

# **3. Der Entwicklungs-/ Wartungsprozeß: allgemeine Aspekte**

## ***Ziele:***

*Festlegung kritischer Arbeitsbereiche*

*Zieleigenschaften des Produkts eines*

*SW-Entwicklungsprozesses*

*SW-Erstellung heißt Modellieren/Strukturieren*

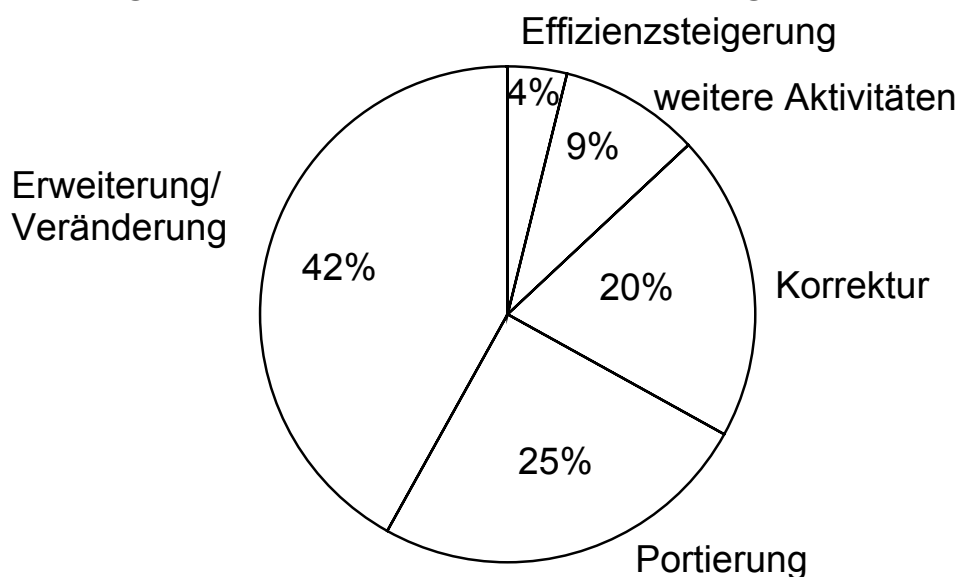
*Klärung Produkt/Prozeß in der ST*

Stand: 25.10.2001

# Zum Problem der Wartung

## Was ist Wartung

- Begriff "Wartung": Software verschleißt nicht  
Software ist langlebig und "beliebig" änderbar
- Veränderung in der Wartung:
  - Erweiterung des Systems (Bediener, Wartungspersonal, Operator)
  - Veränderung der Systemfunktionalität
  - abgemagerte Systeme
  - andere Bedieneroberfläche
  - Übertragung auf andere Basismaschinen (Prozessor, Compiler, Betriebssystem, Dateiverwaltung, Datenbanksystem)
  - Fehlerbeseitigung
  - Effizienzsteigerung
- Aufteilung in Problemklassen der Wartung



## Probleme

- spezifische Probleme der Wartung:
  - Aufbau eines Programmsystems, schwer zu verstehen, modifizieren, überprüfen: Softwarearchitektur/Entwurfsentscheidungen nicht festgehalten
  - Wartungsgeschichte implizit in bestehendem System
  - muß Firmen-/Abteilungskontext, Historie, Denkweise kennen
  - Inkonsistenzen zwischen einzelnen Softwaredokumenten
  - technische Dokumentation unvollständig/inkonsistent
- Begründung der Probleme:
  - Vielzahl technischer Fehler
  - organisatorische Fehler: Entwickler erhalten zu wenig Zeit und Ressourcen
  - Wartung unbeliebt

