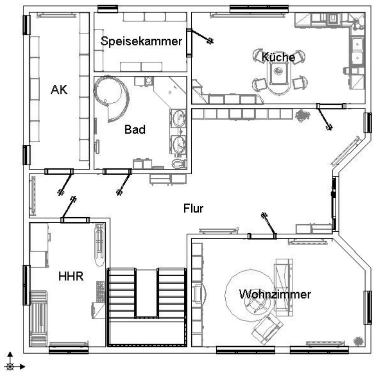


Floorplan and UML Modes!

Sketch:

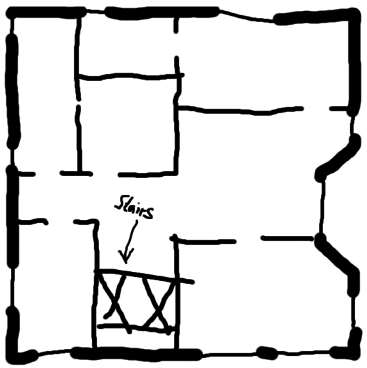


Blueprint:

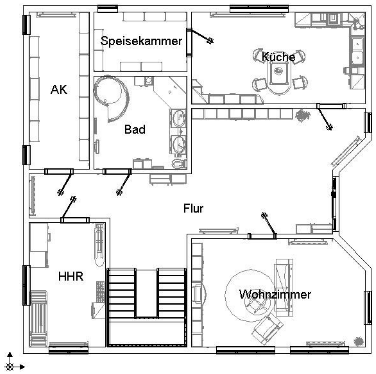


Floorplan and UML Modes!

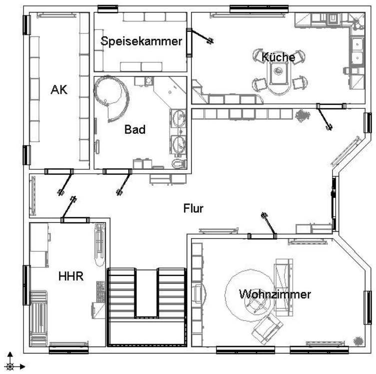
Sketch:



Blueprint:



Program:

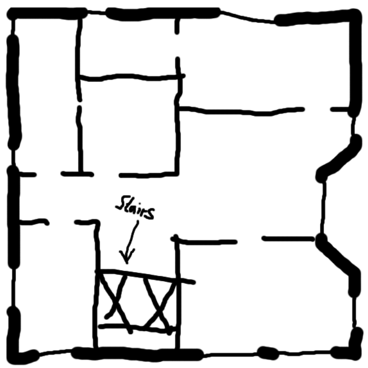


+ wiringplan + windows

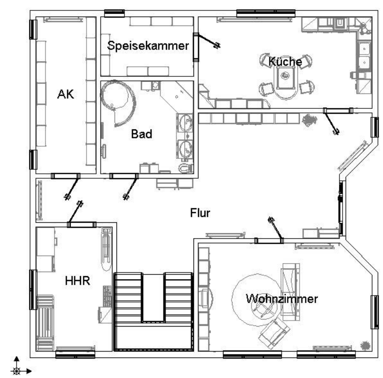
+ ...

Floorplan and UML Modes!

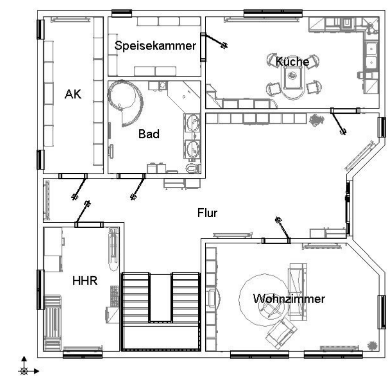
Sketch:



Blueprint:

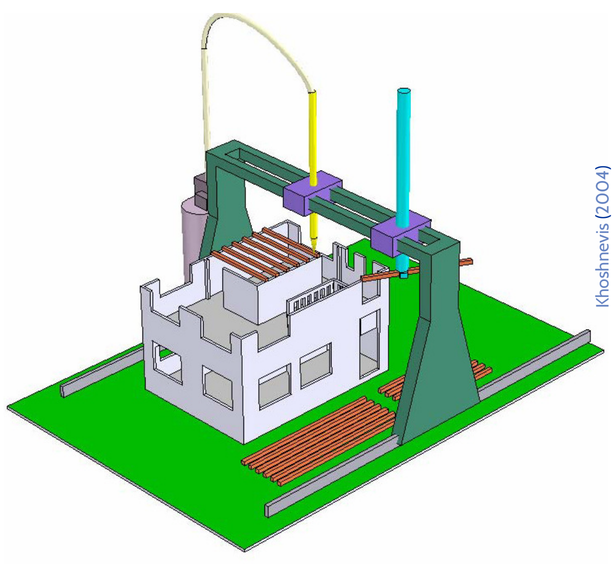


Program:



