



University
of Basel

Software Engineering - Einführung

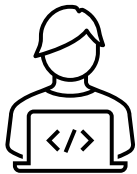
M. Lüthi, Universität Basel, 8. August 2022

Über mich



Anstellung Uni Basel

- Forschung im Bereich Statistische Formmodellierung (25%)
- Lehre (75%)
 - Programmieren
 - Algorithmen und Datenstrukturen
 - Software Engineering
 - Shape modelling and analysis
 - Diverse Seminare (Funktionale Programmierung / Machinelles Lernen)



Erfahrung in Software Engineering

- Anwendungsentwicklung UBS (1997 – 2002)
 - Informatikstudium FH-Bern, Schwerpunkt Software Engineering (1999-2003)
 - Open source Softwareentwicklung (2008 – ...)
 - Scalismo – Software zur statistischen Formmodellierung
-

slido



Please fill in the survey

① Start presenting to display the poll results on this slide.

Agenda 08. August 2022

-
- 1 Kursziele (20')
 - 2 Wie funktioniert der Kurs? (20')
 - 3 Warum ist Software Engineering wichtig? (30')
 - 4 Praktische Übung 1: Softwareumgebung aufsetzen / Studium: No silver bullet (90')
 - 5 Was ist Software Engineering? (20')
 - 6 Kurze Geschichte des Software Engineerings? (30')
-

Kursziele

Bauen

Cheops Pyramide

Gebaut: ca 2500 a.d.

Höhe: 146 m

Arbeiter: Hunderttausende

Bauzeit: ca 20 Jahre



This Photo by Unknown Author is licensed under CC BY-NC

Engineering

Empire state building

Gebaut: 1930

Höhe: 381m

Arbeiter: ca 3000

Bauzeit: 11 Monate



Kursziele

1. Softwareentwicklung als Ingenieursdisziplin zu verstehen

Software Engineering?



"Quite a bit of today's software and its construction process resemble the Egyptian pyramid, but I would dare to say no one currently knows how to organize 3000 programmers to make a major piece of software from scratch in less than 11 months."

Alan Kay, 2001

Kursziele

1. Softwareentwicklung als Ingenieursdisziplin zu verstehen.
 2. Schwierigkeiten und Limitierungen in der Softwareentwicklung zu verstehen.
-

Software Engineering?



This Photo by Unknown Author is licensed under CC BY

Computing spread out much, much faster than educating unsophisticated people can happen. In the last 25 years or so, we actually got something like a pop culture, similar to what happened when television came on the scene and some of its inventors thought it would be a way of getting Shakespeare to the masses. But they forgot that you have to be more sophisticated and have more perspective to understand Shakespeare. What television was able to do was to capture people as they were.

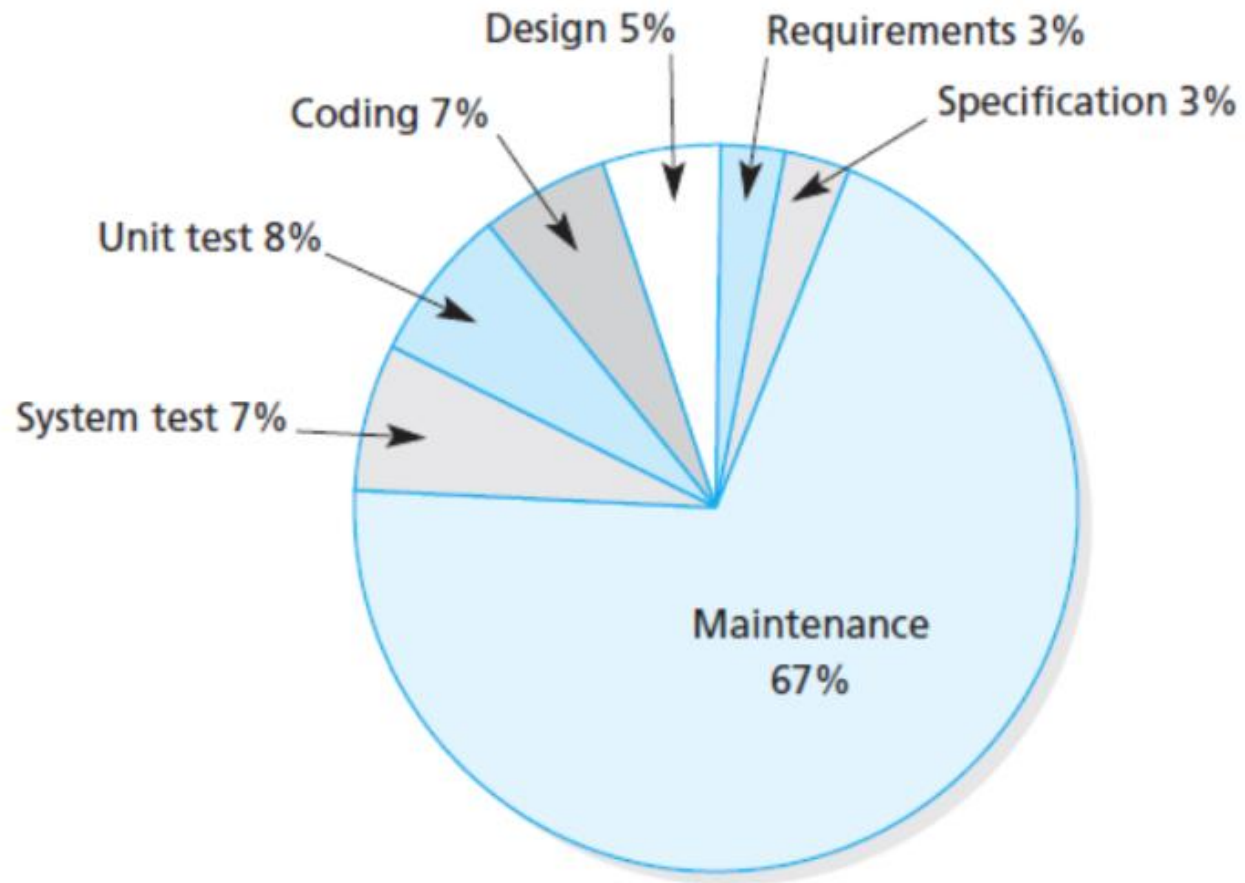
So I think the lack of a real computer science today, and the lack of real software engineering today, is partly due to this pop culture.

