TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Grundlagen Software Engineering

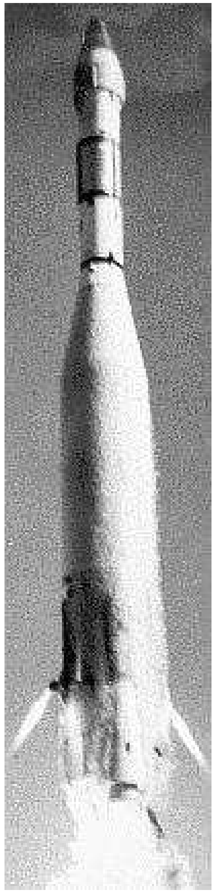
Einführung und Überblick

ENGINEERING
SOFTWARE
DEPENDABILITY

Grundlagen des Software Engineering
Einführung und Überblick

- ☐ Eckdaten der Softwarebranche in Deutschland
- ☐ Was ist Software?
- ☐ Der Wandel im Automobilbereich
- ☐ Mariner 1 und Ariane 5
- ☐ Was ist Softwaretechnik?
- ☐ Ziele der Lehrveranstaltung

Mariner 1



☐ *22. Juli 1962, Cape Canaveral/Florida*

☐ Start der ersten amerikanischen Venussonde Mariner 1

☐ Trägerrakete Atlas-Agena B (NASA, 15. AAB-Start)

Explosion von Mariner 1 Ausschnitt aus dem FORTRAN-Programm zur Steuerung der Flugbahn der Trägerrakete

```
    ...
    IF (TVAL .LT. 0.2E-2) GOTO 40
    DO 40 M = 1, 3
    W0 = (M-1)*0.5
    X = H*1.74533E-2*W0
    DO 20 N0 = 1, 8
    EPS = 5.0*10.0**(N0-7)
    CALL BESJ(X, 0, B0, EPS, IER)
    IF (IER .EQ. 0) GOTO 10
20 CONTINUE
    DO 5 K = 1. 3
    T(K) = W0
    Z = 1.0/(X**2)*B1**2+3.0977E-4*B0**2
    D(K) = 3.076E-2*2.0*(1.0/X*B0*B1+3.0977E-4*
   *(B0**2-X*B0*B1))/Z
    E(K) = H**2*93.2943*W0/SIN(W0)*Z
    H = D(K)-E(K)
 5 CONTINUE
10 CONTINUE
    Y = H/W0-1
40 CONTINUE    ...
```

Explosion von Mariner 1

- Fehler: Komma gegen Punkt vertauscht in der Zeile
  DO 5 K = 1. 3; korrekt wäre: DO 5 K = 1, 3
- Wirkung:
  - Wertzuweisung an eine nicht deklarierte Variable: DO5K = 1.3 (Kein Problem in FORTRAN)
  - Kein Durchlauf der (nicht vorhandenen) Schleife
- Folgen:
  - Abweichung der Trägerrakete von der vorgesehenen Flugbahn
  - Zerstörung der Rakete nach 290 Sekunden
  - Kosten: US$ 18,5 Millionen
- Ursache: Programmiersprache FORTRAN
  - Blanks (Zwischenräume) in Namen und Zahlen erlaubt
  - Variablen-Deklarationen nicht notwendig
  - Strukturierte Schleifen (while ...) nicht möglich

Ariane 5

- ***4. Juni 1996, Kourou / Frz. Guyana:***
  Jungfernflug der neuen europäischen Trägerrakete Ariane 5

```
...
declare
  vertical_veloc_sensor: float;
  horizontal_veloc_sensor: float;
  vertical_veloc_bias: integer;
  horizontal_veloc_bias: integer;
  ...
begin
  declare
    pragma suppress(numeric_error, horizontal_veloc_bias);
  begin
    sensor_get(vertical_veloc_sensor);
    sensor_get(horizontal_veloc_sensor);
    vertical_veloc_bias := integer(vertical_veloc_sensor);
    horizontal_veloc_bias := integer(horizontal_veloc_sensor);
    ...
  exception
    when numeric_error => calculate_vertical_veloc();
    when others => use_irs1();
  end;
end irs2;
```

Ariane 5

- ☐ Ursache:
  - 37 Sekunden nach Zünden der Rakete (30 Sekunden nach Liftoff) erreichte Ariane 5 in 3700 m Flughöhe eine Horizontal-Geschwindigkeit von 32768.0 (interne Einheiten). Die Umwandlung in eine ganze Zahl führte daher zu einem Überlauf, der jedoch nicht abgefangen wurde. Der Ersatzrechner hatte das gleiche Problem schon 72 msec vorher und schaltete sich sofort ab. Daraus resultierte, dass Diagnose-Daten zum Hauptrechner geschickt wurden, die dieser als Flugbahndaten interpretierte. Daraufhin wurden unsinnige Steuerbefehle an die seitlichen, schwenkbaren Feststoff-Triebwerke, später auch an das Haupttriebwerk gegeben. Die Rakete drohte auseinanderzubrechen und sprengte sich selbst.
- ☐ Schaden:
  - 250 Millionen DM Startkosten
  - 850 Millionen DM Cluster-Satelliten
  - 600 Millionen DM für nachfolgende Verbesserungen

Ariane 5

- ☐ Analyseproblem: Nicht erkannt, dass die korrekte Funktion des **wiederverwendeten** Moduls an Rahmenbedingungen geknüpft war, die für die Ariane 5 nicht galten *(Requirements Tracing)*
- ☐ Entwurfsproblem: Homogene Redundanz für Hardware **und** Software
  - Prinzip aus der Hardware-Sicherheitstechnik, das für Software nicht funktioniert
- ☐ Realisierungsproblem: Keine sinnvolle Propagation von Fehlverhaltenscodes, sondern Totalabschaltung
- ☐ Prüfung
  - Keine intensive, systematische Prüfung, da die Software bei der Ariane 4 problemlos funktioniert hatte (Betriebsbewährtheit)

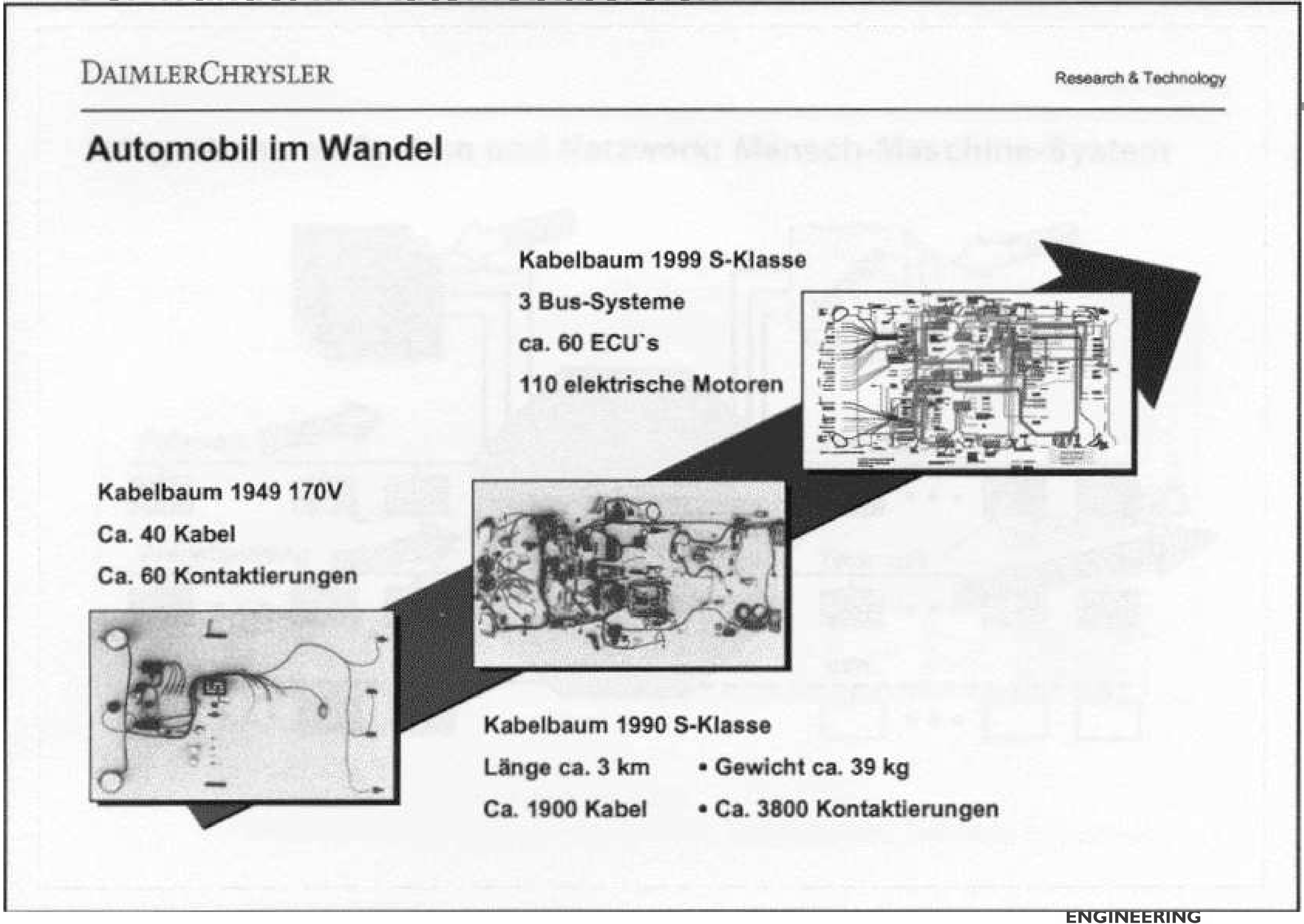
=> Das ist kein **monokausales** Problem, ...

... und daher existiert keine **einfache** Lösung.