\+ wiringplan + windows

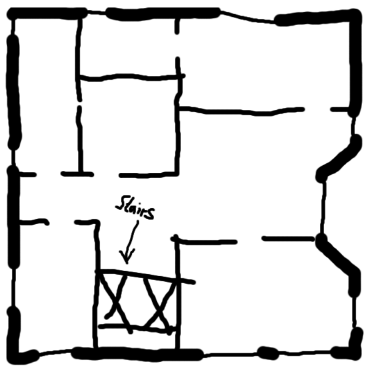
\+ ...

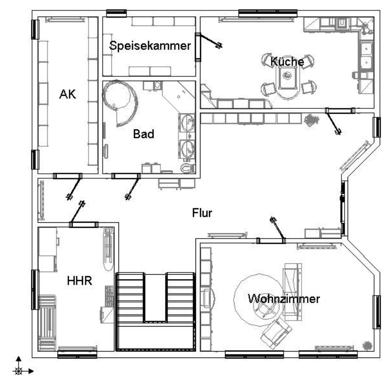


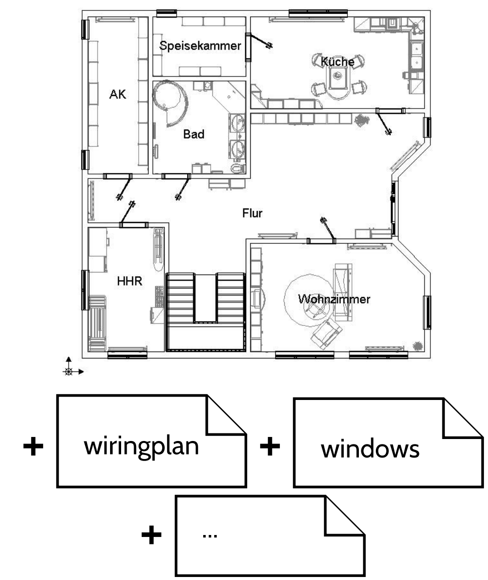
Khoshnevis (2004)

Floorplan and UML Modes!

Sketch: | **Blueprint**: | **Program**:

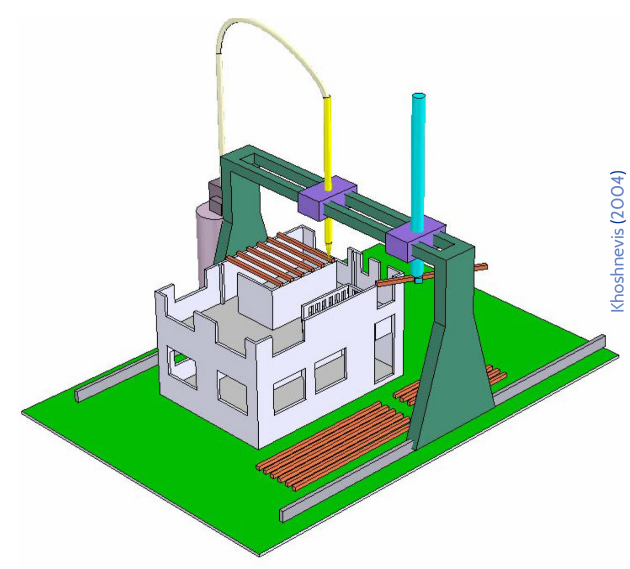






With UML it's the same [`http://martinfowler.com/bliki`]:

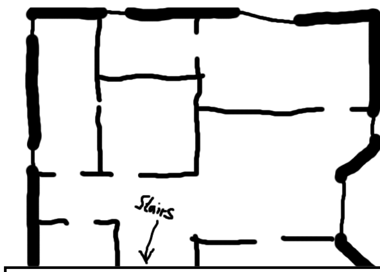
> *"[...] people differ about what should be in the UML because there are* ***differing fundamental views about what the UML should be****.*
>
> *So when someone else's view of the UML seems rather different to yours, it may be because they use a different* ***UmlMode*** *to you."*



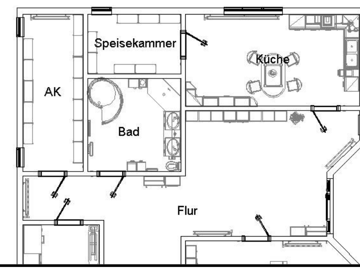
Khoshnevis (2004)

Floorplan and UML Modes!

