8. Softwareentwicklung

(Software Engineering)

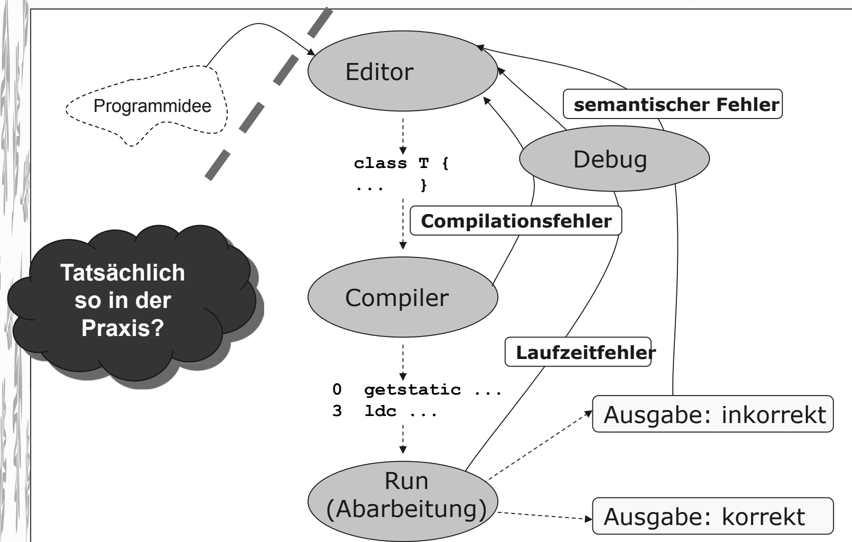
Einbettung der Programmierung in der Gesamtprozess der Softwareentwicklung

Schwerpunkte

- Problematik komplexer Programme
- Statistische Angaben zur Softwareentwicklung
- Eigenschaften von Software
- Qualitätskriterien für SW-Produkte
- Vorgehensmodelle der SW-Entwicklung: Phasen und Produkte

Einbettung der Programmierung in der Gesamtprozess der Softwareentwicklung

Programmentwicklung: Edit-, Compile-, Run-, Debug-Zyklus



Software

gdRead.f90 - Editor

Datei Bearbeiten Format Ansicht ?

```
! $Id: gdRead.f90,v 1.1.1.1 2003/03/06 19:38:16 maine Exp $
! $Name:  $

!-- 19 Oct 92, Richard Maine: Version 1.0.
!-- 30 Jun 93, Richard Maine: add gd_read_trace for diagnostics.
!-- 22 Jul 93, Richard Maine: add special case fast sync.
!-- 30 Dec 93, Richard Maine: jump_tolerance adapts for low sample rates.
!-- 23 Jun 94, Richard Maine: revised status code values.
!-- 7 Jul 94, Richard Maine: revisions in gd_common
!-- 30 Aug 94, Richard Maine: add verbose argument to calc_gd.
!-- 11 Jan 95, Richard Maine: add source argument to file_gd.
!-- 30 Jan 95, Richard Maine: add input%aug_signal, gd%source array
!-- 13 Mar 95, Richard Maine: add signals arg to inquire_gd.
!-- 30 Jun 97, Richard Maine: change default stop_time to 30 days.
!-- 6 Aug 98, Richard Maine: fix fast sync bug, time tol to 1.e-6

module gd_read

  !-- Module for read access to get_data objects.
  !-- A get_data object is an abstraction of a time history data file.
  !-- It may involve, merging, skewing, interpolating, and calculating.
  !-- 6 Aug 98, Richard Maine.

  use precision
  use sysdep_io
  use fdas_string
  use fdas_time
  use gd_tree
  use gd_common
  use gd_filter
  use gd_calc
  use gd_file
  use th_read, only: thr_ok, thr_warn, thr_eod, thr_eoi, thr_error
```