Beispiele: Verbreitung von Softwaresystemen

- Haushalts- und Konsumelektronik
  - „Einfachste“ Geräte wie Kaffeemaschinen, Waschmaschinen und Kühlschränke beinhalten Softwaresysteme
  - Moderne Geräte wie Handys, DVD-Player und Digitalkameras bestehen zum größten Teil aus Software
- Automobilindustrie
  - Betriebliche Abläufe, Verwaltung und Produktion wäre ohne Softwaresysteme nicht mehr möglich
  - In einem Fahrzeug sind heute ca. 100 Mikrocontroller integriert
  - Mehr als die Hälfte aller Fahrzeugpannen lassen sich auf Softwareprobleme zurückführen
- Informationssysteme
  - Anwendungsbranchen: Finanzwesen, Medizinwesen, Verwaltung, …
  - Informationssysteme haben inzwischen einen Durchdringungsgrad in der Geschäftsprozessunterstützung von 60% bis 90%
  - Die Abwicklung eines Geschäftsprozesses erfordert unter Umständen das Zusammenspiel von mehr als 15 Großanwendungen

Der Wandel im Automobilbereich



DAIMLERCHRYSLER

Research & Technology

Automobil im Wandel

Kabelbaum 1999 S-Klasse

3 Bus-Systeme

ca. 60 ECU`s

110 elektrische Motoren

Kabelbaum 1949 170V

Ca. 40 Kabel

Ca. 60 Kontaktierungen

Kabelbaum 1990 S-Klasse

Länge ca. 3 km

Ca. 1900 Kabel

- **Gewicht ca. 39 kg**
- **Ca. 3800 Kontaktierungen**

Software-Katastrophe: Patriot-Rakete (1)

Ein fataler Software-Fehler im Golfkrieg II:

> "During the Gulf war, a computer failure was responsible for the failure of a patriot missile to stop a scud missile that hit an American military barracks in Dharan … 28 dead …"

[Quelle: ACM SIGSOFT Software Engineering Notes, vol. 16, no. 3 (1991), S.19f]

Ursache:

- der Steuercomputer lief 4 Tage ununterbrochen (statt der vorgeschriebenen maximal 14 Stunden)
- dadurch lief das interne Timer-Register über 24 Bit hinaus und es entstanden Rundungsfehler bei der Bahnberechnung
- wäre das Timer-Intervall 1/8 statt 1/10 Sekunde gewesen hätte es keine Rundungsfehler gegeben
- das Intervall wurde entgegen der ursprünglichen Programmierung nachträglich von einem Manager auf 1/10 Sek. geändert

Software-Katastrophe: Patriot-Rakete (2)

Schlussfolgerungen aus dem „Patriot-Missile“ -Beispiel:

- Software soweit wie möglich gegen Fehlbedienungen absichern (z.B. Warnungen nach 14 Stunden Laufzeit)
- Software soweit wie möglich gegen typische Programmierfehler absichern (Zählerüberläufe etc. durch geeignete Plausibilitätsprüfungen und „exception handling“ abfangen)
- Wichtige Entwurfsentscheidungen sind für spätere Wartung zu dokumentieren („1/8 Sek. Timer-Intervall wurde gewählt, weil…“)
- Für Software-Entwicklung feste Vorgehensweisen und Zuständigkeiten festlegen (um ad-hoc Änderungen durch unqualifizierte Personen zu verhindern)

[Quelle: Mark Minas, Vom Bild zum Programm, S.12f]

Software-Katastrophe: Kein Einzelfall (1)

- ☐ 1981: US Air Force Command & Control Software überschreitet Kostenvoranschlag fast um den Faktor 10: ***3,2 Mio. US-$***.
- ☐ 1987-1993: Integration der kalifornischen Systeme zur Führerschein- und KFZ-Registrierung abgebrochen:
  44 Mio. US-$.
- ☐ 1992: Integration des Reservierungssystems SABRE mit anderen Reservierungssystemen abgebrochen: ***165 Mio. US-$.***
- ☐ 1997: Entwicklung des Informationssystems SACSS für den Staat Kalifornien abgebrochen: ***300 Mio. US-$.***
- ☐ 1994: Eröffnung des Denver International Airport um 16 Monate verzögert wegen Softwareproblemen im Gepäcktransport-System: ***655 Mio. US-$.***
- ☐ 2005: Das deutsche Maut Erfassungssystem "Toll Collect" konnte nur mit erheblicher Verzögerung (Vertragsabschluss: September ‘02, geplanter Starttermin: 31. August 2003), am 1. Januar 2005 in technisch reduzierter Form in Betrieb genommen werden: ~***6,5 Mrd. €***.

Software-Katastrophe: Kein Einzelfall (2)

- ☐ 1988: Ein Airbus schießt über die Landebahn hinaus, da sich bei Aquaplaning die Schubumkehr nicht einschalten ließ.
- ☐ 1999: Verlust der Sonde "Mars Climate Orbiter" wegen falscher Einheitenumrechnung.
- ☐ 1999: 20.500 3er BMWs müssen wegen eines Software-Bugs in der Airbag-Steuerung zurückgerufen werden. 50% aller Autopannen sind bereits auf Ausfälle der Bordelektronik zurückzuführen, Tendenz steigend.
- ☐ 2002: Aufgrund eines Softwareproblems konnten mit Postbank-EC-Karten bei allen anderen Geldinstituten außer der Postbank selbst mit beliebigen Pincodes Euro abgehoben werden, ohne dass das Sparkonto mit der abgehobenen Summe belastet wurde.
- ☐ 2004: Siemens S65 wird wegen Softwarefehlern, die Hörschäden verursachen können, aus dem Handel genommen.

Zunehmende QS-Anforderungen

- Für 50% des Ausfälle im industriellen Sektor sind Software-Fehler verantwortlich
- Schwierigkeiten mit Zuverlässigkeit durch hohe Komplexität
  - p_k : Wahrscheinlichkeit, dass eine Komponente nicht fehlerhaft ist
  - p_s : Wahrscheinlichkeit, dass das System nicht fehlerhaft ist

Anzahl Komponenten	p_k	p_s
10	0,9	0,35
10	0,99	0,9
100	0,9	0,000027
100	0,99	0,37

- Fehler in 1.000 LOC
  - 1977: 7 - 20
  - 1994: 0,05 – 0,2
- Durchschnittliche Programmgröße (in 1.000 LOC)
  - 1977: 10
  - 1994: 800

IT-Katastrophen – nur Einzelfälle?

- CHAOS Report
  - Jährlicher Bericht seit 1994 über den Erfolg von IT-Projekten
  - Es wurden ca. 100.000 IT-Projekte in den USA untersucht
  - Herausgeber: Standish Group International, Inc.
- CHAOS Report ordnet IT-Projekte in drei Kategorien ein
  - **Successful**: Projekt wurde innerhalb der vorgegebenen Zeit und Budget abgeschlossen. Projektergebnis ist im Einsatz und erfüllt alle Anforderungen.
  - **Challenged**: Projekt ist abgeschlossen. Projektergebnis ist im Einsatz. Zeit, Budget oder Leistung sind aber nicht im vorgegebenen Umfang.
  - **Failed**: Das Projekt wurde vorzeitig abgebrochen oder das Projektergebnis wurde nie eingesetzt.

Erfolgsstatistik von IT-Projekten

	Succeeded	Failed	Challenged
1994	16%	31%	53%
1996	27%	40%	33%
1998	26%	28%	46%
2000	28%	23%	49%

[Quelle: CHAOS Report, Standish Group International, Inc.]

Die 10 wichtigsten Erfolgsfaktoren

1.	Executive support	18%
2.	User involvement	16%
3.	Experienced project manager	14%
4.	Clear business objectives	12%
5.	Minimized scope	10%
6.	Standard software infrastructure	8%
7.	Firm basic requirements	6%
8.	Formal methodology	6%
9.	Reliable estimates	5%
10.	Other criteria	5%

[Quelle: CHAOS Report, Standish Group International, Inc.]

Eckdaten der Softwarebranche in Deutschland[1]

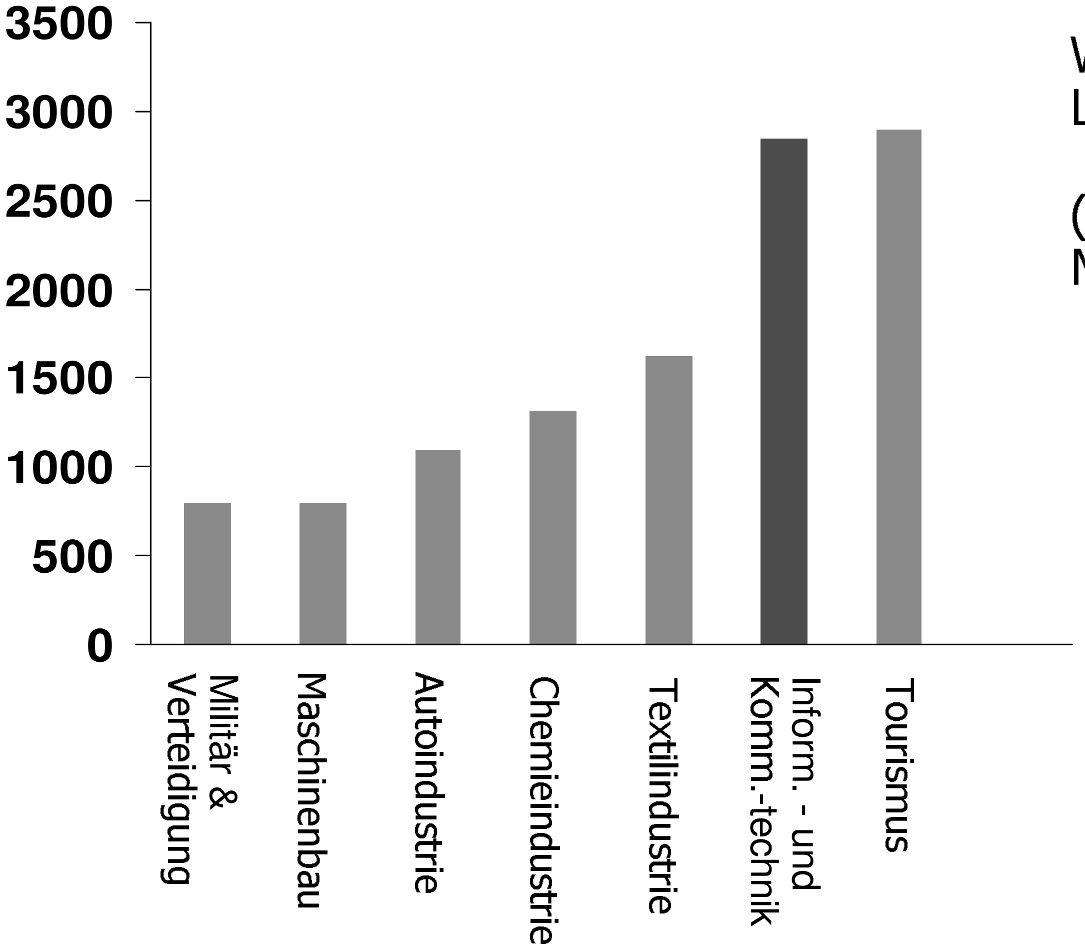
[1] Quelle: Studie „Analyse und Evaluation der Softwareentwicklung in Deutschland“, Bundesministerium für Bildung und Forschung, Dezember 2000