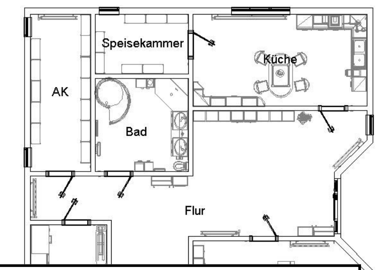
Sketch:



Blueprint:



Program:



Sketch

In this UmlMode developers use the UML to help communicate some aspects of a system. [...]

Sketches are also useful in documents, in which case the focus is communication ra- ther than completeness. [...]

The tools used for sketching are lightweight drawing tools and often peo- ple aren't too particular about keeping to every strict rule of the UML.

Most UML diagrams shown in books, such as mine, are sketches. Their emphasis is on selective communication rather than complete specification.

Hence my sound-bite "comprehensive- ness is the enemy of comprehensibility"

Blueprint

[...] In forward engineering the idea is that blueprints are developed by a designer whose job is to build a detailed design for a programmer to code up.

That design should be sufficiently com- plete that all design decisions are laid out and the programming should follow as a pretty straightforward activity that re- quires little thought. [...]

Blueprints require much more sophisti- cated tools than sketches in order to han- dle the details required for the task. [...]

Forward engineering tools support dia- gram drawing and back it up with a repository to hold the information. [...]

Programming Language

If you can detail the UML enough, and provide semantics for everything you need in software, you can make the UML be your programming language.

Tools can take the UML diagrams you draw and compile them into executable code.

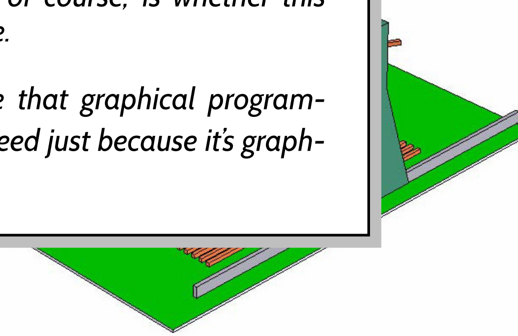
The promise of this is that UML is a higher level language and thus more productive than current programming languages.

The question, of course, is whether this promise is true.

I don't believe that graphical program- ming will succeed just because it's graph- ical. [...]

With

you.



Khoshnevis (2004)

UML-Mode of the Course

So, the "mode" fitting the lecture best is **AsBlueprint**.

UML-Mode of the Course

So, the "mode" fitting the lecture best is **AsBlueprint**.

Aim of the Course:

- show that UML can be **precise** – to **avoid misunderstandings**.
- allow **formal analysis** of models on the **design level** – to **find errors early**.
- be consistent with (informal semantics in) OMG (2011b) as far as possible.

UML-Mode of the Course

So, the "mode" fitting the lecture best is **AsBlueprint**.

Aim of the Course:

- show that UML can be **precise** – to **avoid misunderstandings**.
- allow **formal analysis** of models on the **design level** – to **find errors early**.
- be consistent with (informal semantics in) OMG (2011b) as far as possible.

Side Effects:

After the course, you should...

- have a good working knowledge of UML,
- have a good working knowledge of software modelling,
- be able to **also** efficiently and effectively work in **AsSketch** mode,
- be able to define **your own** UML semantics for **your** context/purpose, or define your own **Domain Specific Languages** as needed.

Content

Formalia

Formalia: Lectures

- **Lecturer:** Dr. Bernd Westphal
- **Support:** Claus Schätzle
- **Homepage:** `http://swt.informatik.uni-freiburg.de/teaching/WS2016-17/sdmauml`
- **Time/Location:** Tuesday, Thursday, 8:00 – 10:00 / here (building 51, room 03-026)

- **Course language:** **English** (slides/writing, presentation, questions/discussions)

- **Presentation:** half slides/half on-screen **hand-writing** – for reasons

- **Script/Media:**
  - slides with annotations on **homepage**,
    typically soon **after** the lecture
  - recording on ILIAS with max. 1 week delay
    (links on **homepage**)

- **Break:**
  - We'll have a **10 min. break** in the middle of each event from now on,
    unless a majority objects now.