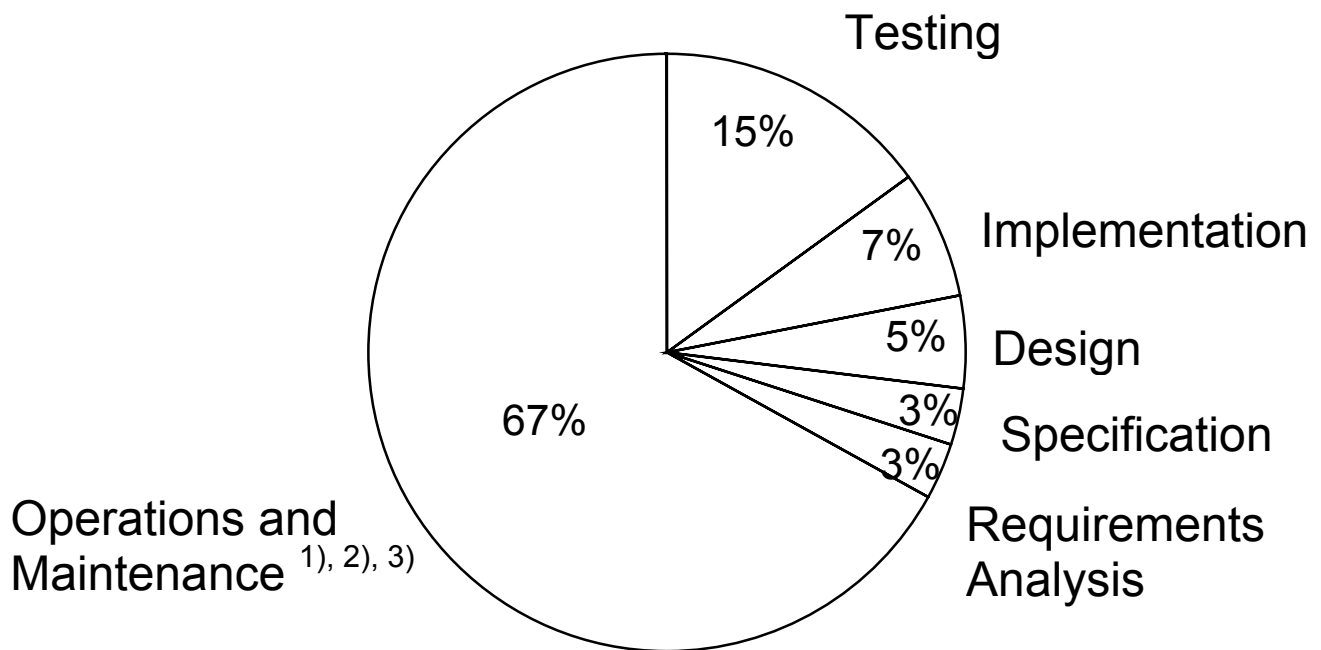
## Lösung

- Lösung durch präventive Wartung: eher Ziel Forderungen:
  - zukünftige Wartungsüberlegungen bei Entwicklung mitberücksichtigen
  - bei Wartung zukünftige Wartungsüberlegungen mitberücksichtigen
  - System stets auf neuestem Stand halten (Funktionalität, Bedieneroberfläche, Softwarearchitektur etc.)
- wichtiger Schritt: Nachdenken über Systemerweiterung
  - bei Erstellung der Anforderungsspezifikation
  - bei Erstellung der Softwarearchitektur
  - bei Modifikation beider