gdRead.f90 - Editor

Datei Bearbeiten Format Ansicht ?

```
    !-- For each input with signals to be filtered...
    input => gd%first_input
    file_loop: do while (associated(input))
      source_number = input%source_number
      n_filtered = count(source_numbers == source_number)
      file_filtered: if (n_filtered /= 0) then

        !-- Open the filter.
        call add_gd_filter(input%filter_list, filter_name, filter_linker, &
             input%source_dt, filter_parameters, &
             pack(output_names, source_numbers==source_number), &
             pack(signal_numbers, source_numbers==source_number), &
             input%aug_signal, input%file_interpolate, &
             gd%source, gd%signal, gd%signal_tree, error)
        if (error) return

      end if file_filtered
      input => input%next_input
    end do file_loop

    !-- Normal exit.
    error = .false.
    return
  end subroutine filter_gd

  subroutine calc_gd (gd_handle, calc_name, calc_linker, &
       calc_parameters, calc_string, verbose, source, error)

    !-- Connect a calculated function set to a get_data object.
    !-- 31 Jan 95, Richard Maine.

    !-------------------- interface.
    type(gd_handle_type), intent(in) :: gd_handle  !-- handle from open_gd.
    character*(*), intent(in) :: calc_name
         !-- calculated function set name.
    character*(*), intent(in), optional :: calc_linker
         !-- calculated function set linker.
    real(r_kind), intent(in), optional :: calc_parameters(:)
         !-- Interpretation of these parameters depends on the calc routines.
         !-- Many calc routines ignore them.
    character*(*), intent(in), optional :: calc_string
         !-- Interpretation of this parameter depends on the calc routines.
         !-- Many calc routines ignore it.
    logical, intent(in), optional :: verbose
         !-- Should extra diagnostic output be printed.  Default is F.
    type(gd_source_type), intent(out), optional :: source
         !-- Corresponding source record.
    logical, intent(out) :: error
         !-- Errors always generate at least one message also.

    !-------------------- local.
    integer :: source_number

    !-------------------- executable code.

    call check_gd_handle(gd_handle,gd)

    gd%maps_built = .false.
    gd%ok_to_read = .false.

    saved_trace = gd_read_trace
    gd_read_trace = 'calc_load'
    call add_gd_calc(gd%calc_list, calc_name, calc_linker, &
           calc_parameters, calc_string, gd%source, &
           gd%signal, gd%signal_tree, verbose, source_number, error)
    gd_read_trace = saved_trace
    if (error) return
    if (present(source)) source = gd%source(source_number)
    return
  end subroutine calc_gd

  subroutine check_gd_handle (gd_handle,gd)

    !-- Check that a gd_handle is for a legal, opened object.
    !-- If ok, return gd as a pointer into gd_table.
    !-- 3 Dec 90, Richard Maine.

    !-------------------- interface.
    type(gd_handle_type), intent(in) :: gd_handle
    type(gd_type), pointer :: gd  !-- intent(out)

    !-------------------- executable code.

    !-- Check for illegal handles.
    gd => gd_handle%gd
    if (.not. associated(gd)) call error_halt('Null gd_handle')

    return
  end subroutine check_gd_handle
end module gd_read
```


gdRead.f90 - Editor

Datei Bearbeiten Format Ansicht ?

Beispiele: Größe von Softwaresystemen

	Mio SLOC
Space shuttle (2010)	0.4
Windows NT 3.1 (1993)	4
Windows NT 4.0 (1996)	12
Windows XP (2001)	40
Windows Vista (2007)	50
Windows 8 (2012)	60
Mac OS X	86
Car software, high-end (embedded system, 2014)	100
SAP NetWeaver (2007)	238

SLOC: Source Lines of Code

Quelle: http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Probleme der Komplexität von Software

Software fehlerhaft, zu spät fertig, teurer als gedacht

- **F18 Kampfflugzeug:** 1983 während einer Übung mit der neu entwickelten Flugzeugsoftware – beim Überflug des Äquators stellte sich das Flugzeug auf den Kopf. Der Grund war ein Vorzeichenfehler im Programm (Wallmüller 1990, S. 1).
- **Flughafen von Denver:** konnte (Anfang 1994) nicht eröffnet werden - die Software für das Gepäcktransportsystem funktionierte nicht (Folgekosten ...).
- **LKW-Maut-System** (Toll Collect) zwei Jahre später fertig als geplant: statt 2003 erst 2005 – Software wurde nicht fertig

