Motivation and Context
"if there is water in stock, it must be possible to buy water at the price of 50 cent"

Among other interactions:
- insert 50 cent coin,
- press ‘WATER’, wait,
- check: yes, water in stock,
- press ‘WATER’ again,
- wait...

What went wrong?
- Was there a misunderstanding of the requirements? $(\rightarrow$ redo all the work$)$
- Is there an error in our design? $(\rightarrow$ redo design and implementation$)$
- Is there a “bug” in our implementation? $(\rightarrow$ fix the implementation$)$

And can we do something to avoid it in the next project...?
**An Analogy: Construction Engineering**

**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:

1. **image attribute** (Abbildungsmerkmal), i.e. there is an entity (called original) whose image or archetype the model is,
2. **reduction attribute** (Verkürzungsmerkmal), i.e. only those attributes of the original that are relevant in the modelling context are represented,
3. **pragmatic attribute**, i.e. the model is built in a specific context for a specific purpose.
Floorplans as Models

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

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

Models in Software Engineering

"if there is water in stock, it must be possible to buy water at the price of 50 cent"

requirements model  design model/structure  design model/behaviour
One Software Modelling Language: UML (OMG, 2011b, 694)

(Our) Premises for Using a Software Modelling Language

(i) We want to know how the words of the language look like: Syntax.
    (In UML: when is a diagram a proper state machine?)
(Our) Premises for Using a Software Modelling Language

(i) We want to know how the words of the language look like: Syntax.
(In UML: when is a diagram a proper state machine?)

(ii) We want to know what a word of the language means: Semantics.
(In UML: can sending event $E$ and then $G$ kill the object?)

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

- 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.
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.”

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 rather than completeness. […] The tools used for sketching are lightweight drawing tools and often people 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 “comprehensiveness 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 complete that all design decisions are laid out and the programming should follow as a pretty straightforward activity that requires little thought. [...] Blueprints require much more sophisticated tools than sketches in order to handle the details required for the task. [...] Forward engineering tools support diagram 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 programming will succeed just because it’s graphical. […]
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 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.

The Lecture: Content and Non-Content
Course Map

Table of Contents

- Introduction (VL 01)
- Semantical Domain (VL 01–02)

Modelling Structure:
- OCL (VL 03–04)
- Object Diagrams (VL 05)
- Class Diagrams (VL 06–09)
- Modelling Guidelines (VL 10)

Modelling Behaviour:
- Constructive:
  - Simple State Machines (VL 11–13)
  - Hierarchical State Machines (VL 14–15)
  - Code Generation (VL 16)
- Reflective:
  - Live Sequence Charts (VL 17–18)

The Rest:
- Inheritance (VL 19–20)
- Meta-Modeling (VL 21)
- Putting it all together: MDA, MDSE (VL 22)
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
  Interesting: inheritance is one of the last lectures.

Formalia
Formalism: Lectures

- **Lecturer:** Dr. Bernd Westphal
- **Support:** Milan Vujinovic
- **Homepage:** http://swt.informatik.uni-freiburg.de/teaching/WS2015-16/sdmauml
- **Time/Location:** Tuesday, Thursday, 10:00 – 12:00 / 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.

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

- **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:**
  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.

- **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).
**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:

  1. Briefly rephrase the task in your own words.
  2. State your claimed solution.
  3. Convince your reader that your proposal is a solution (proofs are very convincing).

**Example:**

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

<table>
<thead>
<tr>
<th>Submission A</th>
<th>Submission B</th>
</tr>
</thead>
<tbody>
<tr>
<td>The length of the longest straight line fully inside the square with side length ( a = 19.1 ) is 27.01 (rounded).</td>
<td>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).</td>
</tr>
</tbody>
</table>

| X | ✔ |

User’s Guide
Literature: Modelling


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

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)


References


