Dr. Michael Eichberg
Software Engineering
Department of Computer Science
Technische Universität Darmstadt

Introduction to Software Engineering

What is Software Engineering?

TECHNISCHE UNIVERSITÄT DARMSTADT

What is Software?

TECHNISCHE UNIVERSITÄT DARMSTADT

> *“**Software**” - The **programs and other operating information used by a computer**.*

New Oxford American Dictionary; 2005

> *[...] **software** is not just the programs but also all **associated documentation** and configuration data that is needed to make these programs operate correctly.*

I. Sommerville

Software Engineering Eighth Edition; Pearson Education, 2007

“ The term software refers to a program and all of the associated information and materials needed to support its...

***installation**,*

operation,

***repair** and*

***enhancement**.*

W. S. Humphrey

The Software Engineering Process: Definition and Scope; ACM SIGSOFT Software Engineering Notes, Vol. 14, Issue 4, 1989

Software is **more than just code**.

- An executable program and its data
- Configuration files
- System documentation
  (e.g. architectural and analysis model, a design document,...)
- User documentation
- A website
  (To inform about issues, download updates,...)
- ...

Properties of Software

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Software has unique properties when compared to any hardware.

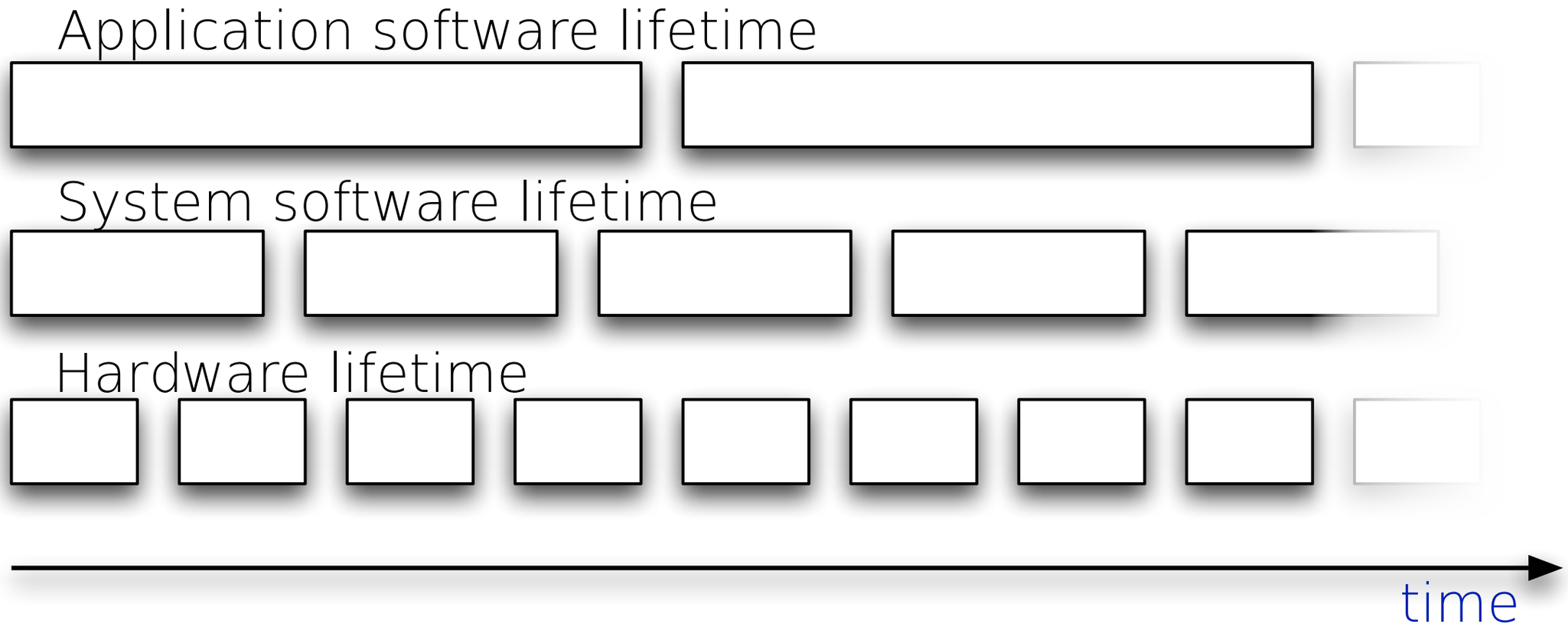
- No “real” physical borders
- Software doesn’t wear out / there are no spare-parts
  Nevertheless, Software has to be constantly updated to cope with changing environments; otherwise the software will become obsolete (software aging).
- Software is hard to “measure”
  How to define the quality of software?
  Are those things (e.g. the lines of code) that can be measured correlated to the quality? How can we measure progress?