Allan Kay, 2004

Kursziele

1. Softwareentwicklung als Ingenieursdisziplin zu verstehen.
 2. Schwierigkeiten und Limitierungen in der Softwareentwicklung zu verstehen.
 3. Die Fundamente und Geschichte von Software Engineering kennenzulernen
 4. Lernen "Modeerscheinungen" von fundamentalen Konzepten zu unterscheiden.
-

Aktivitäten der Softwareentwicklung

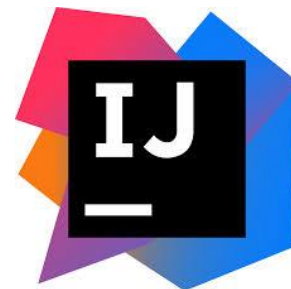


Quelle: Software Engineering for Students, D. Bell

Kursziele

1. Softwareentwicklung als Ingenieursdisziplin zu verstehen.
 2. Schwierigkeiten und Limitierungen in der Softwareentwicklung zu verstehen.
 3. Die Fundamente und Geschichte von Software Engineering kennenzulernen
 4. Lernen "Modeerscheinungen" von fundamentalen Konzepten zu unterscheiden.
 5. Lernen existierende Programme zu verstehen und zu erweitern.
-

Moderne Entwicklung

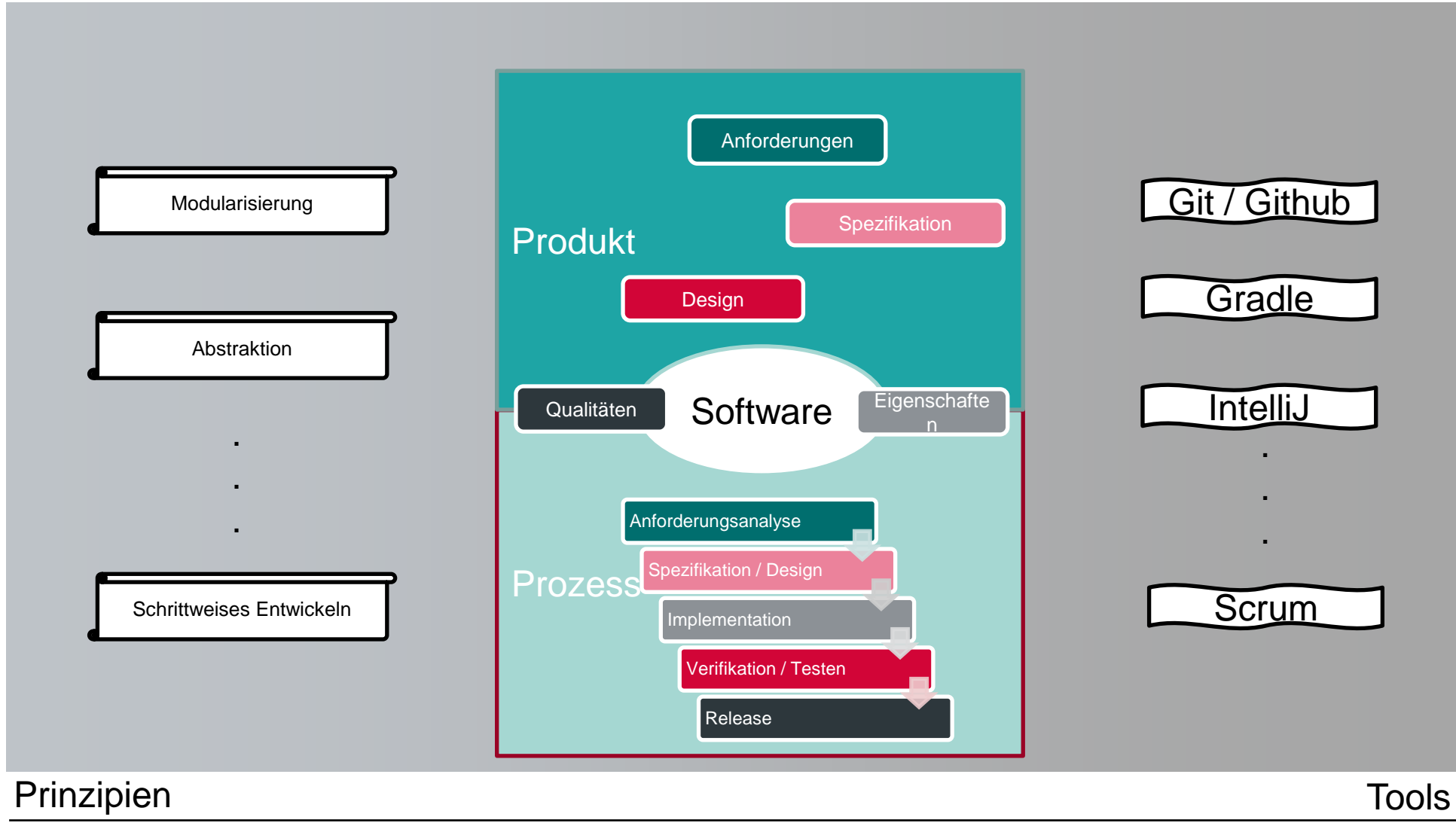


Kursziele

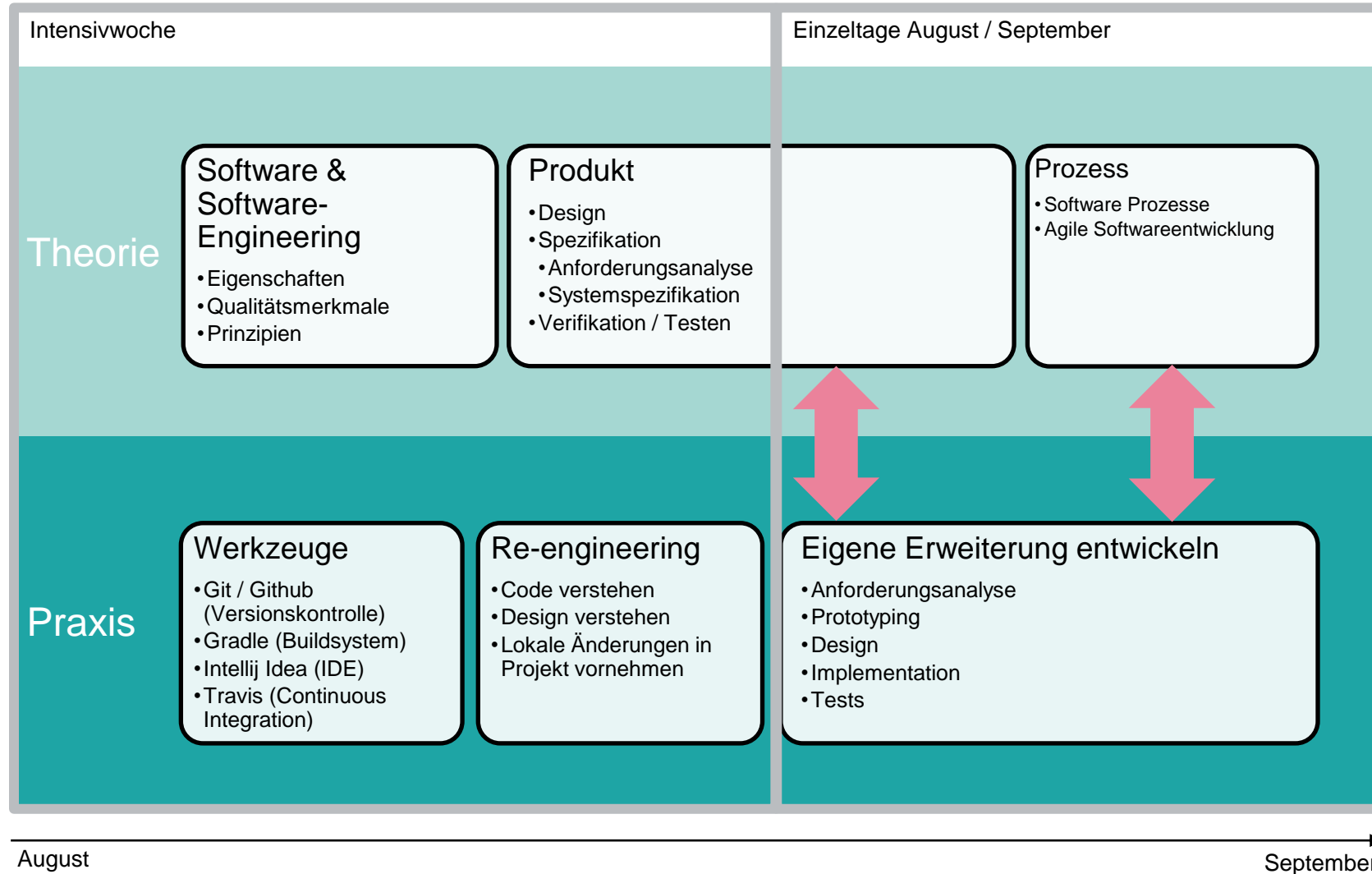
1. Softwareentwicklung als Ingenieursdisziplin zu verstehen.
 2. Schwierigkeiten und Limitierungen in der Softwareentwicklung zu verstehen.
 3. Die Fundamente und Geschichte von Software Engineering kennenzulernen
 4. Lernen "Modeerscheinungen" von fundamentalen Konzepten zu unterscheiden.
 5. Lernen existierende Programme zu verstehen und zu erweitern.
 6. Werkzeuge der modernen Softwareentwicklung kennenlernen
-

Wie funktioniert der Kurs

Kursübersicht



Kursübersicht



Leistungsüberprüfung

Theorie: 50%

- Schriftliche Prüfung zu Theorie des Software Engineerings / Konzepte / Zusammenhänge

Praxis: 50%

Implementation einer eigenen Erweiterung in Open Source Software JabRef

- (Vorbereitende Übungen)
 - Anforderungsanalyse
 - Design
 - Testplan
 - Umsetzung
-



Kursprojekt: JabRef

Stay on top of your Literature

The efficient way to collect, organize & discover

JabRef

File Edit Library Quality Lookup Tools View Options Help

Search...

