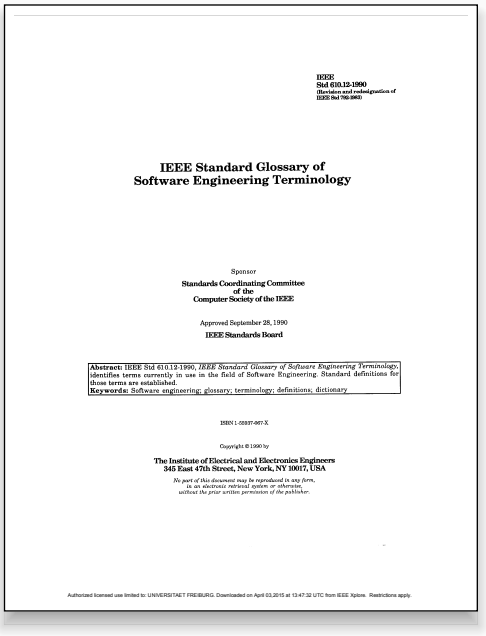
Softwaretechnik / Software-Engineering

Lecture 1: Introduction

2015-04-20

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

Albert-Ludwigs-Universität Freiburg, Germany



IEEE
Std 610.12-1990
(Revision and redesignation of
IEEE Std 729-1983)

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

– 1 – 2015-04-20 – main –

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 Engineering: Multi-person Development of Multi-version Programs.

D. L. Parnas (2011)

software engineering — **1.** the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software. **2.** the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

ISO/IEC/IEEE 24765 (2010)

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.

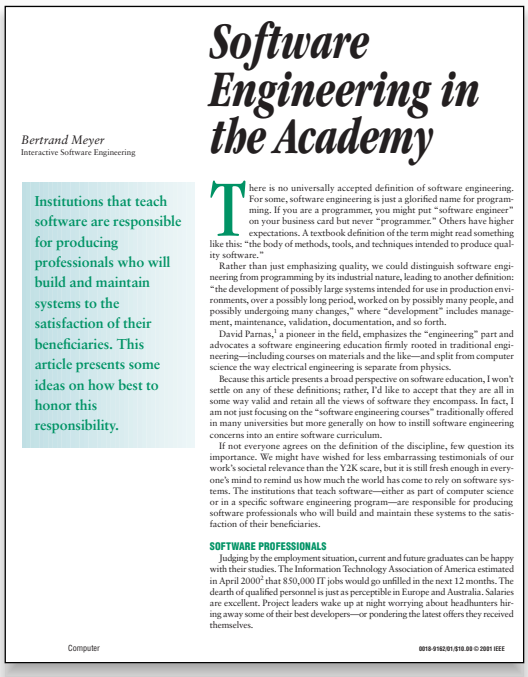
F. L. Bauer (1971)

Software Engineering: Multi-person Development of Multi-version Programs.

D. L. Parnas (2011)

software engineering — **1.** the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software. **2.** the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

ISO/IEC/IEEE 24765 (2010)



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

software engineering — (1) The application of a systematic, disciplined, quantifiable approach to the develop
ware; tha
ware. (2

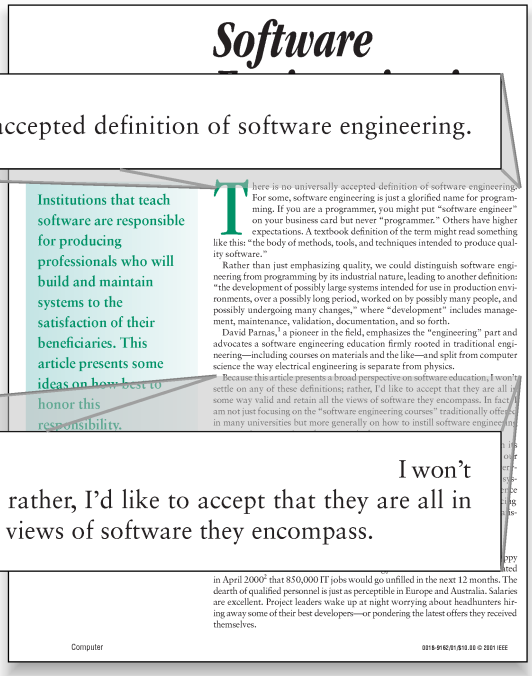
Software engineering — the establishment and use of sound engineering principles to obtain economically software that is reliable and works efficiently on real machines.
F. L. Bauer (1971)

