Karlsruhe Institute of Technology

Welcome!

Software Engineering II („Advanced Software Engineering“)

Prof. Dr. Ralf H. Reussner
together with Prof. Dr. Raffaela Mirandola

Topic 1

Introduction

DSIS – DEPENDABILITY OF SOFTWARE-INTENSIVE SYSTEMS
KASTEL – INSTITUTE OF INFORMATION SECURITY AND DEPENDABILITY

dsis.kastel.kit.edu

SE II – at a Glance and Organisation

Goal

- deepen your understanding of systematic software development
  - by discussing selected state of the art approaches
- develop a critical understanding of principles, methods and tools

Lecture Materials

- Slides available in ILIAS
  - All information available on: https://ilias.studium.kit.edu/goto.php?target=crs_2636839&client_id=produktiv
  - Exercises available in ILIAS
    - **Processing** of exercises is **recommended**
  - merely slides are provided
    - are neither intended to replace attendance or notes
    - nor reading additional literature
      - pointers usually at the end of the slides

SE II – Participation

- We are open for questions during lectures and exercises
  - If you have a question
  - Do **not** hesitate to ask
- Throughout the course, we ask questions
  - Free fields for your answers
  - Individual participation of each student
- Throughout the course, we provide small group tasks
  - Students form Buzz Groups
  - Between 3 to 6 students sitting close
  - Work together on given task
- **Look for the icons**

And You?

- Bachelor/Master?
- Degree Program? (Studiengang?)
  - Computer Science (Informatik)
  - Information Systems (Wirtschaftsinformatik)
  - Computer Science-Teaching Degree (Informatik Lehramt)
  - Information Engineering and Management (Informationswirtschaft)
- Programming Experience? (Programmiererfahrung?)
  - In a company (in Unternehmen)
  - As a student assistant-Hiwi (als Hiwi)
- Attended Software Engineering I? (SWT1 gehört?)
- Attended other courses from the specialization area Software Engineering and Compiler Construction? (Andere Veranstaltungen aus dem Vertiefungsfach Softwaretechnik und Übersetzerbau besucht?)

Overview on First Lecture

- Content
  - a few more words more about the course
  - challenges in software development
  - and some ideas to overcome them
    - i.e. the content overview for this semester

- Learning goals for today, students –
  - have a coarse overview of the course content
    - and understand why it could be important for them
  - have refreshed some core challenges of software development
    - and have an appetite for learning how to overcome them

SE needs to be learned differently…

A word on ***activating teaching methods*** and the learning environment

- Knowledge cannot be „beamed“ from my head into yours
  - as shown by research in didactics
- Rather, it is important that you engage yourself with the content of the course

- I will offer the environment to do this in the classes
  - this might initially feel a bit different for most of you
  - however, you are not obliged to participate
  - feel free to use other ways of learning if they also suit you
    - or better: use my offerings and your approach together
  - Participate in the practical exercises („Übungsblätter“)
- **We will try that later …**

Course Topics

Design & Realisation

- Clean Coding
- Software Architecture & Components
  - Clean Architecture
  - Enterprise Application Architecture
  - Microservices
  - Cloud Architecture
  - Software Container
- Domain-driven design

Processes and Requirements

- Development processes
- Agile development
- Continuous Integration
- Requirements engineering
- Use cases
- NLP

Ensuring Software Quality

- Software Reliabilty
  - Real-time Systems
  - Real-time design patterns
- Software Security

Course Dates

- Classes
  - on Thursdays, at 14:00, Location: 30.46 Chemie, Neuer Hörsaal
  - on Fridays, at 11:30 , Location: 11.40, Tulla-Hörsaal
  - **On Wed 30.04.2025, at 17:30, Location: 11.40, Tulla-Hörsaal**
  - Exam (Written Exam): 10.09.2025, from 08:00 for 90 min

Tentative Dates		
Thu, 24/04/2025	Lecture	Introduction
Fri, 25/04/2025	Lecture	Clean Code
Wed, 30/04/2025 at 17:30	Lecture	Clean Code + Software Architecture
Fri, 02/05/2025	Lecture	Software Architecture
Thu, 08/05/2025	Lecture	Clean Architecture
Fri, 09/05/2025	Exercise	Sheet 1: Clean Code
Thu, 15/05/2025	Lecture	Software Components
Fri, 16/05/2025	Lecture	Software Components
Thu, 22/05/2025	Exercise	Sheet 2: Clean Architecture
Fri, 23/05/2025	Lecture	Microservices

