*Softwaretechnik / Software-Engineering*

# *Lecture 1: Introduction*

*2017-04-24*

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

Albert-Ludwigs-Universität Freiburg, Germany

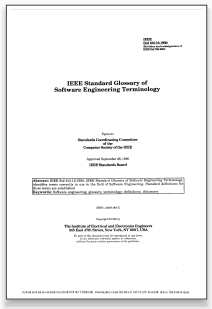
## *Content*





## *Software, Engineering, Software Engineering*





IEEE Standard Glossary of Software Engineering Terminology

Sponsor
Standards Coordinating Committee
of the
Computer Society of the IEEE

Approved September 28, 1990
IEEE Standards Board

Abstract: IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, contains definitions of terms relevant to the field of Software Engineering. Standard definitions for those terms are established.
Keywords: Software engineering, glossary, terminology, definitions, dictionary

ISBN 1-55937-067-X

Published by
The Institute of Electrical and Electronics Engineers
345 East 47th Street, New York, NY 10017, USA

INTERNATIONAL STANDARD

ISO/IEC/ IEEE 24765

Systems and software engineering — Vocabulary

Ingénierie des systèmes et du logiciel — Vocabulaire

ISO IEC

IEEE



**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*

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

(Ludewig and Lichter, 2013)

## 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 –**

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) **ISO/IEC/IEEE 24765 (2010)**

**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)**





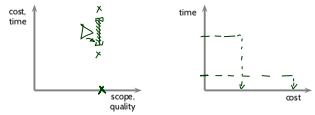

## The course's working definition of Software Engineering

**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)**

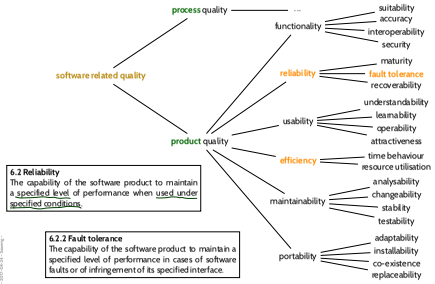
**Software Engineering** – the establishment and use of sound engineering principles to obtain economically software that is reliable and works efficiently on real machines. **F. L. Bauer (1971)**






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

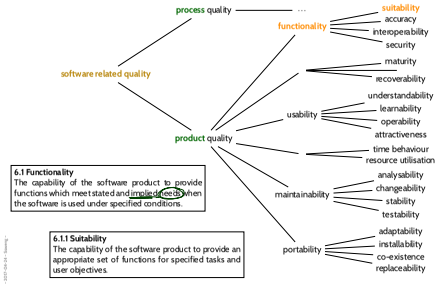
More general: **software of (good) quality** (cf. ISO/IEC 9126-1:2000 (2000))






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

More general: **software of (good) quality** (cf. ISO/IEC 9126-1:2000 (2000))






## 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)**

## Successful Software Development

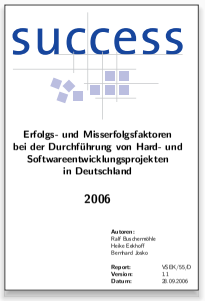


## When is Software Development Successful?


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.

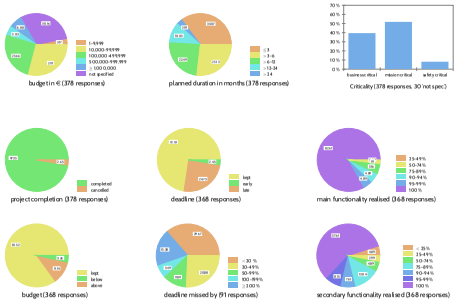


## Is Software Development Always Successful?





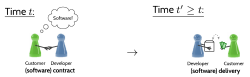

## Some Empirical Findings (Buschermöhle et al. (2006))






## A Closer Look

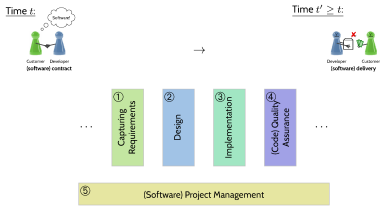
- **Successful:**




- **Unsuccessful:**


**What might've gone wrong?**






Some scenarios:

| ① | ② | ③ | ④ | ⑤ | |
|---|---|---|---|---|---|
| ✗ | ✓ | ✓ | ✓ | ✓ | e.g. misunderstanding of requirements |
| ✓ | ✗ | ✓ | ✓ | ✓ | e.g. non-scalable design, feature forgotten |
| ✓ | ✓ | ✗ | ✓ | ✓ | e.g. programming mistake |
| ✓ | ✓ | ✓ | ✗ | ✓ | e.g. wrongly conducted test |
| ✓ | ✓ | ✓ | ✓ | ✗ | e.g. wrong estimates, bad scheduling |

## In Other Words

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.