Software Engineering: Multi-person Development of Multi-version Programs.
D. L. Parnas (2011)

softwar
plication
methods
tation, t
the appli
able app
maintenance of software; that is, the application of engineering to software.
ISO/IEC/IEEE 24765 (2010)

here 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.


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

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)

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 subjectively possible, 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)

The course's working definition of Software Engineering

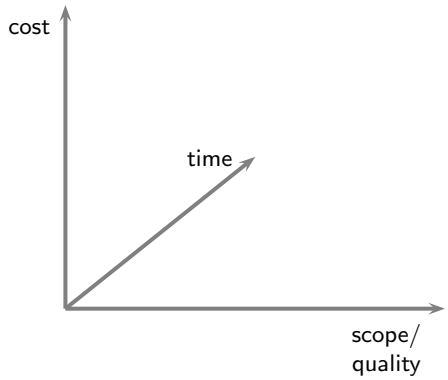
software engineering — (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
(2) The study of approaches as in (1).

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)

"obtain economically" (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).

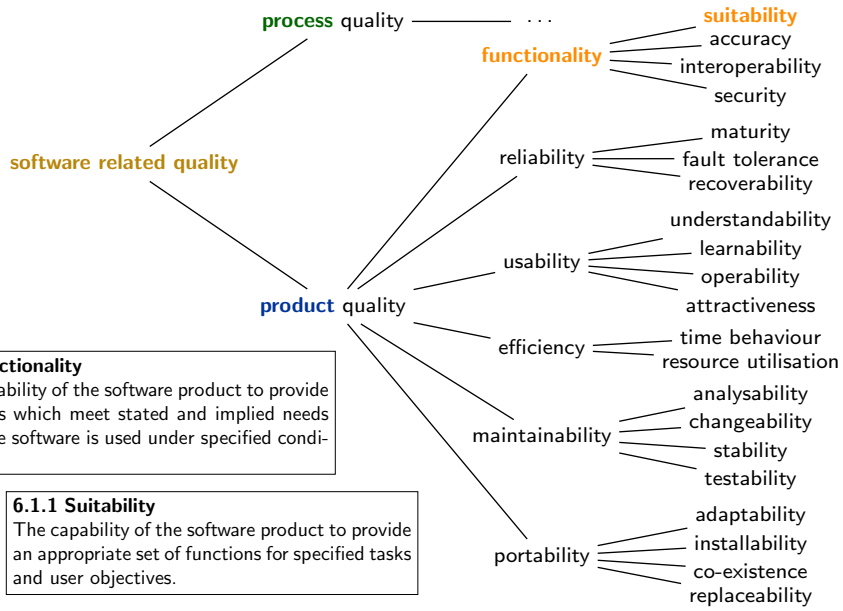
IEEE 610.12 (1990)

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))