Formalia: Exercises and Tutorials

- You should work in groups of **approx. 3**, clearly give **names** on submission.
- Please submit via ILIAS (cf. homepage); **paper submissions** are **tolerated**.

Formalia: Exercises and Tutorials

- You should work in groups of **approx. 3**, clearly give **names** on submission.
- Please submit via ILIAS (cf. homepage); **paper submissions** are **tolerated**.
- **Schedule:**

Week N,	Thursday,	8–10	**Lecture A1**	(exercise sheet A **online**)
Week $N+1$,	Tuesday	8–10	**Lecture A2**	
	Thursday	8–10	**Lecture A3**	
Week $N+2$,	Monday,	12:00		(exercises A **early submission**)
	Tuesday,	8:00		(exercises A **late submission**)
		8–10	**Tutorial A**	
	Thursday	8–10	**Lecture B1**	(exercise sheet B **online**)
	…			

Formalia: Exercises and Tutorials

- You should work in groups of **approx. 3**, clearly give **names** on submission.
- Please submit via ILIAS (cf. homepage); **paper submissions** are **tolerated**.
- **Schedule:**

Week N,	Thursday,	8–10	**Lecture A1**	(exercise sheet A **online**)
Week $N+1$,	Tuesday	8–10	**Lecture A2**	
	Thursday	8–10	**Lecture A3**	
Week $N+2$,	Monday,	12:00		(exercises A **early submission**)
	Tuesday,	8:00		(exercises A **late submission**)
		8–10	**Tutorial A**	
	Thursday	8–10	**Lecture B1**	(exercise sheet B **online**)
	...			

- **Rating system:** "most complicated rating system **ever**"
  - **Admission points** (good-will rating, upper bound)
    ("reasonable proposal given student's knowledge **before** tutorial")
  - **Exam-like points** (evil rating, lower bound)
    ("reasonable proposal given student's knowledge **after** tutorial")

10% bonus for **early** submission.

Formalia: Exercises and Tutorials

- You should work in groups of **approx. 3**, clearly give **names** on submission.
- Please submit via ILIAS (cf. homepage); **paper submissions** are **tolerated**.
- **Schedule:**

Week N,	Thursday,	8–10	**Lecture A1**	(exercise sheet A **online**)
Week $N+1$,	Tuesday	8–10	**Lecture A2**	
	Thursday	8–10	**Lecture A3**	
Week $N+2$,	Monday,	12:00		(exercises A **early submission**)
	Tuesday,	8:00		(exercises A **late submission**)
		8–10	**Tutorial A**	
	Thursday	8–10	**Lecture B1**	(exercise sheet B **online**)
	...			

- **Rating system:** "most complicated rating system **ever**"
  - **Admission points** (good-will rating, upper bound)
    ("reasonable proposal given student's knowledge **before** tutorial")
  - **Exam-like points** (evil rating, lower bound)
    ("reasonable proposal given student's knowledge **after** tutorial")

  10% bonus for **early** submission.

- **Tutorial:** Plenary, **not recorded**.
  - Together develop **one good solution** based on selection of early submissions (anonymous) – there is no "Musterlösung" for modelling tasks.

Formalia: Exam

- **Exam Admission:**

  Achieving 50% of the regular **admission points** in total is **sufficient** for admission to exam.

  Typically, 20 regular admission points per exercise sheet; some exercise sheets have **bonus tasks**.

- **Exam Form:**

  - oral for BSc and on special demand (Erasmus),
  - **written** for everybody else (if sufficiently many candidates remain).

  Scores from the exercises **do not** contribute to the final grade.

- **Exam Date:**

  Please remind me in early December that we need to agree on an exam date.

User's Guide

- **Approach:**

  The lectures is supposed to work as a **lecture**: **spoken word** $+$ **slides** $+$ **discussion**

  It is **not our goal** to make any of the three work in isolation.

User's Guide

- **Approach:**

  The lectures is supposed to work as a **lecture**: **spoken word** + **slides** + **discussion**

  It is **not our goal** to make any of the three work in isolation.

- **Interaction:**

  Absence often moaned but **it takes two**: **please ask/comment immediately**.

User's Guide

- **Approach:**

  The lectures is supposed to work as a **lecture**: **spoken word** $+$ **slides** $+$ **discussion**

  It is **not our goal** to make any of the three work in isolation.