“To Code is to Design”...

Several types of software can be distinguished.

- **Generic products**
  (in the past referred to as shrink-wrapped software)
  e.g. Microsoft Word, Open Office, Acrobat,...
  to shrink-wrap =dt. einschweißen; in Schrumpffolie verpacken
- **Customized products**
  (individual software, build-to-order software)
  e.g. TUCaN (Campusnet), an Air Traffic Control System, ...
- **Open-Source products**

The borders are blurring (e.g. Enterprise Resource Planning (ERP) software is often customized to match the workflows in a particular company).



Balzert

Lehrbuch der Softwaretechnik; Spektrum Akademischer Verlag, 1996

What is Software Engineering?

TECHNISCHE UNIVERSITÄT DARMSTADT

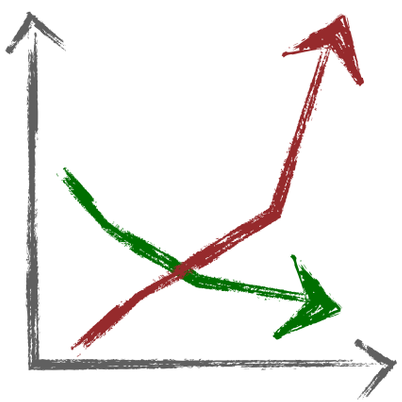
*(Hardware) "**Engineering**"*

*The branch of science and technology concerned with the **design**, **building**, and **use** of engines, machines, and structures.*

New Oxford American Dictionary; 2005

The term “**Software Crisis**” was coined in the 60’s and refers to multiple problems.

- The costs for hardware were falling, but the costs for software were rising significantly
- Software projects were not in-time, were not in-budget and contained too many errors
- Technological issues
  - Lack of suitable programming languages
  - Lack of methods
  - Lack of tool support
  - …



The term “**Software Engineering**” was coined at the end of the sixties and is often attributed to F.L. Bauer.

The NATO Software Engineering Conference
(Garmisch, Germany, 7-11 Oct 1968)
http://homepages.cs.ncl.ac.uk/brian.randell/NATO/N1968/index.html

(Software Engineering ~ dt. Softwaretechnik / Softwaretechnologie)

> *Software Engineering refers to the disciplined application of engineering, scientific, and mathematical principles and methods to the* ***economical production of quality software****.*
>
> *[...] quality refers to the degree to which a product meets its users' needs.*

W. S. Humphrey

The Software Engineering Process: Definition and Scope; ACM SIGSOFT Software Engineering Notes, Vol. 14, Issue 4, 1989

“***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 Standards Board

IEEE Standard Glossary of Software Engineering Terminology
Std. 610.12-1990, 1990

*"**Software Engineering**"*

*Software engineering is a systematic and disciplined approach to developing software. It applies both computer science and engineering principles and practices to the **creation, operation, and maintenance of software systems**.*

[Computer Science is concerned with the theories and methods that underlie computers and software systems, software engineering is concerned with the practical problems of producing software.]

> *[…] Der Begriff Software-Engineering steht für die Auffassung, dass die Erstellung, Anpassung und Wartung von Programmsystemen* ***kein "künstlerischer", sondern vorwiegend ein ingenieurmäßig verlaufender Prozess ist…***

Duden Informatik; 1993

"This year's [2009] results show a marked decrease in project success rates, with 32% of all projects succeeding which are delivered on time, on budget, with required features and functions.

44% were challenged which are late, over budget, and/or with less than the required features and functions and 24% failed which are cancelled prior to completion or delivered and never used.

These numbers represent a downtick in the success rates from the previous study, as well as a significant increase in the number of failures[...]

Standish Group, Boston, Massachusetts, April 23, 2009

CHAOS Summary 2009

New Standish Group report shows more project failing and less successful projects.