Course Dates

- Classes

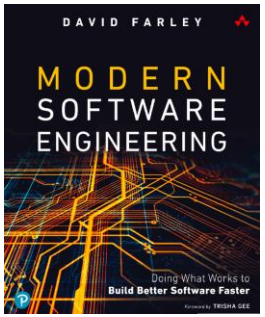
Tentative Dates		
Fri, 30/05/2025	Exercise	Sheet 3: Software Architectures and Components
Thu, 05/06/2025	Lecture	Patterns for Enterprise Application Architectures
Fri, 06/06/2025	Lecture	Cloud Computing
Fri, 20/06/2025	Exercise	Sheet 4: Patterns for Enterprise Application Architectures
Thu, 26/06/2025	Lecture	Domain Driven Design + Reviews
Fri, 27/06/2025	Lecture	Software Development Processes
Thu, 03/07/2025	Lecture	Agile Development
Fri, 04/07/2025	Exercise	Sheet 5: Domain Driven Design
Thu, 10/07/2025	Lecture	Requirements Engineering
Fri, 11/07/2025	Exercise	Sheet 6: Agile Development
Thu, 17/07/2025	Lecture	Guest Lecture
Fri, 18/07/2025	Lecture	Use Cases
Thu, 24/07/2025	Lecture	NLP
Fri, 25/07/2025	Lecture	Software Quality – Reliability
Thu, 31/07/2025	Exercise	Sheet 7: Requirements Engineering and Use Cases
Fri, 01/08/2025	Lecture	Ethic + Wrap-Up

General Literature References

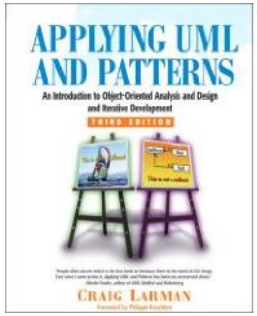
Recommended materials

- **Software Architecture Design**
  - Ralf Reussner et al., *Modeling and Simulating Software Architectures*
    - MIT Press, 2015

- **General SE**
  - David Farley, *Modern Software Engineering*, Addison-Wesley, 2022
  - Ian Sommerville, *Software Engineering* 9th edition, Addison-Wesley, 2010
  - *Digital Libraries* of ACM and IEEE et al.
  - *Software Engineering Body of Knowledge (SWEBOK)*



- **Object-oriented analysis and design**
  - Craig Larman, *Applying UML and Patterns* 3rd edition, Prentice Hall, 2004



Further references will be listed

➔ *at the end of each lecture*

More about SE II (1)

- **Core module of Specialization Area 6 Software Engineering** (Stammmodul des VF 6 Softwaretechnik)
- **Semester Weekly Hours** (SWS): 4

- **Examination**:
  - September 10th, 2025
- The assessment is carried out in the form of a written examination with a duration of 90 minutes in accordance with § 4 paragraph 2 number 1 of the examination regulations (Nachweis: Die Erfolgskontrolle erfolgt in Form einer schriftlichen Prüfung im Umfang von 90 Minuten nach § 4 Abs. 2 Nr. 1 SPO.)

- Credit points (Leistungspunkte - ECTS): 6

6 ECTS = 180 h Time commitment

Preparation and follow-up times: 1.5 hours per semester week hour (SWS)
(4 SWS + 1,5 x 4 SWS) x 15 + 30 h Exam preparation = 180 h

From SE1: Short Definition of Software Engineering

Software engineering is the science of software construction: the systematic development and maintenance of software systems (from SE 1, Tichy).

SE1 provided an overview of the methods of software construction and covers essential foundational knowledge for any computer scientist working in an engineering manner.
“Software engineering is the discipline of how to transform often unclear requirements into a description of the problem that a dumb machine can execute and then work with this description” – [Reussner]

From SE1: Overview

- Software engineering provides/seeks answers to the following questions:
  - How do you determine what properties the software to be constructed should have (requirements gathering)?
  - How do you then describe these properties (specification)?
  - How do you structure the software so that it can be easily built and flexibly modified (design)?
  - How do you modify software that lacks such structure or whose structure is no longer understood (maintenance, restoration, reengineering)?
  - How do you avoid defects in software or uncover them (quality assurance, testing)?
  - How do you organize the work of a software company or department to consistently achieve cost-effective and high-quality results on time (process management)?
  - …and other questions.

From SE 1 lecture by Walter F. Tichy and Ina Schaefer.

From SE1: Software Engineering is more than programming.

- Planning and cost estimation are essential.
- Requirements analysis, specification, and design also occur before the actual programming.
- Quality assurance extends to all activities (it is, for example, carried out by reviewing or inspecting the produced work products).
- The organization of small and large development teams is added.

- **Analogy of a garden house:** A hobbyist building a garden shed will not be able to build a skyscraper with their method.

