Softwaretechnik / Software-Engineering

Lecture 1: Introduction

2019-04-25

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
Engineering, Software, Software Engineering
**Engineering** – The application of a systematic, disciplined, quantifiable approach to structures, machines, products, systems, or processes.  

IEEE 610.12 (1990)

---

**Engineering** – is the application of knowledge in the form of science, mathematics, and empirical evidence, to the innovation, design, construction, operation and maintenance of structures, machines, materials, software, devices, systems, processes, and organizations.  

Wikipedia
## Non-Engineering vs. Engineering

<table>
<thead>
<tr>
<th></th>
<th>Non-Engineering (studio / artwork)</th>
<th>Engineering (workshop / techn. product)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deadlines</strong></td>
<td>cannot be planned due to dependency on artist’s inspiration</td>
<td>can usually be planned with sufficient precision</td>
</tr>
<tr>
<td><strong>Price / Cost</strong></td>
<td>determined by market value, not by cost</td>
<td>oriented on cost, thus calculable</td>
</tr>
<tr>
<td><strong>Evaluation and comparison</strong></td>
<td>is only possible subjectively, results are disputed</td>
<td>can be conducted using objective, quantified criteria</td>
</tr>
<tr>
<td><strong>Norms and standards</strong></td>
<td>are rare and, if known, not respected</td>
<td>exist, are known, and are usually respected</td>
</tr>
<tr>
<td><strong>Warranty and liability</strong></td>
<td>are not defined and in practice hardly enforceable</td>
<td>are clearly regulated, cannot be disclaimed</td>
</tr>
<tr>
<td><strong>Mental prerequisite</strong></td>
<td>artist’s inspiration, among others</td>
<td>the existing and available technical know-how</td>
</tr>
<tr>
<td><strong>Author</strong></td>
<td>considers the artwork as part of him/herself</td>
<td>remains anonymous, often lacks emotional ties to the product</td>
</tr>
</tbody>
</table>

*(Ludewig and Lichter, 2013)*
Content

- Terminology
  - Engineering, Software, Software Engineering

- Motivation: Successful Software Development
  - Working definition: success
  - Unsuccessful software development exists
  - Common reasons for non-success

- Course
  - Content
    - Topic areas
    - Structure of topic areas
    - Emphasis: formal methods
    - Relation to other courses
    - Literature
  - Organisation
    - Lectures
    - Tutorials
    - Exam
Software – Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

See also: application software; support software; system software.

Contrast with: hardware.  

IEEE 610.12 (1990)

Software –

1. all or part of the programs, procedures, rules, and associated documentation of an information processing system. […]

2. see 610.12

3. program or set of programs used to run a computer. […]

cf. application software

NOTE: includes firmware, documentation, data, and execution control statements.  

IEEE 24765 (2010)
Software Engineering — This Course’s Working Definition

Software Engineering —

1. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

2. The study of approaches as in (1).

IEEE 610.12 (1990)

Software Engineering —

1. the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software.

2. see IEEE 610.12 (1)


Software Engineering —

Multi-person development of multi-version programs.

D. L. Parnas (2011)

Software Engineering — the establishment and use of sound engineering principles to obtain economically software that is reliable and works efficiently on real machines.

F. L. Bauer (1971)
Content

- Terminology
  - Engineering, Software, Software Engineering

- Motivation: Successful Software Development
  - Working definition: success
  - Unsuccessful software development exists
  - Common reasons for non-success

- Course
  - Content
    - Topic areas
    - Structure of topic areas
    - Emphasis: formal methods
    - Relation to other courses
    - Literature
  - Organisation
    - Lectures
    - Tutorials
    - Exam
Successful Software Development
When is Software Development Successful?

A software development project is successful if and only if developer, customer, and user are happy with the result at the end of the project.
Which Result? Which Project?

- **Successful:**

  - Time $t$: Customer (software) contract
  - Software!

  - Time $t' \geq t$: Developer (software) delivery

- **Unsuccessful:**

  - Time $t$: Customer (software) contract
  - Software!

