Introduction

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

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

Judging by the employment situation, current and future graduates can be happy with their studies. The Information Technology Association of America estimated software professionals who will build and maintain these systems to the satisfaction of their beneficiaries.

If not everyone agrees on the definition of the discipline, few question its necessity. Advocates of software engineering 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 somehow valid and retain all the views of software they encompass. In fact, I'm not just focusing on the "software engineering courses" traditionally offered in many universities but more generally on how to instill software engineering into an entire software curriculum.

For some, software engineering is just a glorified name for programming. If you are a programmer, you might put yourself in the category of "software developers" and never "software engineers." Some use the terms interchangeably, and many people believe that software engineering is just another name for systems engineering, and in fact it can be. However, it is not identical to systems engineering, in part because systems engineering deals with a broader context and at a wider scale. If systems engineering is the product of systematic decomposition, then software engineering is the product of systematic specification. If software engineering is the product of systematic abstraction, then systems engineering is the product of systematic generalization. Systems engineering is concerned with the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software.

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 somehow valid and retain all the views of software they encompass. In fact, I'm not just focusing on the "software engineering courses" traditionally offered in many universities but more generally on how to instill software engineering into an entire software curriculum.
The course's working definition of 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) scope, quality, cost, time

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?

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

Report: VSEK/55/D

Version: 1.1

Datum: 28.09.2006

Some Empirical Findings (Buschermöhle et al. 2006)

<table>
<thead>
<tr>
<th>Budget in € (378 responses)</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-9,999</td>
<td>33.07</td>
<td>2.91</td>
<td>10.05</td>
<td>22.49</td>
<td>25.13</td>
<td>≤ 3</td>
<td>&gt; 3-6</td>
<td>&gt; 6-12</td>
<td>&gt; 12-24</td>
<td>&gt; 24</td>
<td></td>
</tr>
<tr>
<td>10,000-99,999</td>
<td>72.01</td>
<td>24.73</td>
<td>2.45</td>
<td>1.9</td>
<td>25-49 %</td>
<td>50-74 %</td>
<td>75-89 %</td>
<td>90-94 %</td>
<td>95-99 %</td>
<td>100 %</td>
<td></td>
</tr>
<tr>
<td>100,000-499,999</td>
<td>5.89</td>
<td>82.61</td>
<td>4.89</td>
<td>4.89</td>
<td>5.16</td>
<td>≤ 25 %</td>
<td>25-49 %</td>
<td>50-74 %</td>
<td>75-89 %</td>
<td>90-94 %</td>
<td>95-99 %</td>
</tr>
</tbody>
</table>
| 500,000-999,999             | 97.35 | 2.65 | completed | cancelled | project completion (378 responses, 30 ' not spec. ')
| ≥ 1,000,000                 | 29.67 | 15.38 | 5.49 | 9.89 | 20.88 | < 20 % | 20-49 % | 50-99 % | 100-199 % | ≥ 200 % |
| Not Specified               | 0.27 | 4.89 | 4.89 | 5.16 | 1.9 | 25-49 % | 50-74 % | 75-89 % | 90-94 % | 95-99 % | 100 % |

Deadline (368 responses)

<table>
<thead>
<tr>
<th>Main Functionality Realised</th>
<th>81.52</th>
<th>11.14</th>
<th>3.26</th>
<th>kept</th>
<th>below</th>
<th>above</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secondary Functionality Realised (368 responses)</td>
<td>0.27</td>
<td>4.89</td>
<td>4.89</td>
<td>5.16</td>
<td>1.9</td>
<td>25-49 %</td>
</tr>
</tbody>
</table>
All engineering disciplines face the same questions:
• How to describe requirements / avoid misunderstandings with the customer?
• How to describe design ideas / avoid misunderstandings with the implementers?
• How to ensure that the product is built right / that the right product is built?
  (→ How to measure the quality of the product?)
• How to schedule activities properly?
At best: are there procedures which promise to systematically avoid certain mistakes or costs?