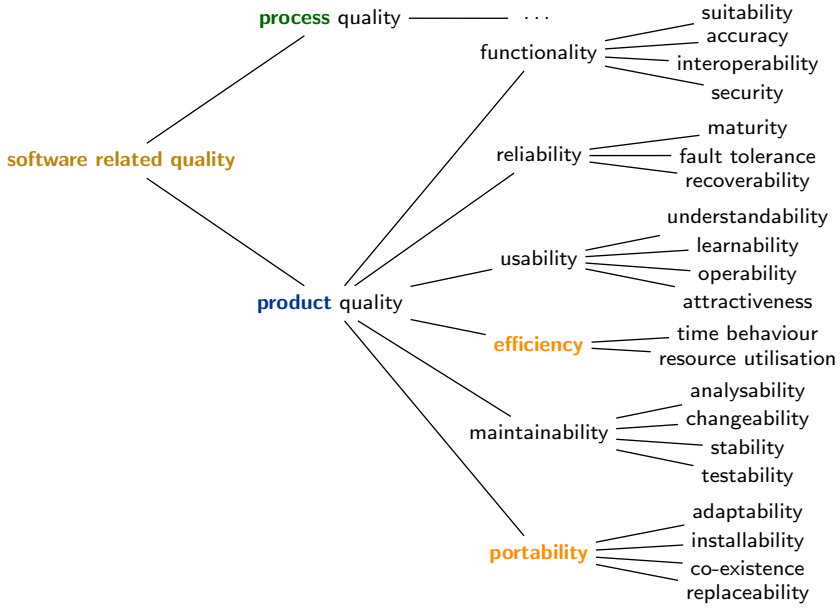


6.1 Functionality
The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.

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

"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)

"software"

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)

Note: not all software created in a software project is visible in the final product, e.g. **build scripts**, **test drivers**, **stubs**, etc.

Some Empirical Findings


success

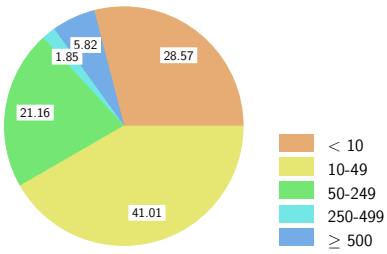
Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Softwareentwicklungsprojekten in Deutschland

2006

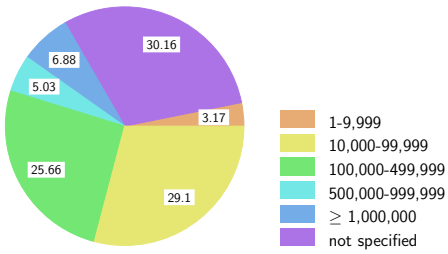
Autoren:
Ralf Buschermöhle
Heike Eekhoff
Bernhard Josko

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

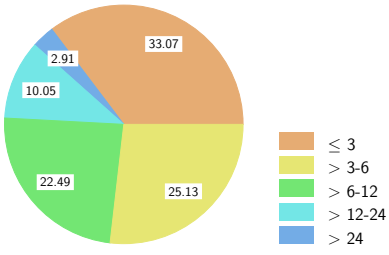
Characteristics of Software Projects in SUCCESS



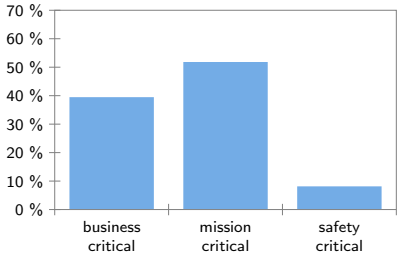
employees in company (378 responses)



budget in € (378 responses)

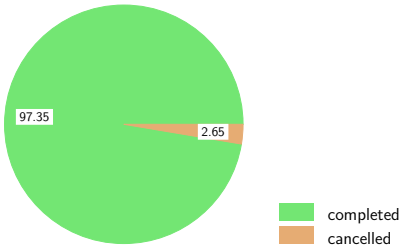


planned duration in months (378 responses)

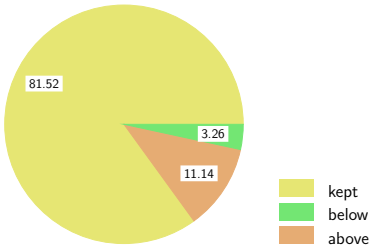


Criticality (378 responses, 30 'not spec.')