  - Time $t' \geq t$: Developer (software) delivery

Does ’unsuccessful’ happen? If yes: How can we avoid it?
Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Softwareentwicklungsprojekten in Deutschland

2006

Autoren:
Ralf Buschemöhle
Heike Eekhoff
Bernhard Josko

Report: VSEK/55/D
Version: 1.1
Datum: 28.09.2006
Some Empirical Findings (Buschermöhle et al. (2006))

- Budget in € (378 responses):
  - 1-9,999: 32.56%
  - 10,000-99,999: 25.66%
  - 100,000-499,999: 29.1%
  - 500,000-999,999: 11.16%
  - ≥ 1,000,000: 0.93%
  - Not specified: 2.65%

- Planned duration in months (378 responses):
  - ≤ 3: 2.46%
  - > 3-6: 24.73%
  - > 6-12: 54.73%
  - > 12-24: 11.14%
  - > 24: 0.68%

- Criticality (378 responses, 30 'not spec. '):
  - Business critical: 82.61%
  - Mission critical: 17.39%
  - Safety critical: 0.00%

- Project completion (378 responses):
  - Completed: 97.35%
  - Cancelled: 2.65%

- Deadline (368 responses):
  - Kept: 72.01%
  - Early: 24.73%
  - Late: 3.26%

- Main functionality realised (368 responses):
  - Kept: 81.52%
  - Below: 11.14%
  - Above: 7.34%

- Secondary functionality realised (368 responses):
  - < 25%: 57.61%
  - 25-49%: 13.04%
  - 50-74%: 13.04%
  - 75-89%: 4.89%
  - 90-94%: 4.89%
  - 95-99%: 4.89%
  - 100%: 4.89%
Causes for Unsuccessful Projects: First Approximation

Possible causes (by phase):

1. Capturing Requirements
   - e.g. misunderstanding of requirements; contradicting requirements

2. Design
   - e.g. non-scalable design; feature forgotten; designer misunderstood requirement

3. Implementation
   - e.g. programmer misread design specification; simple programming mistake

4. Quality Assurance
   - e.g. wrongly conducted test; tester misunderstood requirement

5. Project Management
   - e.g. wrong cost estimation; bad scheduling; team member was not aware of responsibilities
### Causes for Unsuccessful Projects: Once Again

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

- e.g. **misunderstanding** of requirements; **contradicting** requirements
- e.g. **non-scalable** design; feature forgotten; designer **misunderstood** requirement
- e.g. programmer **misread** design specification; simple programming **mistake**
- e.g. **wrongly conducted** test; tester **misunderstood** requirement
- e.g. **wrong** cost estimation; bad scheduling; team member was **not aware of** responsibilities

And that’s this course:

- Discuss **typical Software-Engineering problems**,  
  - like **communication**, **misunderstandings**, etc.
  - like **technical errors**, **quality issues**, etc.

- and (state-of-the-art) **generic mitigation approaches**  
  - like **precise description languages** (e.g. for requirements),
  - like **analysis techniques** (e.g. for program correctness),
  - by development phase (Requirements, Design, etc.).
Content

- Terminology
  - Engineering, Software, Software Engineering

- Motivation: Successful Software Development
  - Working definition: success
  - Unsuccessful software development exists
  - Common reasons for non-success

- Course
  - Content
    - Topic areas
    - Structure of topic areas
    - Emphasis: formal methods
    - Relation to other courses
    - Literature
  - Organisation
    - Lectures
    - Tutorials
    - Exam
Course: Content
Course Content (Tentative)

