Software Engineering 2
A practical course in software engineering

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science
I. Introduction
Introduction

- Motivation
- The role of models in software engineering
- Software engineering & management

- Organisation of this course

- Project & tutorials
  - The task
  - Technology tutorials
  - Forming the groups
- Overview of Cloud and IBM Bluemix
# Weekly Schedule (roughly)

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-10</td>
<td></td>
<td>lecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13-15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15-17</td>
<td></td>
<td></td>
<td></td>
<td>project</td>
</tr>
</tbody>
</table>

- Plus actual work on the project!!

There will be many exceptions from the rule! See web pages for details!
1. Motivation

- Objectives of this course: **Skills** in software engineering!

- What is “software engineering”?  
- What is “software”?  
  
- software ≠ program
- software engineering ≠ programming

**SE 2 (02162 e015), L 01**
Program vs. Software

Software >> Program

Software Engineering >>> Programming

is much more than

is much much more than
If somebody has built a garage, would we let him build a skyscraper? No, never!

If somebody has written a program, would we let him build software? Yes we would!

Program
Programmer
Programming

But, of course we should not!
Software Engineering is

... much more than programming!
... listening and understanding!
... analytic and conceptual work!
... communication!
... a social process!
... acquiring and using new technologies!

... a discipline with proven concepts, methods, notations, and tools!
... and ever new technologies emerging!

but, still evolving
Software Engineering requires much experience!

This experience
- can not be taught theoretically!
- will be provided in this course!

→ project
→ tutorial (new technologies)
→ and (only) backed by the lectures
Analogy revisited

Effort per participant
• 10 ECTS = ca. 270h work
• ca. 20h/week

The experience of a big project cannot be replaced by the experience of many small ones.
Objective

Practice the concepts, methods, notations and tools for software engineering

- improve programming skills
- understanding of the software engineering process
- experiences with problems and concepts for solving them
- writing and creating documents and models
- use of methods and tools
- practice communication and presentation skills
- capability of teamwork and leadership
- acquire new technologies
- ...

If in doubt:
1. think
2. search ("google")
3. ask
Excursion: CDIO

Conceive
Design
Implement
Operate
Co-evolution

“What” should the software do?

“How” is it realized?

Reality is even more complicated, but keeping "what" and "how" separate in your mind is a first step.
Questions

- Why do so many software projects fail?

- Why is software development so hard (or at least harder as we believe)?

- BTW: What is software?
Die Menge aller **Programme, Prozeduren und Objekte**, zusammen mit den zugehörigen **Daten** und der **Dokumentation**, die für eine lauffähige Anwendung nötig oder wünschenswert sind.

[frei nach Informatik DUDEN und Hesse]
The sum of all **programs**, **procedures** and **objects** along with the associated **data** and **documentation**, which are necessary (or at least desirable) for running an application on a computer system.
Software …

is becoming more and more complex!

Exponential growth of software (in „lines of code“ LOC) within the same product line:

- Apollo (NASA‘s Apollo programme)
- Cars (automotive software)
- …

Technology needs to keep up with that!
Software ...

cannot be „programmed“ by a single person anymore; a single person cannot fully comprehend all details of software any more.

Efforts of 10 to 100 person years (PYs) are quite standard in software development.
Software …

is intangible.

You cannot touch, see or feel software. Humans lack a “natural feeling” of software and its complexity.
Software does not wear out, but becomes of age anyway (in relation to the environment it is running in and the expectations of the end user)!

Software needs „maintenance“! But, this does not mean the same as in traditional engineering (e.g. in mechanical engineering, where systems physically wear out).