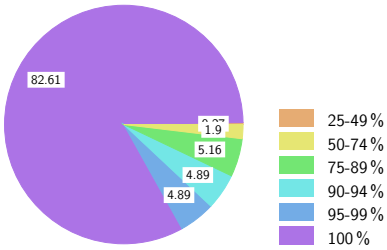
Projectsuccess, Budget, Functionality



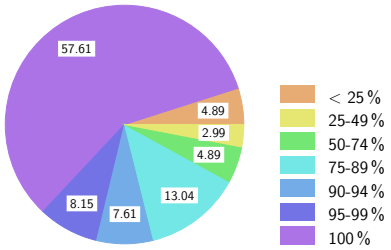
project completion (378 responses)



budget (368 responses)

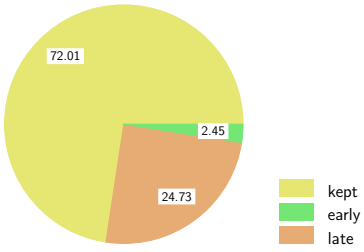


main functionality realised (368 responses)

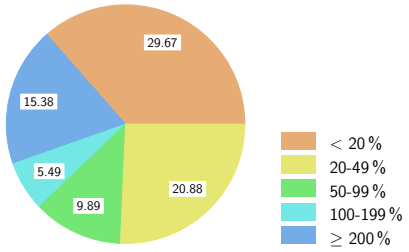


secondary functionality realised (368 responses)

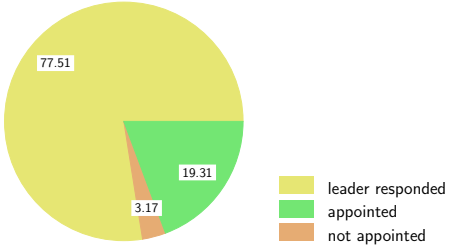
Deadlines, Project Leader, Process Model



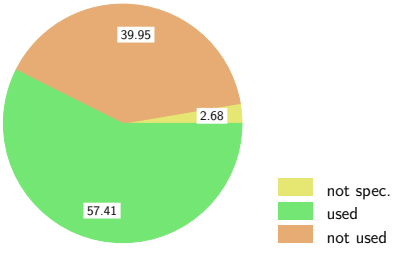
deadline (368 responses)



deadline missed by (91 responses)



existence of project leader (378 responses)



use of process model (378 responses)

Course Goals and Content

Course Goals and Content

- **First of all**:
  - communicate/cooperate with "real" software engineers
  - enable further study of today's software engineering research
- **To this end**:
  - provide a broad overview over software engineering research
  - point out areas, landmarks and elaborate example techniques/formalisms/tools
- ... with an emphasis on **formal methods**

Introduction	L 1:	20.4., Mo
	T 1:	23.4., Do
Development Process, Metrics	L 2:	27.4., Mo
	L 3:	30.4., Do
	L 4:	4.5., Mo
	T 2:	7.5., Do
Requirements Engineering	L 5:	11.5., Mo
	-	14.5., Do
	L 6:	18.5., Mo
	L 7:	21.5., Do
	-	25.5., Mo
	-	28.5., Do
	T 3:	1.6., Mo
Design Modelling & Analysis	-	4.6., Do
	L 8:	8.6., Mo
	L 9:	11.6., Do
	L 10:	15.6., Mo
	T 4:	18.6., Do
Implementation, Testing	L 11:	22.6., Mo
	L 12:	25.6., Do
	L 13:	29.6., Mo
	T 5:	2.7., Do
Formal Verification	L 14:	6.7., Mo
	L 15:	9.7., Do
	L 16:	13.7., Mo
	T 6:	16.7., Do
The Rest	L 17:	20.7., Mo
	L 18:	23.7., Do

Course Goals and Content

- **First of all**:
  - communicate/cooperate with "real" software engineers
  - enable further study of today's software engineering research
- **To this end**:
  - provide a broad overview over software engineering research
  - point out areas, landmarks and e
    example techniques/formalisms/
- ... with an emphasis on **formal m**

Example "Requirements Engineering":

- introduction to RE
- common notions, problems, goals, approaches (informal, abstract)
- formalisation and formal analysis of requirements (formal, concrete)
- point out further reading

Introduction	L 1:	20.4., Mo
	T 1:	23.4., Do
Development Process, Metrics	L 2:	27.4., Mo
	L 3:	30.4., Do
	L 4:	4.5., Mo
	T 2:	7.5., Do
Requirements Engineering	L 5:	11.5., Mo
	-	14.5., Do
	L 6:	18.5., Mo
	L 7:	21.5., Do
	-	25.5., Mo
Verification	T 6:	16.7., Do
The Rest	L 17:	20.7., Mo
	L 18:	23.7., Do

