Softwaretechnik / Software-Engineering

Lecture 1: Introduction

2016-04-18

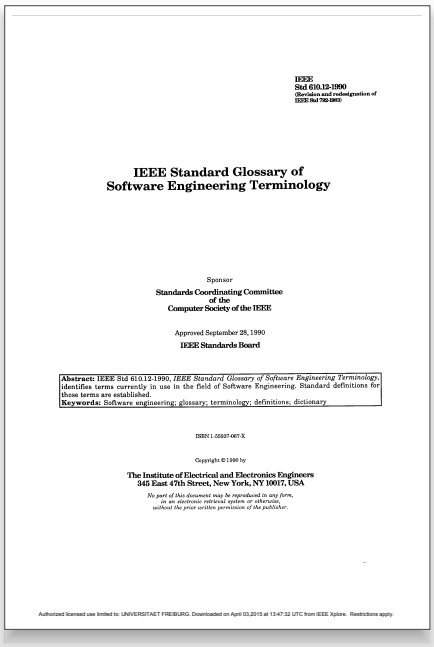
Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

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, Engineering, Software Engineering



IEEE
Std 610.12-1990
(Revision and redesignation of
IEEE Std 792-1983)

IEEE Standard Glossary of Software Engineering Terminology

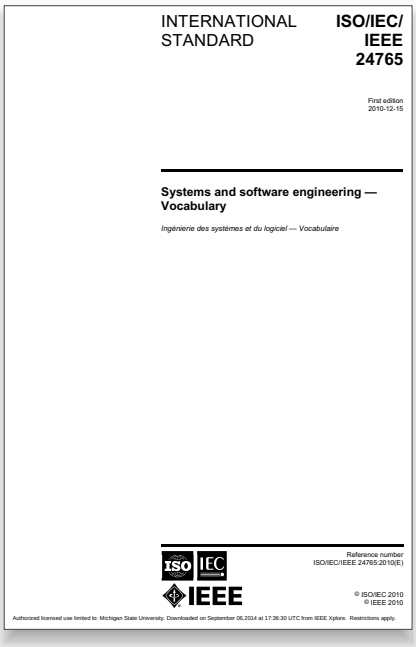
Sponsor
Standards Coordinating Committee
of the
Computer Society of the IEEE

Approved September 28, 1990
IEEE Standards Board

Abstract: IEEE Std 610.12-1990, *IEEE Standard Glossary of Software Engineering Terminology*, identifies terms currently in use in the field of Software Engineering. Standard definitions for those terms are established.
Keywords: Software engineering; glossary; terminology; definitions; dictionary

ISBN 1-55937-067-X

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



INTERNATIONAL STANDARD

ISO/IEC/ IEEE 24765

First edition
2010-12-15

Systems and software engineering — Vocabulary

Ingénierie des systèmes et du logiciel — Vocabulaire

Reference number
ISO/IEC/IEEE 24765:2010(E)

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	oriented on 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)

(ff informatik.uni-bremen.de/agbkb/2010/assets/images/)

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)

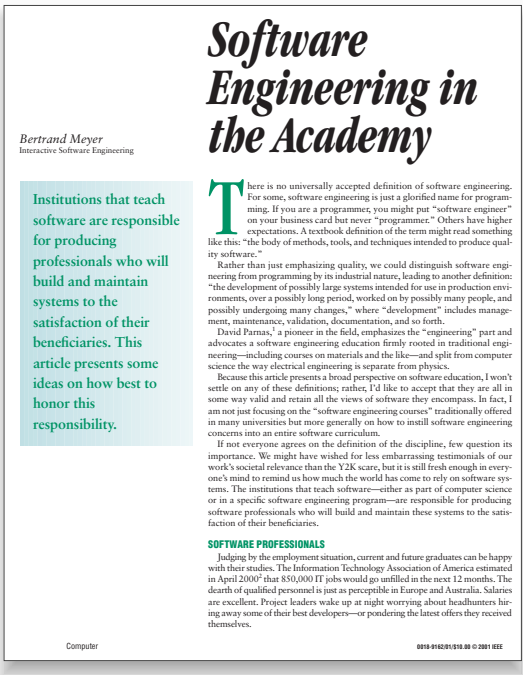
(commons.wikimedia.org (CC-by-sa 3.0))

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 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)**



Software Engineering in the Academy

Bertrand Meyer
Interactive 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 best to honor this responsibility.