- **Interaction:**

  Absence often moaned but **it takes two**: **please ask/comment immediately**.

- **Exercise submissions:**

  Each task is a **tiny little scientific work**:

  (i) Briefly rephrase the task in your own words.

  (ii) State your claimed solution.

  (iii) Convince your reader that your proposal is a solution (proofs are very convincing).

- **App**

  The

  It is

Example:

Task: Given a square with side length $a = 19.1$. What is the length of the longest straight line fully inside the square?

Submission A:

27

Submission B:

The length of the longest straight line fully inside the square with side length $a = 19.1$ is 27.01 (rounded).

The longest straight line inside the square is the diagonal. By Pythagoras, its length is $\sqrt{a^2 + a^2}$. Inserting $a = 19.1$ yields 27.01 (rounded).

- **Inte**

  Abs

- **Exercise submissions:**

  Each task is a **tiny little scientific work**:

  (i) Briefly rephrase the task in your own words.

  (ii) State your claimed solution.

  (iii) Convince your reader that your proposal is a solution (proofs are very convincing).

- **App**

  The

  It is

Example:

Task: Given a square with side length $a = 19.1$. What is the length of the longest straight line fully inside the square?

Submission A:

Submission B:

The length of the longest straight line fully inside the square with side l[illegible]gth $a = 19.1$ is 27.01 (rounded).

The longest s[illegible]ht[illegible] inside the square is the diagona[illegible]thagoras, its length is $\sqrt{a^2 + a^2}$. Ins[illegible]ng $a = 19.1$ yields 27.01 (rounded).

- **Inte**

  Abs

- **Exercise submissions:**

  Each task is a **tiny little scientific work**:

  (i) Briefly rephrase the task in your own words.

  (ii) State your claimed solution.

  (iii) Convince your reader that your proposal is a solution (proofs are very convincing).

Literature

Literature: Modelling

- 

  - W. Hesse, H. C. Mayr: Modellierung in der Softwaretechnik: eine Bestandsaufnahme, Informatik Spektrum, 31(5):377-393, 2008.
  - O. Pastor, S. Espana, J. I. Panach, N. Aquino: Model-Driven Development, Informatik Spektrum, 31(5):394-407, 2008.
  - M. Glinz: Modellierung in der Lehre an Hochschulen: Thesen und Erfahrungen, Informatik Spektrum, 31(5):408-424, 2008.

  `http://www.springerlink.com/content/0170-6012`

- U. Kastens, H. Kleine Büning: Modellierung – Grundlagen und Formale Methoden, 2. Auflage, Hanser-Verlag, 2008.

Literature: UML

- OMG: Unified Modeling Language Specification, Infrastructure, 2.4.1
- OMG: Unified Modeling Language Specification, Superstructure, 2.4.1
- OMG: Object Constraint Language Specification, 2.0

  All three: `http://www.omg.org` (cf. hyperlinks on course homepage)

- A. Kleppe, J. Warmer: The Object Constraint Language,
  Second Edition, Addison-Wesley, 2003.

- D. Harel, E. Gery: Executable Object Modeling with Statecharts,
  IEEE Computer, 30(7):31-42, 1997.
- B. P. Douglass: Doing Hard Time, Addison-Wesley, 1999.
- B. P. Douglass: ROPES: Rapid Object-Oriented Process for Embedded Systems, i-Logix Inc., Whitepaper, 1999.

- B. Oesterreich: Analyse und Design mit UML 2.1,
  8. Auflage, Oldenbourg, 2006.
- H. Stoerrle: UML 2 für Studenten, Pearson Studium Verlag, 2005.

References

References

Booch, G. (1993). *Object-oriented Analysis and Design with Applications*. Prentice-Hall.

Dobing, B. and Parsons, J. (2006). How UML is used. *Communications of the ACM*, 49(5):109–114.

Glinz, M. (2008). Modellierung in der Lehre an Hochschulen: Thesen und Erfahrungen. *Informatik Spektrum*, 31(5):425–434.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.

Harel, D., Lachover, H., et al. (1990). Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414.

Jacobson, I., Christerson, M., and Jonsson, P. (1992). *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison-Wesley.

Khoshnevis, B. (2004). Automated construction by contour crafting – related robotics and information technologies. *Journal of Automation in Construction*, 13:5–19.

OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.

OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1990). *Object-Oriented Modeling and Design*. Prentice Hall.