Course Goals and Content

- **First of all**:
  - communicate/cooperate with "real" software engineers
  - enable further study of today's software engineering research
- **To this end**:
  - provide a broad overview over software engineering research
  - point out areas, landmarks and elaborate example techniques/formalisms/tools
- ... with an emphasis on **formal methods**

Introduction	L 1:	20.4., Mo
	T 1:	23.4., Do
Development Process, Metrics	L 2:	27.4., Mo
	L 3:	30.4., Do
	L 4:	4.5., Mo
	T 2:	7.5., Do
Requirements Engineering	L 5:	11.5., Mo
	-	14.5., Do
	L 6:	18.5., Mo
	L 7:	21.5., Do
	-	25.5., Mo
	-	28.5., Do
	T 3:	1.6., Mo
Design Modelling & Analysis	-	4.6., Do
	L 8:	8.6., Mo
	L 9:	11.6., Do
	L 10:	15.6., Mo
	T 4:	18.6., Do
Implementation, Testing	L 11:	22.6., Mo
	L 12:	25.6., Do
	L 13:	29.6., Mo
	T 5:	2.7., Do
Formal Verification	L 14:	6.7., Mo
	L 15:	9.7., Do
	L 16:	13.7., Mo
	T 6:	16.7., Do
The Rest	L 17:	20.7., Mo
	L 18:	23.7., Do

– 1 – 2015-04-20 – main –

A Glimpse of Formal Methods

Formal Methods (in the Software Development Domain)

... back to "'technological paradise' where 'no acts of God can be permitted' and everything happens according to the blueprints".

(Kopetz, 2011; Lovins and Lovins, 2001)

Definition. [Bjørner and Havelund (2014)]
A method is called **formal method** if and only if its techniques and tools can be explained in mathematics.

Example: If a method includes, as a tool, a specification language, then that language has

- a **formal syntax**,
- a **formal semantics**, and
- a **formal proof system**. (at best)

Formal, Rigorous, or Systematic Development

"The **techniques** of a formal method help

- **construct** a specification, and/or
- **analyse** a specification, and/or
- **transform** (**refine**) one (or more) specification(s) into a **program**.

The **techniques** of a formal method, (besides the specification languages) are typically software packages that help developers use the techniques and other tools.

The aim of developing software, either

- **formally** (all arguments are formal) or
- **rigorously** (some arguments are made and they are formal) or
- **systematically** (some arguments are made on a form that can be made formal)

is to (be able to) **reason in a precise manner about properties** of what is being developed." (Bjørner and Havelund, 2014)

Software, formally

Definition. Software is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) computation paths of the form

$$\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$$

where

- $\sigma_i \in \Sigma$, $i \in \mathbb{N}_0$, is called **state** (or **configuration**), and
- $\alpha_i \in A$, $i \in \mathbb{N}_0$, is called **action** (or **event**).

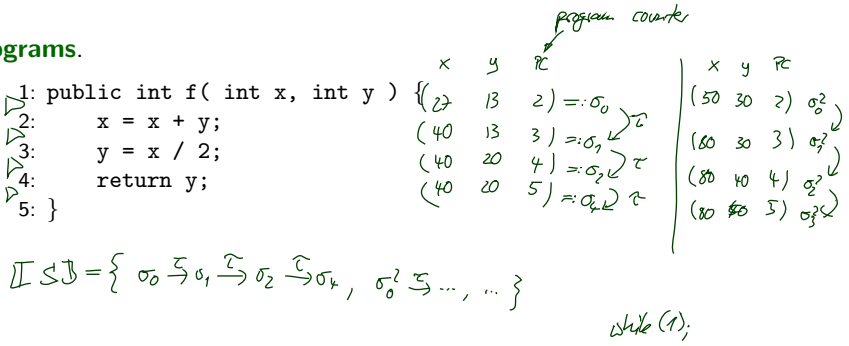
The (possibly partial) function $[\![\cdot]\!] : S \mapsto [\![S]\!]$ is called **interpretation** of S.