# Kritische Bereiche des Prozesses

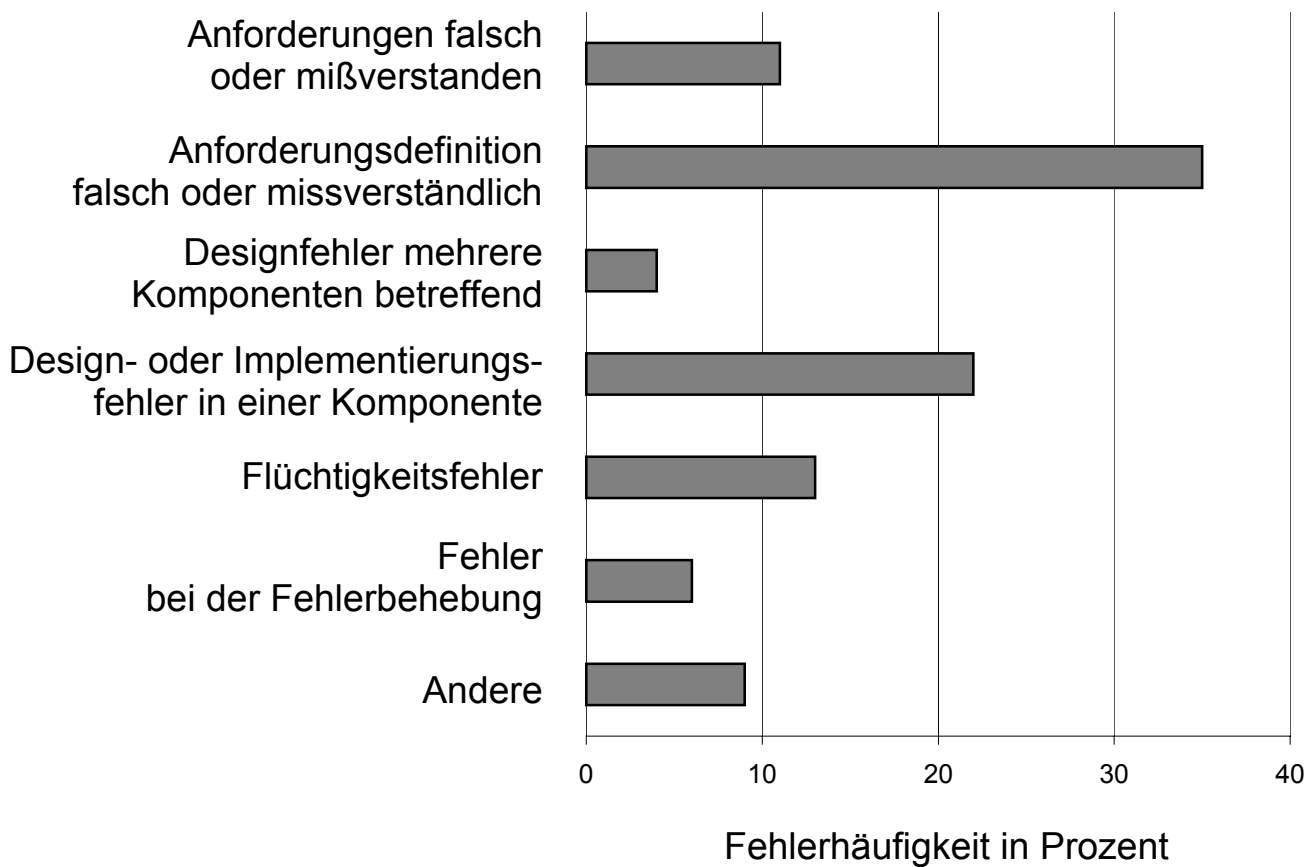
## Aufwand und Fehler

- Aufwand für Phasen



aus /Ze 79/

- Fehleraufteilung im Erstellungs-/Wartungsprozeß

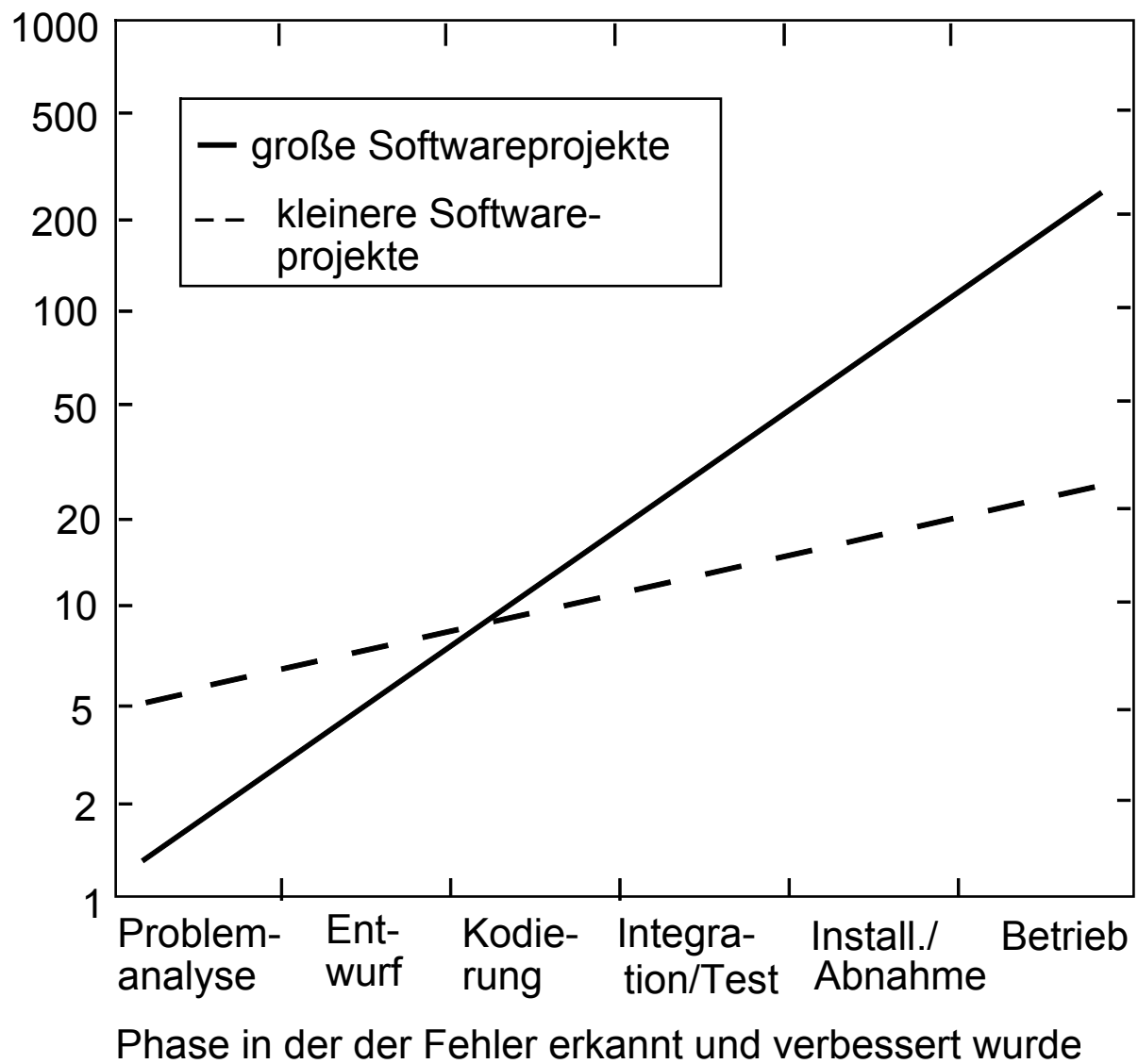


aus C.V. Ramamoorthy: Software Engineering - Problems and Perspectives, Computer 10/84

Fehler:

Verständnisprobleme	}	in Modellierung und Kommunikation
Verständigungsprobleme		
Ausdrucksprobleme		

- relative Kosten von Fehlern:



aus /Boe 84/

## **Folgerungen:**

- mehr Aufwand
  - für RE, damit nicht an den Anforderungen vorbeientwickelt wird, die Validierung einfacher wird
  - für Architekturmodellierung, da dort die wesentlichen Eigenschaften des Systems festgelegt werden
  - für Kosten-, Personalschätzung, damit Projekt wirtschaftlich erfolgreich ist bzw. überhaupt durchgeführt werden kann
- Nutzen bei mittelfristiger Optimierung:  
kurzfristige Optimierung führt zu mittelfristigen Schäden