There is no universally accepted definition of software engineering. For some, software engineering is just a glorified name for programming. If you are a programmer, you might put "software engineer" on your business card but never "programmer." Others have higher expectations. A textbook definition of the term might read something like this: "the body of methods, tools, and techniques intended to produce quality software."

Rather than just emphasizing quality, we could distinguish software engineering from programming by its industrial nature, leading to another definition: "the development of possibly large systems intended for use in production environments, over a possibly long period, worked on by possibly many people, and possibly undergoing many changes," where "development" includes management, maintenance, validation, documentation, and so forth.

David Parnas,[1] a pioneer in the field, emphasizes the "engineering" part and advocates a software engineering education firmly rooted in traditional engineering—including courses on materials and the like—and split from computer science the way electrical engineering is separate from physics.

Because this article presents a broad perspective on software education, 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. In fact, I am not just focusing on the "software engineering courses" traditionally offered in many universities but more generally on how to instill software engineering concerns into an entire software curriculum.

If not everyone agrees on the definition of the discipline, few question its importance. We might have wished for less embarrassing testimonials of our work's societal relevance than the Y2K scare, but it is still fresh enough in everyone's mind to remind us how much the world has come to rely on software systems. The institutions that teach software—either as part of computer science or in a specific software engineering program—are responsible for producing software professionals who will build and maintain these systems to the satisfaction of their beneficiaries.

SOFTWARE PROFESSIONALS

Judging by the employment situation, current and future graduates can be happy with their studies. The Information Technology Association of America estimated in April 2000[2] that 850,000 IT jobs would go unfilled in the next 12 months. The dearth of qualified personnel is just as perceptible in Europe and Australia. Salaries are excellent. Project leaders wake up at night worrying about headhunters hiring away some of their best developers—or pondering the latest offers they received themselves.

Computer

0018-9162/01/$10.00 © 2001 IEEE

- 1 - 2016-04-18 - Sswerg -

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 610.12 (1). **ISO/IEC/IEEE 24765** (2010)

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

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 best to honor this responsibility.

There is no universally accepted definition of software engineering. For some, software engineering is just a glorified name for programming. If you are a programmer, you might put "software engineer" on your business card but never "programmer." Others have higher expectations. A textbook definition of the term might read something like this: "the body of methods, tools, and techniques intended to produce quality software."

Rather than just emphasizing quality, we could distinguish software engineering from programming by its industrial nature, leading to another definition: "the development of possibly large systems intended for use in production environments, over a possibly long period, worked on by possibly many people, and possibly undergoing many changes," where "development" includes management, maintenance, validation, documentation, and so forth.

David Parnas,[1] a pioneer in the field, emphasizes the "engineering" part and advocates a software engineering education firmly rooted in traditional engineering—including courses on materials and the like—and split from computer science the way electrical engineering is separate from physics.

Because this article presents a broad perspective on software education, 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. In fact, I am not just focusing on the "software engineering courses" traditionally offered in many universities but more generally on how to instill software engineering

in April 2000[2] that 850,000 IT jobs would go unfilled in the next 12 months. The dearth of qualified personnel is just as perceptible in Europe and Australia. Salaries are excellent. Project leaders wake up at night worrying about headhunters hiring away some of their best developers—or pondering the latest offers they received themselves.