- Primärbranchen (DV-Dienstleister, Hersteller von Datenverarbeitungsgeräten und -einrichtungen)
  - Rund 10.550 Unternehmen
  - Ca. 300.000 Erwerbstätige
  - Überwiegend kleine Unternehmen mit 1-9 Mitarbeitern
- Sekundärbranchen (Maschinenbau, Elektrotechnik, Fahrzeugbau, Telekommunikation und Finanzdienstleistungen)
  - Rund 8.650 Unternehmen
  - 2,5 Millionen Erwerbstätige
  - Eher mittlere und größere Unternehmen
  - Heutige Produkte ohne Software oft undenkbar

... daß schon jetzt mehr als die Hälfte der Wertschöpfung von Siemens auf Software-Leistungen entfällt. Diese Entwicklung geht weiter ...

Heinrich von Pierer, Siemens AG

Wirtschaftliche Bedeutung von Softwaresystemen



Weltweite
Leit-Industrien

(Umsatz 1994 in
Milliarden US-$)

[Quelle: Balzert, Bd. 1, S. 28 Abb. 6]

Was ist Software?

- Software (engl., eigtl. »weiche Ware«), Abk. SW, Sammelbezeichnung für Programme, die für den Betrieb von Rechensystemen zur Verfügung stehen, einschl. der zugehörigen Dokumentation *(Brockhaus Enzyklopädie)*
- Software: die zum Betrieb einer Datenverarbeitungsanlage erforderlichen nichtapparativen Funktionsbestandteile *(Fremdwörter-Duden).*

Was ist Software?
Definition

- Software ist eine Menge von Programmen oder Daten zusammen mit begleitenden Dokumenten, die für die Anwendung notwendig oder hilfreich sind
- Beispiele
  - Microsoft Office
  - Linux
  - Waschmaschinensteuerung
- Programme sind
  - „weiche“ Ware
  - immateriell
  - „vermeintlich“ leicht zu ändern
  - haben keinen Verschleiß, keine „klassische“ Wartung

Was ist ein Softwaresystem?
Definition

- Ein Softwaresystem ist ein aus mehreren Teilen zusammengesetztes Ganzes. Es besteht aus Software, Hardware und unterstützenden Elementen zusammen mit begleitenden Dokumenten, die für die Anwendung notwendig oder hilfreich sind
- Beispiele
  - Flugzeug
  - Kontoauszugdrucker
  - Versicherungsverwaltungssystem
- Gegenbeispiel
  - Kugelschreiber

Klassifizierung von Software und Softwaresystemen (1)

- ☐ Größe
  - Klein: Gerätetreiber mit 5.000 LOC
  - Mittel: Mobiltelefon mit 200.000 LOC
  - Groß: SAP R/3 mit 6 MIO LOC (50.000 Funktionen; 17.000 Menüleisten)

- ☐ Anwendungsgebiet
  - Informationssystem: SAP, Banksystem, …
  - Eingebettete Systeme: Air-bag-Controller, …
  - Technische Systeme: Betriebssysteme (Linux, …), …

Klassifizierung von Software und Softwaresystemen (2)

- ☐ Hardwarenähe
  - Anwendungssysteme : Microsoft Office, …
  - Middleware: EJB-Server, DB2, …
  - Systemsoftware: Linux, Gerätetreiber, …

- ☐ Verbreitung und Spezialisierungsgrad
  - Standardsoftware: Microsoft Windows XP, SAP, …
  - Individualsoftware: Banksystem, Eingebettete SW

Was ist ein Projekt?
Definition

- Ein Projekt ist ein einmaliges Vorhaben mit einem gewissen Risiko. Ein vorgegebenes Ziel muss innerhalb einer vorgegebenen Zeit unter Einsatz von vorhandenen, meist beschränkten Mitteln erarbeitet werden
- Beispiel
  - Umzug
  - Mondlandung
  - Entwicklung von Software- Systemen
- Gegenbeispiel
  - Betrieb eines Rechenzentrums (SCI)

Klassifizierungen von Projekten (1)

☐ Größe und Dauer

- Klein: 1 PJ, 1-2 Bearbeiter, Entwicklung für Eigenbedarf
- Mittel: 1-10 PJ, 3-10 Bearbeiter, Compiler, Steuerprogramme, Entwicklung für Kunden
- Groß: 5-50 PJ, 10 bis 30 Bearbeiter, Datenbanken, Spiele, Individual-SW-Systeme
- Riesig: 50-5.000 PJ, 20-1000 Bearbeiter, Gesamtlösungen für Unternehmen

Klassifizierungen von Projekten (2)

- ☐ Zielsetzung und Ergebniserwartung
    - Kurzfristige ökonomische Interessen
    - Strategisches Investitionsprojekt
    - Forschungsprojekt
- ☐ Anwendungsdomäne
    - Finanzwesen
    - Verwaltung
    - Militär
- ☐ Technologien
    - Programmiersprachen, Hardware, Systemsoftware

Was ist Ingenieurswesen?
Definition

☐ Engineering ist die Anwendung von naturwissenschaftlichen Erkenntnissen und praktischen Erfahrungen, um für die Menschheit sinnvolle Dinge zu entwickeln und bereit zu stellen

☐ Beispiele

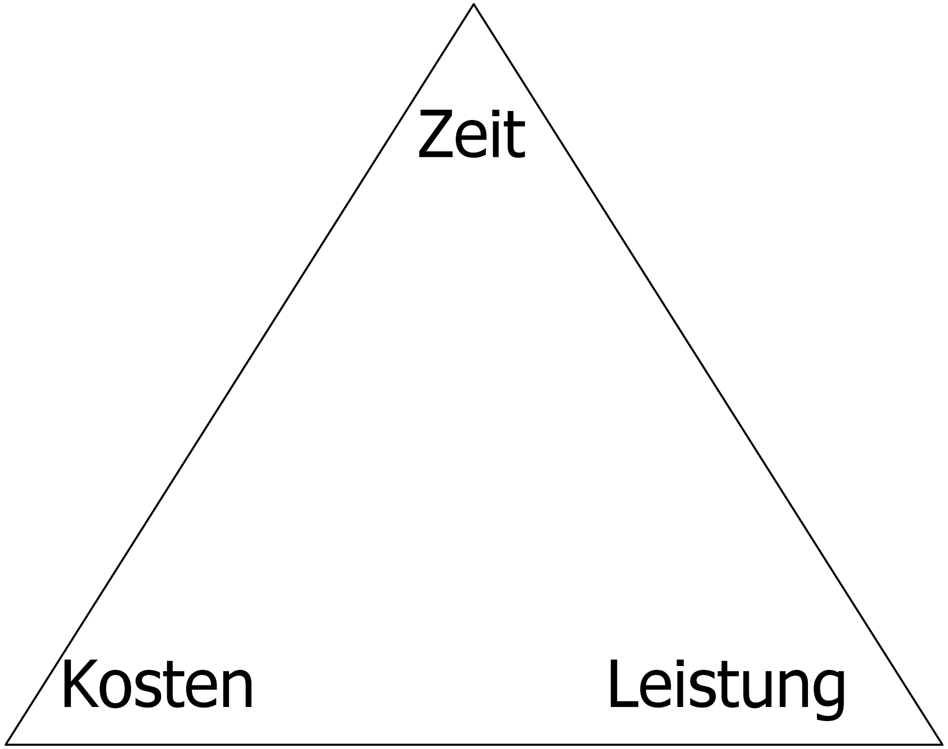
  - Maschinenbau
  - Bauingenieurswesen
  - Architektur
  - Software Engineering

Was will Software Engineering?

- ☐ Softwaresysteme sind ein zentrales Rückrat unserer Gesellschaft!
- ☐ Die Fähigkeit IT-Projekte durchzuführen muss verbessert werden!

➔ Ziel: Verbesserung der Beherrschbarkeit des "magischen Dreiecks"

- ☐ Ansatz: Disziplin Software Engineering



Was ist Software Engineering?
Definition

- Software Engineering ist die zielorientierte Bereitstellung und Verwendung von **systematischen, ingenieurmäßigen und quantifizierbaren Vorgehensweisen** für **Entwicklung, Betrieb, Wartung und Stilllegung von Softwaresystemen**

- Zielorientiert bedeutet dabei die Berücksichtigung von
  - Zeit
  - Kosten
  - Leistung (Qualität)

Was ist Software Engineering?
Definition

- ☐ Das ingenieurmäßige Entwerfen, Herstellen und Implementieren von Software sowie die ingenieurwissenschaftliche Disziplin, die sich mit Methoden und Verfahren zur Lösung der damit verbundenen Problemstellungen befasst
(Brockhaus Enzyklopädie)

Was ist Software Engineering nicht?