This course is about Software Engineering, so we should discuss:
• How to describe requirements on software precisely?
• How to describe design ideas for software precisely?
• How to ensure that software is built right?
  (→ How to measure the quality of software?)
• How to schedule software development activities properly?
What are procedures to systematically avoid certain mistakes or costs in software development?

Example: Nightly Builds

Scenario:
• Program P compiles successfully at time t.
• Programmers work for duration d on P, yielding program P’ at time t + d.
• P’ does not compile at time t + d.
  → the reason for not compiling any more must be among the changes during d.

Experience:
• If d is large, it can be very difficult (and time consuming) to identify the cause.

Proposal: "Nightly Builds"
• Set up a procedure, which (at best: automatically) tries to compile the current state of the development each day over night.
• Promise: with "nightly builds", d is effectively limited to be smaller or equal to one day, so the number of possible causes for not compiling should be manageable.

→ Software Engineering as a defensive discipline (measures against failures and "catastrophes"):
• if program P always compiles, the effort for "nightly builds" was strictly speaking wasted.
• if a compilation issue occurs during the project, the caused damage is bounded.
Same holds for documentation: if no maintenance is ever needed, documentation effort may be wasted.

Course: Content

Software Engineering is a young discipline: plenty of proposals for each question.
So the course will focus on the problems and discuss example proposals.
Course Content (Tentative)

- Capturing Requirements
- Design
- Implementation
- Code Quality Assurance

Structure of Topic Areas

Example: Requirements Engineering

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

Techniques:
- informal
- semi-formal
- formal

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 ms (±ε).

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

Developer B: ‘within’ means between 300 − ε and 300 + ε.

• Fix observables:
  - crashdetected: Time → {0, 1}
  - fireairbag: Time → {0, 1}

• Formalise requirement:
  ∀t, t′ ∈ Time • crashdetected(t) ∧ airbagfired(t′) ⇒ t′ ∈ [t + 300 − ε, t + 300 + ε]

→ no more misunderstandings, sometimes tools can objectively decide: requirement satisfied yes/no.
Any Questions So Far?

Course Software-Engineering vs. Softwarepraktikum

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

Introduction

- L 1: 16.4., Mon
  Scales, Metrics
- L 2: 19.4., Thu
  Costs
- L 3: 23.4., Mon
  Development Process
- L 4: 30.4., Mon
  Requirements Engineering
- L 5: 3.5., Thu
  T 1: 26.4., Thu
- L 6: 7.5., Mon
  - 10.5., Thu
  Software-Modelling
- L 7: 14.5., Mon
  Arch. & Design
- L 12: 18.6., Mon
  T 4: 21.6., Thu
- L 13: 25.6., Mon
  Software Modelling, T 5: 5.7., Thu
- L 14: 28.6., Thu
  Patterns
- L 15: 2.7., Mon
  QA
- L 16: 9.7., Mon
  T 6: 19.7., Thu
  (Testing, Formal Verification)
- L 17: 12.7., Thu
  Wrap-Up
- L 18: 16.7., Mon
  T 7: 19.7., Thu
  (Testing, Formal Verification)

Today’s 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

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?
The length of the longest straight line fully inside the square with side length $a$ is 27.01 (rounded). The longest straight line inside the square is the diagonal. By Pythagoras, its length is $\sqrt{a^2 + a^2} = \sqrt{2a^2} = 2a$. The working definition of success is at least 24 admission points for 100%.

However, the longest straight line inside the square is the diagonal, with a length of $\sqrt{2a}$.
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 yourself and your tutor of the correctness of your solution (at best: prove it).

Example:

**Task**: What is the length of the longest line inside the square with side length $a = 19.1$?

**Submission A**

The length of the longest straight line fully inside the square with side length $a = 19.1$ is 27.01 (rounded). The longest straight line inside the square is the diagonal. By Pythagoras, its length is $\sqrt{a^2 + a^2}$. Inserting $a = 19.1$ yields 27.01 (rounded).