From SE 1 Lecture by Walter F. Tichy and Ina Schaefer

SE II is …

... important for your future career

- as will be demonstrated in a second

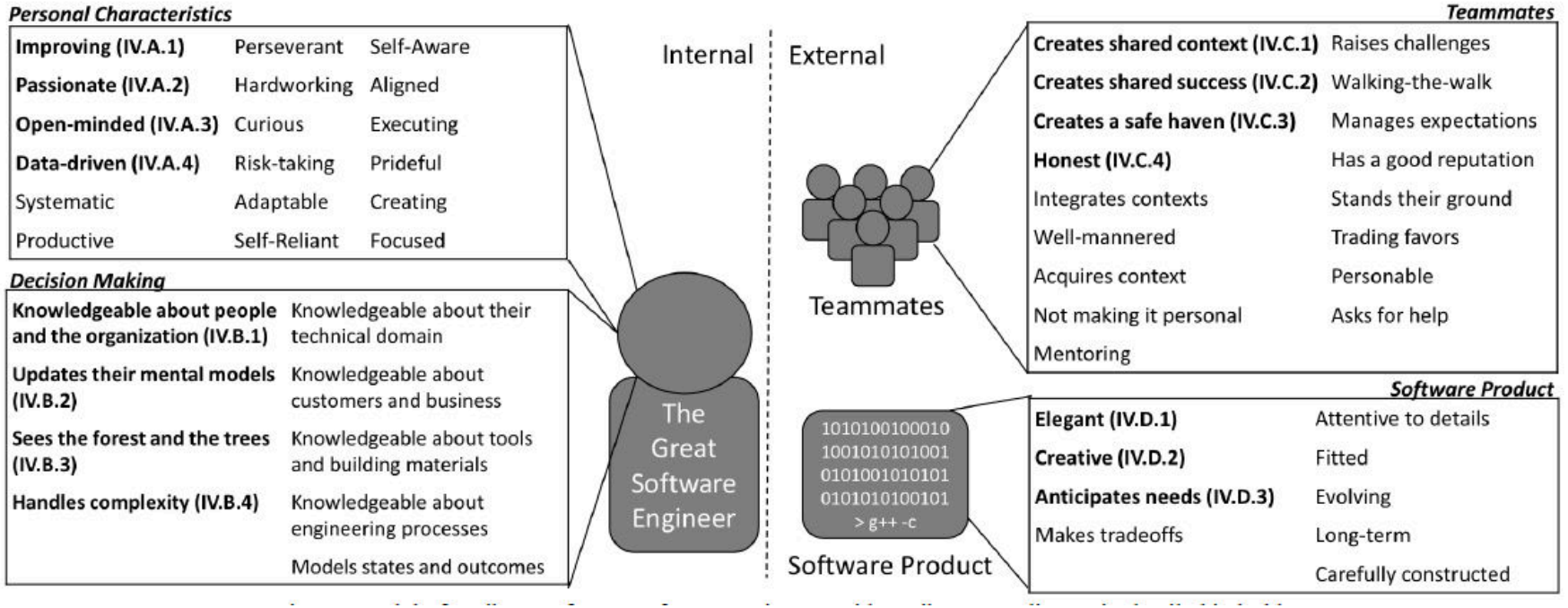
… a signpost for later specialized lectures in the area of software engineering at KIT, e.g.

- Software Testing and Quality Management (Software-Test and Qualitätsmanagement)
- Software Architecture and Quality (Software-Architektur and Qualität)
- Engineering Self-Adaptive Systems Software Evolution (Software-Evolution)
- Model-driven Software Development (Modellgetriebene Software-Entwicklung)
- Software Product Line Development (Software-Produktlinien-Entwicklung)
- Secure Software Engineering (Secure Software Engineering)
- ...

[Wikipedia]

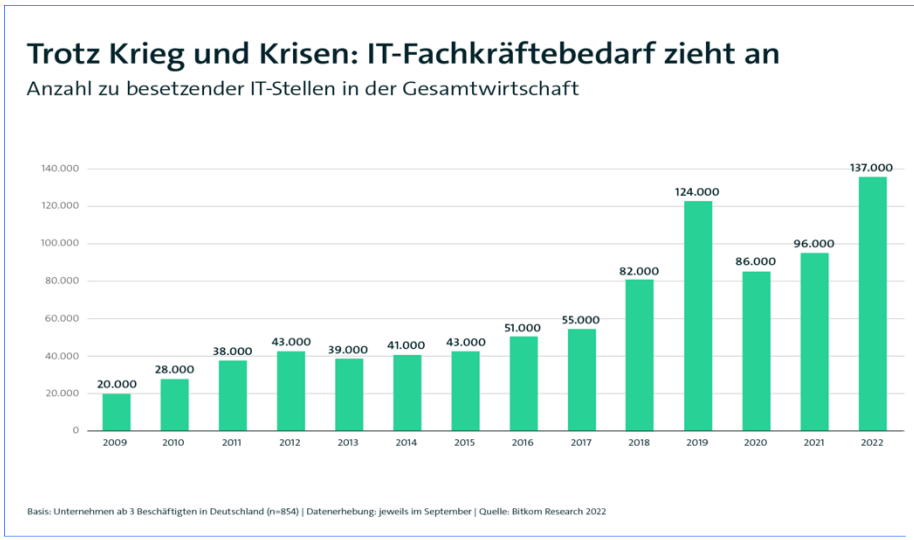
What makes a great Software Engineer?