Computer

0018-9162/01/$10.00 © 2001 IEEE

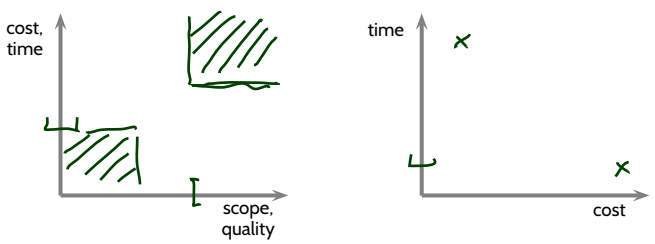
There is no universally accepted definition of software engineering.

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.

The course's working definition of Software Engineering

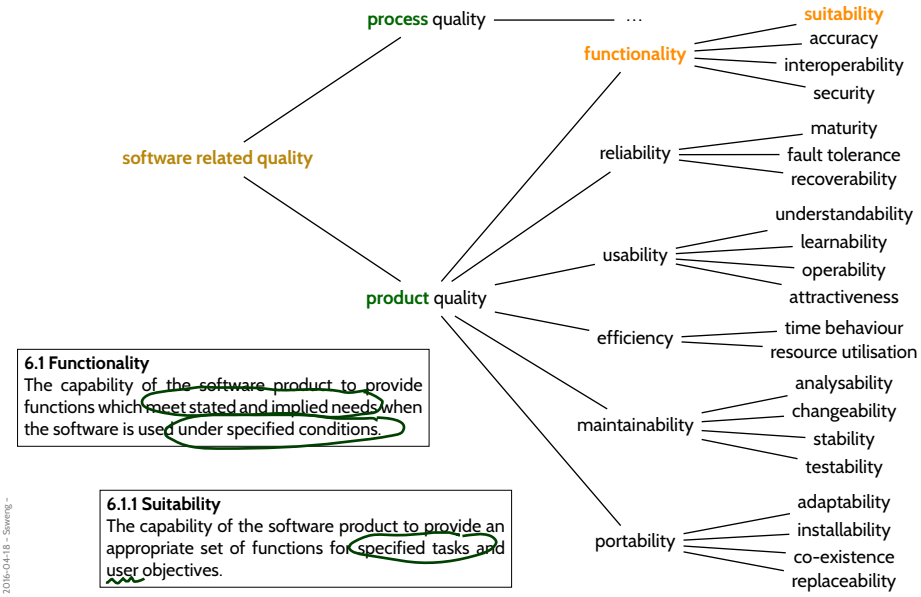
Software Engineering –
(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
(2) The study of approaches as in (1). **IEEE 610.12** (1990)

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)



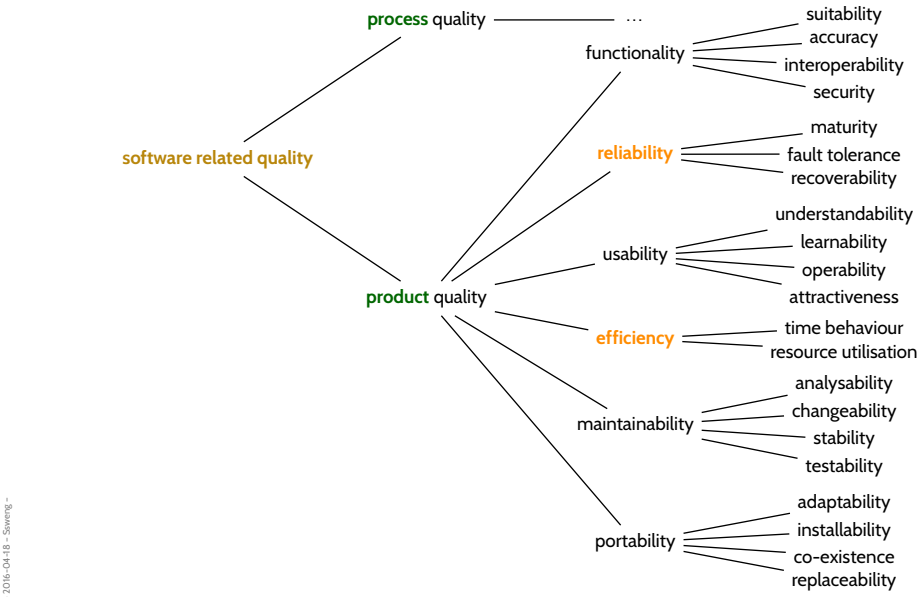
"software that is reliable and works efficiently" (Bauer, 1971)