## Was sind "Fehler"?

- Anforderungen nicht getroffen, nicht sauber beschrieben
- Projekt macht Verlust
- Projekt wird nicht zeitgerecht fertig
- Projekt ist in gegebener Zeit oder bei vorhandener Kompetenz überhaupt nicht durchführbar
- System realisiert Anforderungen nicht
- System ist ineffizient
- System erfüllt bestimmte Eigenschaften nicht, wie Übertragbarkeit, Erweiterbarkeit
- Baustein erfüllt nicht seine Beschreibung
- Komponentenbeschreibungen (Modulspezifikationen) passen nicht zusammen mit Programmcode
- Syntaxfehler
- Laufzeitfehler
- Dokumentation paßt nicht zur vorhandenen Realisierung
  - allgemeiner: Inkonsistenzen zwischen Dokumenten
- ...
- System wird nicht angenommen: Zwänge, ungewünschte Funktionalität, Ergonomie



# Eigenschaften von Programmsystemen

## Zu Eigenschaften des Produkts des Erstellungsprozesses

- Erstellung/Wartung:

*Anforderungsdefinition* → *Programmsystem in höherer  
Programmiersprache*  
?

Wie überhaupt?

Welche unterschiedlichen Möglichkeiten?

Auswahl nach welchen Kriterien?

- Viele Möglichkeiten:

- Softwarearchitektur

- Innenleben von Modulen: Datenstrukturen, Ablaufstrukturen,

...

- Bezeichnerwahl

Welche Eigenschaften fordern wir von Softwaresystem?

Auf welche Eigenschaft muß der Prozeß der Erstellung/  
Wartung hin ausgerichtet werden?

- Eigenschaften in der Anforderungsdefinition festgelegt:

Bedieneroberfläche, Effizienzparameter, ...

oft nicht oder nur teilweise festgelegt: Wahlmöglichkeit

# Eigenschaften des erstellten Programms

- Zuverlässigkeit
  - Korrektheit:  
Entwurfsspezifikation entspricht Anforderungsdefinition  
Implementation entspricht Entwurfsspezifikation
  - Robustheit:  
gegen falsche Eingabedaten (Anzahl, Aufbau, Wertebereich, Abhängigkeit)
  - Ausfallsicherheit:  
Hardware-, Softwarefehler (reproduzierbar, sporadisch)
- Bedienerfreundlichkeit (Außenverhalten)
  - Verständlichkeit
  - Angemessenheit
  - vernünftiges Fehlerverhalten
  - Uniformität

}

der  
Bediener-  
funktionen

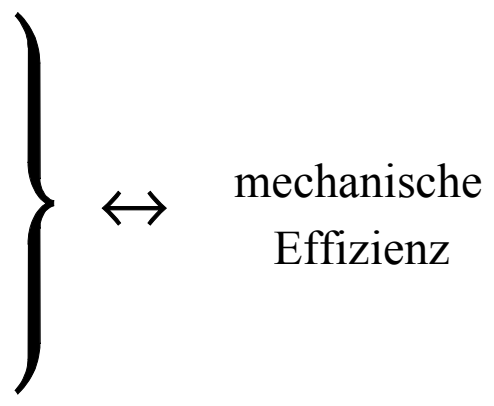
}

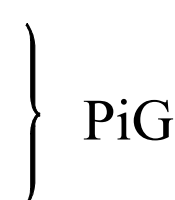
Prinzip der  
geringsten  
Verwunderung
- Flexibilität
  - Adaptabilität (bei Veränderungen des Außenverhaltens)
  - Portabilität (bei Veränderungen der Basismaschine)
- Lesbarkeit/Einfachheit
  - Forderung an Softwarearchitektur
  - Forderung an Programmiersprache
  - Programmierdisziplin

auch an andere Dokumente, Sprachen, Disziplin

- Effizienz
  - mechanische Effizienz: Laufzeitverhalten, Speicherplatzbedarf
    - Messen
    - Ausrechnen (Schranken oft größenordnungsmäßig:  
z.B. obere Schranke für schlechtesten Fall)
  - Tradeoff Laufzeit  $\leftrightarrow$  Datenspeicherplatz
  - Vernachlässigt:
    - Programmspeicher
    - Übersetzungs-/Neuübersetzungsaufwand
    - Programmerstellungsaufwand
  - Fazit:
    - Beschränkt sich auf dynamische Eigenschaften des Produkts
    - Effizienzeigenschaften des Prozesses unberücksichtigt