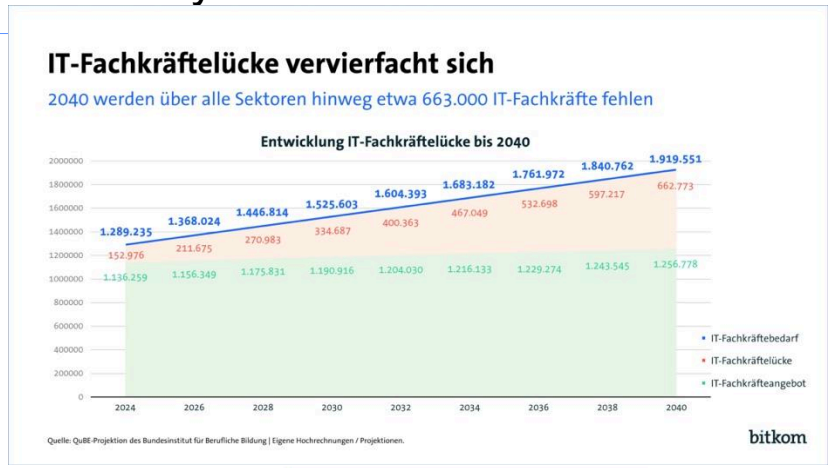
- This lecture focusses on technical skills, which are very important.
- But soft-skills needed for success, too.
- Interesting (fully optional) paper:
  - Li, Paul Luo, Andrew J. Ko, and Jiamin Zhu. "What makes a great software engineer?." In Proceedings of the 37th International Conference on Software Engineering-Volume 1, pp. 700-710. IEEE Press, 2015. http://dl.acm.org/citation.cfm?id=2818839



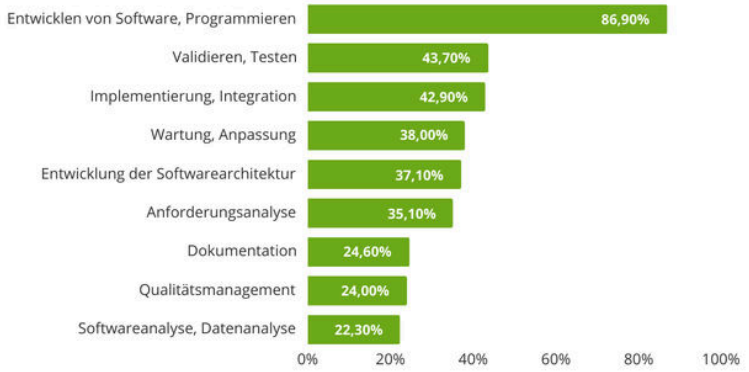
Career Outlook

- BITKOM surveys regularly underline a lack of IT specialists
  - ***149,000 open positions*** in Germany alone (2023)
  - predicted ***663,000 open positions*** in Germany until 2040





Which IT specialists are wan
[in the ICT sector]? (2024)

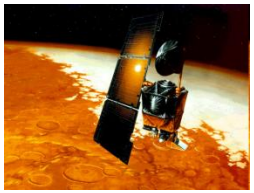


[https://www.bitkom.org/Presse/Presseinformation/Deutschland-fehlen-[illegible]]
[https://www.dekra-akademie.de/info/dekra-arbeitsmarkt-report]

Software development is difficult

Ariane 5

FISCUS



Mars Climate Orbiter

hochschulstart.de
Stiftung für Hochschulzulassung

Therac-25

A2LL

Patriot

Inpol-neu

Baggage transport
Denver Airport

TOLL COLLECT
service on the road

Boing 737
MAX 8

Airbus 320
Fly-By-Wire
System

It is a very humbling experience to make a multi-million-dollar mistake, but it is also very memorable.