Fragen: Statistische Angaben zur Softwareentwicklung (1)

(Fragen nach Mynatt: Software Engineering, S. 1)

1. **Ein typisches (durchschnittliches) Softwareentwicklungsprojekt dauert**
   - a) 1-6
   - b) 6-12
   - ➡ c) 12-24
   - d) 24-48 Monate
2. **Für ein Software-System mittlerer Größe werden im Durchschnitt**
   - ➡ a) weniger als 10
   - b) 10-100
   - c) 100-300
   - d) mehr als 300

   Zeilen Quellcode pro Person und pro Tag während der gesamten Periode der Entwicklungszeit des Systems produziert.

Fragen: Statistische Angaben zur Softwareentwicklung (2)

(Fragen nach Mynatt: Software Engineering, S. 1)

3. **Die durchschnittliche Anzahl von Fehlern in 1000 Zeilen Quellcode während der Entwicklung eines Software-Systems beträgt**
   - a) weniger als 30
   - b) 30-40
   - c) 40-50
   - ➡ d) 50-60
4. **Die durchschnittliche Anzahl von Fehlern in 1000 Zeilen Quellcode in einem ausgelieferten System beträgt**
   - ➡ a) weniger als 4
   - b) 4-8
   - c) 8-12
   - d) mehr als 12

Fragen: Statistische Angaben zur Softwareentwicklung (3)

5. **Wie viele Software-Systeme, mit deren Entwicklung begonnen wurde, werden auch beendet?**
   - a) 90-100 %
   - b) 80-90 %
   - ➡ c) 70-80 %
   - d) 60-70 %
6. **Die meisten Software-Fehler sind zurückzuführen auf**
   - a) Programmierfehler
   - ➡ b) Probleme beim Verständnis des Problems
   - c) Zeitdruck
   - d) Fehler im Entwurf der Software (Struktur)

Fragen: Statistische Angaben zur Softwareentwicklung (4)

7. **Wie hoch ist der zeitliche Anteil der Programmierung bei der Softwareentwicklung?**
   a) 99 % b) 70 % c) 50 %
   ➡ d) 20 % e) 10 %

8. **Die Kosten für die Wartung von Software sind typischerweise**
   a) halb so b) genauso
   ➡ c) doppelt so
   hoch wie die Entwicklungskosten für die Software

   (*Wartung* = Änderung, Weiterentwicklung von Software nach ihrer Fertigstellung)

Frage: Defektrate

Defektrate
= Anzahl der Fehler auf 1000 Zeilen Quellcode

- Die Defektrate hat sich aufgrund der steigenden Komplexität der Software von 1977 bis 1994 wie folgt entwickelt:
  a) auf das 10fache gestiegen
  b) auf das Doppelte gestiegen
  c) gleich geblieben
  ➡ d) auf 1% gesunken
  (Grund: Fortschritte des Software Engineering)

Konsequenzen

→ Nach Erhalt einer SW-Entwicklungsaufgabe:
- nicht sofort "drauflos" programmieren
- zunächst gründlich die Aufgabe durchdenken, Unklarheiten beim Kunden nachfragen ...

→ Programmierung macht bei der SW-Entwicklung nur einen kleineren Anteil aus.
Genauso wichtig sind Fähigkeiten wie
'exakte Problembeschreibung finden', ‚systematische Fehlersuche' u. a.