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



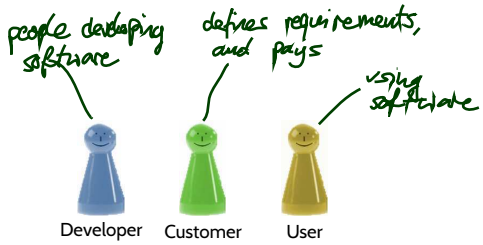
"software that is reliable and works efficiently" (Bauer, 1971)

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



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?


success

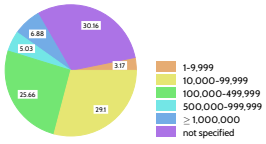
Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Softwareentwicklungsprojekten in Deutschland

2006

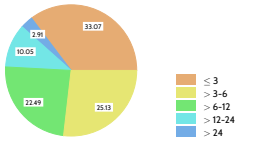
Autoren:
Ralf Buschermöhle
Heike Eekhoff
Bernhard Josko

Report: VSEK/55/D
Version: 1.1
Datum: 28.09.2006

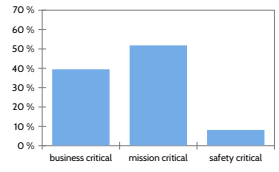
Some Empirical Findings (Buschermöhle et al. (2006))



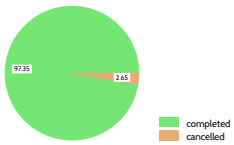
budget in € (378 responses)



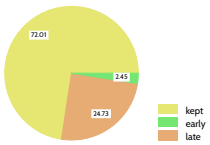
planned duration in months (378 responses)



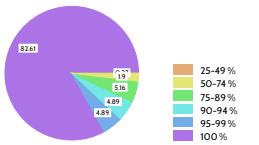
Criticality (378 responses, 30 'not spec.')



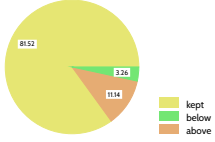
project completion (378 responses)



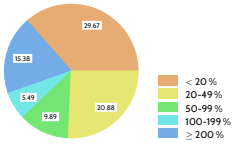
deadline (368 responses)



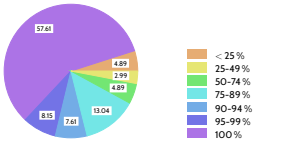
main functionality realised (368 responses)



budget (368 responses)



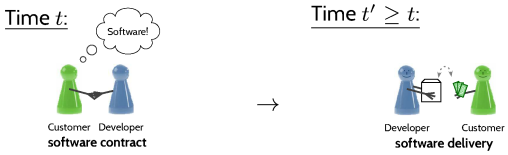
deadline missed by (91 responses)



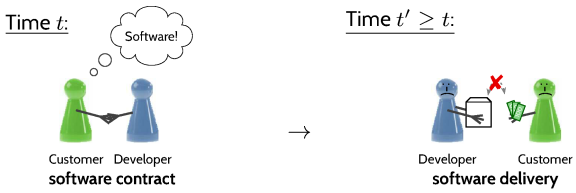
secondary functionality realised (368 responses)