Groups

- Filter groups
- All entries (210)
- By status (13)
- Last read (2)
- Skimmed (2)
- To be read (9)
- By rating (36)
- For Project X (2)
- For teaching (9)
- My papers (8)

Literature.bib jabref-authors.bib X

| Author/Editor | Title | Year |
|-----------------------|--|------|
| Schönberger et al. | Has WS-I's Work Resulted in WS-* Interoperability? | 2011 |
| Schönberger et al. | QoS-Enabled Business-to-Business Integration Using ebBP to WS-BPEL Translations | 2009 |
| Schönberger et al. | Proceedings of the 4th Central-European Workshop on Services and their Compositio... | 2012 |
| Sheikh and Gustafsson | Linear Programming Design of Coefficient Decimation FIR Filters | 2012 |
| Sheikh and Gustafsson | Design of narrow-band and wide-band frequency-response masking filters using spar... | 2010 |
| Simon et al. | Analyzing the Importance of Jabref Features from the User Perspective | 2019 |
| Skoien et al. | A computer vision approach for detection and quantification of feed particles in mari... | 2014 |
| Strauch et al. | Cloud Data Patterns for Confidentiality | 2012 |
| Strauch et al. | Non-functional data layer patterns for Cloud applications | 2012 |
| Strauch et al. | A Taxonomy for Cloud Data Hosting Solutions | 2011 |
| Sungur et al. | Supporting Informal Processes | 2014 |
| Sungur et al. | Extending BPMN for Wireless Sensor Networks | 2013 |

Required fields Optional fields Optional fields 2 General Abstract Comments Annotations biblatex source

Author Martin Simon and Linus W. Dietz and Tobias Diez and Oliver Kopp

Bibtexkey SimonDietzDiezEtAl2019 Generate

Booktitle ZEUS

Date 2019

Title Analyzing the Importance of Jabref Features from the User Perspective

Web search

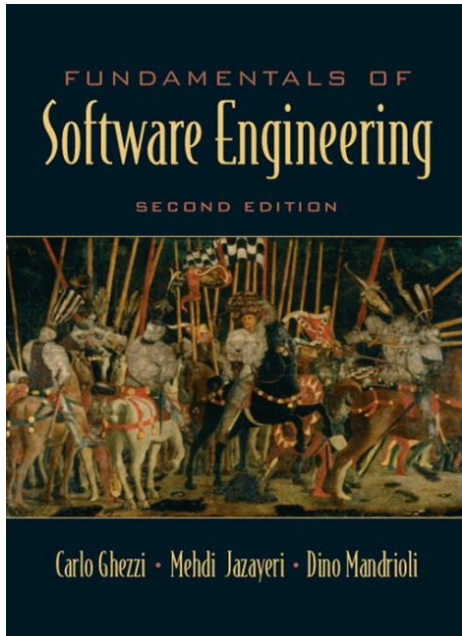
ArXiv

Search...

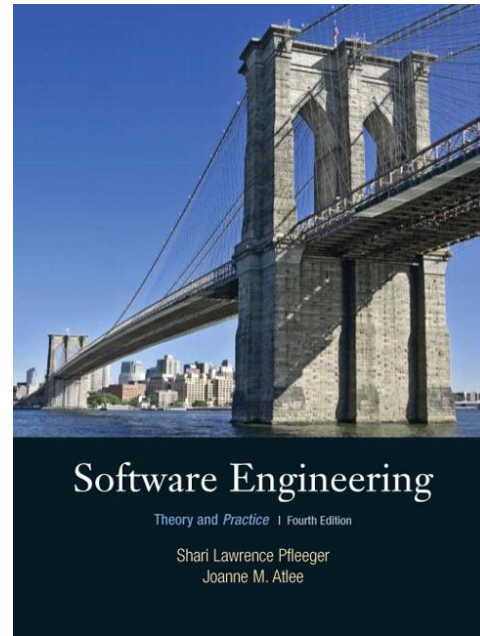
Search

Literatur

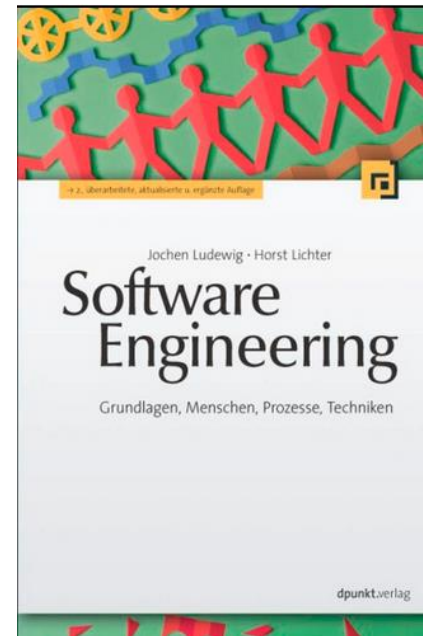
Unterrichtsmaterialien & Internet ausreichend für Vorlesung