→ Bereits bei der Programmierung muss an die Zeit nach der Auslieferung der SW gedacht werden (Wartung):
lesbare, gut strukturierte, gut kommentierte Programme

→ SW-Entwicklung bedeutet immer 'Teamarbeit'

Komplexität von Software

- "Softwaresysteme gehören zu den komplexesten Gebilden, die je von Menschenhand geschaffen wurden. Strukturen und Abläufe in großen Systemen sind im einzelnen oft nicht mehr überschaubar. Man kann sie weder im vorhinein, beim Entwurf, noch im nachhinein, beim Testen, im Betrieb und in der Wartung vollständig verstehen."
  (Denert, Software-Engineering, S. 4)

- "Das entscheidende Charakteristikum der industriell einsetzbaren Software ist, daß es für den einzelnen Entwickler sehr schwierig, wenn nicht gar unmöglich ist, alle Feinheiten des Designs zu verstehen. Einfach ausgedrückt, überschreitet die Komplexität solcher Systeme die Kapazität der menschlichen Intelligenz."
  (Booch, Objektorientierte Analyse und Design, S.18)

Eigenschaften von Software:

(vgl. Balzert, S. 26 f., Denert)

Software =
Programme, Daten, Dokumentation

- **kein Verschleiß**
  (keine Abnutzung beim Einsatz)
- **leicht kopierbar**
  (→ auch Fehler)
- **altert** *)
  (SW wird ständig angepasst)
- **lange in Benutzung** **)
- **schwer zu vermessen**
  (Metriken: Qualität, Quantität)
- **äußerst komplex**
  (in der Praxis)

***)** nach 10 Jahren Einsatzzeit existiert oft keine Originalzeile mehr (Faustregel, nach Pagel, Six S. 35)

****)** durchschnittlich 10 - 15 Jahre, (Wallmüller 90, S. 3)
bis zu 30 Jahren (Hausi Müller)

Klassifikation der Softwaregröße:

(A. Macro, S. 70)

klein:	bis 2000 Zeilen Quelltext
mittel:	2000 - 100 000
groß:	100 000 - 1 000 000
sehr groß:	> 1 Mio

Durchschnittliche Größe verwalteter Software unter den 100 größten USA-Firmen:
35 Mio Zeilen Quellcode
(Hausi Müller, S. 3-12)

Qualitätskriterien für SW-Produkte

(nach Pagel/Six, Pomberger, Meyer, Marciniak)

- Korrektheit
- Robustheit (z. B. bei Fehlbedienung)
- Effizienz
- Benutzerfreundlichkeit
- Modifizierbarkeit
- Lesbarkeit (Verständlichkeit)
  (Kommentare, sinnvolle Bezeichnerwahl, Formatierung ...)
- Wiederverwendbarkeit
- Modularität
  (zerlegt in Module / Komponenten)
- Portabilität
  übertragbar auf anderen Rechner
- ...

Modelle der Softwareentwicklung

Alternative Begriffe:

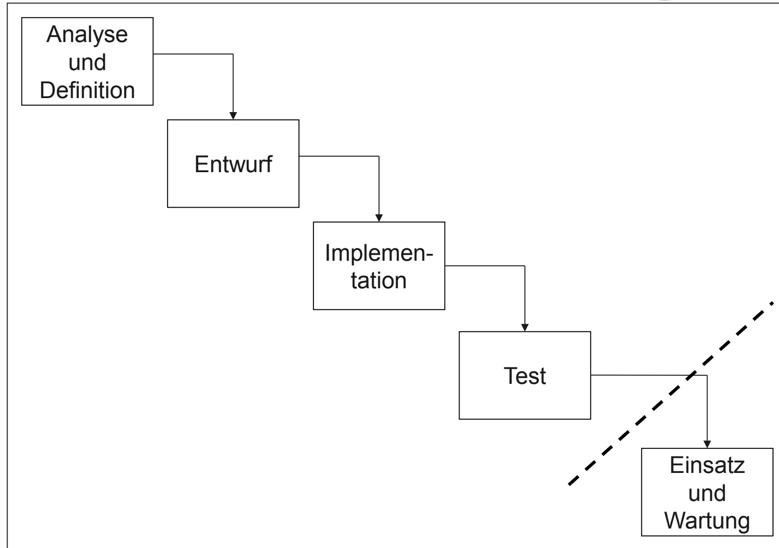
- Vorgehensmodelle
- Phasenmodelle
- Lebenszyklusmodelle

→ Ziel: systematische Software-Entwicklung
→ wesentlich für Entwicklung komplexer Software

(vgl. Probleme: SW ist fehlerhaft, zu spät fertig ...)

