Softwaretechnik / Software-Engineering

Lecture 1: Introduction

2015-04-20

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
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.

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

<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 subjectively possible, 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)

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)
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).

**Software engineering** — the establishment and use of sound engineering principles to obtain economically software that is reliable and works efficiently on real machines.

*IEEE 610.12 (1990)*

*B. L. Bauer (1971)*
“software that is reliable and works efficiently” (Bauer, 1971)


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

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

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.’)
**Projects success, Budget, Functionality**

- **Project Completion (378 responses):**
  - Completed: 77.83%
  - Cancelled: 2.45%

- **Budget (368 responses):**
  - Kept: 93.52%
  - Below: 6.48%
  - Above: 0.01%

- **Main Functionality Realised (368 responses):**
  - 25-49%: 81.52%
  - 50-74%: 1.14%
  - 75-89%: 3.26%
  - 90-94%: 1.56%
  - 95-99%: 1.90%
  - 100%: 1.90%

- **Secondary Functionality Realised (368 responses):**
  - < 25%: 57.61%
  - 25-49%: 25.11%
  - 50-74%: 15.38%
  - 75-89%: 5.49%
  - 90-94%: 9.89%
  - 95-99%: 20.88%
  - ≥ 100%: 2.58%

**Deadlines, Project Leader, Process Model**

- **Deadline (368 responses):**
  - Kept: 72.01%
  - Late: 24.73%

- **Deadline Missed by (91 responses):**
  - ≤ 20%: 29.67%
  - 20-49%: 15.38%
  - 50-74%: 5.49%
  - 75-99%: 9.89%
  - ≥ 100%: 20.88%

- **Existence of Project Leader (378 responses):**
  - Leader Responded: 77.51%
  - Appointed: 19.93%
  - Not Appointed: 1.55%

- **Use of Process Model (378 responses):**
  - Not Specified: 57.43%
  - Used: 39.95%
  - Not Used: 2.62%
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
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

**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
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 least)*
"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 $\llbracket S \rrbracket$ of (finite or infinite) computation paths of the form $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$.

$\sigma_i$: state/configuration; $\alpha_i$: action/event.

- Programs.

```java
private void f(int x, int y) {
    x = x + y;
    y = x / 2;
    return y;
}
```

- HTML.

```html
<html>
<head>
<title>SWT 2015</title>
</head>
<body/>
</html>
```

$\llbracket S \rrbracket = \{ \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \xrightarrow{\alpha_3} \cdots \}$
Software is a finite description \( S \) of a (possibly infinite) set \( [S] \) of (finite or infinite) computation paths of the form \( \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \).

\( \sigma_i \): state/configuration; \( \alpha_i \): 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 $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \ldots$.

$\sigma_i$: state/configuration; $\alpha_i$: action/event.

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

Software Specification, formally

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

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

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


Example: Software Specification

Alphabet:
- \( M \) – dispense cash only,
- \( C \) – return card only,
- \( M C \) – dispense cash and return card.

- **Customer 1** "don’t care"
  \[
  (M.C \mid C.M \mid \begin{array}{c}
    M \\
    C
  \end{array})
  \]

- **Customer 2** "you choose, but be consistent"
  \[
  (M.C) \text{ or } (C.M)
  \]

- **Customer 3** "consider human errors"
  \[
  (C.M)
  \]

---

Formal Software Development

\[
\begin{align*}
S_1 &= \{ (M.C, \{ \cdot, j_1 \}), (C.M, \{ \cdot, j_1 \}) \} \\
S_2 &= \{ (M.M, \{ \cdot, j_1 \}), (C.T.C.M, \{ \cdot, j_1 \}) \} \\
S &= \{ \sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \ldots \}
\end{align*}
\]
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/swtv1

- **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**”:
    1. try to solve yourself
    2. discuss with colleagues
    3. Exercises: contact tutor (cf. homepage)
    4. 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 (→ 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