Lecture 1: Introduction

2017-04-24
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

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)
Systems and software engineering — Vocabulary

Software: Ingénierie des systèmes et du logiciel — Vocabulaire

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

© ISO/IEC 2010
© IEEE 2010
Software Engineering

The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

IEEE 610.12 (1990)

Software Engineering —

the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software.

see IEEE 610.12 (1)


Software Engineering —

Multi-person Development of Multi-version Programs.

D. L. Parnas (2011)

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

More general:


software related quality

process quality

. . .

product quality

functionality

suitability

accuracy

interoperability

security

maturity

recoverability

usability

understandability

learnability

operability

attractiveness

efficiency

time behaviour

resource utilisation

maintainability

analysability

changeability

stability

testability

portability

adaptability

installability

co-existence

replaceability

6.2 Reliability

The capability of the software product to maintain a specified level of performance when used under specified conditions.

6.2.2 Fault tolerance

The capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.
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?

Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Softwareentwicklungsprojekten in Deutschland 2006

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

<table>
<thead>
<tr>
<th>Budget in €</th>
<th>1-9,999</th>
<th>10,000-99,999</th>
<th>100,000-499,999</th>
<th>500,000-999,999</th>
<th>≥ 1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Responses</td>
<td>33.07</td>
<td>2.91</td>
<td>10.05</td>
<td>22.49</td>
<td>25.13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Planned Duration in Months</th>
<th>≤ 3</th>
<th>&gt; 3-6</th>
<th>&gt; 6-12</th>
<th>&gt; 12-24</th>
<th>&gt; 24</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Responses</td>
<td>72.01</td>
<td>24.73</td>
<td>2.45</td>
<td></td>
<td>97.35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Criticality</th>
<th>Business Critical</th>
<th>Mission Critical</th>
<th>Safety Critical</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Responses</td>
<td>0.27</td>
<td>82.61</td>
<td>4.89</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project Completion</th>
<th>Completed</th>
<th>Cancelled</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Responses</td>
<td>97.35</td>
<td>2.65</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deadline</th>
<th>Kept</th>
<th>Early</th>
<th>Late</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Responses</td>
<td>72.01</td>
<td>24.73</td>
<td>2.45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Main Functionality Realised</th>
<th>Kept</th>
<th>Below</th>
<th>Above</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Responses</td>
<td>81.52</td>
<td>11.14</td>
<td>3.26</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Budget</th>
<th>Completed</th>
<th>Below 20%</th>
<th>20-49%</th>
<th>50-99%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Responses</td>
<td>29.67</td>
<td>15.38</td>
<td>5.49</td>
<td>9.89</td>
<td>20.88</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Secondary Functionality Realised</th>
<th>Completed</th>
<th>Below 25%</th>
<th>25-49%</th>
<th>50-74%</th>
<th>75-89%</th>
<th>90-94%</th>
<th>95-99%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Responses</td>
<td>4.89</td>
<td>57.61</td>
<td>8.15</td>
<td>7.61</td>
<td>13.04</td>
<td>4.89</td>
<td>2.99</td>
<td>0.27</td>
</tr>
</tbody>
</table>
This course is about formal software engineering and software project management.

**Example: Nightly Builds**

Set up a procedure, which (at best: automatically) tries to compile the current state of the development each day over night.

Properly?→

- Avoid certain mistakes or costs?
- Measure the quality of the product?
- Ensure that the right product is built properly?
- ...and discuss.

Software Engineering is a young discipline:

- The discipline is characterized by a large number of proposals. The same holds for documentation: if no maintenance is ever needed, documentation effort may be wasted.
- For each question, plenty of proposals are available.
- To decide whether each proposal is suitable for the project or not, "defensive discipline" is necessary.
- Defensive discipline means, that the damage is bounded during the project, the caused by the change during the project is always manageable.
- If program does not compile at time "t" at time "t′" does not compile at time "t′" does not compile at time "t′"
- "Nightly Builds" is large, it can be very difficult (and time consuming) to identify the cause.
- The number of possible causes is large, it can be very difficult (and time consuming) to identify the cause.
- The number of possible causes is large, it can be very difficult (and time consuming) to identify the cause.
- If program does not compile at time "t′" does not compile at time "t′" does not compile at time "t′"
- "Nightly Builds"

All engineering disciplines face the same questions:

- How to describe requirements precisely?
- How to describe design ideas precisely?
- How to schedule activities properly?
- How to avoid misunderstandings with the customer?
- How to avoid misunderstandings with the implementers?
- How to manage properly?
- How to measure properly?
- How to ensure that the right product is built properly?
- How to avoid certain mistakes or costs systematically?
Example: Requirements Engineering, Airbag Controller

DaimlerChrysler AG, CC BY-SA 3.0

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

Developer A: ‘within’ means ‘$\leq$’; so 100 ms is okay, too.