Erfahrung: Ohne ein systematisches Herangehen an die SW-Entwicklung keine qualitativ hochwertige Software!

Klassisches Wasserfallmodell für die Softwareentwicklung



SW-Entwicklung: Phasen und Produkte

- **Analyse und Definition**

  Analyse des Problems + Definition der Anforderungen an das SW-Produkt
  Gegenstand: Außenverhalten des SW-Systems
  Kooperation: Auftraggeber ↔ Auftragnehmer
  ➔ Produktdefinition
  (Systemspezifikation, Anforderungsdefinition, Pflichtenheft)

- **Entwurf (Design)**

  Struktur, Aufbau, Komponenten der SW und ihre Beziehungen festlegen
  ➔ Software-Architektur

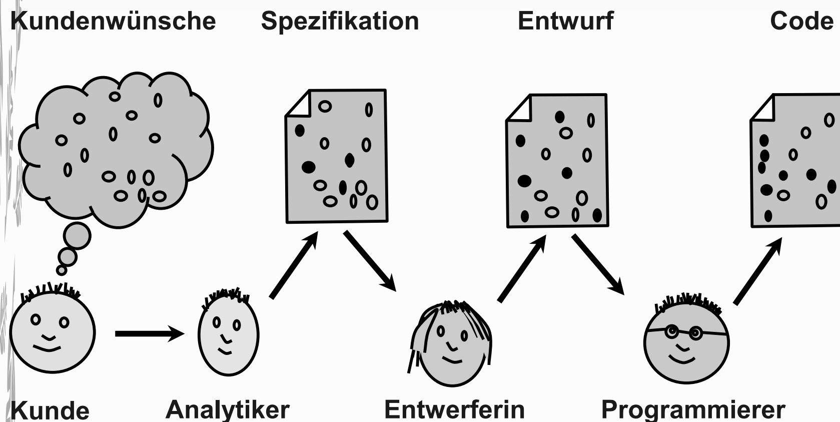
- **Implementation** GdP

  Software-Architektur "ausgefüllt": Programmierung der Komponenten
  ➔ Programm

- **Test**

  Test der Komponenten, Test ihrer Integration, Systemtest
  ➔ Testprotokolle: Testfälle, Ergebnisse des Testlaufs