- Capturing Requirements
- Design
- Implementation
- Code Quality Assurance

Software Project Management

<table>
<thead>
<tr>
<th>Course Topic</th>
<th>Dates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>22.4., Mon</td>
</tr>
<tr>
<td>L 1: Metrics, Costs, Development Process</td>
<td>25.4., Thu</td>
</tr>
<tr>
<td>L 2: Metrics, Costs, Development Process</td>
<td>29.4., Mon</td>
</tr>
<tr>
<td>L 3: Development Process</td>
<td>2.5., Thu</td>
</tr>
<tr>
<td>L 4: Development Process</td>
<td>6.5., Mon</td>
</tr>
<tr>
<td>T 1: Development Process</td>
<td>9.5., Thu</td>
</tr>
<tr>
<td>L 5: Requirements Engineering</td>
<td>13.5., Mon</td>
</tr>
<tr>
<td>L 6: Requirements Engineering</td>
<td>16.5., Thu</td>
</tr>
<tr>
<td>L 7: Requirements Engineering</td>
<td>20.5., Mon</td>
</tr>
<tr>
<td>T 2: Requirements Engineering</td>
<td>23.5., Thu</td>
</tr>
<tr>
<td>L 8: Requirements Engineering</td>
<td>27.5., Mon</td>
</tr>
<tr>
<td>L 9: Requirements Engineering</td>
<td>3.6., Mon</td>
</tr>
<tr>
<td>T 3: Requirements Engineering</td>
<td>6.6., Thu</td>
</tr>
<tr>
<td>T 4: Requirements Engineering</td>
<td>10.6., Mon</td>
</tr>
<tr>
<td>T 5: Requirements Engineering</td>
<td>13.6., Thu</td>
</tr>
<tr>
<td>T 6: Requirements Engineering</td>
<td>17.6., Mon</td>
</tr>
<tr>
<td>T 7: Requirements Engineering</td>
<td>20.6., Thu</td>
</tr>
<tr>
<td>L 10: Arch. &amp; Design, Modelling, Patterns, QA</td>
<td>24.6., Mon</td>
</tr>
<tr>
<td>T 8: Arch. &amp; Design, Modelling, Patterns, QA</td>
<td>27.6., Thu</td>
</tr>
<tr>
<td>L 11: Arch. &amp; Design, Modelling, Patterns, QA</td>
<td>1.7., Mon</td>
</tr>
<tr>
<td>T 9: Arch. &amp; Design, Modelling, Patterns, QA</td>
<td>4.6., Thu</td>
</tr>
<tr>
<td>L 12: Arch. &amp; Design, Modelling, Patterns, QA</td>
<td>8.7., Mon</td>
</tr>
<tr>
<td>T 10: Arch. &amp; Design, Modelling, Patterns, QA</td>
<td>11.7., Thu</td>
</tr>
<tr>
<td>(Testing, Formal Verification)</td>
<td>15.7., Mon</td>
</tr>
<tr>
<td>T 11: (Testing, Formal Verification)</td>
<td>18.7., Thu</td>
</tr>
<tr>
<td>L 13: (Testing, Formal Verification)</td>
<td>22.7., Mon</td>
</tr>
<tr>
<td>T 12: (Testing, Formal Verification)</td>
<td>25.7., Thu</td>
</tr>
</tbody>
</table>
Literature (Preview)
Structure of Topic Areas  (Example: Requirements Eng.)

Sommerville (2010), Balzert (2009), Ludewig and Lichter (2013), etc.:  

Vocabulary  
- e.g. consistent, complete, tacit, etc.  

Techniques  
- informal  
- e.g. Natural Language  
- e.g. Nat. Language Patterns  
- semi-formal  
- e.g. Use Cases  
- e.g. Use Case Diagrams  
- formal  
- e.g. Decision Tables  

This course:  

Vocabulary  

Techniques  
- informal  
- e.g. Natural Language  
- e.g. Nat. Language Patterns  
- semi-formal  
- e.g. Use Cases  
- e.g. Use Case Diagrams  
- formal  
- e.g. Decision Tables  
- simple  
- complex  
- e.g. Decision Tables (formal)  
- e.g. Live Sequence Charts (proper subset, formal)
**Excursion: Informal vs. Formal Techniques**

**Example**: Requirements Engineering, Airbag Controller

**Requirement specification, informal**: Whenever a crash is detected, the airbag has to be fired within 300 ms ($\pm \varepsilon$).