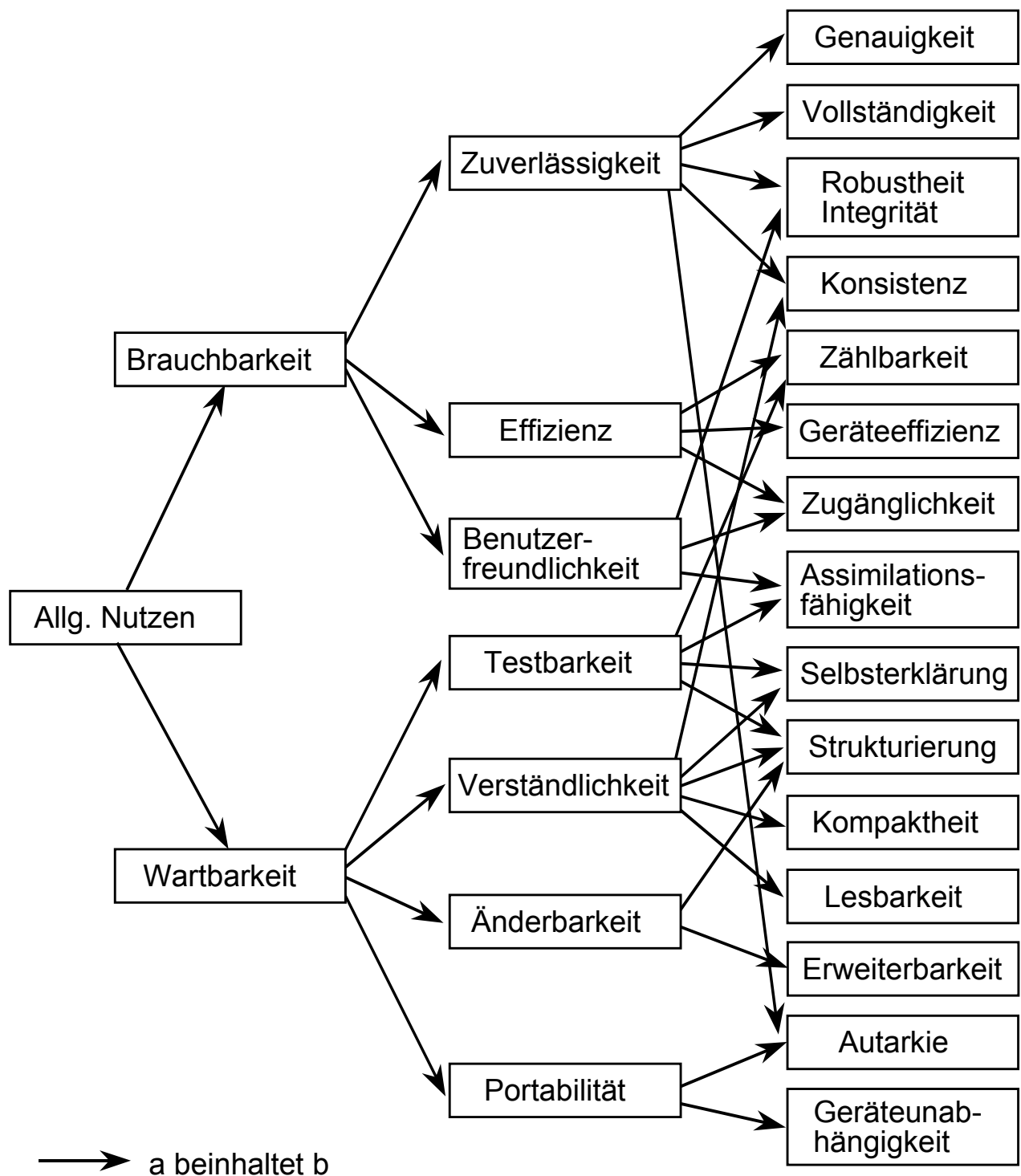
## Erfüllen der Eigenschaften

- Zielkonflikte
  - Korrektheit stets einzuhalten
  - Robustheit
  - Ausfallsicherheit
  - Verständlichkeit
  - vernünftiges Fehlerverhalten
  - Adaptabilität
  - Portabilität
  - Lesbarkeit/Einfachheit