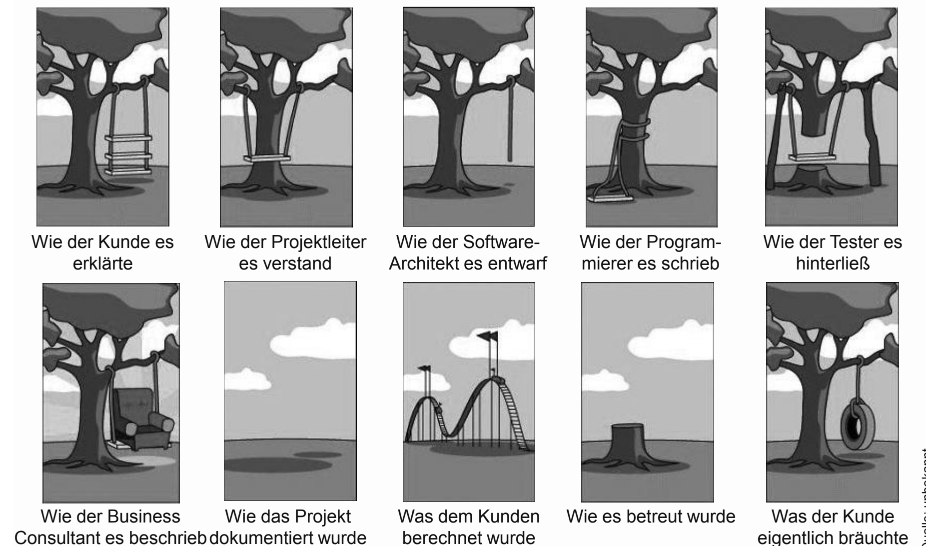
Dokumente der SW-Entwicklung



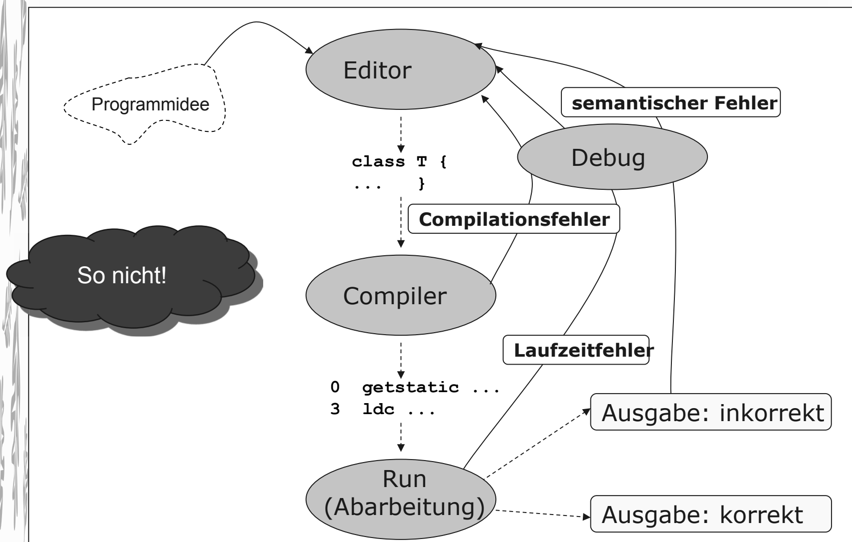
Geforderte Eigenschaften (leer) gehen verloren, unnötige (gefüllt) kommen hinzu

