University of Basel

Software Engineering

Marcel Lüthi, Universität Basel

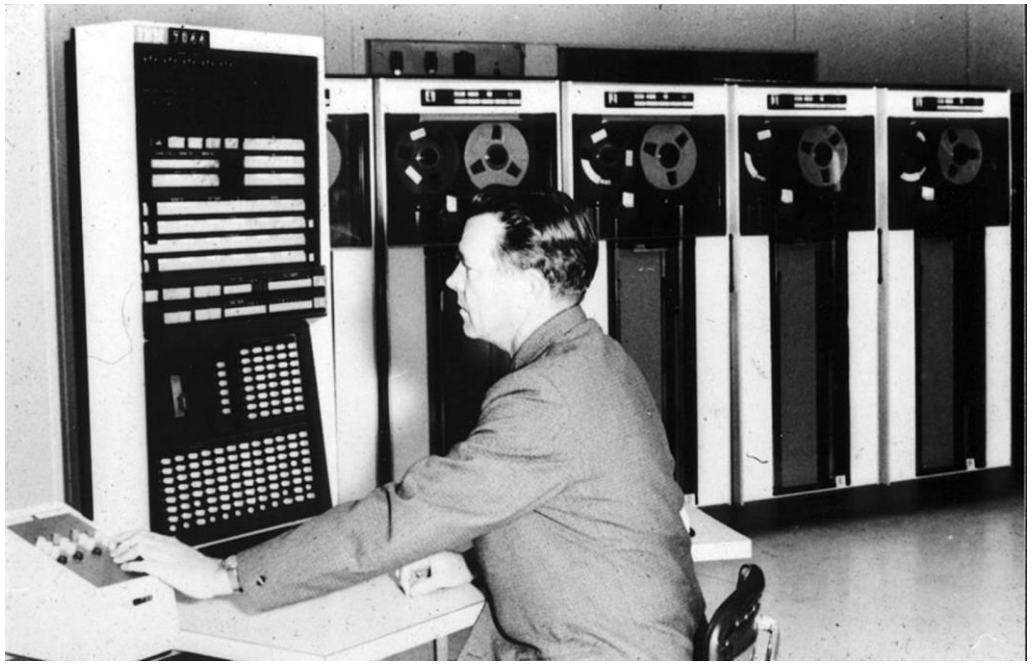
Kurze Geschichte des Software Engineerings

Historischer Kontext

Innovation löst immer ein Problem.

- Problem und Lösung entstehen in Kontext.

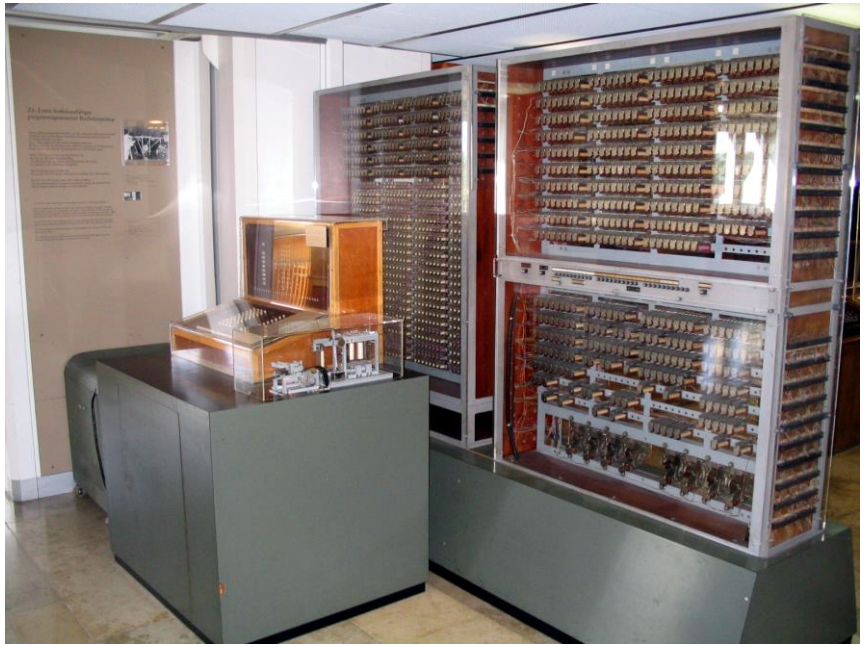
Historischer Kontext hilft Methodik und Ansatz einzuordnern.



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
                  ZAP    N,=P'1'
 LOOPN    CP     N,NN
                  BH     ENDLOOPN
                  LA     R1,PARMLIST
                  L      R15,=A(FACT)
                  BALR   R14,R15
ZAP    F,0(L'R,R1)
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'
B      LOOPN
ENDLOOPN EQU    *
RETURN   EQU    *
                  L      R13,4(0,R13)
                  LM     R14,R12,12(R13)
                  XR     R15,R15
                  BR     R14
```

```
FACT     EQU
                  L      R2,0(R1)
                  L      R3,12(R2)
                  ZAP    L,0(L'N,R2)
                  ZAP    R,=P'1'
                  ZAP    I,=P'2'
 LOOP     CP     I,L
                  BH     ENDLOOP
MP     R,I
AP     I,=P'1'
B      LOOP
 ENDLOOP  EQU    *
                  LA     R1,R
                  BR     R14
                  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

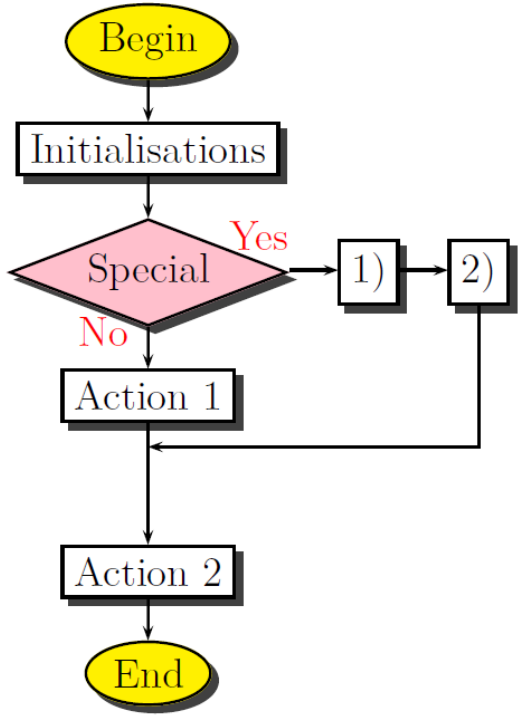
Probleme:

- 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.

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 - ???

- Big Data / Cloud computing
  - Programmieren über Computergrenzen hinweg

- Deep-Learning / Large Language models
  - Programme werden aus Daten gelernt
  - Spezifikation wird wichtig

- Internet of Things
  - Security wird wichtig