mechanische Effizienz
- neben Forderungen an ganze Programmsysteme, Forderungen an seine Teile (Module, Teilsysteme)
  - Wiederverwendbarkeit
  - Kombinierbarkeit
  - Generierbarkeit
- alle obigen Ziele mit Ausnahme der mechanischen Effizienz haben mit Veranstaltung oder anderen Softwaretechnik-Veranstaltungen zu tun
  - Korrektheit: RE, PiG
  - Lesbarkeit/Einfachheit: PiG, PiK
  - Robustheit, Ausfallsicherheit, Benutzerfreundlichkeit  
indirekt: RE, werden oft übersehen und nachträglich hinzugefügt, bedingen Eigenschaften der Softwarearchitektur
  - Flexibilität: PiG
  - Effizienz des Prozesses (Strukturierung, Flexibilität): ST, PiG
  - Wiederverwendbarkeit
    - Kombinierbarkeit
    - Generierbarkeit

PiG

# Produkteigenschaften allgemeiner



aus B. Boehm et al.: Quantitative Evaluation of Software Quality, entnommen aus /Bal 82/, S. 10-14, leicht verändert

folgende Begriffswelt nicht 1:1 mit der sonst in der Vorlesung auftauchenden konsistent

- **Brauchbarkeit (usability), Nutzen ohne Produktänderungen (as-is utility):**  
Grad, in dem ein Produkt zuverlässig, effizient und benutzerfreundlich ist.
- **Wartbarkeit (maintainability):**  
Grad der Wahrscheinlichkeit, in dem ein in Einsatz befindliches Produkt nach Auftreten eines Fehlers innerhalb eines gegebenen Zeitintervalls wieder in funktionsfähigen Zustand gebracht werden kann.
- **Portabilität (portability):**  
Erforderlicher Aufwand, um ein Produkt von einer Hardware- und/oder Softwareumgebung in eine andere zu überführen.
- **Zuverlässigkeit (reliability):**  
Grad der Wahrscheinlichkeit, in dem ein im Einsatz befindliches Produkt während eines vorgesehenen Zeitintervalls unter den vorgesehenen Bedingungen die Funktion erfüllt und die Leistungen erbringt, die in der Anforderung spezifiziert sind.
- **Effizienz (efficiency):**  
Grad, in dem ein Produkt seine Aufgaben ohne Verschwendung von Ressourcen erfüllt.
- **Benutzerfreundlichkeit ("human engineering"):**  
Grad, in dem das Produkt seine Aufgaben erfüllt, ohne Zeit und Energie des Benutzers zu verschwenden bzw. ohne seine Motivation zu vermindern.
- **Testbarkeit (testability):**  
Erforderlicher Aufwand, ein Produkt zu testen, um sicherzustellen, daß es die geforderten Funktionen richtig ausführt.
- **Verständlichkeit (understandability):**  
Grad, in dem der Zweck eines Produkts einem Außenstehenden bei der Durchsicht der Unterlagen klar ist.
- **Änderbarkeit (modifiability):**  
Grad, in dem ein Produkt den Einbau von Änderungen erleichtert, wenn die Art der gewünschten Änderung festgelegt ist.
- **Geräteunabhängigkeit (device-independence):**  
Grad der Abhängigkeit eines Produkts von spezifischen Hardwareeigenschaften und/oder -konfigurationen.