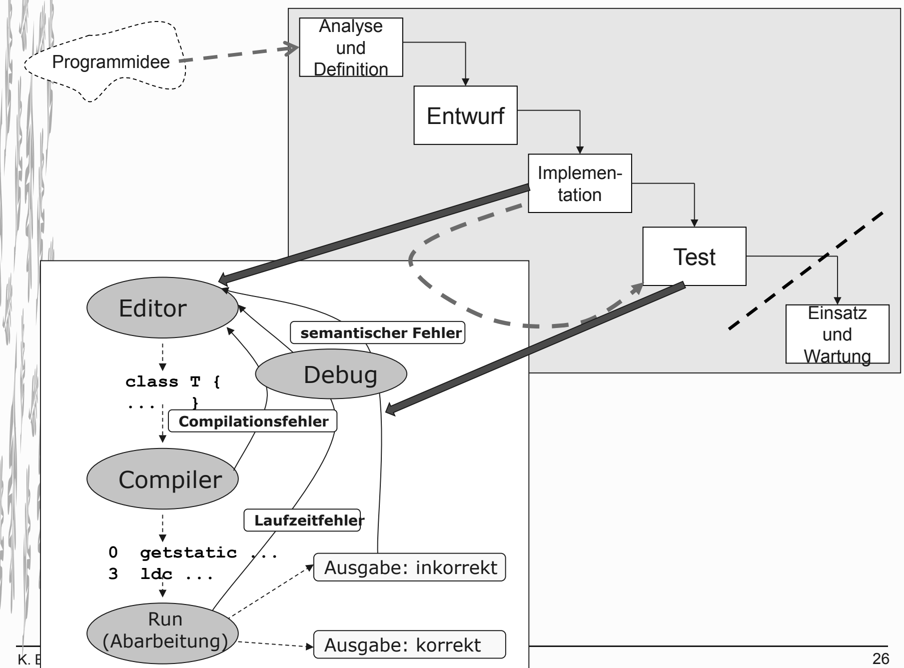
Quelle: Schneider, SEUH 43, S. 123

Informationsübertragung im Laufe eines Software-Projektes



Programmentwicklung: Edit-, Compile-, Run-, Debug-Zyklus





Studentische Softwareentwicklung
- zu Beginn des Studiums -

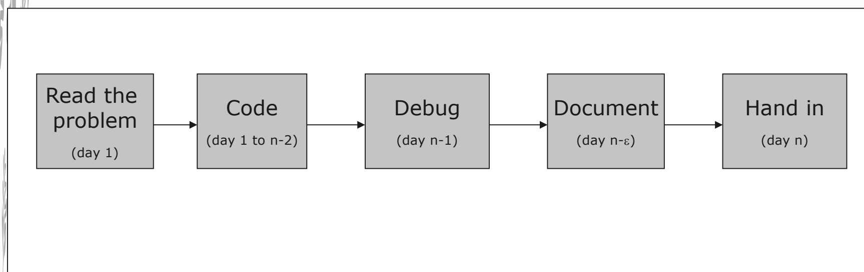


Fig. 3. "Normal" student view of software life cyle

(Quelle: Modesitt, in LNCS, S. 42)

Softwareentwicklung in Job-Angeboten

The Legacy. The Technology. The Opportunity.

Northrop Grumman Electronic Systems in Rolling Meadows, Illinois is a worldwide leader in smart electronic systems research, design, development and manufacture. With a diverse array of projects and a growing team of talented people, we're continuing the tradition of technological innovation. The opportunity is now at Northrop Grumman.

Software Engineering Opportunities

The following opportunities require a BS in CS/Physics/EE/Math and a minimum of 3 years 'C' programming, software development for real-time multi-tasking/multi-processor and embedded systems experience. Electronics warfare industry experience a plus.

Software Architects

Initiate design projects and perform requirements analyses, algorithm development and high level design. An advanced technical degree and a minimum 5 years experience in the development of large-scale real-time embedded software systems required. Knowledge of computer hardware architectures, performance simulation, modeling and exposure to knowledge based expert systems and object-oriented programming techniques a plus.

C++, C & ADA Software Developers

Design, implement and test real-time embedded and non-embedded software systems using C, C++ and Ada. Knowledge of modern software design methodologies preferred, however active process and design training activities are available.

Embedded Software Developer

Design, code, test and implement embedded real-time software systems using modern software design methodologies.

Software Test Engineers

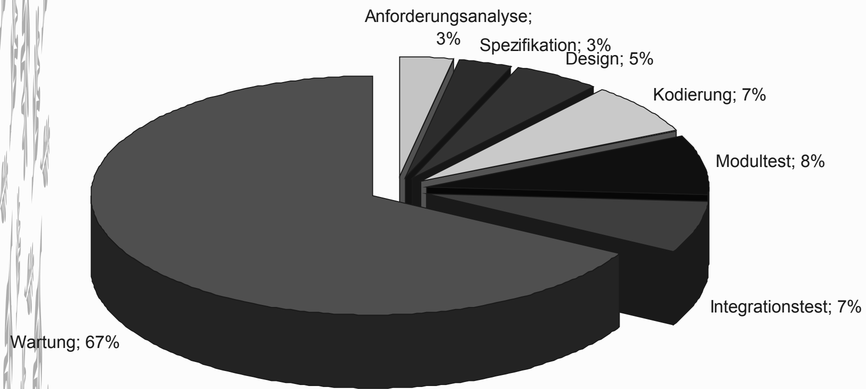
Analyze test requirements, develop test documentation and perform test dry runs. Familiarity with auditing/verifying compliance via SEI/ISO9000 and other industry/military standards essential. 8 years experience in software development with the three most recent years in a significant contributing capacity preferred.