- 'within' means '$\leq$'; so 100 ms is okay, too
- 'within' means between $300 - \varepsilon$ and $300 + \varepsilon$

**Requirement specification, formal**:  
- Fix observables: $\text{crashdetected} : \text{Time} \to \{0, 1\}$ and $\text{fireairbag} : \text{Time} \to \{0, 1\}$
- Formalise requirement:

\[
\forall t, t' \in \text{Time} \quad \text{crashdetected}(t) \land \text{airbagfired}(t') \implies t' \in [t + 300 - \varepsilon, t + 300 + \varepsilon]
\]

→ no more misunderstandings, sometimes tools can objectively decide: requirement satisfied yes/no.
Sign says:
Welcome to **formal methods paradise**:  
- No more **misunderstandings**!  
- Let **tools** decide things **objectively**!
...more on the course homepage.
Content

- **Terminology**
  - Engineering, Software, Software Engineering

- **Motivation: Successful Software Development**
  - Working definition: success
  - Unsuccessful software development exists
  - Common reasons for non-success

- **Course**
  - **Content**
    - Topic areas
    - Structure of topic areas
    - Emphasis: formal methods
    - Relation to other courses
    - Literature
  - **Organisation**
    - Lectures
    - Tutorials
    - Exam
The lecturer points out connections to other topic areas (e.g. research, praxis).

- totally agree
- strongly disagree

- Optimisation
- Info I
- Info II
- Info III
- Logic
- Graph Theory
- Maths I
- Maths II
- Databases
- Networks
- Comp. Arch.
**Agreement** between ‘Fachschaft’ and the chair for software engineering: strong(er) coupling between both courses.

---

**Course Software-Engineering vs. Softwarepraktikum**

**Zeitplan**

<table>
<thead>
<tr>
<th>Woche</th>
<th>Organisation</th>
<th>Ort</th>
<th>Mon</th>
<th>Dien</th>
<th>Mitt</th>
<th>Don</th>
<th>Fre</th>
<th>War</th>
<th>Wann und Ort</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;Organisation und Prozess&quot;, (Einführungsvorlesung)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;Game Design Document (GDD)&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;Softwarearchitektur&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;Architektur von Videospielen&quot;, Ausarbeitung (beta)</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;Softwarearchitektur&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;ArchiObjectiver Gestalt&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;Softwarearchitektur&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;ArchiObjectiver Gestalt&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;Softwarearchitektur&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;ArchiObjectiver Gestalt&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;Softwarearchitektur&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;ArchiObjectiver Gestalt&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;Softwarearchitektur&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;ArchiObjectiver Gestalt&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;Softwarearchitektur&quot;, Vorlesung Hausaufgaben</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorlesung: &quot;ArchiObjectiver Gestalt&quot;, Vorlesung Hausaufgaben</td>
</tr>
</tbody>
</table>

---

**Introduction**

**Metrics, Costs, Development Process**

**Requirements Engineering**

**Arch. & Design**

**Software-**

**Modelling, Patterns**

**QA**

**(Testing, Formal Verification)**

**Wrap-Up**

---

- 22.4., Mon
- L 1: 25.4., Thu
- L 2: 29.4., Mon
- L 3: 2.5., Thu
- L 4: 6.5., Mon
- T 1: 9.5., Thu
- L 5: 13.5., Mon
- L 6: 16.5., Thu
- L 7: 20.5., Mon
- T 2: 23.5., Thu
- L 8: 27.5., Mon
- L 9: 30.5., Thu
- T 3: 3.6., Mon
- T 4: 6.6., Thu
- L 10: 17.6., Mon
- L 11: 24.6., Mon
- L 12: 1.7., Mon
- L 13: 4.6., Thu
- L 14: 8.7., Mon
- T 5: 11.7., Thu
- L 15: 15.7., Mon
- L 16: 18.7., Thu
- L 17: 22.7., Mon
- T 6: 25.7., Thu
Any Questions So Far?
Content