A Closer Look

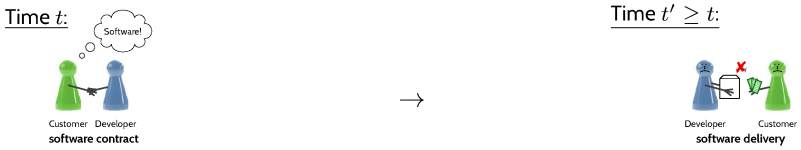
- **Successful:**

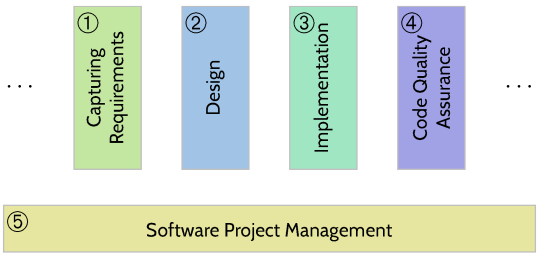


- **Unsuccessful:**



What might've gone wrong?



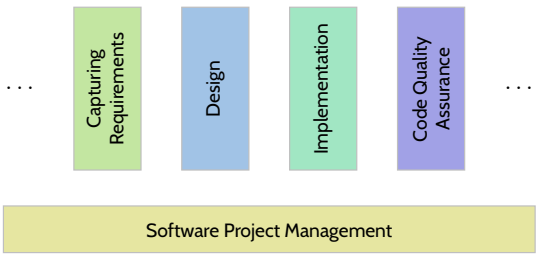


Some scenarios:

①	②	③	④	⑤	
✘	✔	✔	✔	✔	e.g. misunderstanding of requirements
✔	✘	✔	✔	✔	e.g. non-scalable design
✔	✔	✘	✔	✔	e.g. programming mistake
✔	✔	✔	✘	✔	e.g. wrongly conducted test
✔	✔	✔	✔	✘	e.g. wrong estimates

Course: Content

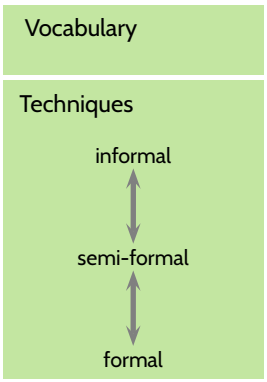
Course Content



Introduction	L 1:	18.4., Mon
Scales, Metrics, Costs	L 2:	21.4., Thu
	L 3:	25.4., Mon
	T 1:	28.4., Thu
Development	L 4:	2.5., Mon
	-	5.5., Thu
Process	L 5:	9.5., Mon
	L 6:	12.5., Thu
	-	16.5., Mon
	-	19.5., Thu
	T 2:	23.5., Mon
	-	26.5., Thu
Requirements Engineering	L 7:	30.5., Mon
	L 8:	2.6., Thu
	L 9:	6.6., Mon
	T 3:	9.6., Thu
Architecture & Design	L10:	13.6., Mon
	L 11:	16.6., Thu
	L12:	20.6., Mon
	T 4:	23.6., Thu
Software Modelling	L13:	27.6., Mon
	L14:	30.6., Thu
	L15:	4.7., Mon
	T 5:	7.7., Thu
Quality Assurance (Testing, Formal Verification)	L16:	11.7., Mon
	L17:	14.7., Thu
	L18:	18.7., Mon
Wrap-Up	L19:	21.7., Thu

Structure of Topic Areas

Example: Requirements Engineering



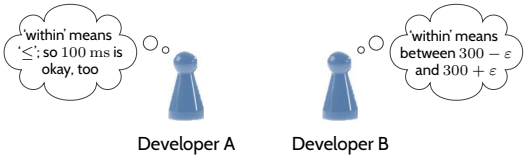
e.g. consistent, complete, tacit, etc.

