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
- literature
IEEE Standard Glossary of Software Engineering Terminology

Systems and software engineering — Vocabulary

ISO/IEC/IEEE 24765:2010(E)
© ISO/IEC 2010
© IEEE 2010

INTERNATIONAL STANDARD
ISO/IEC/IEEE 24765

The Institute of Electrical and Electronics Engineers, Inc.

Systems and software engineering — Vocabulary only in English — Terminology

First edition 2010-12-15

Software, Engineering, Software Engineering
**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 – 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 – (1) The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software; that is, the application of software 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 – 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 in the Academy

F. L. Bauer

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.

Bertrand Meyer

Software professionals should be immutable beings, or even living artifacts, who build and maintain their systems as an extension of their beneficiaries.

Software professionals: The software professionals' main concern is to deliver software that is reliable and works efficiently on real machines.

Institutions that teach software should be 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.

Bertrand Meyer

Software professionals should be immutable beings, or even living artifacts, who build and maintain their systems as an extension of their beneficiaries.

Software professionals: The software professionals' main concern is to deliver software that is reliable and works efficiently on real machines.
There 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 best to define this profession.

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)

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)


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)

6.1 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.
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))

- Budget in € (378 responses)
- Planned duration in months (378 responses)
- Project completion (378 responses)
- Main functionality realised (368 responses)
- Secondary functionality realised (368 responses)
- Deadline (368 responses)
- Deadline missed by (91 responses)
- Criticality (378 responses, 30 'not spec.')
- Budget (368 responses)
- Completed
- Late
- Early
- Over
- Under
- 0-10 %
- 10-25 %
- 25-49 %
- 50-99 %
- 100 %
A Closer Look

- **Successful:**

  \[ \text{Time } t \rightarrow \text{Time } t' \geq t \]

  Customer - Developer
  software contract

  

- **Unsuccessful:**

  \[ \text{Time } t \rightarrow \text{Time } t' \geq t \]

  Customer - Developer
  software contract

  

  What might've gone wrong?

---

Some scenarios:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>①</td>
<td>Capturing Requirements</td>
<td>②</td>
<td>Design</td>
<td>③</td>
<td>Implementation</td>
</tr>
</tbody>
</table>

- 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

- Capturing Requirements
- Design
- Implementation
- Code Quality Assurance

Software Project Management
**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></td>
</tr>
<tr>
<td>informal</td>
<td></td>
</tr>
<tr>
<td>semi-formal</td>
<td></td>
</tr>
<tr>
<td>formal</td>
<td></td>
</tr>
</tbody>
</table>

---

**Excursion: Informal vs. Formal Techniques**

**Example:** Requirements Engineering, Airbag Controller

**Requirement:**

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

vs.

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

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

→ no more misunderstandings, sometimes tools can objectively decide requirement satisfied yes/no.
Structure of Topic Areas

Example: Requirements Engineering

<table>
<thead>
<tr>
<th>Vocabulary</th>
<th>Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td></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. consistent, complete, tacit, etc.
- e.g. “Whenever a crash...”
- e.g. “Always, if (crash) at t...”
- e.g. “∀ t, t’ ∈ Time • ...”

→ no more misunderstandings, sometimes **tools** can **objectively** decide: requirement satisfied yes/no.
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
On popular demand, the chair for software engineering agreed on: strong(er) coupling between both courses.

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Scales, Metrics, Costs</th>
<th>Development</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>L 1: 18.4., Mon</td>
<td>L 2: 21.4., Thu</td>
<td>L 4: 2.5., Mon</td>
<td>L 5: 5.5., Thu</td>
</tr>
<tr>
<td>T 1: 28.4., Thu</td>
<td>T 2: 19.5., Thu</td>
<td>T 3: 23.5., Mon</td>
<td>T 4: 2.5., Thu</td>
</tr>
<tr>
<td>L 9: 26.5., Thu</td>
<td>L 10: 26.5., Thu</td>
<td>L 11: 30.5., Mon</td>
<td>L 12: 28.5., Mon</td>
</tr>
<tr>
<td>T 5: 29.5., Thu</td>
<td>L 13: 30.5., Mon</td>
<td>T 6: 3.6., Mon</td>
<td>T 7: 6.6., Mon</td>
</tr>
<tr>
<td>L 14: 6.6., Mon</td>
<td>L 15: 9.6., Thu</td>
<td>L 16: 9.6., Thu</td>
<td>L 17: 12.6., Mon</td>
</tr>
<tr>
<td>L 18: 12.6., Thu</td>
<td>T 8: 15.6., Thu</td>
<td>L 19: 15.6., Thu</td>
<td>L 20: 18.6., Mon</td>
</tr>
<tr>
<td>L 24: 24.6., Thu</td>
<td>T 10: 27.6., Thu</td>
<td>L 25: 27.6., Thu</td>
<td>L 26: 30.6., Mon</td>
</tr>
<tr>
<td>L 27: 30.6., Mon</td>
<td>T 11: 33.6., Thu</td>
<td>L 28: 33.6., Thu</td>
<td>L 29: 36.6., Mon</td>
</tr>
<tr>
<td>L 30: 36.6., Mon</td>
<td>T 12: 39.6., Thu</td>
<td>L 31: 39.6., Thu</td>
<td>L 32: 42.6., Mon</td>
</tr>
<tr>
<td>L 33: 42.6., Mon</td>
<td>T 13: 45.6., Thu</td>
<td>L 34: 45.6., Thu</td>
<td>L 35: 48.6., Mon</td>
</tr>
<tr>
<td>L 36: 48.6., Mon</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Literature**

- More on the course homepage.
Any Questions So Far?

Course: Organisation
Organisation: Lectures

- **Homepage:** [http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl](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](http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl) with beginning of lecture the latest
  - slides with annotations on [homepage](http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl) typically soon after the lecture
  - recording on [ILIAS](http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl) (stream and download) with max. 2 days delay (cf. link on [homepage](http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl))
- **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](http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl) (group) forum, contact tutor
  - b) Everything else: contact lecturer (cf. [homepage](http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl)) 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](http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl) and [ILIAS](http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl) 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](http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl); 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)
  - Exam-like points (evil rating, lower bound)
  - “reasonable grading given student’s knowledge before tutorial”
  - “reasonable grading given student’s knowledge after tutorial”
- **Tuition:** 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](http://swt.informatik.uni-freiburg.de/teaching/SS2016/swtvl).
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.
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