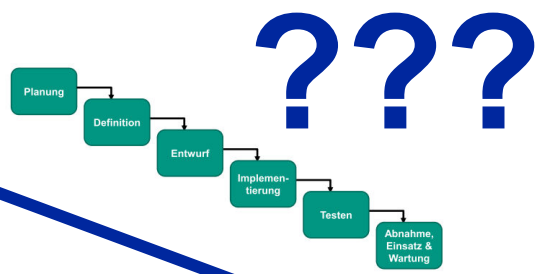
Turing Award Winner
Fred Brooks

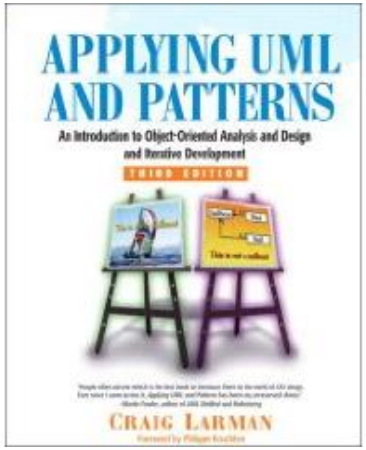
Other examples you know?

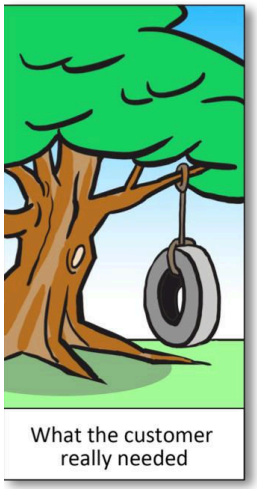
Teaser I: A Kind of Magic?

How can we bridge the gap between a problem in the
- real world and code?
  - in a systematic manner?

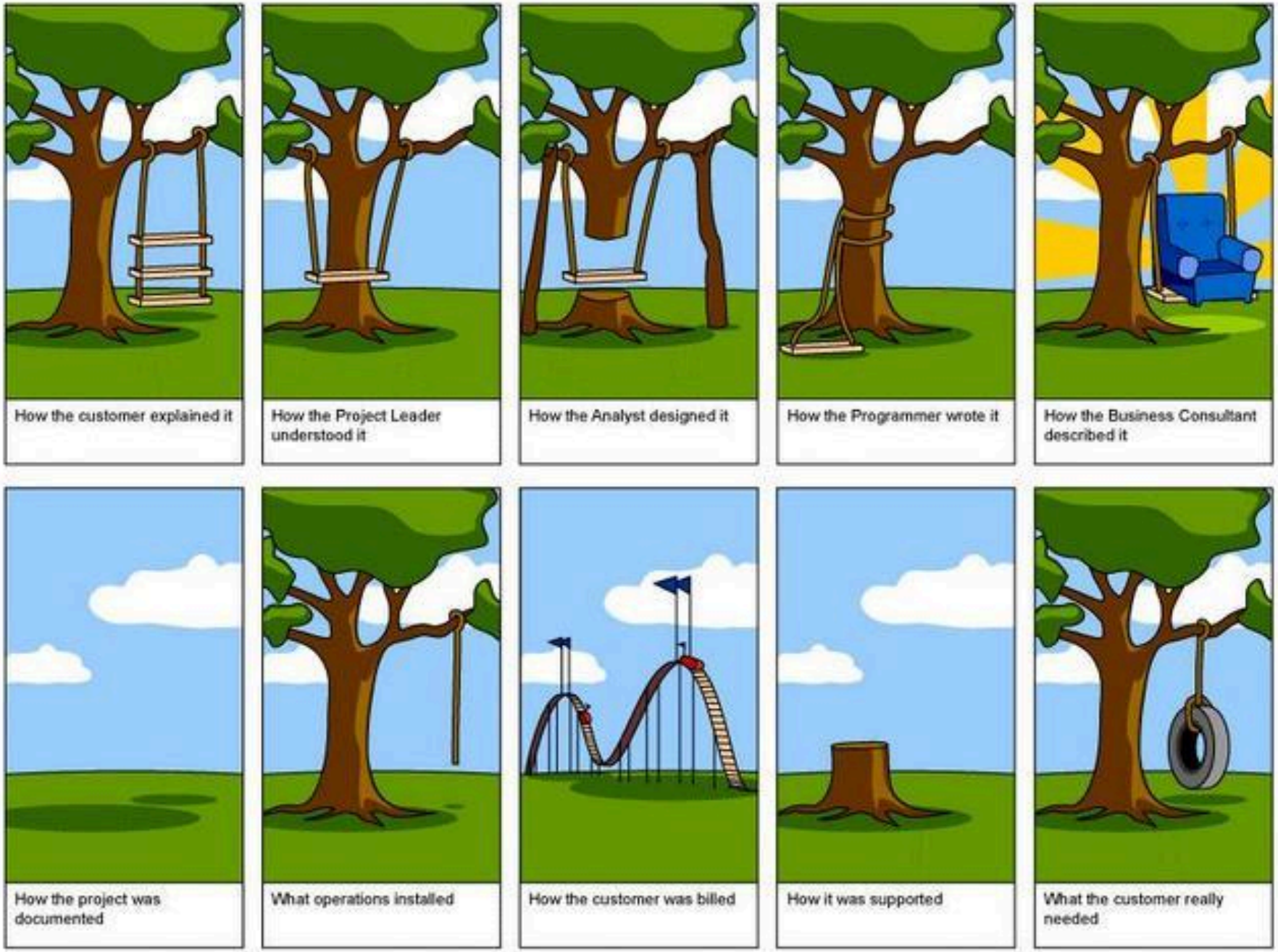
Problem / opportunity in the real world







Code


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Will LLMs change SE?