Grundlage der
Vorlesung



Modernes
Standardwerk



Deutsches Lehrbuch

Vorlesungsseite

[View on GitHub](#) 

Software Engineering - GymInf 22

Vorlesungsseite für die Vorlesung Softwaretechnik im Rahmen des Programms GymInf

Softwaretechnik

Departement Mathematik und [Informatik](#), Universität Basel

Dozent: Marcel Lüthi (marcel.luethi@unibas.ch)

[Diese Vorlesungsseite auf Github](#)

[Diskussionsforum](#)

Kurzbeschreibung

Die Vorlesung gibt eine erste Einführung in das Software Engineering. Das Hauptziel der Vorlesung ist die Studierenden mit den Grundprinzipien der Softwaretechnik, sowie, zu einem gewissen Grad, auch mit der Geschichte des Gebiets vertraut zu machen. Aktuelle Methoden und Tools werden im Rahmen eines vorlesungsbegleitenden Projekts eingeführt.

Projekt

Das Open Source Projekt Jabref ([Offizielles Github Repositories](#)) dient als Grundlage für die praktischen Übungen. Die Teilnehmenden bringen kleine Änderungen am Projekt an, studieren den Source Code und Implementieren dann eigene Erweiterungen.

Einen Überblick über die einzelnen Projektschritte finden Sie auf der [Projektseite](#).

Programm

| | Themen | Bemerkungen |
|---------------------------|--------------------------------------|-------------|
| Halbtag 1 | Einführung / Administratives | |
| | Was ist Software engineering | |
| Halbtag 2 | | |
| | Eigenschaften von Software | |
| Halbtag 3 | Prinzipien des Software engineerings | |

<https://unibas-marcelluethi.github.io/software-engineering-gyminf/>

Fragen / Anmerkungen

Was habe ich vergessen?

Was ist noch nicht ganz klar?

Warum ist Software Engineering wichtig?



In short, software is eating the world

— *Marc Andreessen* —

AZ QUOTES

Bekannte Beispiele

Welche dieser Branchen wird von Softwareunternehmen dominiert? Nenne ein Unternehmen.

- Buchhandel
 - Videoverleih
 - Filmindustrie
 - Recruiting
 - Transport
 - Hotel
-

Warum jetzt?

- 80 Jahre Computer
- 60 Jahre Mikroprozessoren
- 40 Jahre Internet
- 20 Jahre Smartphones

Technologien sind reif

Technische Infrastruktur besteht

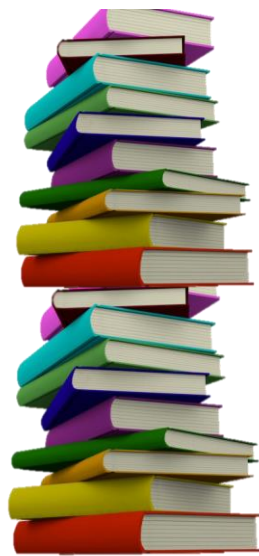


Von Taxiarchos228 - Eigenes Werk, FAL, <https://commons.wikimedia.org/w/index.php?curid=44866422>