Example: Software, formally

Software is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) computation paths of the form $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$.
σ_i: **state**/**configuration**; α_i: **action**/**event**.

- **Programs**.

```
1: public int f( int x, int y ) {
2:     x = x + y;
3:     y = x / 2;
4:     return y;
5: }
```



Example: Software, formally

Software is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) computation paths of the form $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$.
σ_i: **state**/**configuration**; α_i: **action**/**event**.

- **Programs**.
- **HTML**.

```
1: <html>
2: <head>
3: <title>SWT 2015</title>
4: </head>
5: <body/>
6: </html>
```

Example: Software, formally

Software is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) computation paths of the form $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$.
σ_i: **state**/**configuration**; α_i: **action**/**event**.

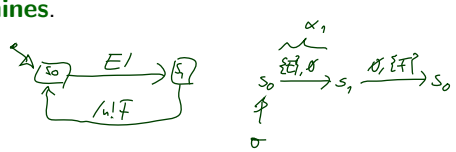
- **Programs**.
- **HTML**.
- **Global Invariants**.

  $x \geq 0$

Example: Software, formally

Software is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) computation paths of the form $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$.
σ_i: **state**/**configuration**; α_i: **action**/**event**.

- **Programs**.
- **HTML**.
- **Global Invariants**.
- **State Machines**.



– 1 – 2015-04-20 – main –
23/37

Example: Software, formally

Software is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) computation paths of the form $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$.
σ_i: **state**/**configuration**; α_i: **action**/**event**.

- **Programs**.
- **HTML**.
- **Global Invariants**.
- **State Machines**.
- **User's Manual.**

Software Specification, formally

Definition. A **software specification** is a finite description $\mathscr{S}$ of a (possibly infinite) set $[\![\mathscr{S}]\!]$ of softwares, i.e.

$$[\![\mathscr{S}]\!] = \{(S_1, [\![\cdot]\!]_1), \ldots\}.$$

The (possibly partial) function $[\![\cdot]\!] : \mathscr{S} \mapsto [\![\mathscr{S}]\!]$ is called **interpretation** of $\mathscr{S}$.

Example: Software Specification

Alphabet:

- M – dispense cash only,
- C – return card only,
- $\begin{matrix} M \\ C \end{matrix}$ – dispense cash and return card.

- **Customer 1** "don't care"

$$\left(M.C \middle| C.M \middle| \begin{matrix} M \\ C \end{matrix}\right)$$

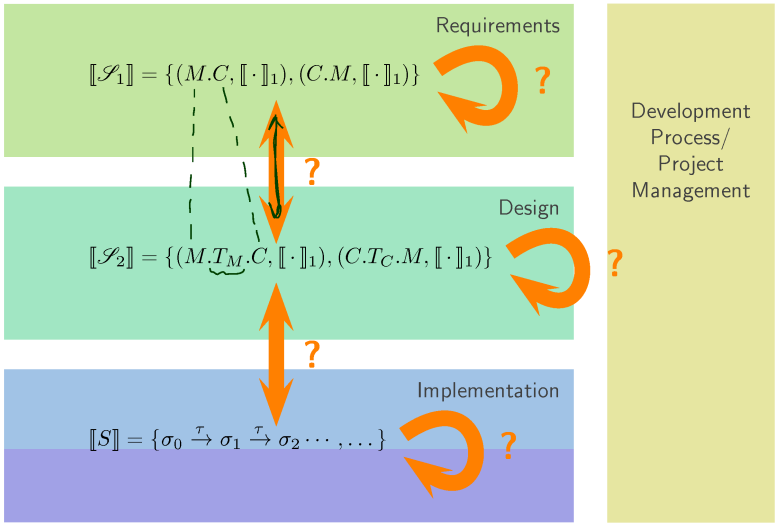
- **Customer 2** "you choose, but be consistent"