(or: a slide to laugh about in 5-10 years?)
(LLM == large language model. e.g., chatGPT)

Facts:

- LLMs do / will change SE
  - small standard programming tasks work well
    - however, similar to (google/stack overflow) or github Co-Pilot
    - obstacles concern more legal than technical issues: privacy, copyright, liability
  - OO design works sometimes well
    - challenge: specialised domains

Essential difference to current tools: LLMs take natural language texts as input !

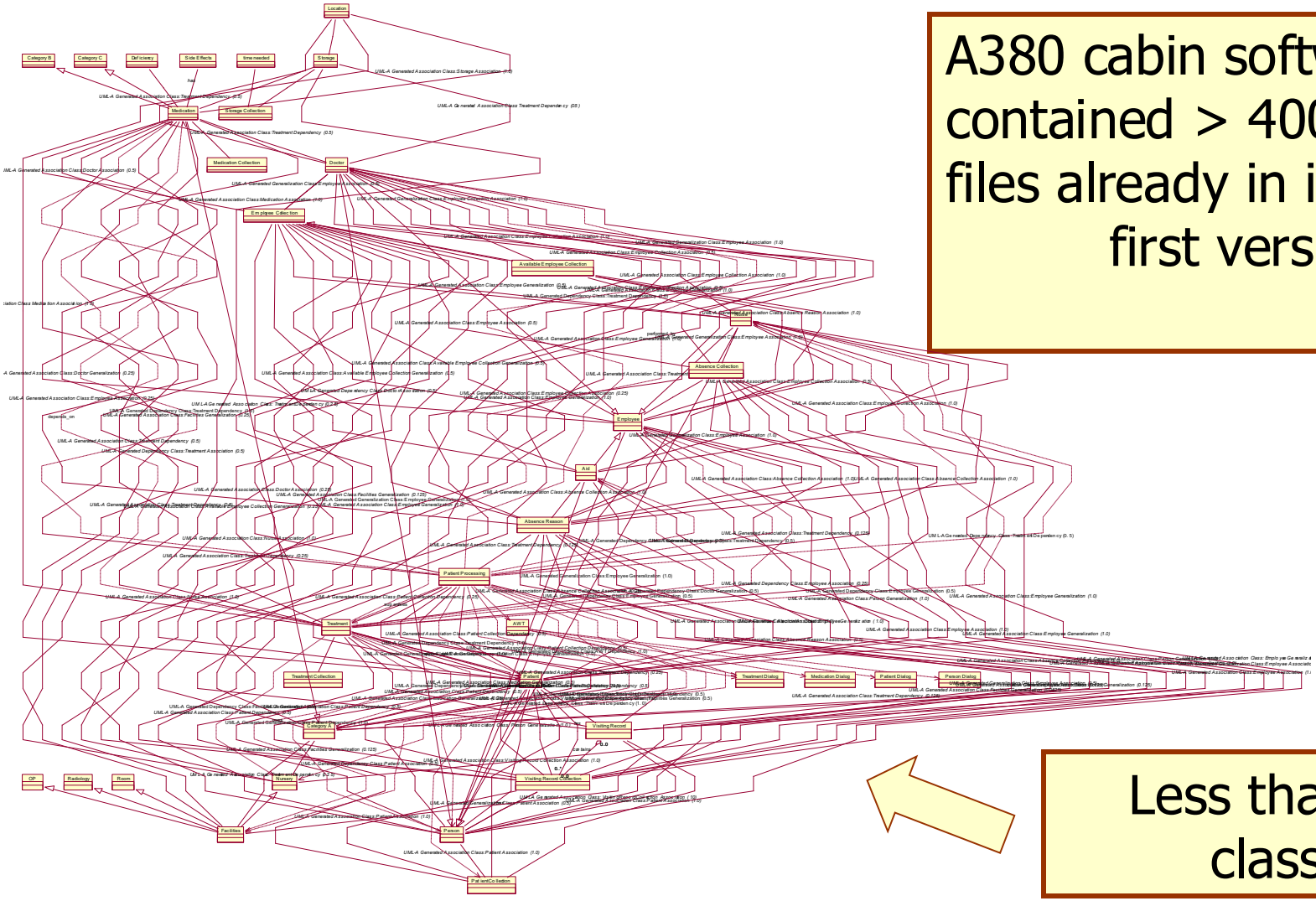
→ **Speculation:**

- Architectural design with LLMs ?
- Automatisation of whole SE process ?
  - what about requirements engineering?
  - what about quality assurance?