Komplexität von Software



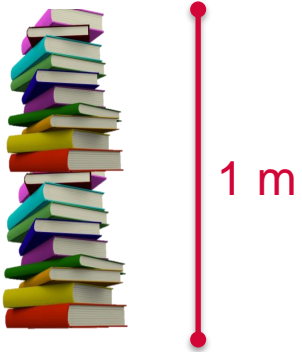
Buch mit 400 Seiten
20'000 Zeilen Code



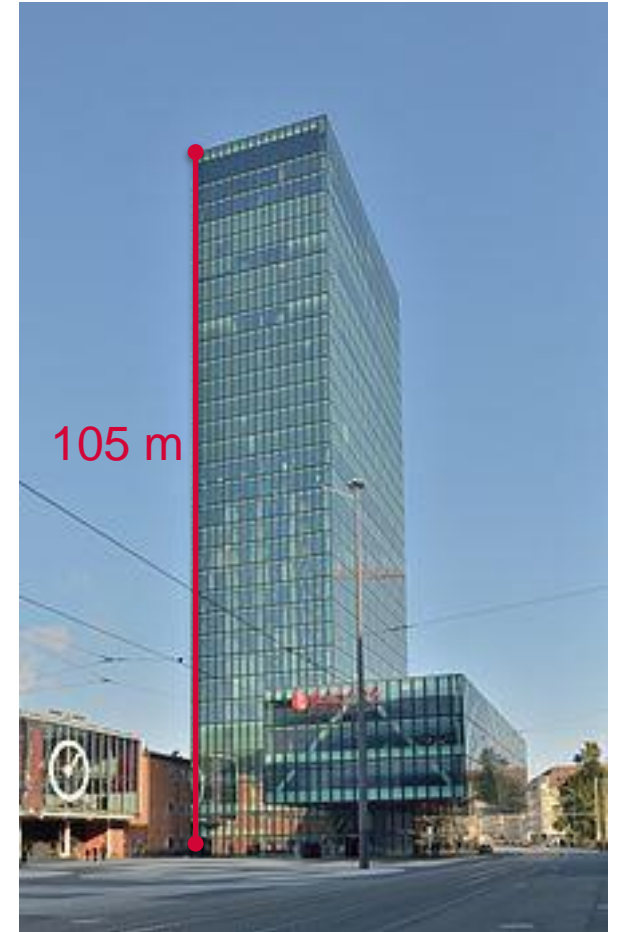
1m

1 Meter Stapel mit Büchern
1 Million Zeilen Code

Komplexität von Software



| | |
|---------------|---------------------------------------|
| JabRef: | 150'000 Zeilen Code (7 Bücher, 15cm) |
| VSCode: | 1 Million Zeilen Code (1 m) |
| Firefox: | 16 Millionen Zeilen (16 m) |
| Linux Kernel: | 32.4 Millionen Zeilen (32 m) |
| Mac OSX: | 85 Millionen Zeilen (85 m) |



Probleme der Softwarekomplexität

Wie *versteht* man ein Programm das über 1000 Bücher verteilt ist?

Wie *ändert* man ein Programm das über 1000 Bücher verteilt ist?

Wie *findet man Fehler* in einem Programm das über 1000 Bücher verteilt ist?



Quelle: Pinterest
<https://www.pinterest.com/pin/438256607464529059/>

Praktisch Übung: Softwareumgebung aufsetzen

Aufgabe (Zeit: 90 Minuten)

[View on GitHub](#) 

Software Engineering - GymInf 22

Vorlesungsseite für die Vorlesung Softwaretechnik im Rahmen des Programms GymInf

Halbtag 1: Einführung in Softwaretechnik

Slides

- Kursziele und Administratives
 - Slides: [pdf](#)
- Software Engineering
 - Slides: [pdf](#)

Übungen

- Aufsetzen von JabRef gemäss dieser [Anleitung](#)
- Lesen des Artikels [No silver bullet - Essence and Accidence in Software Engineering](#)

Literatur

- [No silver bullet - Essence and Accidents in Software Engineering](#)

Was ist Software Engineering

Software Engineering

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

IEEE Standard Glossary of Software Engineering Terminology

The multi-person construction of multi-version software.

David Parnas, 1978

Programmieren vs. Software Engineering



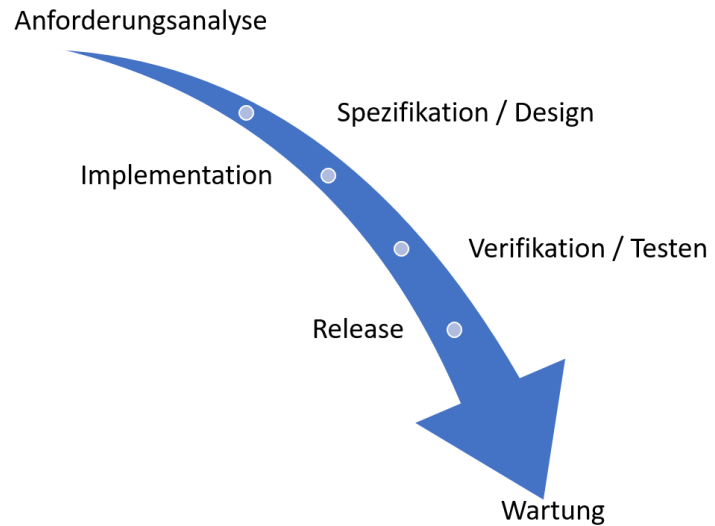
Programmieren im Kleinen



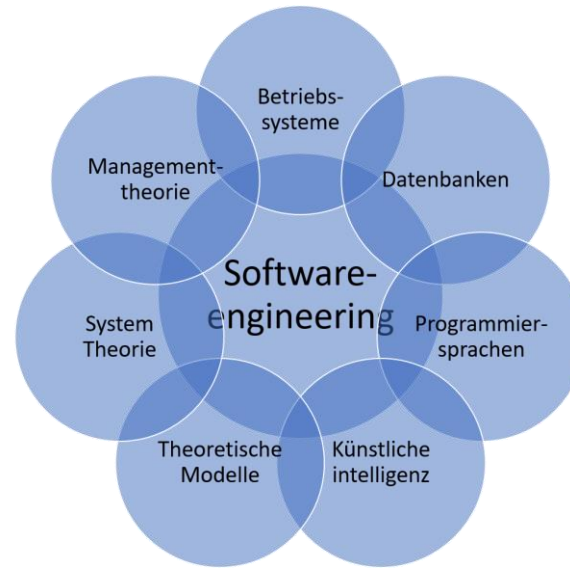
Programmieren im Grossen
= Software Engineering



Software-Engineering ist ...