## In Other Words

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**?

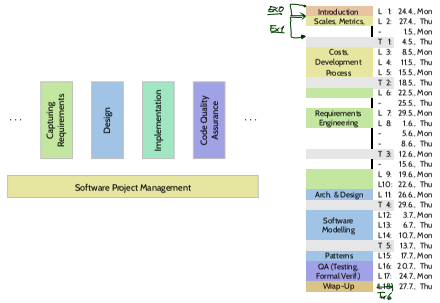
Software Engineering is a young discipline: **plenty of proposals** for each question.
So the course will **focus on the problems** and discuss **example proposals**.



## Course: Content



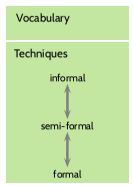
## Course Content (Tentative)






## Structure of Topic Areas

**Example**: Requirements Engineering




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

## Excursion: Informal vs. Formal Techniques

**Example**: Requirements Engineering, Airbag Controller

**Requirement**:

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


vs.

- Fix observables: crashdetected : Time $\to \{0,1\}$ and fireairbag : Time $\to \{0,1\}$
- Formalise requirement:

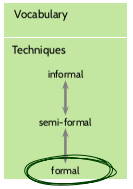
$$\forall t, t' \in \mathrm{Time} \bullet \mathrm{crashdetected}(t) \land \mathrm{airbagfired}(t') \implies t' \in [t+300-\varepsilon, t+300+\varepsilon]$$

→ no more misunderstandings, sometimes tools can objectively decide: requirement satisfied yes/no.




## Structure of Topic Areas

**Example**: Requirements Engineering

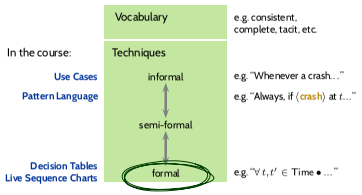



e.g. consistent, complete, tacit, etc.



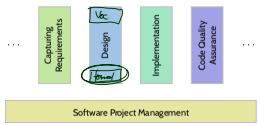
## Structure of Topic Areas

**Example**: Requirements Engineering






## Course Content (Tentative)



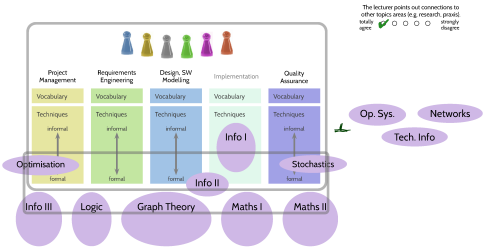

| | | | |
|---|---|---|---|
| Introduction | L 1: | 24.4., | Mon |
| Scales, Metrics, | L 2: | 27.4., | Thu |
| | - | 1.5., | Mon |
| | T 1: | 4.5., | Thu |
| Costs, | L 3: | 8.5., | Mon |
| Development | L 4: | 11.5., | Thu |
| Process | L 5: | 15.5., | Mon |
| | T 2: | 18.5., | Thu |
| | L 6: | 22.5., | Mon |
| | - | 25.5., | Thu |
| Requirements | L 7: | 29.5., | Mon |
| Engineering | L 8: | 1.6., | Thu |
| | - | 5.6., | Mon |
| | - | 8.6., | Thu |
| | T 3: | 12.6., | Mon |
| | - | 15.6., | Thu |
| | L 9: | 19.6., | Mon |
| | L10: | 22.6., | Thu |
| Arch. & Design | L 11 | 26.6., | Mon |
| | T 4: | 29.6., | Thu |
| Software | L12: | 3.7., | Mon |
| Modelling | L13: | 6.7., | Thu |
| | L14: | 10.7., | Mon |
| | T 5: | 13.7., | Thu |
| Patterns | L15: | 17.7., | Mon |
| QA (Testing, Formal Verif.) | L16: | 20.7., | Thu |
| | L17: | 24.7., | Mon |
| Wrap-Up | L18 | 27.7., | Thu |