☐ Programmierkurs, Programmier - Know - How

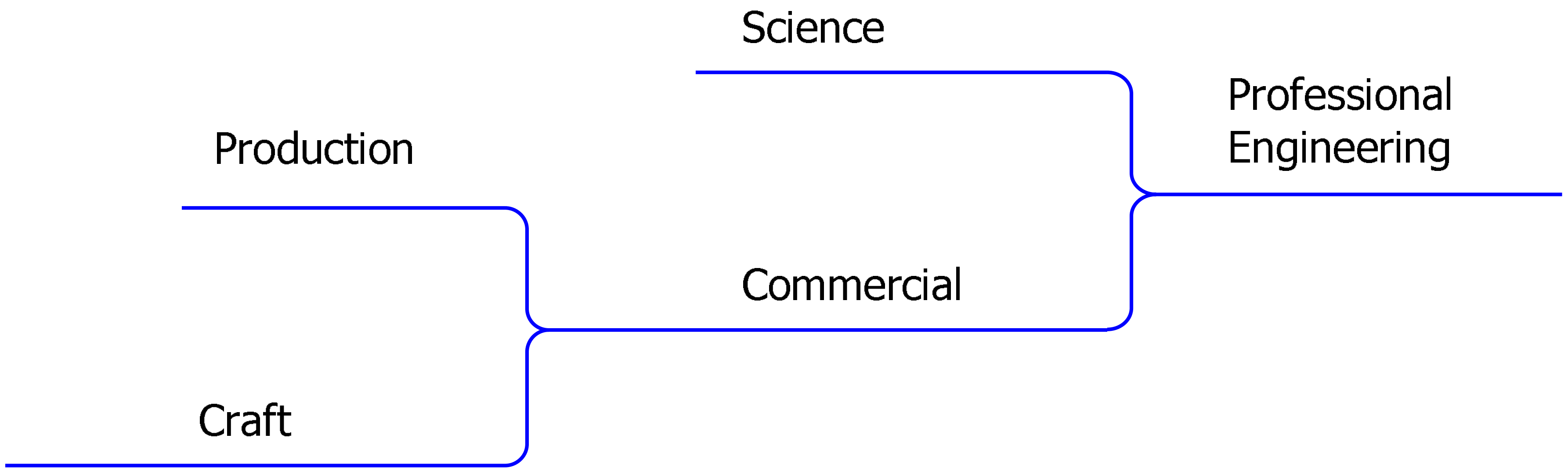
☐ AnwenderInnen-Kurs

☐ abstrakte Wissenschaft

☐ "A fool with a tool is still a fool"

☐ "Silver Bullet"

Entwicklung von Ingenieursdisziplinen



- Virtuosos and talented amateurs
- Intuition and brute force
- Haphazard progress
- Casual transmission
- Extravagant use of available materials
- Manufacture for use rather than sale

- Skilled craftsmen

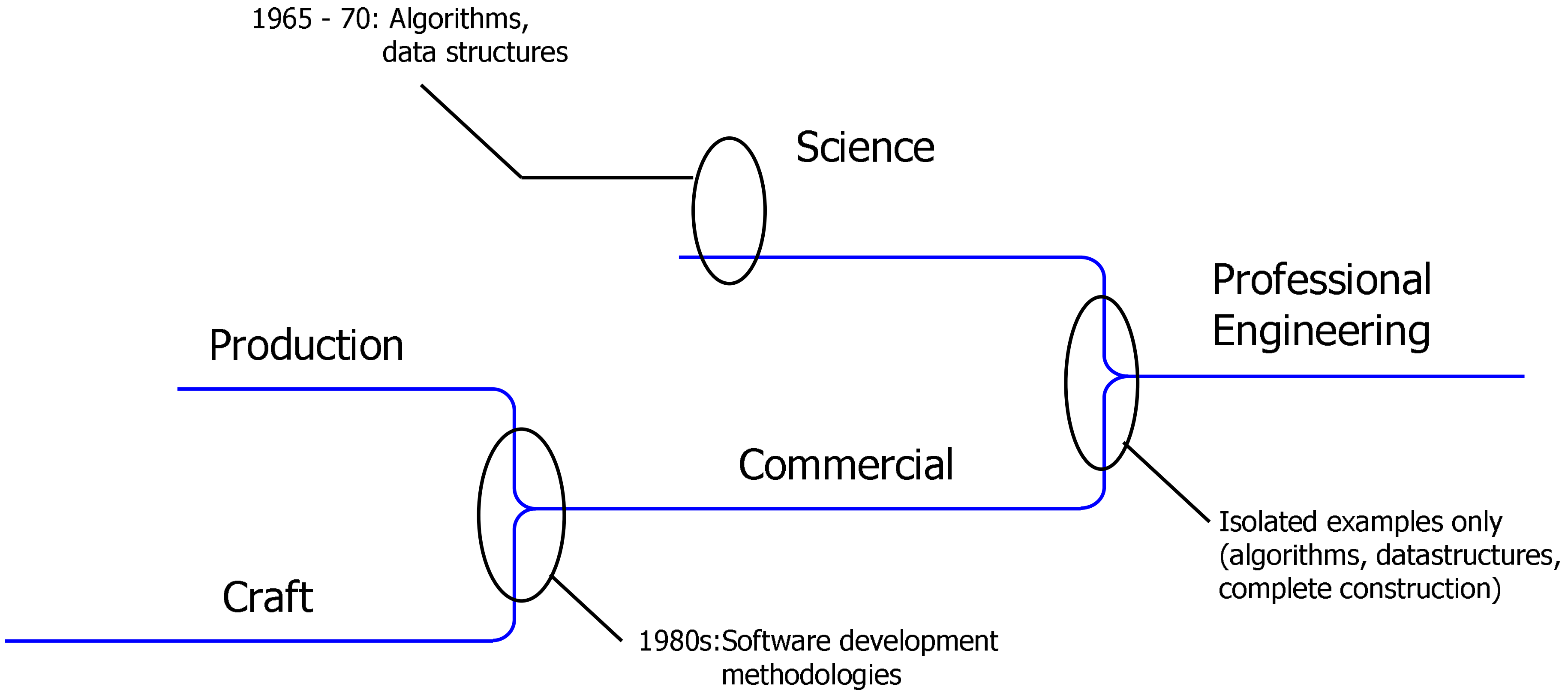
- Established procedure
- Pragmatic refinement
- Training in mechanics
- Economic concern for cost and supply of materials
- Manufacture for sale

- Educated professionals

- Analysis and theory
- Progress relies on science
- Educated professionals
- Enabling new applications through analysis
- Market segmentation by product variety

[Quelle: Shaw96, S.8]

Entwicklung von Software Engineering



1965 - 70: Algorithms,
data structures

Science

Production

Professional
Engineering

Commercial

Craft

Isolated examples only
(algorithms, datastructures,
complete construction)

1980s:Software development
methodologies

[Quelle: Shaw96, S.8]

Unterschiede zwischen anderen Ingenieursdisziplinen und Software Engineering (1)

☐ Software unterliegt keinen physikalischen Gesetzen, Software ist immateriell

➔ Software Engineering ist die einzige Ingenieursdisziplin, die sich mit nicht fassbaren Systemen beschäftigt

☐ Unterschiede und Missverständnisse (1):

- ▪ Nur Entwicklung, keine Produktion
- 💣 Software ist leicht änderbar

Unterschiede zwischen anderen Ingenieursdisziplinen und Software Engineering (2)

- Unterschiede und Missverständnisse (2):
  - Softwaresysteme haben (fast) keinen Verschleiß

  💣 Betrieb und Nutzung kann vernachlässigt werden

  - Softwaresysteme haben einen relativ kurzen Entwicklungszyklus

  💣 Große Softwaresysteme sind kurzlebig

Ethische Verantwortung (1)

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

- ☐ 1 PUBLIC - Software engineers shall act consistently with the public interest.
- ☐ 2 CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.
- ☐ 3 PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

[Quelle: IEEE: http://www.computer.org/tab/seprof/code.htm]

Ethische Verantwortung (2)

- ☐ 4 JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
- ☐ 5 MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
- ☐ 6 PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
- ☐ 7 COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.
- ☐ 8 SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Rechtliche Verantwortung

Ausschnitt aus Safety - Handbuch der BW
SIL Safety Integrity Level

	HOCH			NIEDRIG	
Attributes	**SIL 4**	**SIL 3**	**SIL 2**	**SIL 1**	**Appl. HW SW**
Requirements and Design Specification	Formal (Mathematical)	Semiformal	Informal (e.g. Natural Language)	Informal (e.g. Natural Language)	H/S
Configuration Management	Full (Au-tomated for development and production	Full (Au-tomated for development and production	Yes	Manual	H/S
Structured Design Method; e.g. data	Yes	Yes	Preferred	Optional	H/S

Verpflichtend durchzuführende Aktivitäten

Werden diese nicht durchgeführt ist der Ingenieur haftbar!

Gilt auch wenn nicht nach State-of-the-art ent-wickelt wird.