$(M.C)$ or $(C.M)$

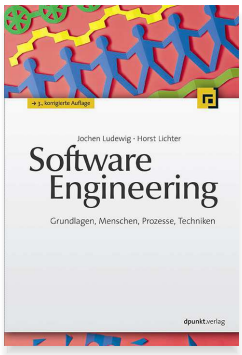
- **Customer 3** "consider human errors"

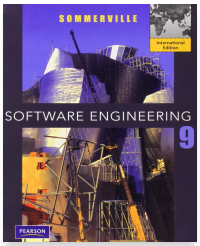
$(C.M)$

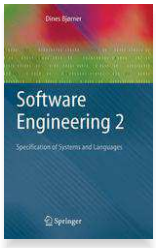
Formal Software Development



Literature







. . . more on lecture's homepage.

Any **questions** so far?

Formalia

Who's Who

- **Lecturer**: Dr. Bernd Westphal
- **Assistant**: Sergio Feo Arenis, MSc
- **Tutors**: Betim, Claus, Jan, Michael

- **Homepage:**

  `http://swt.informatik.uni-freiburg.de/teaching/SS2015/swtvl`

- **Course language:** tja, **English** or **German**...?
- **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. 1 week delay
    (link on **homepage**)

Questions and Interaction

- **Interaction:**
  absence often moaned but **it takes two**, so please ask/comment immediately.

- **Questions:**
  - **"online"**: ask immediately or in the break
  - **"offline"**:

    (i) try to solve yourself

    (ii) discuss with colleagues

    (iii)
    - Exercises: contact tutor (cf. homepage)
    - Rest: contact lecturer (cf. homepage)
      or just drop by: Building 52, Room 00-020

- **Break:**
  - We'll have a **10 min. break** in the middle of each lecture from now on,
    unless a majority objects **now**.

Exam

- **Exam Admission:**

  Achieving 50% of the regular **admission points** ($\rightarrow$ next slide) in total is sufficient for admission to exam.

  Typically, 20 regular admission points per exercise sheet.

- **Exam Form:**
  - **written** exam
  - Friday, September, 11th, 2015, 9:00 c.t.
  - Building 101, Room: 026+036
  - Scores from the exercises **do not** contribute to the final grade.

Exercises & Tutorials

- **Schedule/Submission:**
  - exercises **online** with first lecture of a block,
    early turn in 24h before tutorial (usually Wednesday, 12:15, local time),
    regular turn in right before tutorial (usually Thursday, 12:15, local time).
  - should work in groups of **approx. 3**, clearly give **names** on submission
  - please submit **electronically** via **ILIAS**; paper submissions are **tolerated**

- **Rating system:** "most complicated rating system **ever**"
  - **Admission points** (good-will rating, upper bound)
    ("reasonable proposal given student's knowledge **before** tutorial")
  - **Exam-like points** (evil rating, lower bound)
    ("reasonable proposal given student's knowledge **after** tutorial")

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

- **Tutorial:** **Plenary**.
  - Together develop **one** good proposal,
    starting from discussion of the early submissions (anonymous).
  - Tutorial notes provided as print-outs in subsequent lecture.

Evaluation of the Course

- **Mid-term Evaluation(s):**
  - In addition to the mandatory final evaluation, we will have **intermediate evaluation(s)**.
  - If you decide to leave the course earlier you may want to **do us a favour** and tell us the reasons – by participating in the evaluation(s) (will be announced on homepage).

- **Note**: we're **always** interested in

  comments/hints/proposals/wishes/...

  concerning **form** or **content**.

  Feel free to approach us (tutors, Sergio, me) in any form. **We don't bite.**

References

References

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

Bjørner, D. and Havelund, K. (2014). 40 years of formal methods. talk.

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).

Jones, C. B. et al., editors (2011). *Dependable and Historic Computing - Essays Dedicated to Brian Randell on the Occasion of His 75th Birthday*, volume 6875 of *LNCS*. Springer.

Kopetz, H. (2011). What I learned from Brian. In Jones et al. (2011).

Lovins, A. B. and Lovins, L. H. (2001). *Brittle Power - Energy Strategy for National Security*. Rocky Mountain Institute.

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 et al. (2011), pages 413–427.