## 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

## Course Software-Engineering vs. Other Courses



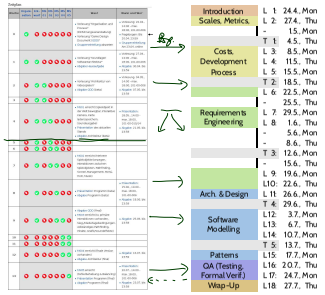



## Any Questions So Far?



## Course Software-Engineering vs. Softwarepraktikum

**Agreement** between 'Fachschaft' and the chair for software engineering:
**stronger] coupling between both courses**.




| | | | |
|---|---|---|---|
| L 1 | 24.4., Mon | T 3 | 12.6., Mon |
| L 2 | 27.4., Thu | - | 15.6., Thu |
| - | 1.5., Mon | L 9 | 19.6., Mon |
| T 1 | 4.5., Thu | L10 | 22.6., Thu |
| L 3 | 8.5., Mon | L 11 | 26.6., Mon |
| L 4 | 11.5., Thu | T 4 | 29.6., Thu |
| L 5 | 15.5., Mon | L12 | 3.7., Mon |
| T 2 | 18.5., Thu | L13 | 6.7., Thu |
| L 6 | 22.5., Mon | L14 | 10.7., Mon |
| - | 25.5., Thu | T 5 | 13.7., Thu |
| L 7 | 29.5., Mon | L15 | 17.7., Mon |
| L 8 | 1.6., Thu | L16 | 20.7., Thu |
| - | 5.6., Mon | L17 | 24.7., Mon |
| - | 8.6., Thu | L18 | 27.7., Thu |



## Course: Organisation



## Literature


…more on the course homepage.

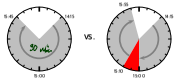


## Content

- **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://swt.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": (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.




vs.





## Organisation: Exercises & Tutorials

- **Schedule/Submission:**
  - exercises online (homepage and ILIAS) with first lecture of a block.
  - early submission 24 h 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 | |
| | - | 1.5., Mon | |
| | T 1 | 4.5., Thu | |
| Costs, 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 | |
| | - | 25.5., Thu | |
| Requirements Engineering | L 7 | 29.5., Mon | |
| | L 8 | 1.6., Thu | |
| | - | 5.6., Mon | |
| | - | 8.6., Thu | |
| | T 3 | 12.6., Mon | |
| | - | 15.6., Thu | |
| | L 9 | 19.6., Mon | |
| | L10 | 22.6., Thu | |
| Arch. & Design | L 11 | 26.6., Mon | |
| | T 4 | 29.6., Thu | |
| Software Modelling | L12 | 3.7., Mon | |
| | L13 | 6.7., Thu | |
| | L14 | 10.7., Mon | |
| | T 5 | 13.7., Thu | |
| Patterns | L15 | 17.7., Mon | |
| QA (Testing, Formal Verif.) | L16 | 20.7., Thu | |
| | L 17 | 24.7., Mon | |
| Wrap-Up | L18 | 27.7., Thu | |



## Organisation: Exam

- **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. . .


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

- 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*


Bauer, F. L. (1971). Software engineering. In *IFIP Congress (1)*, pages 530–538.

Buschermöhle, R., Eekhoff, H., and Josko, B. (2006). success – Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Softwareentwicklungsprojekten in Deutschland. Technical Report VSEK/55/D.

IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology*. Std 610.12-1990.

ISO/IEC FDIS (2000). *Information technology – Software product quality – Part 1: Quality model*. 9126-1:2000(E).

ISO/IEC/IEEE (2010). *Systems and software engineering – Vocabulary*. 24765:2010(E).

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Parnas, D. L. (2011). Software engineering: Multi-person development of multi-version programs. In Jones, C. B. et al., editors, *Dependable and Historic Computing*, volume 6875 of *LNCS*, pages 413–427. Springer.