Ziele dieser Lehrveranstaltung

- ☐ Softwareentwicklungsprozesse kennen, beschreiben und bewerten können
- ☐ Wissen wie Software arbeitsteilig in Teams entwickelt wird, welche Rollen und welche Zusammenhänge existieren
- ☐ Lerninhalte aus SE 1 und SE 2 vertiefen und ausweiten
- ☐ Weitere wichtige Methoden und Techniken lernen, die in SE 1 bzw. SE 2 noch nicht behandelt wurden
- ☐ Wichtige Sachverhalte (Grundregeln) des Software Engineering kennen

Inhalte dieser Lehrveranstaltung

☐ Teile der in diesem Semester verwendeten Unterlagen basieren auf

- der Lehrveranstaltung „Software-Konstruktion“ von P. Liggesmeyer (Univ. Potsdam),
- der Lehrveranstaltung „Grundlagen des Software Engineering“ von A. Rausch (TU KL),
- der Lehrveranstaltung „Software Engineering“ von D. Rombach (TU KL),
- dem *Lehrbuch der Software-Technik* (Band 1, 2. Aufl.) von H. Balzert, Spektrum-Verlag, Heidelberg 2000, ISBN 3-8274-0480-0
- dem Buch *Software-Qualität* von P. Liggesmeyer, Spektrum-Verlag, Heidelberg 2000, ISBN 3-8274-1118-1

Software Engineering
Einleitung

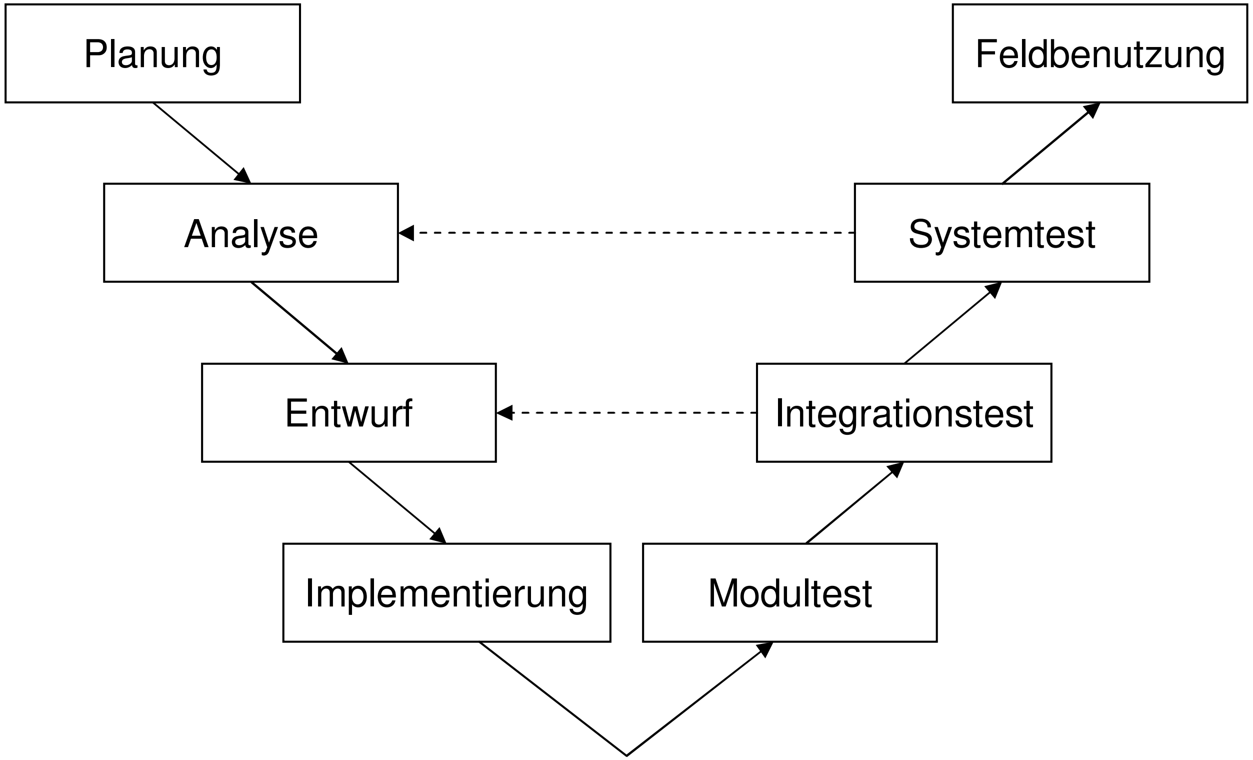
Eine umfangreiche Softwareentwicklung erfordert ...

- ☐ ... einen Plan, dessen Verfolgung am Ende ein kosten-, zeit- und qualitätsgerechtes Ergebnis erwarten läßt
- ☐ ... Projektmanagement (Zuordnung von Personal und Sachmitteln, Ermittlung von Aufwänden und deren Kontrolle)
- ☐ ... Qualitätsmanagement (Qualitätsplanung und –kontrolle)
- ☐ ... Entwicklungsschritte mit definierten Eingaben, Inhalten und Ausgaben
- ☐ ... Überprüfungsschritte zur Kontrolle von Qualitätseigenschaften
- ☐ ... Werkzeuge zur Unterstützung der Entwicklungs- und Überprüfungsschritte

Einleitung
Prozessmodelle

- Ein Prozessmodell legt fest:
  - Reihenfolge des Arbeitsablaufs
    - Entwicklungsstufen
    - Phasenkonzepte
  - Jeweils durchzuführende Aktivitäten
  - Definition der Teilprodukte einschließlich Layout und Inhalt
  - Fertigstellungskriterien
  - Notwendige Mitarbeiterqualifikationen
  - Verantwortlichkeiten und Kompetenzen
  - Anzuwendende Standards, Richtlinien, Methoden und Werkzeuge

Einleitung
Prozessmodelle


Planung
Feldbenutzung
Analyse
Systemtest
Entwurf
Integrationstest
Implementierung
Modultest

Einleitung
Die Planung

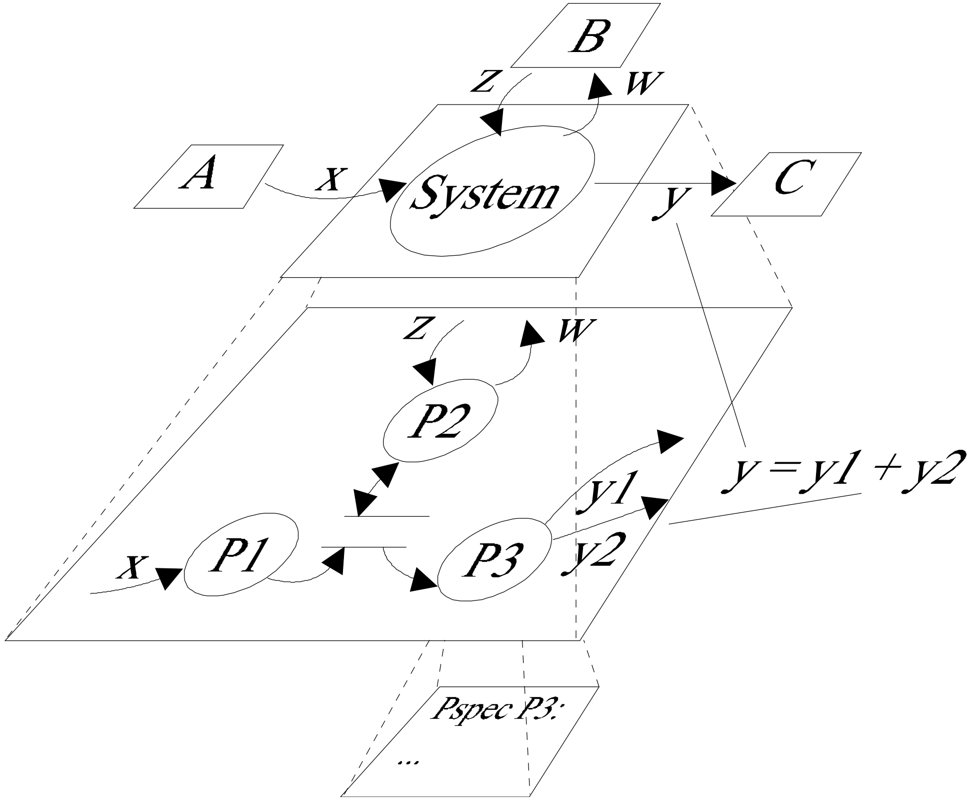
- ☐ Prüfen der Machbarkeit (technische Machbarkeit, genügend Ressourcen (insb. richtig qualifiziertes Personal))
- ☐ Prüfen der Rentabilität des Entwicklungsvorhabens (Marktsituation, Konkurrenz)
- ☐ **Aufwandsschätzung** (Wie viele Mitarbeitermonate werden voraussichtlich benötigt werden?)
- ☐ Erstellen eines Projektplans (Schritte, Ressourcen, Zeiten)

Einleitung
Die Analyse

- Festlegung der Eigenschaften der zu entwickelnden Software (es geht allein um das "Was"; nicht um das "Wie")
  - Gewünschte Funktionalität: "Was soll die Software tun?"
  - Leistungsdaten: Zeitverhalten (besonders kritisch bei Echtzeitsystemen), Mengengerüste
  - Qualitätseigenschaften (sogen. Qualitätszielbestimmung): "Welche Qualitätseigenschaften sind in welcher Weise zu beachten?"
- Ermittlung der Anforderungen (Requirements Engineering)
- Beschreibung der Anforderungen in Form von Analysedokumenten:
  - Funktional dekomponierender Ansatz (z. B. Strukturierte Analyse: SA)
  - Objektorientierter Ansatz (OOA: z. B. Unified Modeling Language: UML)
- **Beachtung ergonomischer Regeln und Forderungen**

Einleitung: Die Analyse
Beispiel: Funktional dekomponierende Techniken

Beispiel SA



+ Universelle Einsetzbarkeit
+ Gute Abstraktions-, Modularisierungs- und Hierarchisierungsmechanismen
+ Gute Visualisierung
+ Automatische Konsistenzprüfung möglich
+ Unterstützung von Erweiterbarkeit und Änderbarkeit
- Methodisch nicht konsistent verfeinerbar (Methodenbruch)
- Semantik interpretierbar
- Leistungsanforderungen nur eingeschränkt beschreibbar