Mehr als Programme schreiben



Im Kern der Informatik,
aber mehr als nur Informatik



Teamarbeit (Interdisziplinär)

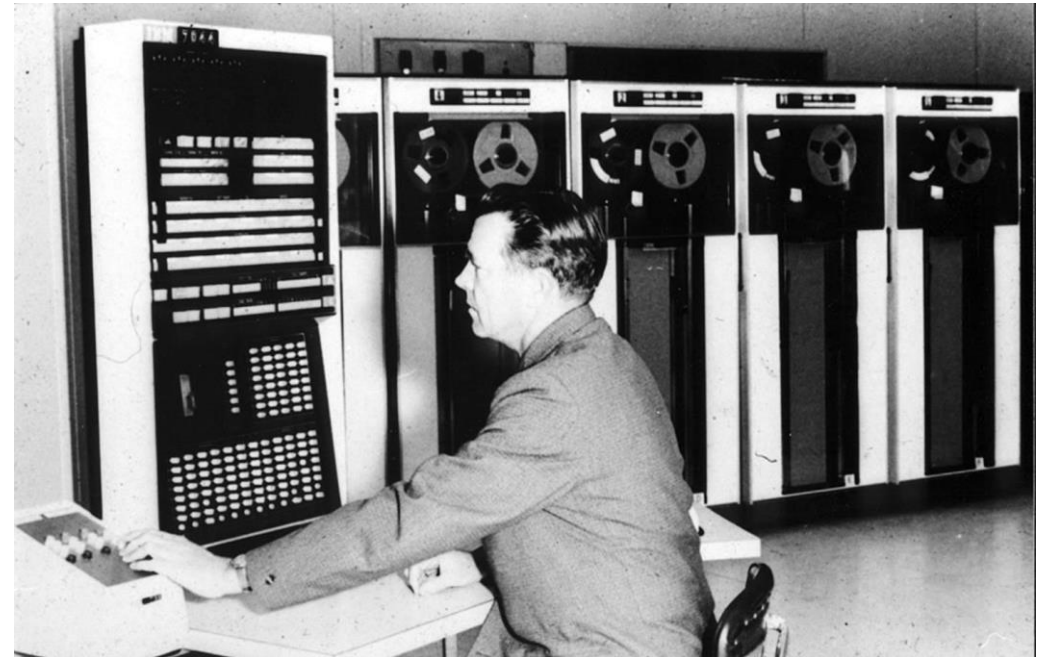
Kurze Geschichte des Software Engineerings

Historischer Kontext

Innovation löst immer ein Problem

- Problem und Lösung entstehen in Kontext

Historischer Kontext hilft Methodik / Ansatz einzuordnern



This Photo by Unknown Author is licensed under [CC BY-ND](#)

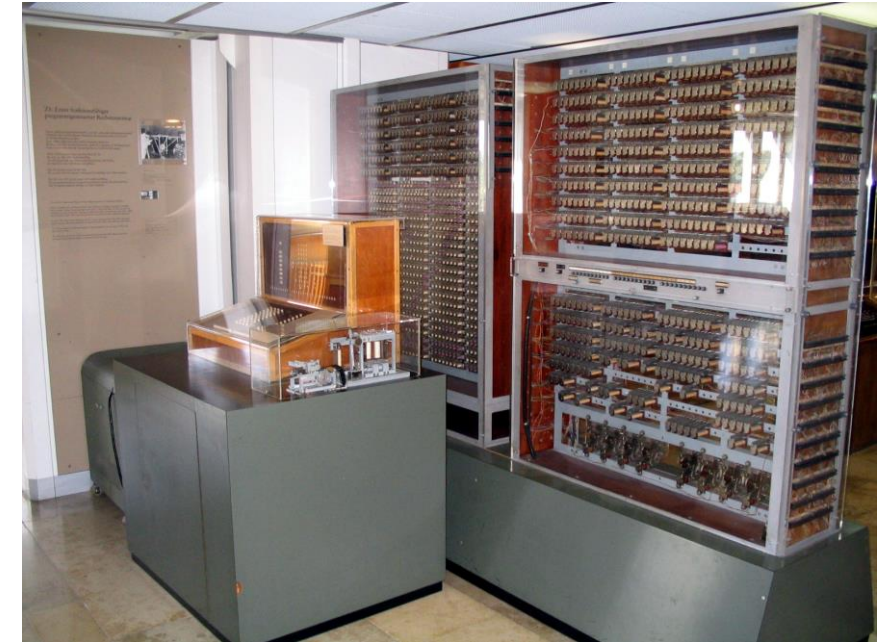
Vor 1950

1:1 zwischen Programmierer und Computer

- Kein Software-Engineering - Nur Programmierung
- Wohldefinierte Probleme
 - Beispiel: Lösen einer Differentialgleichung
- Programmierer waren meistens Physiker

Programmiersprachen

- Assembler



Vor 1950: Beispielprogramm

```
FACTO CSECT
      USING FACTO,R13
SAVEAREA B STM-SAVEAREA(R15)
      DC 17F'0'
      DC CL8'FACTO'
STM STM R14,R12,12(R13)
      ST R13,4(R15)
      ST R15,8(R13)
      LR R13,R15      base register and savearea pointer
      ZAP N,=P'1'      n=1
LOOPN CP N,NN      if n>nn
      BH ENDLOOPN      then goto endloop
      LA R1,PARMLIST
      L R15,=A(FACT)
      BALR R14,R15      call fact(n)
ZAP F,0(L'R,R1)      f=fact(n)
DUMP EQU *
MVC S,MASK
ED S,N
      MVC WTOBUF+5(2),S+30
MVC S,MASK
ED S,F
      MVC WTOBUF+9(32),S
      WTO MF=(E,WTOMSG)
AP N,=P'1'      n=n+1
B LOOPN
ENDLOOPN EQU *
RETURN EQU *
      L R13,4(0,R13)
      LM R14,R12,12(R13)
      XR R15,R15
      BR R14
```

```
FACT EQU *      function FACT(I)
      L R2,0(R1)
      L R3,12(R2)
      ZAP L,0(L'N,R2)      l=n
      ZAP R,=P'1'      r=1
      ZAP I,=P'2'      i=2
LOOP CP I,L      if i>l
      BH ENDLOOP      then goto endloop
MP R,I      r=r*i
AP I,=P'1'      i=i+1
B LOOP
ENDLOOP EQU *
      LA R1,R      return r
      BR R14      end function FACT
      DS 0D
NN DC PL16'29'
N DS PL16
F DS PL16
C DS CL16
II DS PL16
PARMLIST DC A(N)
S DS CL33
MASK DC X'40',29X'20',X'212060' CL33
WTOMSG DS 0F
      DC H'80',XL2'0000'
WTOBUF DC CL80'FACT(..)=.....'
L DS PL16
R DS PL16
I DS PL16
      LTORG
      YREGS
      END FACTO
```