Actually, maintenance is a big factor in the cost of IT systems.
„lives“ longer than its creators expect it to live.
is everywhere and many lifes depend on it.
Software Engineering is

... much more than programming!
... listening and understanding!
... analytic and conceptual work!
... communication!
... a social process!
... acquiring new technologies!
Problems

- imprecise requirements
- mistakable and unclear requirements
- inconsistent requirements
- changing requirements

- changing environments (software / hardware)
- different versions and configurations
- changing tools, notations, languages, methods, concepts, technologies

- collective knowledge only
- communication
- …
is the sum of all means, facilities, procedures, processes, notations, methods, concepts for developing, operating and maintaining a software system.
Software engineering

Branches:

- **Development**: actual development of the software product
- **Management**: Manage (control and improve) the development process
- **Quality management**: Planning and implementing measures that guarantee that the software meets the required quality
- **Software maintenance**: Remove faults occurring in operation, adapt software to changing requirements and environments
Process models (life cycle models) are the „distilled“ experience of successful software projects.

They define a functional procedure along with appropriate documents.

- **What** should be done
document, notation
- **when**, phase
- **by whom** and role
- **how!** method
Problem: Often process models are used very mechanical and in a „meaningless“ way.

→ documents just for the sake of the process
→ (UML) diagrams just for the sake of UML
→ comments just for the sake of comments

Therefore:
1. Think!
2. What is reasonable?
Rule of thumb

When producing and compiling a document, ask yourself:

- What should the document be good for?
- Who should be addressed?
- Which information is expected?
- What is the common „pragmatics“?
- …

**In short:** What is reasonable?
... and a glimpse of how software can be developed by using models – without doing any programming at all.
Models are the “floor plans” of software engineers, and are the key to the success of software projects.
A Model (Petri net)
Stages

- Examples
- Taxonomy (done on blackboard)
- Glossary
- Model (see next slide)

Rule: Never ever start making a UML model without having seen some examples first and naming the main concepts (taxonomy)!
Models and Meta Models

Meta model for Petri nets

PetriNet

context Arc inv:
( self.source.oclIsKindOf(Place) and self.target.oclIsKindOf(Transition) )
or
( self.source.oclIsKindOf(Transition) and self.target.oclIsKindOf(Place) )

Object

Node

Transition

Place

Arc

Token

1 source

1 target

Meta model for Petri nets

Model for Petri nets

Meta model for Petri nets

Place

Transition

Arc
Don’t think models as Java

Rule: Don’t think of "HOW" (programming) for now! These models are "WHAT" only: which concepts are there in Petri nets?

context Arc inv: ( self.source.oclIsKindOf(Place) and self.target.oclIsKindOf(Transition) ) or ( self.source.oclIsKindOf(Transition) and self.target.oclIsKindOf(Place) )
Syntax (abstract and concrete)

- **graphical / concrete syntax**

**abstract syntax** (as an UML object diagram)

- Petri net
  - Transition
  - Arc
  - Place
  - Token
  - Source
  - Target
Overview

meta model

build-time

is an
instance of

model

runtime
Benefits of Modelling

- Better understanding
- Mapping of instances to XML syntax (XMI)
- Automatic code generation
  - API for creating, deleting and modifying model
  - Methods for loading and saving models (in XMI)
  - Standard mechanisms for keeping track of changes (observers)
The term “meta” model makes more sense now!

Class Diagrams are Models too

Note: The real model of UML is much more complicated.

UML model

Meta model for UML (class diagrams)

The term “meta” model makes more sense now!
Different Meta-levels: MOF

Where does the concrete syntax come from?
Answers:

- Program an editor
  Not a good answer here!

- Standard technology for mapping abstract to concrete syntax: EMF / GMF / EMFT
EMF/GMF Technology

meta model

is instance of

This will, however, not be covered in this year’s course

generate an editor
Benefits of Modelling (cntd.)

- Better Understanding

- Mapping of instances to XML syntax (XMI)

- Automatic Code Generation
  - API for creating, deleting and modifying model
  - Methods for loading and saving models (in XMI)
  - Standard mechanisms for keeping track of changes (observers)
  - Editors and GUIs
Theses

- We will always have programming and programmers!
- We should always teach programming!
- But, software engineers should be trained in their engineering and modelling skills!
- And this is where they should be at their best!
- Most of the rest can be automated!

- Eventually, programming will be for software engineers as assembler is today for programmers.