Einleitung: Die Analyse
Beispiel: Objektorientierte Techniken

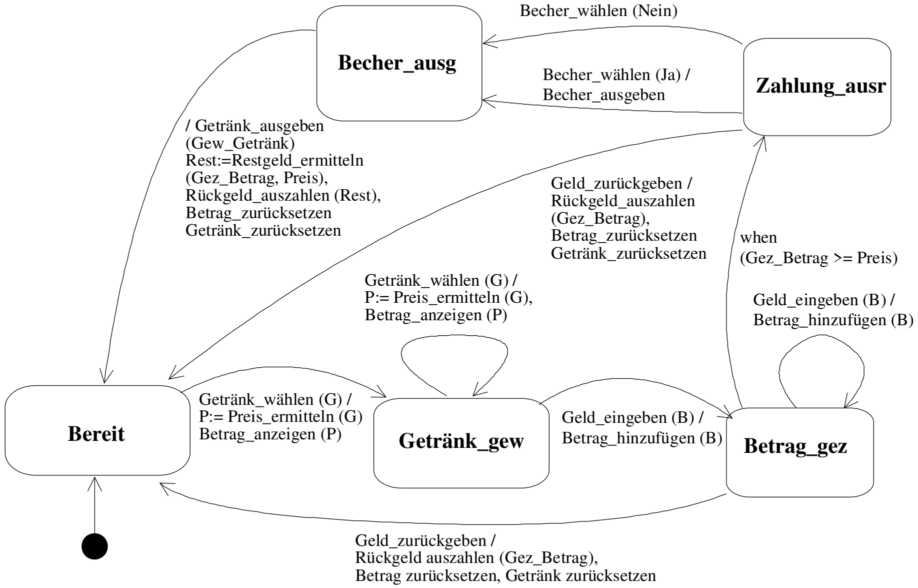
Beispielausschnitt UML

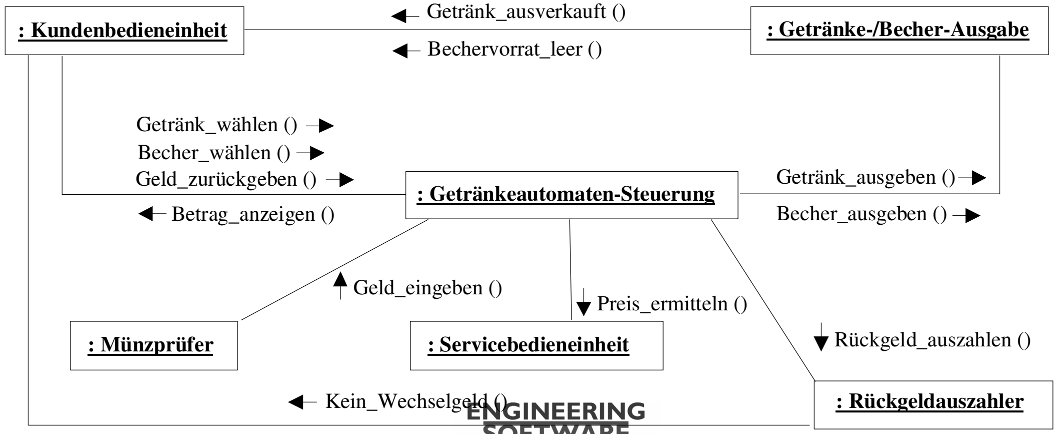
- + Universelle Einsetzbarkeit
- + Ausgezeichnete Abstraktions-, Modularisierungs- und Hierarchisierungsmechanismen
- + Exzellente Visualisierung
- + Automatische Konsistenzprüfung möglich
- + Unterstützung von Erweiterbarkeit und Änderbarkeit
- + Methodisch konsistent verfeinerbar
- - Semantik interpretierbar
- - Leistungsanforderungen nur eingeschränkt beschreibbar

Getränkeautomaten-Steuerung

Akt_Zustand
Gew_Getränk
Preis
Gez_Betrag

Getränk_wählen ()
Becher_wählen ()
Geld_eingeben ()
Geld_zurückgeben ()
# Betrag_zurücksetzen
# Getränk_zurücksetzen
# Restgeld_ermitteln

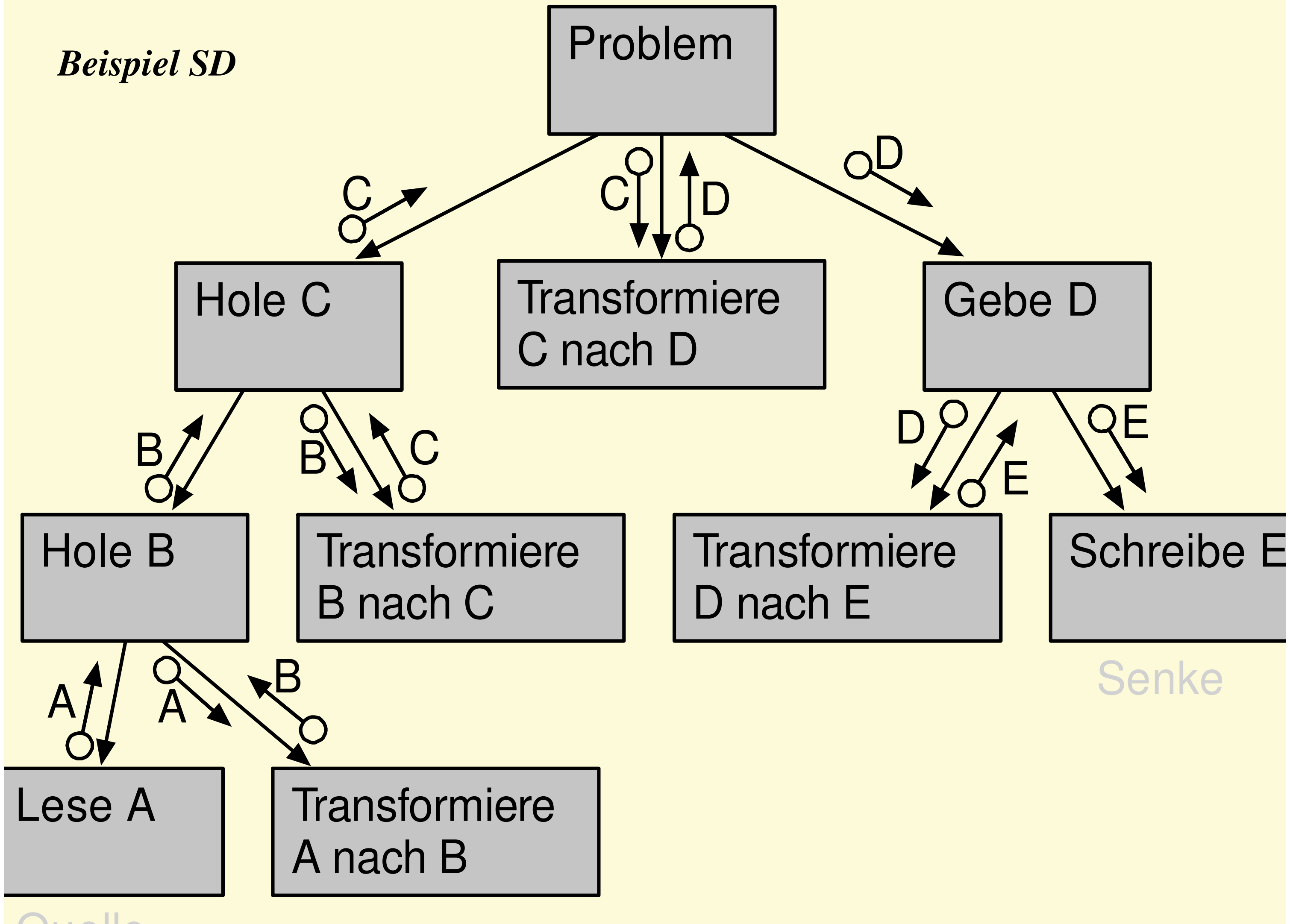

Becher_wählen (Nein)
Becher_ausg
Becher_wählen (Ja) / Becher_ausgeben
Zahlung_ausr
/ Getränk_ausgeben (Gew_Getränk) Rest:=Restgeld_ermitteln (Gez_Betrag, Preis), Rückgeld_auszahlen (Rest), Betrag_zurücksetzen Getränk_zurücksetzen
Geld_zurückgeben / Rückgeld_auszahlen (Gez_Betrag), Betrag_zurücksetzen Getränk_zurücksetzen
when (Gez_Betrag >= Preis)
Getränk_wählen (G) / P:= Preis_ermitteln (G), Betrag_anzeigen (P)
Geld_eingeben (B) / Betrag_hinzufügen (B)
Bereit
Getränk_wählen (G) / P:= Preis_ermitteln (G) Betrag_anzeigen (P)
Getränk_gew
Geld_eingeben (B) / Betrag_hinzufügen (B)
Betrag_gez
Geld_zurückgeben / Rückgeld auszahlen (Gez_Betrag), Betrag zurücksetzen, Getränk zurücksetzen


: Kundenbedieneinheit
Getränk_ausverkauft ()
Bechervorrat_leer ()
: Getränke-/Becher-Ausgabe
Getränk_wählen ()
Becher_wählen ()
Geld_zurückgeben ()
Betrag_anzeigen ()
: Getränkeautomaten-Steuerung
Getränk_ausgeben ()
Becher_ausgeben ()
Geld_eingeben ()
Preis_ermitteln ()
Rückgeld_auszahlen ()
: Münzprüfer
: Servicebedieneinheit
: Rückgeldauszahler
Kein_Wechselgeld ()

Einleitung
Der Entwurf

- ☐ Festlegung der Struktur der zu entwickelnden Software ("Wie" soll die Software realisiert werden)
  - Welche Subsysteme (Grobentwurf) und Module (Feinentwurf) soll die Software haben?
  - Wie ist der Zusammenhang zwischen den Komponenten (Architektur der Software)?
  - Welche Funktion sollen diese Komponenten besitzen und wie sind ihre Schnittstellen beschaffen?
  - Welche Subsysteme sollen wie realisiert werden (Datenbank vs. Dateien, handgeschriebener Parser vs. generierter Parser)?
  - Welche Fehlermöglichkeiten sollen wo abgefangen werden?