Software projects fail due to several different reasons.

(A software project is considered to have failed as soon as the project is not on-time or is not in-budget).

- The requirements and system dependencies are not well-defined
- Changing the requirements during the development is much, much easier for software than for hardware; (Software has to accommodate for hardware “issues”.)
- Lack of tools, methods, education, planning, ...

However, other complex and innovative **hardware systems are also often behind schedule** (e.g. the Airbus A380, the Boeing Dreamliner, the white iPhone).
Engineering Software is about getting the design right and less about building the 42nd A380.

"In the just-released report, CHAOS Manifesto 2011, The Standish Group's shows a marked increase in project success rates from 2008 to 2010. These numbers represent an uptick in the success rates from the previous study, as well as a decrease in the number of failures. [...]

This year's results represent the highest success rate in the history of the CHAOS Research.

[...] "We clearly are entering a new understanding of why projects succeed or fail." This understanding is spelled out in the CHAOS Manifesto research report.

Standish Group, Boston, Massachusetts, March 3, 2011

CHAOS Manifesto 2011

New Standish Group report shows more projects are successful and less projects failing.

Software engineering encompasses several areas.

- Software Requirements
  The requirements define what the system is expected to do.
- Software Design
  How the system is designed.
- Software Testing
  The systematic identification (and elimination) of errors.
- Software Maintenance
- Software Configuration Management
  The management of different versions and configuration of a software.
- Software Engineering Process
  Definition and improvement of software development processes.
- Software Engineering Tools and Methods
- Software Quality
- Software Ethics

Software engineering encompasses several areas.

- Software Requirements
  The requirements define what the systems is expected to do.
- Software Design
  How the system is designed.
- Software Testing
  The systematic elimination of errors.
- Software Maintenance
- Software Configuration Management
  The management of different versions and configuration of a software.
- Software Engineering Process
  Definition and improvement of software development processes.
- Software Engineering Tools and Methods
- Software Quality

Primary focus of this lecture.

Systems Engineering ↔ Software Engineering

- **System related activities**, such as defining the overall system objectives and requirements, allocating system functions between hardware and software, defining hardware / software interfaces, full system acceptance tests are essential, but they **are part of systems engineering**
- Software Engineering is a part of systems engineering

We will not talk about systems engineering in this lecture.

What is Software Engineering?

- A Critical View of Software Engineering

TECHNISCHE UNIVERSITÄT DARMSTADT

viewpoints

Contact Editor: Dennis Taylor ■ dtaylor@computer.org

Software Engineering:

An Idea Whose Time Has Come and Gone?

Tom DeMarco

We're now just past the 40th anniversary of the NATO Conference on Software Engineering in Garmisch, Germany, where the discipline of software engineering was first proposed. Because some of my early work became part of that new discipline, this seems like an appropriate moment for reassessment.

My early metrics book, *Controlling Software Projects: Management, Measurement, and Estimation* (Prentice Hall/Yourdon Press, 1982), played a role in the way many budding software engineers quantified work and planned their projects. In my reflective mood, I'm wondering, was its advice correct at the time, is it still relevant, and do I still believe that metrics are a must for any successful software development effort? My answers are no, no, and no.

The book for me is a curious combination of generally true things written on every page but combined into an overall message that's wrong. It's as though the book's young author had never met a metric he didn't like. The book's deep message seems to be, metrics are good, more would be better, and most would be best. Today we all understand that software metrics cost money and time and must be used with careful moderation. In addition, software development is inherently different from a natural science such as physics, and its metrics are accordingly much less precise in capturing the things they set out to describe. They must be taken with a grain of salt, rather than trusted without reservation.

We welcome your letters. Send them to software@computer.org. Include your full name, title, affiliation, and email address. Letters are edited for clarity and space.

Compelled to Control

The book's most quoted line is its first sentence: "You can't control what you can't measure." This line contains a real truth, but I've become increasingly uncomfortable with my use of it. Implicit in the quote (and indeed in the book's title) is that control is an important aspect, maybe the most important, of any software project. But it isn't. Many projects have proceeded without much control but managed to produce wonderful products such as GoogleEarth or Wikipedia.

To understand control's real role, you need to distinguish between two drastically different kinds of projects:

- Project A will eventually cost about a million dollars and produce value of around $1.1 million.
- Project B will eventually cost about a million dollars and produce value of more than $50 million.

