Content

- Software, Engineering, Software Engineering
- 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. […]

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

IEEE 24765 (2010)

Engineering vs. Non-Engineering

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

(Ludewig and Lichter, 2013)
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)

Software Engineering

multi-person Development of Multi-version Programs.

D. L. Parnas (2011)

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)

Here is no universally accepted definition of software engineering.

Institutions that teach software are responsible for producing professionals who will build and maintain systems to the satisfaction of their beneficiaries. This article presents some ideas on how but also raises issues.

I won’t settle on any of these definitions; rather, I’d like to accept that they are all in some way valid and retain all the views of software they encompass.
**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** — 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)


6.2 Reliability

The capability of the software product to maintain a specified level of performance when used under specified conditions.

6.2.2 Fault tolerance

The capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.

“software that is reliable and works efficiently” (Bauer, 1971)
"software that is reliable and works efficiently" (Bauer, 1971)


6.1 Functionality
The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.

6.1.1 Suitability
The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives.

**The course’s working definition of Software Engineering**

**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 – 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](#))
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.
Some Empirical Findings (Buschermöhle et al. (2006))
A Closer Look

- **Successful:**
  
  - Time $t$
    
    - Customer
    
    - Developer
    
    - Software contract
  
  →
  
  - Time $t' \geq t$
    
    - Customer
    
    - Developer
    
    - Software delivery

- **Unsuccessful:**
  
  - Time $t$
    
    - Customer
    
    - Developer
    
    - Software contract
    
  →
  
  - Time $t' \geq t$
    
    - Customer
    
    - Developer
    
    - Software delivery
    
    - Cross

What might’ve gone wrong?

Time $t$

- Capturing Requirements
- Design
- Implementation
- (Code) Quality Assurance
- (Software) Project Management

... 

Some scenarios:

1. $\times$ $\times$ $\times$ $\times$ e.g. misunderstanding of requirements
2. $\times$ $\times$ $\times$ $\times$ e.g. non-scalable design, feature forgotten
3. $\times$ $\times$ $\times$ $\times$ e.g. programming mistake
4. $\times$ $\times$ $\times$ $\times$ e.g. wrongly conducted test
5. $\times$ $\times$ $\times$ $\times$ e.g. wrong estimates, bad scheduling
All engineering disciplines face the same questions:

- How to describe requirements / avoid misunderstandings with the customer?
- How to describe design ideas / avoid misunderstandings with the implementers?
- How to ensure that the product is built right / that the right product is built? (→ How to measure the quality of the product?)
- How to schedule activities properly?

At best: are there procedures which promise to systematically avoid certain mistakes or costs?

This course is about Software Engineering, so we should discuss:

- How to describe requirements on software precisely?
- How to describe design ideas for software precisely?
- How to ensure that software is built right? (→ How to measure the quality of software?)
- How to schedule software development activities properly?

What are procedures to systematically avoid certain mistakes or costs in software development?

---

**Example: Nightly Builds**

**Scenario:**

- Program $P$ compiles successfully at time $t$.
- Programmers work for duration $d$ on $P$, yielding program $P'$ at time $t + d$.
- $P'$ does not compile at time $t + d$.

→ the reason for not compiling any more must be among the changes during $d$.

**Experience:**

- If $d$ is large, it can be very difficult (and time consuming) to identify the cause.

**Proposal:** “Nightly Builds”

- Set up a procedure, which (at best: automatically) tries to compile the current state of the development each day over night.
- Promise: with ‘nightly builds’, $d$ is effectively limited to be smaller or equal to one day, so the number of possible causes for not compiling should be manageable.

→ Software Engineering as a defensive discipline (measures against failures and “catastrophes”):

- if program $P$ always compiles, the effort for ‘nightly builds’ was strictly speaking wasted.
- if a compilation issue occurs during the project, the caused damage is bounded.

Same holds for documentation: if no maintenance is ever needed, documentation effort may be wasted.
All engineering disciplines face the same questions:

- How to **describe requirements** / **avoid misunderstandings** with the customer?
- How to **describe design ideas** / **avoid misunderstandings** with the implementers?
- How to **ensure** that the **product** is built **right** / that the **right product** is built?
  \(\rightarrow\) How to **measure** the quality of the product?
- How to **schedule activities** properly?

At best: are there procedures which promise to **systematically** avoid certain mistakes or costs?

This course is about **Software Engineering**, so we should discuss:

- How to **describe requirements** on **software** precisely?
- How to **describe design ideas for software** precisely?
- How to **ensure** that **software** is built **right**?
  \(\rightarrow\) How to **measure** the quality of **software**?
- How to **schedule software development activities** properly?