Einleitung: Der Entwurf
Beispiel: Funktional dekomponierende Techniken



Beispiel SD

Problem

Hole C

Transformiere C nach D

Gebe D

Hole B

Transformiere B nach C

Transformiere D nach E

Schreibe E

Lese A

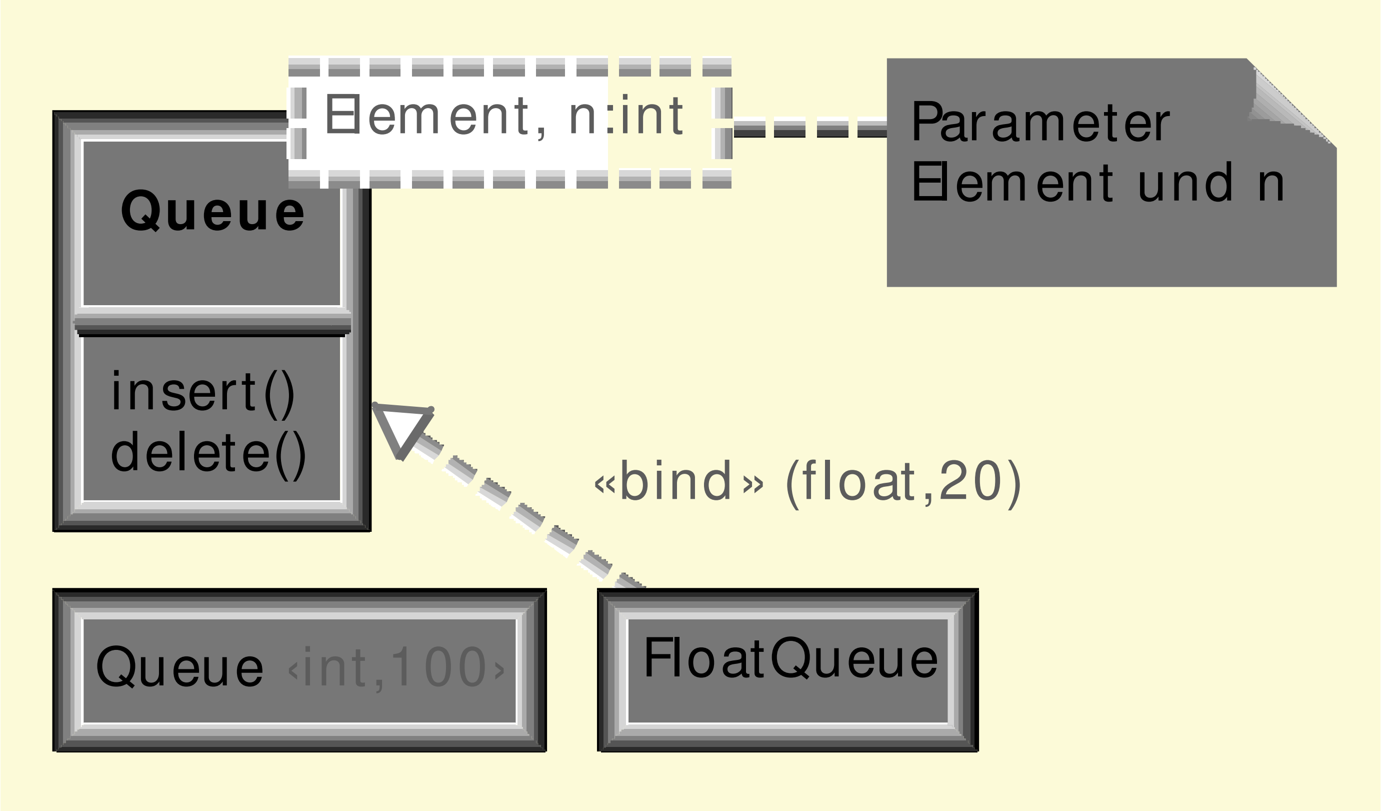
Transformiere A nach B

Senke

Quelle

Einleitung: Der Entwurf
Beispiel: Objektorientierte Techniken

- ☐ Hinzufügen "technischer" Klassen; z. B. Warteschlange (Queue)


Element, n:int
Parameter Element und n
Queue
insert()
delete()
«bind» (float,20)
Queue ‹int,100›
FloatQueue

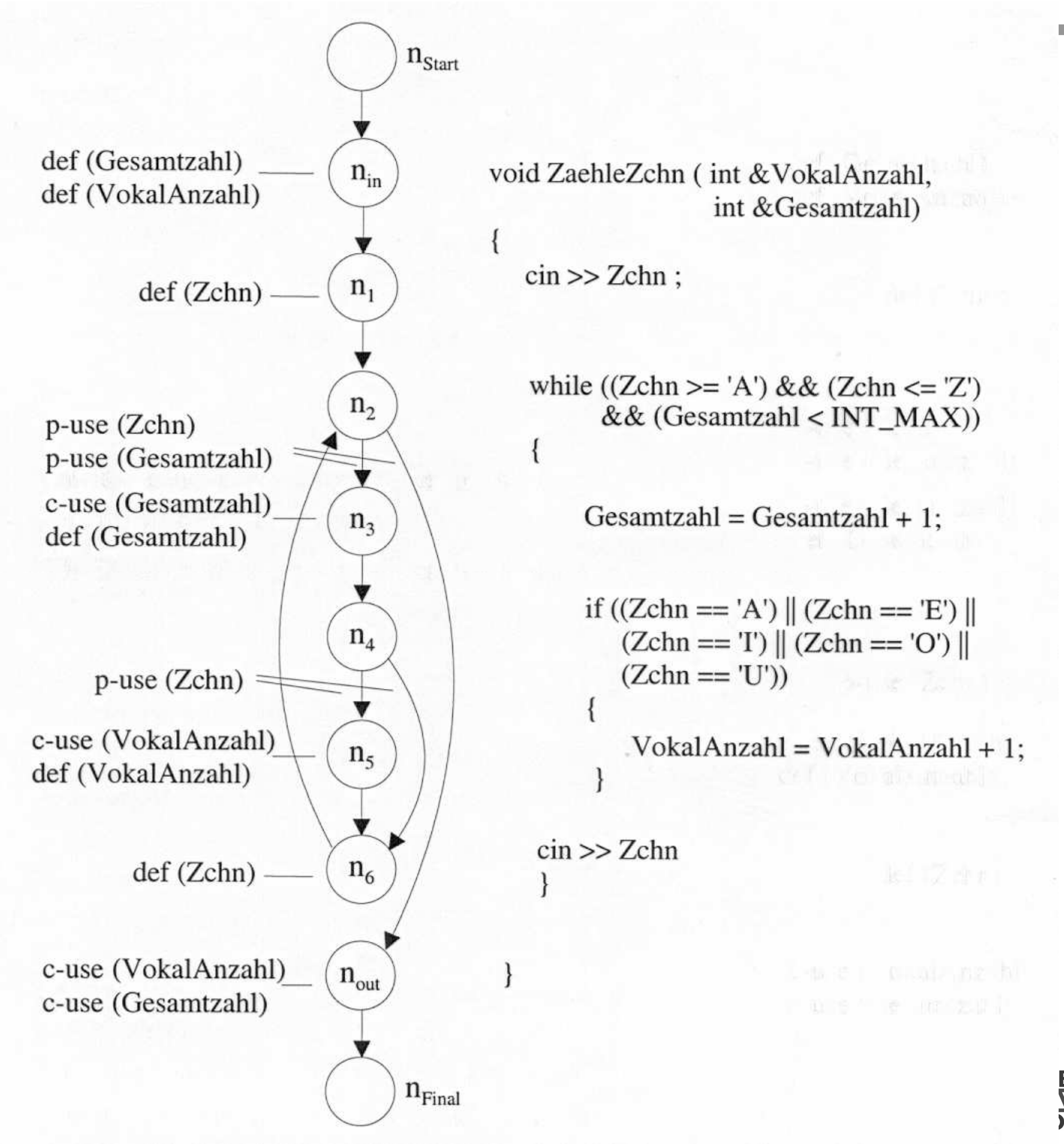
Einleitung
Die Implementierung

- Realisierung des Entwurfs in einer Programmiersprache
  - Auswahl der Datenstrukturen
  - Realisierung der Kontroll-Logik
- ggf. Aufbau der Datenbank, usw.