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:

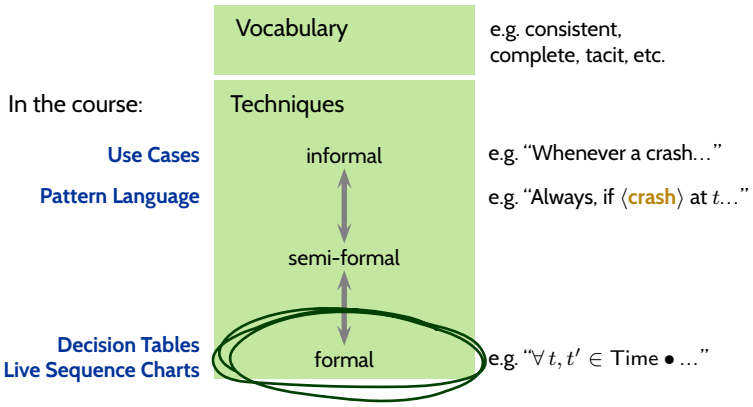
$$\forall\, t, t' \in \mathrm{Time} \bullet \mathrm{crashdetected}(t) \wedge \mathrm{airbagfired}(t') \implies t' \in [t + 300 - \varepsilon, t + 300 + \varepsilon]$$

($\leq$)

$\to$ no more misunderstandings, sometimes **tools** can **objectively** decide: requirement satisfied yes/no.