Developer B: ‘within’ means between $300 - \varepsilon$ and $300 + \varepsilon$.

vs.

- Fix observables:
  - crashdetected: $\text{Time} \to \{0, 1\}$
  - fireairbag: $\text{Time} \to \{0, 1\}$

- Formalise requirement:
  $$\forall t, t' \in \text{Time} \cdot \text{crashdetected}(t) \land \text{fireairbag}(t') \Rightarrow t' \in [t + 300 - \varepsilon, t + 300 + \varepsilon]$$

→ no more misunderstandings, sometimes tools can objectively decide: requirement satisfied yes/no.
Course Software-Engineering vs. Softwarepraktikum – 1 – 2017-04-24 – Srel –

Agreement between ‘Fachschaft’ and the chair for software engineering: stronger coupling between both courses.

Introduction
- L 1: 24.4., Mon
- Scales, Metrics
- L 2: 27.4., Thu
- Performance
- T 1: 4.5., Thu
- Development
- L 3: 8.5., Mon
- Costs
- L 4: 11.5., Thu
- Process
- L 5: 15.5., Mon
- T 2: 18.5., Thu
- L 6: 22.5., Mon
- Requirements
- L 7: 29.5., Mon
- L 8: 1.6., Thu
- L 9: 19.6., Mon
- L 10: 22.6., Thu
- Arch. & Design
- L 11: 26.6., Mon
- L 12: 3.7., Mon
- L 13: 6.7., Thu
- Software Modelling
- L 14: 10.7., Mon
- T 5: 13.7., Thu
- Patterns
- L 15: 17.7., Mon
- L 16: 20.7., Thu
- QA (Testing, Formal Verif.)
- L 17: 24.7., Mon
- Wrap-Up
- L 18: 27.7., Thu

Literature

- Project Management
- Vocabulary
- Techniques
- informal
- formal

- Requirements Engineering
- Vocabulary
- Techniques
- informal
- formal

- Design, SW Modelling
- Vocabulary
- Techniques
- informal
- formal

- Quality Assurance
- Vocabulary
- Techniques
- informal
- formal

... more on the course homepage.

Any Questions So Far?

Course: Organisation

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
your tutor of (at best: prove) the correctness of your solution.

• convince

• state

still, humans are at the heart of software engineering.

note:

the task in your own words,

rephrase

• enable objective, tool-based assessment

• Basic rule for high quality submissions:

• avoid misunderstandings,

• Formal (vs. informal) methods

• a tiny little scientific work

Every exercise task is

• motivates content of the course – for the case of software

• project management

quality assurance,

• implementation,

• design,

• gather requirements,

• (Fun) fact

• some definitions are neither formal nor universally agreed

• totally agree

• strongly disagree

I have improved my skills in scientific

• problem solving.

I have improved my skills in scientific

• good-will rating

now

vs.

15:00

Wrap-UpL 18: 27.7., Thu

starting from discussion of the early submissions (anonymous),

we’ll have a

5-10 min. break

• break:

• contact lecturer (cf. homepage)

• (group) forum, contact tutor

• (iii) a)

• (evil rating, lower bound)

• (good-will rating, upper bound)

most complicated grading system ever

Grading system:

• Exam Form:

• (i) try to solve yourself

• (ii) discuss with colleagues

• (iii) a)

• (stream and download) with max. 2 days delay (cf. link on

Homepage:

• ILIAS:

• scores from the exercises

• permitted exam aids: one A4 paper (max. 21 x 29.7 x 1 mm) of notes, max. two sides inscribed

• date, time, place: tba

Exam Admission:

• 10 regular admission points on sheets 0 and 1, and

• 20 regular admission points on exercise sheets 2–6

10 admission points in total

• admission points

 Achieving 50% of the

Regular admission points

-ilias via

Homepage:

• http://swt.informatik.uni-freiburg.de/teaching/SS2017/swtvl

Script/Media:

• arch. & design

• engineering

• software

• software engineering,

• arch. & design

• software

• software engineering

Scales, Metrics, L 2: 27.4., Thu

Introduction:

• organisation: exercises & tutorials

• organisation: lectures

Course Code:

• 60250

ECTS:

• 5

Objective:

• students learn concepts of software development and

• software engineering

• students learn to use software development tools

• students learn to take care of the quality of the delivery product

• students improve listening, speaking, and writing skills

• students improve skills to work in teams

• students learn to present project work effectively

• students learn how to work independently

One Last Word on The Exercises...

Tell Them What You’ve Told Them...
References