If you want to take your talent to new levels and work on projects that are important, join the professionals at Northrop Grumman ESID. As a part of our team, you will enjoy a competitive salary and benefits package which includes health/major medical/dental/life insurance, 401(k) and pension plans, as well as opportunity for advancement. Excellent relocation package also available. Please send/fax/e-mail (ASCII text only) your resume to: **Northrop Grumman ESID, Dept. ACM298, 600 Hicks Rd., M/S U3100, Rolling Meadows, IL 60008-1098. FAX: 847/590-3189. E-mail: resumes@eiws.esid.northgrum.com** U.S. citizenship required for most positions. An equal opportunity employer m/f/d/v. **Visit our website at: www.northgrum.com/recruit**

NORTHROP GRUMMAN

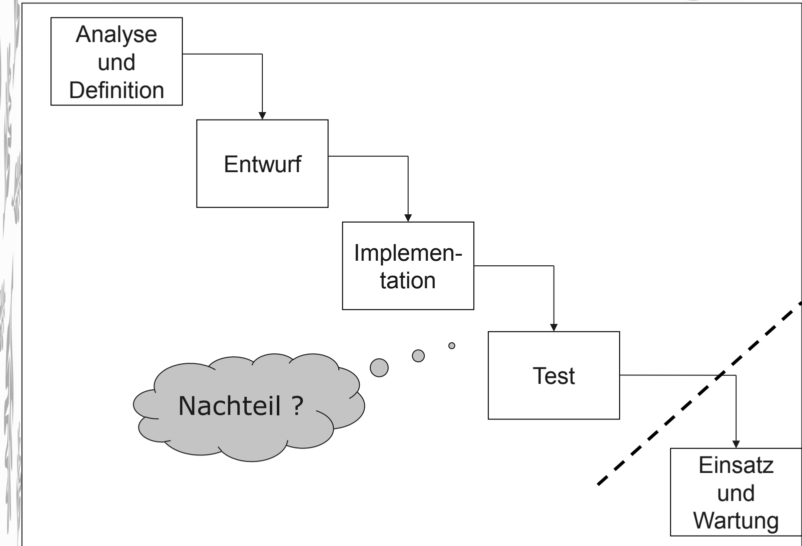
(Quelle: Communications of the ACM)

Kostenverteilung im Software Life-Cycle

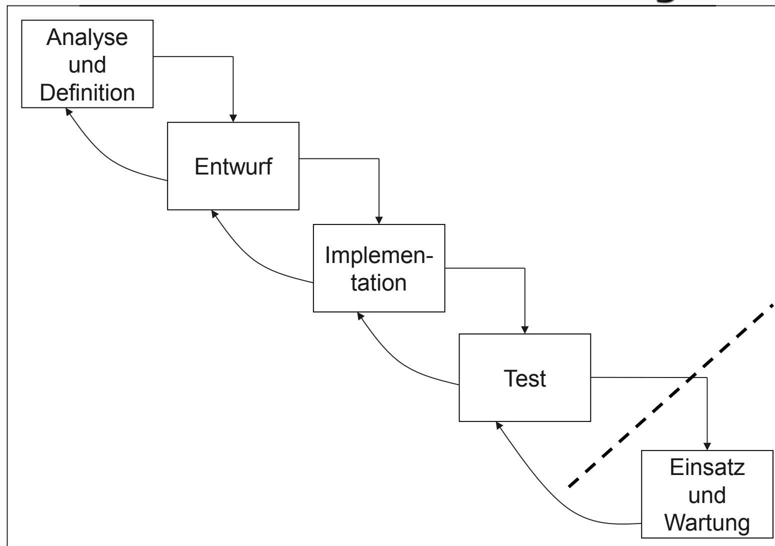


Quelle: Klösch, Gall, 1995, S.3

Klassisches Wasserfallmodell für die Softwareentwicklung



Iteratives Phasenmodell für die Softwareentwicklung



Vorgehensmodelle: Überblick

- Klassisches Phasenmodell (Wasserfallmodell)
- Iteratives Phasenmodell (Lebens**zyklus**)
- Spiralmodell
- Prototyping (evolutionäre Softwareentwicklung)
- Inkrementelle Softwareentwicklung (z.B. RUP)
- V-Modell

➔ Vorlesung "Software Engineering"