1950-1960

- Programmierer:in wird zum Beruf
- Programmieren bleibt single-player Game
 - Neu: Programmierer != User
- Erste grosse Software Projekte entstehen

Anforderungen müssen kommuniziert werden

Programmiersprachen

- Fortan, Cobol, Lisp



1950 - 1960

Programmiersprachen

Fortran, LISP

```
FUNCTION FACT(N)
INTEGER N,I,FACT
FACT=1
DO 10 I=1,N
10 FACT=FACT*I
END
```

```
(defun fact (n)
  (if (< n 2)
      1
      (* n (fact(- n 1)))))
```

1960-1970

- Erste grosse, kommerzielle Softwaresysteme
- Grenzen des Programmierens werden ersichtlich.
 - Programmiertechniken skalieren nicht
- Begriff der Softwarekrise

Probleme:

- Kommunikationsoverhead
 - Was passiert wenn Programmierer geht
 - Teueres "on boarding"
 - Änderungen in einem System beeinflusst andere Systeme
-

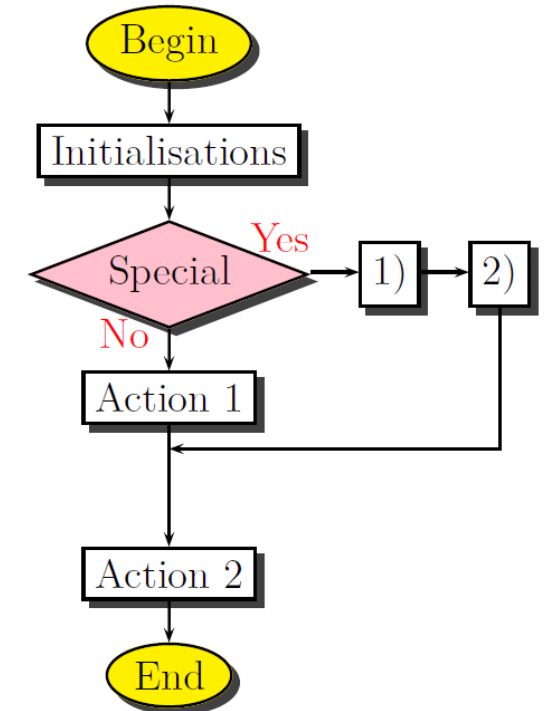
1960-1970

Lösungsansätze

- Teamorganisation
- Neue Programmiersprachen
- Programmierrichtlinien
- Formale Modelle

Software Engineering wird erfunden

Kommunikation/Informationsfluss und Modularisierung werden wichtig.



This Photo by Unknown Author is licensed under [CC BY-SA](#)

1960-1970

Programmiersprachen

Simula, Basic, PL/I

```
factorial: procedure (N) returns (fixed decimal (30));  
  declare N fixed binary nonassignable;  
  declare i fixed decimal (10);  
  declare F fixed decimal (30);  
  
  if N < 0 then signal error;  
  F = 1;  
  do i = 2 to N;  
    F = F * i;  
  end;  
  return (F);  
end factorial;
```

1970-1990

Stetiger Fortschritt

- Bessere Sprachen / Tools
 - Mainstream: Strukturierte Programmierung
 - Forschung: Objektorientierte Programmierung
- Besseres Verständnis der Prozesse

Einsicht: Es ist schwierig! (There is no silver bullet)



1970-1990

Programmiersprachen

- C, Smalltalk, ML

```
int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; ++i)  
        result *= i;  
    return result;  
}
```

```
fun factorial n =  
    if n <= 0 then 1  
    else n * factorial (n-1)
```


1990 - Heute

Computer sind günstig und allgegenwärtig

- Internet immer verfügbar
- Open source als Entwicklungsmodell
- Programmiersprachen und Tooling verbessern sich enorm

Agile Methoden statt schwerfälliger Prozesse.

1990 - Heute

Programmiersprachen

- Python, Haskell, Java

```
factorial :: Integral -> Integral
factorial 0 = 1
factorial n = n * factorial (n-1)
```

```
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
```

Heute - ???

- BigData / Cloud computing
 - Programmieren über Computergrenzen hinweg
- Deep-Learning / Differenzierbares und probabilistisches Programmieren
 - Neue Programmiermodelle ergänzen traditionelles Programmieren
- Programmsynthese wird besser – Large Language Models
 - Spezifikation wird wichtig
- Internet of Things
 - Security wird wichtig