Einleitung
Der Modultest

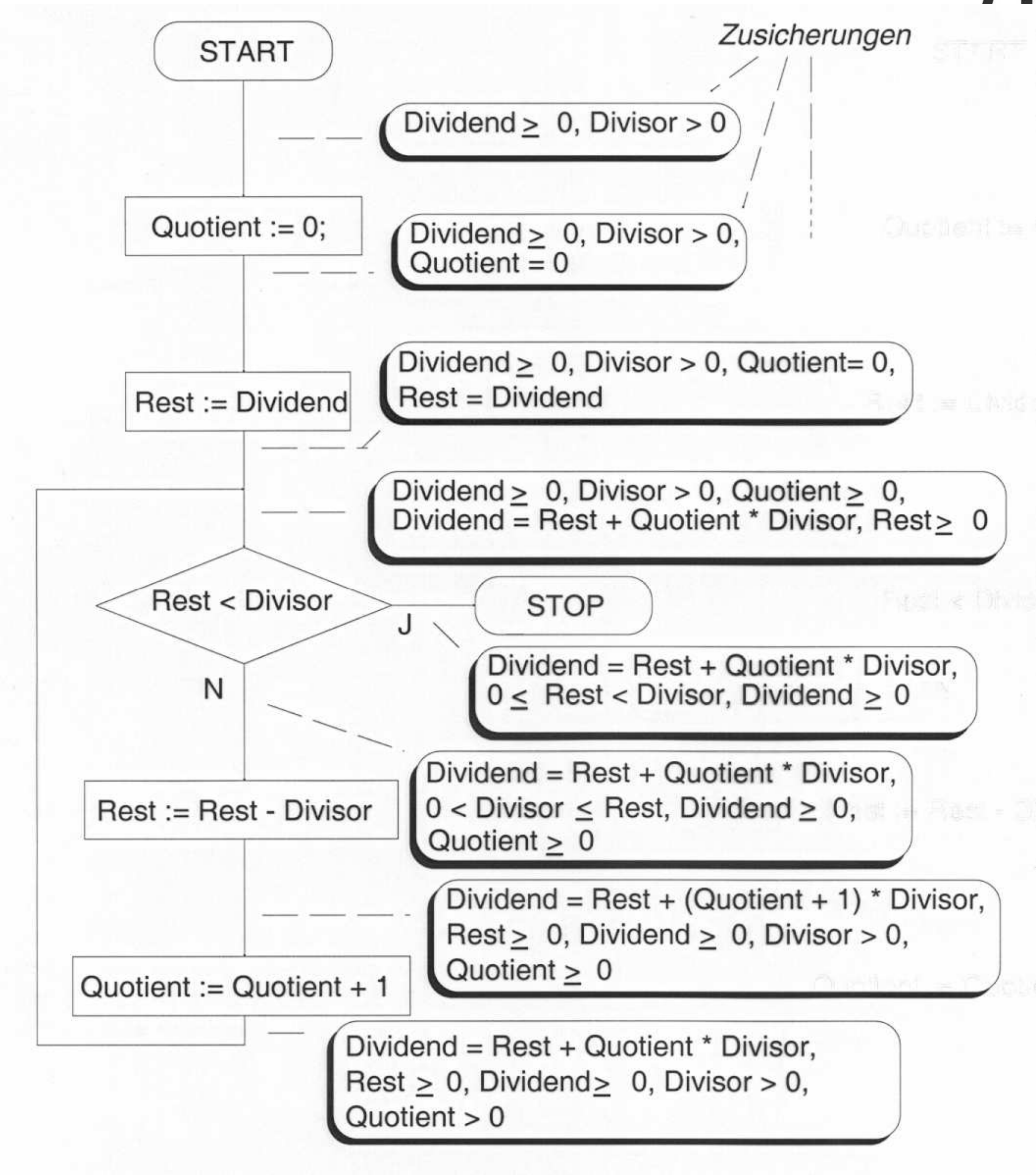
- Überprüfung der korrekten Funktion einzelner Module oder eines kleinen Verbunds von Modulen
  - Dynamisches Testen (Ausführung mit konkreten Testfällen)
    - Funktionsorientierter Test
    - Strukturorientierter Test
    - Diversifizierender Test
  - Statische Analysen (z.B. Aufspüren bestimmter Fehler unter Verzicht auf die Ausführung der Software)
    - Inspektionstechniken
    - Datenflußanomalieanalyse
    - ...
  - Formale Verifikation (Nachweis der Konsistenz zwischen dem Programmcode des Moduls und der (formalen) Modulspezifikation)

Einleitung
Der Modultest
Kontrollflußgraph mit Datenflußattributen (Datenflußorient. Test)


n_{Start}
def (Gesamtzahl)
def (VokalAnzahl)
n_{in}
def (Zchn)
n_1
n_2
p-use (Zchn)
p-use (Gesamtzahl)
c-use (Gesamtzahl)
def (Gesamtzahl)
n_3
n_4
p-use (Zchn)
c-use (VokalAnzahl)
def (VokalAnzahl)
n_5
def (Zchn)
n_6
c-use (VokalAnzahl)
c-use (Gesamtzahl)
n_{out}
n_{Final}

```
void ZaehleZchn ( int &VokalAnzahl,
                  int &Gesamtzahl)
{
   cin >> Zchn ;


   while ((Zchn >= 'A') && (Zchn <= 'Z')
        && (Gesamtzahl < INT_MAX))
   {

      Gesamtzahl = Gesamtzahl + 1;

      if ((Zchn == 'A') || (Zchn == 'E') ||
         (Zchn == 'I') || (Zchn == 'O') ||
         (Zchn == 'U'))
      {
         VokalAnzahl = VokalAnzahl + 1;
      }

   cin >> Zchn
   }

}
```

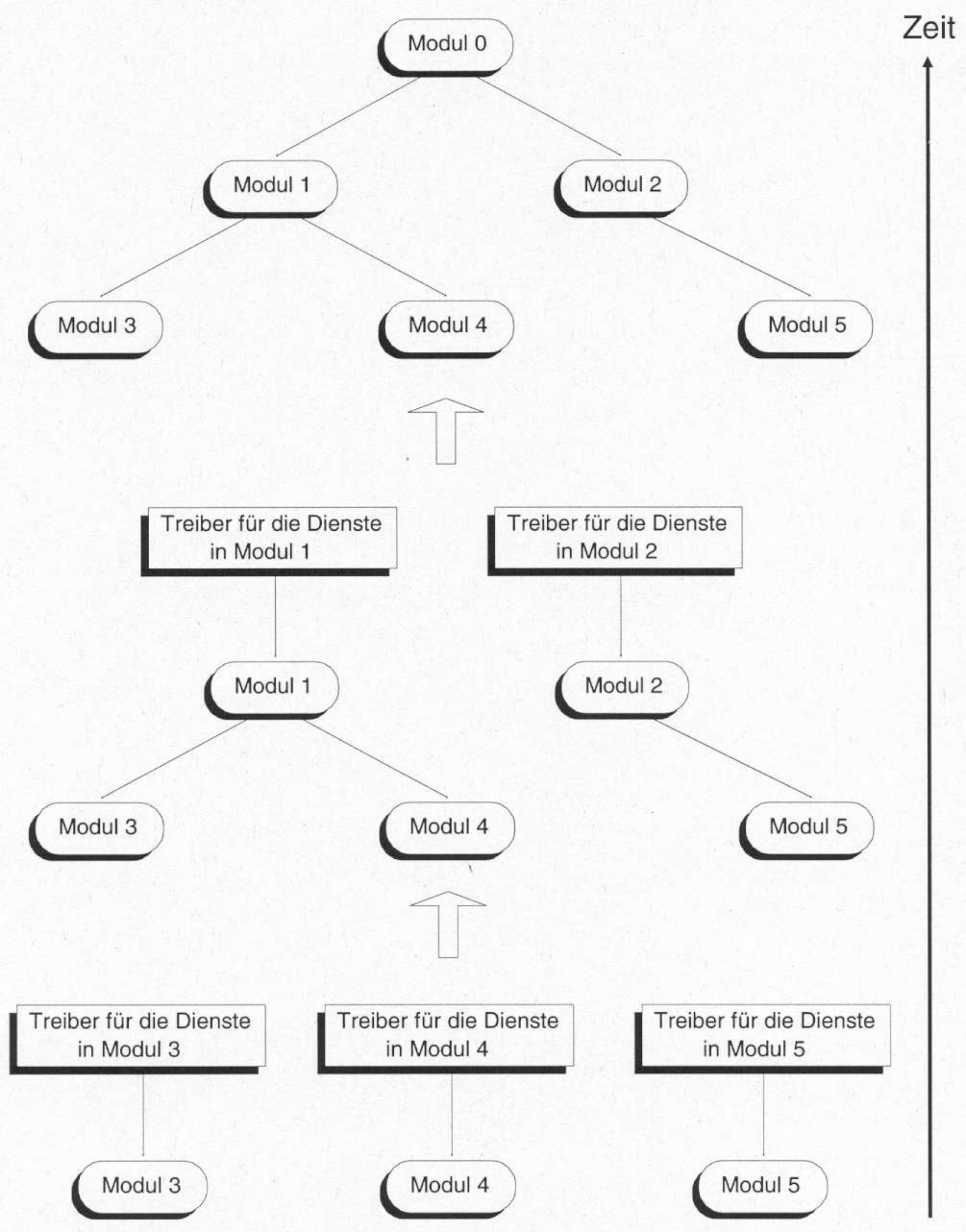
Einleitung
Der Modultest
Formale Verifikation


START
Zusicherungen
Dividend ≥ 0, Divisor > 0
Quotient := 0;
Dividend ≥ 0, Divisor > 0, Quotient = 0
Rest := Dividend
Dividend ≥ 0, Divisor > 0, Quotient= 0, Rest = Dividend
Dividend ≥ 0, Divisor > 0, Quotient ≥ 0, Dividend = Rest + Quotient * Divisor, Rest ≥ 0
Rest < Divisor
J
STOP
Dividend = Rest + Quotient * Divisor, 0 ≤ Rest < Divisor, Dividend ≥ 0
N
Dividend = Rest + Quotient * Divisor, 0 < Divisor ≤ Rest, Dividend ≥ 0, Quotient ≥ 0
Rest := Rest - Divisor
Dividend = Rest + (Quotient + 1) * Divisor, Rest ≥ 0, Dividend ≥ 0, Divisor > 0, Quotient ≥ 0
Quotient := Quotient + 1
Dividend = Rest + Quotient * Divisor, Rest ≥ 0, Dividend ≥ 0, Divisor > 0, Quotient > 0

Einleitung
Der Integrationstest

- Schrittweises Zusammenfügen der Module (sogenannte Integrationsstrategie)
- Überprüfung der korrekten Interaktion zwischen Modulen über ihre Schnittstellen
  - Dynamisches Testen
  - Statische Analysen

Einleitung
Der Integrationstest


Zeit
Modul 0
Modul 1
Modul 2
Modul 3
Modul 4
Modul 5
Treiber für die Dienste in Modul 1
Treiber für die Dienste in Modul 2
Modul 1
Modul 2
Modul 3
Modul 4
Modul 5
Treiber für die Dienste in Modul 3
Treiber für die Dienste in Modul 4
Treiber für die Dienste in Modul 5
Modul 3
Modul 4
Modul 5

Einleitung
Der Systemtest und Abnahmetest

- Überprüfung der korrekten Funktion, der Leistung und der Qualität des Softwaresystems "von Außen"
- Funktionstest: Funktionsorientierte Testfallerzeugung auf Basis der Anforderungsdefinition
- Leistungstest: Das System wird in Grenzbereiche gebracht:
  - Wie ist das Antwortzeitverhalten unter Vollast wenn gleichzeitig an 100 Terminals gearbeitet wird?
- Streßtest: Das System wird überlastet: Deadlocks, Ressourcenlecks?
- Alpha Test: Test unter Kundenbeteiligung in den Prüflabors des Herstellers
- Beta Test: Installation des Systems bei einigen speziell ausgewählten Pilotkunden