• Terminology
  • Engineering, Software, Software Engineering

• Motivation: Successful Software Development
  • Working definition: success
  • Unsuccessful software development exists
  • Common reasons for non-success

• Course
  • Content
    • Topic areas
    • Structure of topic areas
    • Emphasis: formal methods
    • Relation to other courses
    • Literature
  • Organisation
    • Lectures
    • Tutorials
    • Exam
Course: Organisation
Organisation: Lectures

- **Homepage:** [http://swt.informatik.uni-freiburg.de/teaching/SS2019/swtvl](http://swt.informatik.uni-freiburg.de/teaching/SS2019/swtvl)
- **Course language:** German (since we are in an odd year)
- **Script/Media:**
  - slides *without* annotations on homepage with beginning of lecture the latest
  - slides *with* annotations on homepage typically soon after the lecture
  - recording on ILIAS (stream and download) with max. 2 days delay (cf. link on homepage)
- **Schedule:** topic areas à three 90 min. lectures, one 90 min. tutorial (with exceptions)
- **Interaction:** absence often moaned; but it takes two, so please ask/comment immediately.
- **Questions/comments:**
  - “online”: ask immediately or in the break
  - “offline”:
    - (i) try to solve yourself
    - (ii) discuss with colleagues
    - (iii) a) **Exercises:** ILIAS (group) forum, contact tutor
    - b) **Everything else:** contact lecturer (cf. homepage)
       or just drop by: Building 52, Room 00-020

- **Break:** we’ll have a 5-10 min. break in the middle of each lecture (from now on), unless a majority objects now.
### Schedule/Submission:
- exercises **online** ([homepage](#)) and **ILIAS**) with first lecture of a block,
- **early submission** 24h before tutorial (usually Wednesday, 12:00, local time),
- **regular submission** right before tutorial (usually Thursday, 12:00, local time).
- please submit **electronically** via **ILIAS**
- should work in teams of **2-3 people**, clearly give **names** on submission

<table>
<thead>
<tr>
<th>Activity</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>22.4., Mon</td>
</tr>
<tr>
<td>Metrics, Costs, Development Process</td>
<td>25.4., Thu</td>
</tr>
<tr>
<td>L 1:</td>
<td>29.4., Mon</td>
</tr>
<tr>
<td>L 2:</td>
<td>2.5., Thu</td>
</tr>
<tr>
<td>L 3:</td>
<td>6.5., Mon</td>
</tr>
<tr>
<td>T 1:</td>
<td>9.5., Thu</td>
</tr>
<tr>
<td>L 5:</td>
<td>13.5., Mon</td>
</tr>
<tr>
<td>L 6:</td>
<td>16.5., Thu</td>
</tr>
<tr>
<td>L 7:</td>
<td>20.5., Mon</td>
</tr>
<tr>
<td>T 2:</td>
<td>23.5., Thu</td>
</tr>
<tr>
<td>L 8:</td>
<td>25.4., Mon</td>
</tr>
<tr>
<td>-</td>
<td>27.5., Mon</td>
</tr>
<tr>
<td>-</td>
<td>30.5., Thu</td>
</tr>
<tr>
<td>L 9:</td>
<td>3.6., Mon</td>
</tr>
<tr>
<td>T 3:</td>
<td>6.6., Thu</td>
</tr>
<tr>
<td>-</td>
<td>10.6., Mon</td>
</tr>
<tr>
<td>-</td>
<td>13.6., Thu</td>
</tr>
<tr>
<td>L10: Arch. &amp; Design,</td>
<td>17.6., Mon</td>
</tr>
<tr>
<td>-</td>
<td>20.6., Thu</td>
</tr>
<tr>
<td>L 11: Software-</td>
<td>24.6., Mon</td>
</tr>
<tr>
<td>T 4:</td>
<td>27.6., Thu</td>
</tr>
<tr>
<td>L 12: Modelling, Patterns</td>
<td>1.7., Mon</td>
</tr>
<tr>
<td>L 13:</td>
<td>4.6., Thu</td>
</tr>
<tr>
<td>L 14:</td>
<td>8.7., Mon</td>
</tr>
<tr>
<td>T 5:</td>
<td>11.7., Thu</td>
</tr>
<tr>
<td>L15: (Testing, Formal Verification)</td>
<td>15.7., Mon</td>
</tr>
<tr>
<td>L16:</td>
<td>18.7., Thu</td>
</tr>
<tr>
<td>L17:</td>
<td>22.7., Mon</td>
</tr>
<tr>
<td>T 6:</td>
<td>25.7., Thu</td>
</tr>
</tbody>
</table>
Schedule/Submission:

- exercises **online** ([homepage](#) and ILIAS) with first lecture of a block,
- **early submission** 24h before tutorial (usually Wednesday, 12:00, local time),
- **regular submission** right before tutorial (usually Thursday, 12:00, local time).
- please submit **electronically** via ILIAS
- should work in teams of **2–3 people**, clearly give **names** on submission

Grading system: **“most complicated grading system ever”**

- **Admission points** (good-will rating, upper bound)
  ("reasonable grading given student's knowledge before tutorial")
- **Exam-like points** (evil rating, lower bound)
  ("reasonable grading given student's knowledge after tutorial")

10% **bonus** for **early** submission.

Tutorial: **Four groups** (central assignment), hosted by tutor.

- Starting from discussion of the early submissions (anonymous), develop **one** good proposal together,
- **tutorial notes provided via ILIAS**.
Exam Admission:

Achieving 50% of the **regular admission points** of Exercise Sheets 0–3 and 50% of the **regular admission points** of Exercise Sheets 4–6 is sufficient for admission to exam.

5 + 15 regular admission points on sheets 0 and 1, and 20 regular admission points on exercise sheets 2–6

→ 120 **regular** admission points for 100%.

(plus plenty of **admission bonus points** in both blocks, 0–3 and 4–6)

Exam Form:

- **written** exam
- date, time, place: tba
- permitted exam aids: one A4 paper (max. 21 x 29.7 x 1 mm) of notes, max. two sides inscribed
- scores from the exercises **do not** contribute to the final grade.
- example exam available on **ILIAS**
One Last Word on The Exercises... 

- Every exercise task is a tiny little scientific work!

- Basic rule for high quality submissions:
  - **rephrase** the task in your own words,
  - **state** your solution,
  - **convince** yourself and your tutor of the correctness of your solution (at best: prove it).

**Example:**

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

**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).
One Last Word on The Exercises...

- Every exercise task is **a tiny little scientific work**!
- Basic rule for high quality submissions:
  - **rephrase** the task in your own words,
  - **state** your solution,
  - **convince** yourself and your tutor of the correctness of your solution (at best: prove it).

**Example:**

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

**Submission A:**

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

**Submission B:**

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).
One Last Word on The Exercises... 

- Every exercise task is **a tiny little scientific work!**

- Basic rule for high quality submissions:
  - **rephrase** the task in your own words,
  - **state** your solution,
  - **convince** yourself and your tutor of the correctness of your solution (at best: prove it).

---

**I have improved my skills in scientific problem solving.**

- totally agree
- strongly disagree

**I have improved my skills in scientific problem solving.**

- totally agree
- strongly disagree
Tell Them What You’ve Told Them…

- **Basic vocabulary:**
  - software, engineering, software engineering,
  - customer, developer, user,
  - successful software development

  → **note:** some definitions are neither formal nor universally agreed

- **(Fun) fact:** software development is not always successful

- **Basic activities of (software) engineering:**
  - gather requirements,
  - design,
  - implementation,
  - quality assurance,
  - project management

  → motivates content of the course – for the case of software

- **Formal (vs. informal) methods**
  - avoid misunderstandings,
  - enable objective, tool-based assessment

  → **note:** still, humans are at the heart of software engineering.

- **Course content and organisation**
Any (More) Questions?
References
References