Structure of Topic Areas

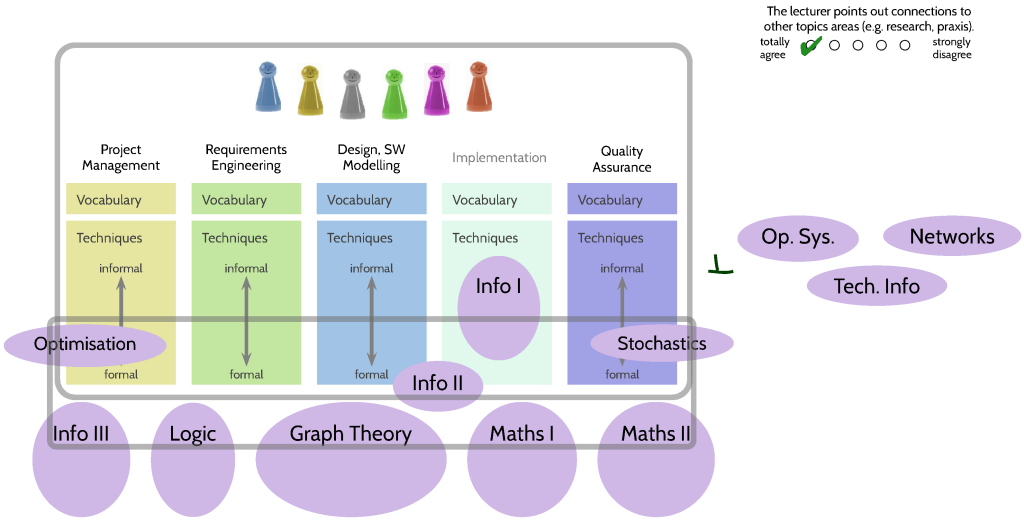
Example: Requirements Engineering



Content

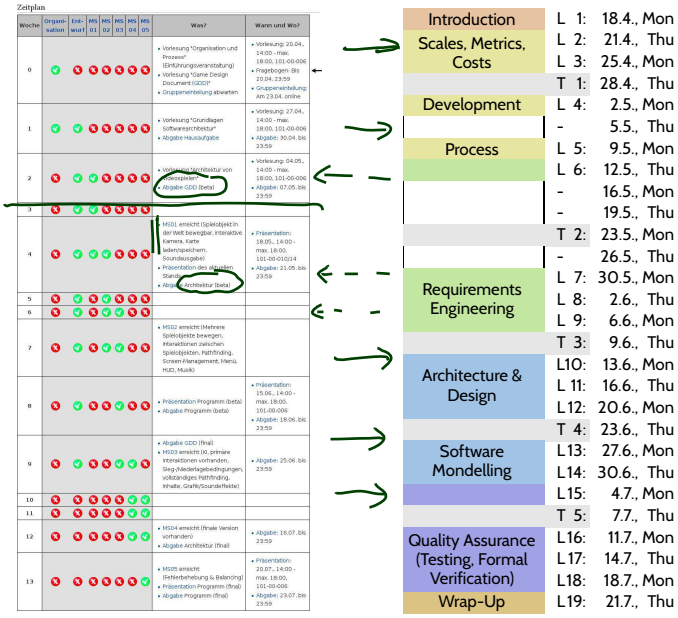
- **Software, Engineering, Software Engineering**
- **Successful Software Development**
  - working definition: success
  - unsuccessful software development exists
  - common reasons for non-success
- **Course**
  - Content
    - topic areas
    - structure of topic areas
    - emphasis: formal methods
    - relation to other courses
    - literature
  - Organisation
    - lectures
    - tutorials
    - exam

Course Software-Engineering vs. Other Courses



Course Software-Engineering vs. Softwarepraktikum

On popular demand, the chair for software engineering **agreed** on: strong(er) **coupling between both courses**.



Literature

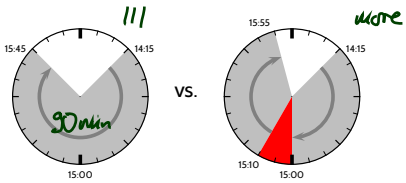
…more on the course homepage.

Any Questions So Far?

Course: Organisation

Organisation: Lectures

- **Homepage:** `http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl`
- **Course language:** English (since we are in an even 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.



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")

  ~~10%~~ 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	L 1:	18.4., Mon
Scales, Metrics, Costs	L 2:	21.4., Thu
	L 3:	25.4., Mon
	T 1:	28.4., Thu
Development	L 4:	2.5., Mon
	-	5.5., Thu
Process	L 5:	9.5., Mon
	L 6:	12.5., Thu
	-	16.5., Mon
	-	19.5., Thu
	T 2:	23.5., Mon
	-	26.5., Thu
Requirements Engineering	L 7:	30.5., Mon
	L 8:	2.6., Thu
	L 9:	6.6., Mon
	T 3:	9.6., Thu
Architecture & Design	L10:	13.6., Mon
	L 11:	16.6., Thu
	L12:	20.6., Mon
	T 4:	23.6., Thu
Software Mondelling	L13:	27.6., Mon
	L14:	30.6., Thu
	L15:	4.7., Mon
	T 5:	7.7., Thu
Quality Assurance (Testing, Formal Verification)	L16:	11.7., Mon
	L17:	14.7., Thu
	L18:	18.7., Mon
Wrap-Up	L19:	21.7., Thu

Organisation: Exam

- **Exam Admission:**

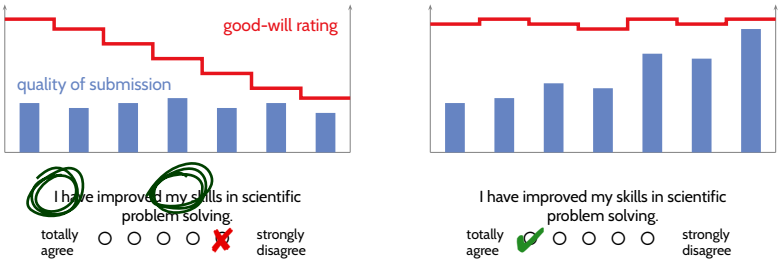
  Achieving 50% of the **regular admission points** in total is sufficient for admission to exam.

  20 regular admission points on exercise sheets 1–6, and 10 regular admission points on sheets 0 and 7

  → 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: in many cases, 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
- **Formal (vs. informal) methods**
  - avoid misunderstandings,
  - enable objective, tool-based assessment

  → note: humans are at the heart of software engineering.
- **Course content and organisation**

Any (More) Questions?

References

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.