Current Assumption: the more general the task, the better the LLM
(the more domain specific / context dependent, the worse!)

You need to review the result before using it!

Teaser II: Taming Complexity



A380 cabin software contained > 4000 files already in its first version ...

Less than 50 classes!!!

Teaser III: Laws of SE

Or: We do know something after 50 years of software engineering

1. Putting more people on a late project makes it _____
2. Errors are more frequent during _____ phases
3. Testing can only show the presence of errors, but not their _____
4. A system that is used will be _____
5. Only what is _____ can be changed without risk

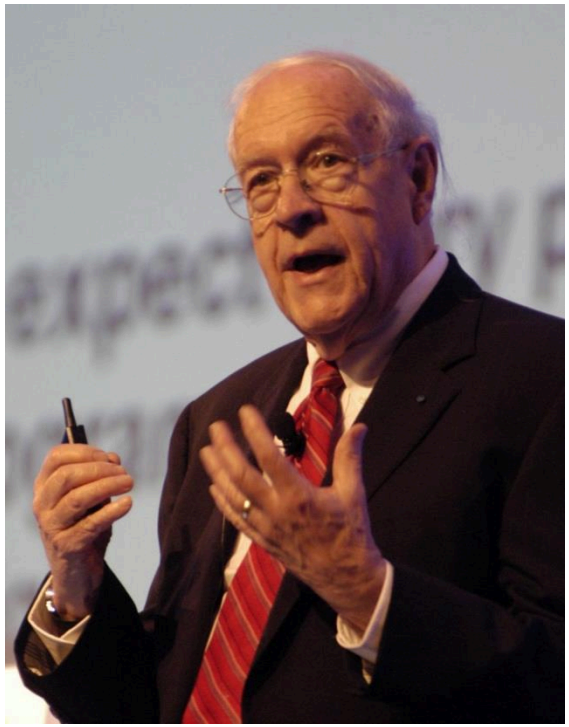
Teaser III: Laws of SE

Or: We do know something after 50 years of software engineering

1. Putting more people on a late project makes it ___later___
2. Errors are more frequent during ___requirements & design___ phases
3. Testing can only show the presence of errors, but not their ___absence___
4. A system that is used will be ___changed___
5. Only what is ___hidden___ can be changed without risk

Brook‘s Law

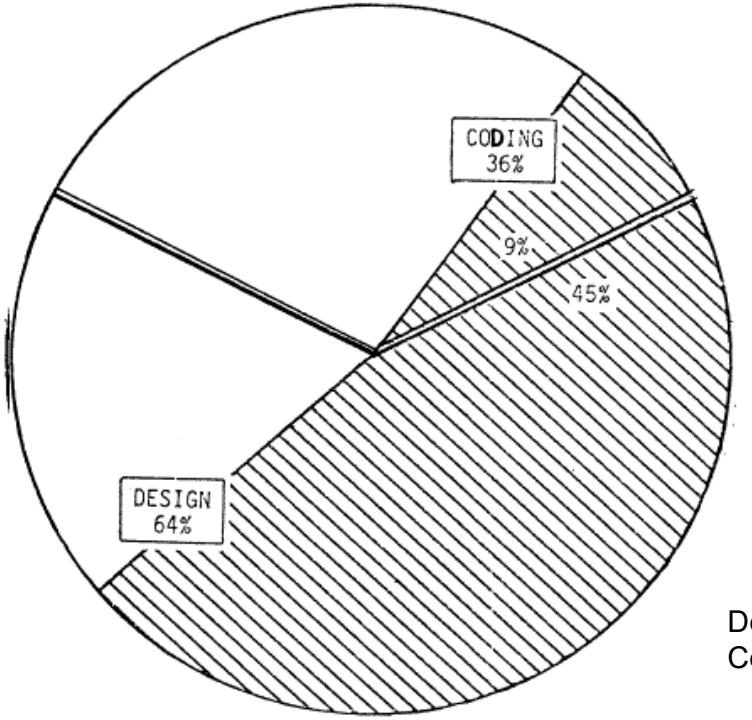
- “Adding manpower to a late software project makes it later”
  - From the book “Mythical Man Month” [Brooks 1975]
  - Reasons: Ramp-up time, communication overhead



Frederik Brooks [Image: Wikipedia]

Boehm‘s First Law

- Errors more frequent during requirements and design



Barry Boehm
[Image: Wikipedia]