What are procedures to **systematically** avoid certain mistakes or costs in **software development**?

Software Engineering is a young discipline: **plenty of proposals** for each question.

So the course will **focus on the problems** and discuss **example proposals**.
**Course Content (Tentative)**

- **Capturing Requirements**
- **Design**
- **Implementation**
- **Code Quality Assurance**

**Software Project Management**

---

**Structure of Topic Areas**

**Example:** Requirements Engineering

**Vocabulary**

- **Techniques**
  - informal
  - semi-formal
  - formal

- e.g. consistent, complete, tacit, etc.
**Example:** Requirements Engineering, Airbag Controller

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

- **Developer A** means `$\leq$'; so 100 ms is okay, too.
- **Developer B** means between $300 - \varepsilon$ and $300 + \varepsilon$.

**VS.**
- Fix observables: $\text{crashdetected} : \text{Time} \rightarrow \{0, 1\}$ and $\text{fireairbag} : \text{Time} \rightarrow \{0, 1\}$
- Formalise requirement:
  \[
  \forall t, t' \in \text{Time} \cdot \text{crashdetected}(t) \land \text{airbagfired}(t') \implies t' \in [t + 300 - \varepsilon, t + 300 + \varepsilon]
  \]

\[
\rightarrow \text{no more misunderstandings, sometimes tools can objectively decide: requirement satisfied yes/no.}
\]
Structure of Topic Areas

Example: Requirements Engineering

<table>
<thead>
<tr>
<th>Vocabulary</th>
<th>e.g. consistent, complete, tacit, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Techniques</td>
<td>informal</td>
</tr>
<tr>
<td></td>
<td>semi-formal</td>
</tr>
<tr>
<td></td>
<td>formal</td>
</tr>
</tbody>
</table>

In the course:

- Use Cases
- Pattern Language
- Decision Tables
- Live Sequence Charts

- e.g. “Whenever a crash...”
- e.g. “Always, if ⟨crash⟩ at t...”
- e.g. “∀ t, t’ ∈ Time • ...”
Content

- **Software, Engineering, Software Engineering**

- **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
Agreement between 'Fachschaft' and the chair for software engineering: strong(er) coupling between both courses.
...more on the course homepage.

Any Questions So Far?
Course: Organisation

Content

- Software, Engineering, Software Engineering
- 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
**Organisation: Lectures**

- **Homepage:** [http://svt.informatik.uni-freiburg.de/teaching/SS2017/swtvl](http://svt.informatik.uni-freiburg.de/teaching/SS2017/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":
    1. try to solve yourself
    2. discuss with colleagues
  - 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**.

---

**Organisation: Exercises & Tutorials**

- **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; paper submissions are tolerated
  - should work in teams of **approx. 3**, 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")
  - **20% bonus for early submission.**

- **Tutorial:** **Three groups** (central assignment), hosted by tutor.
  - Starting from discussion of the early submissions (anonymous),
  - develop one good proposal together,
  - tutorial notes provided via ILIAS.

---

**Introduction Scales, Metrics**

- L 1: 24.4., Mon
- L 2: 27.4., Thu
- T 1: 1.5., Mon

**Costs and Development Process**

- L 3: 8.5., Mon
- L 4: 11.5., Thu
- L 5: 15.5., Mon
- T 2: 18.5., Thu
- L 6: 22.5., Mon
- T 3: 25.5., Thu
- L 7: 29.5., Mon
- L 8: 1.6., Thu
- T 4: 22.6., Thu

**Requirements Engineering**

- L 9: 19.6., Mon
- L 10: 22.6., Thu
- L 11: 26.6., Mon
- T 4: 29.6., Thu
- L 12: 3.7., Mon
- L 13: 6.7., Thu
- L 14: 10.7, Mon
- T 5: 13.7, Thu

**Arch & Design**

- L 15: 17.7., Mon
- L 16: 20.7., Thu
- L 17: 24.7., Mon
- L 18: 27.7., Thu

**Software Modelling**

- L 19: 30.7., Thu

**Patterns**

- L 20: 3.8., Mon
- L 21: 6.8., Mon

**QA (Testing, Formal Verif.)**

- L 22: 10.8., Mon
- L 23: 13.8., Mon

**Wrap-Up**

- L 24: 17.8, Mon

---
Exam Admission:

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

10 regular admission points on sheets 0 and 1, and 20 regular admission points on exercise sheets 2–6 → 120 regular admission points for 100%.

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 your tutor of (at best: prove) the correctness of your solution.
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