What's immediately apparent is that control is really important for Project A but almost not at all important for Project B. This leads us to the odd conclusion that strict control is something that matters a lot on relatively useless projects and much less on useful projects. It suggests that the more you focus on control, the more likely you're working on a project that's striving to deliver something of relatively minor value.

To my mind, the question that's much more important than how to control a software project is, why on earth are we doing so many projects that deliver such marginal value?

Continued on p. 95

1982

2009

Tom DeMarco

IEEE Software, July/August 2009 (vol. 26 no. 4)

Freely available at:

http://www2.computer.org/cms/Computer.org/ComputingNow/homepage/2009/0709/rW_SO_Viewpoints.pdf

“You can’t control what you can’t measure.” **1982**

Read: “You can’t control software projects without taking extensive quantitative data…..”

Tom DeMarco

Controlling Software Projects: Management, Measurement, and Estimation; Prentice Hall/Yourdon Press, 1982

"[...] My early metrics book,[...]. I'm wondering, was its advice correct at the time, is it still relevant, and do I still believe that metrics are a must for any successful software development effort?

My answers are no, no, and no.

The book for me is a curious combination of generally true things written on every page but combined into an overall message that's wrong.[...]"

1982

2009

Tom DeMarco

Software Engineering
An Idea Whose Time Has Come and Gone?

"[...] the more you focus on control, the more likely you're working on a project that's striving to deliver something of relatively minor value.

[...] we need to reduce our expectations for exactly how much we're going to be able to control [...]."

E.g., the value of a project where the goal is to "just" replace a legacy technology is often very limited.

1982

2009

Tom DeMarco

Software Engineering
An Idea Whose Time Has Come and Gone?

"So, how do you manage a project without controlling it? Well, you manage the people and control the time and money.

*[...] Your job is to go about the **project incrementally**, adding pieces to the whole in the **order of their relative value**, and doing integration and documentation and acceptance testing incrementally as you go."*

1982

2009

Tom DeMarco

Software Engineering
An Idea Whose Time Has Come and Gone?

1982

"I still believe it makes excellent sense to engineer software. But that isn't exactly what software engineering has come to mean.

The term encompasses a specific set of disciplines including…

- *defined process,*
- *inspections and walkthroughs,*
- *requirements engineering,*
- *traceability matrices,*
- *metrics,*
- *precise quality control,*
- *rigorous planning and tracking, and*
- *coding and documentation standards."*

2009

Tom DeMarco

Software Engineering
An Idea Whose Time Has Come and Gone?

[...] Software development is and always will be somewhat experimental.

The actual software construction isn't necessarily experimental, but its conception is.

1982

2009

Tom DeMarco

Software Engineering
An Idea Whose Time Has Come and Gone?

Fifteen Principles of Software Engineering

TECHNISCHE UNIVERSITÄT DARMSTADT

1. *Make quality number 1*
2. *High-quality software is possible*
3. *Give products to customers early*
4. *Determine the problem before writing the requirements (...before starting to code)*
5. *Evaluate design alternatives*
6. *Use an appropriate process model*
7. *Use different languages for different phases*
8. *...*

1994

Alan M. Davis

Fifteen Principles of Software Engineering; IEEE Software 1994

“

7. ...
8. *Minimize intellectual distance*

The distance between the real-wold problem and the computerized solution to the problem..

9. *Put technology before tools*
   (Before you use a tool, you should understand and be able to follow appropriate software technique.)
10. *Get it right before you make it faster*
11. *Inspect code*
   (... Sometimes code inspections are claimed to be more effective than testing ...)
12. ...

Alan M. Davis

Fifteen Principles of Software Engineering; IEEE Software 1994

11. ...

12. Good management is more important than good technology

(... Management style must be adapted to the situation...)

13. People are the key to success

14. Follow with care

(Just because everybody is doing it, does not make it right for you...)

15. Take responsibility

Alan M. Davis

Fifteen Principles of Software Engineering; IEEE Software 1994

The goal of this lecture is to enable you to systematically carry out small(er) commercial or open-source projects.

Engineering software is hard; this lecture teaches you why and (to some extent) how to tackle common problems.

Software engineering is about designing software and not about building software.