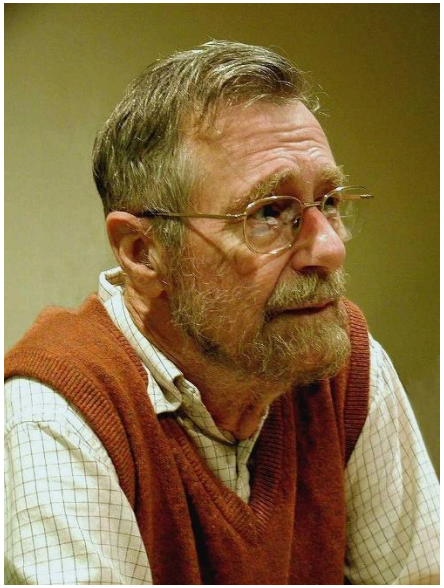
Design: Errors fixed by changing the design
Coding: Errors fixed by changing the code

From [Boehm 1975]

- And more difficult to fix
- ❼ Lectures on **requirements, architecture,** and **design**

Dijkstra 1969

- *"Testing shows the presence, not the absence of bugs"* [Dijkstra 1969]
- See also SE1 Chapter 5 5.1 – Testing and Verification



Edsger W. Dijkstra
[Image: Wikipedia]

(One of) Lehman‘s Laws

„A system that is used will be changed“

[phrasing as used by Endres and Rombach 2003]

- One of Lehman's eight Laws of Software Evolution
- Original wording: *“A program that is used and that, as an implementation of its specification, reflects some other reality, undergoes continuing change or becomes progressively less useful. The change or decay process continues until it is judged more cost effective to replace the program with a recreated version.”*

  – [Lehman 1980]

⇨ Lecture on **clean code**

Parnas' Law

- Information hiding principle, encapsulation principle
- *"Only what is hidden can be changed without risk"* [Parnas 1972]
  - phrasing as used by [Endres and Rombach, 2003]

- Related to modularization of software

- Previous view:
  - module should be small enough to be understandable
  - interfaces should be clearly specified

- Parnas' suggestion as described by [Endres and Rombach, 2003]
  - Modules should be independent
  - Each module should be changeable without impacting the others
  - Implementation details should be hidden
  - Interfaces should "reveal as little as possible about its inner workings"

- Lectures on **software architecture and quality**
- More about SE laws: [Endres and Rombach 2003]

Course Topics

Design & Realisation

- Clean Coding
- Software Architecture & Components
  - Clean Architecture
  - Enterprise Application Architecture
  - Microservices
  - Cloud Architecture
  - Software Container
- Domain-driven design

Processes and Requirements

- Development processes
- Agile development
- Continuous Integration
- Requirements engineering
- Use cases
- NLP

Ensuring Software Quality

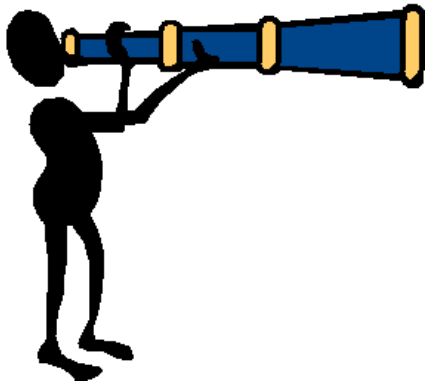
- Software Reliabilty
  - Real-time Systems
  - Real-time design patterns
- Software Security

Conclusion

- **Software development is a challenging activity**
  - still no routine

A variety of novel approaches have been developed

- to overcome this challenge
  - *state of the art will be presented in this course*



Clean Code

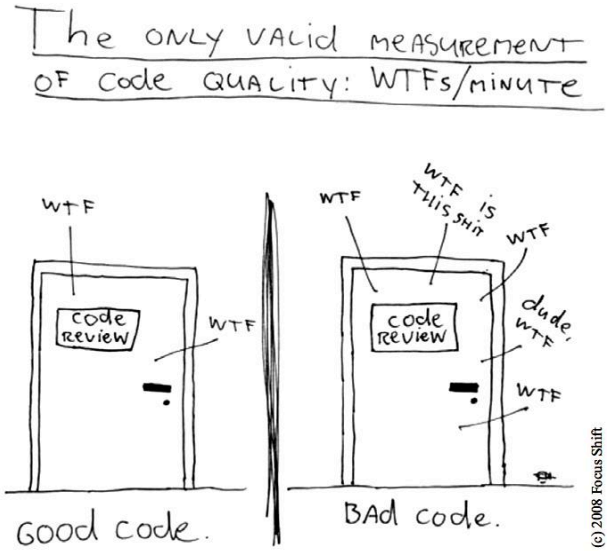


Image source: [Martin2008, p. xxv]