- Autarkie (self-containedness):  
Grad, in dem ein Produkt von der Existenz anderer gleichrangiger Softwareprodukte bzw. ihrer Funktion unabhängig ist.
- Genauigkeit (accuracy):  
Grad, in dem die Berechnungen und die Produktausgabe ausreichend präzise sind, um den gewünschten Gebrauch sicherzustellen.
- Vollständigkeit (completeness):  
Grad, in dem ein Produkt den definierten Anforderungen entspricht.
- Robustheit (robustness):  
Grad, in dem ein Produkt eine wohlverständliche Reaktion bei nicht vorgesehener Verwendung erbringt und sein Funktionsfähigkeit bewahrt.
- Konsistenz (consistency):  
Grad, in dem ein Produkt nach einheitlichen Entwurfs- und Implementierungstechniken sowie in einheitlicher Notation entwickelt wurde.
- Zählbarkeit (accountability), Instrumentierung (instrumentation):  
Grad, in dem ein Produkt die Messung der Gebrauchshäufigkeit von Codeteilen oder die Identifizierung von Fehlern unterstützt.
- Zugänglichkeit (accessibility):  
Grad, in dem ein Produkt den selektiven Gebrauch von Produktteilen auch für andere Zwecke erleichtert.
- Assimilationsfähigkeit (communicativeness):  
Grad, in dem Form und Inhalt von Ein- und Ausgaben leicht geändert und angepaßt werden können.
- Selbsterklärung (self-descriptiveness):  
Grad, in dem ein Produkt genügend Information für den Leser enthält, um seine Objekte, seine Annahmen, Einschränkungen, Eingaben, Ausgaben, Komponenten und seinen Status zu bestimmen oder zu "verifizieren".
- Strukturierung (structuredness):  
Grad, in dem ein Produkt ein erkennbares Organisationsmuster seiner voneinander abhängigen Teile besitzt.
- Kompaktheit (conciseness):  
Grad, in dem ein Produkt nur notwendige, d.h. keine überflüssigen Informationen enthält.

- Lesbarkeit (legibility):  
Grad, in dem die Funktionen eines Produkts beim Lesen des Codes leicht erkannt werden können.
- Erweiterbarkeit (augmentability):  
Grad, in dem ein Produkt leicht um zusätzliche Anforderungen erweitert werden kann.

Folgende Begriffe haben sich ebenfalls im Sprachgebrauch eingebürgert:

Wiederverwendbarkeit (reuseability):

Grad, in dem Produktteile in anderen Anwendungen wiederverwendet werden können.

Kopplungsfähigkeit (interoperability):

Erforderlicher Aufwand, um ein Produkt mit einem anderen zu verbinden.

Allgemeingültigkeit (generality):

Grad, in dem ein Produkt ein breites Funktionsspektrum abdeckt.

Systemsoftware-Unabhängigkeit (software system independence):

Grad, in dem ein Produkt von der Systemsoftware-Umgebung (Betriebssystem, Dienstprogramme, Ein-/Ausgaberoutinen usw.) unabhängig ist.

Fehlertoleranz (error tolerance):

Grad, in dem ein Produkt auf jegliche Art von Fehlern in einen wohldefinierten Zustand übergeht, von dem aus seine Funktionsfähigkeit wieder hergestellt werden kann.

(ähnlich: graceful degradation)



# Die Modellierungsproblematik

## Allgemeines zur Modellierung

- Modellieren auf 

RE-	}	Niveau
PiG-		
PiK-		
PO-		
QS-		
DOK-		

gesucht: Prinzipien für Modellierung (Prozeß) und für dessen Ergebnisse (Softwaredokumente) und ihren Zusammenhang

- geeignete Modellierung
  - Voraussetzung für spezifische Qualitätsmerkmale  
z.B. für Softwaresysteme des letzten Abschnitts
  - Voraussetzung auch für allgemeine Eigenschaften von Softwaresystemen (und zugehörige Festlegungs-, Planungs- etc. Dokumenten)

Verständlichkeit	}	fußen auf	{	Vollständigkeit
Überprüfbarkeit				Widerspruchsfreiheit
Veränderbarkeit				„Minimalität“

- Voraussetzung für Zusammenhang verschiedener Modellierungsebenen

z.B. Vollständigkeit	}	der Anforderungsdefinition
		der Entwurfsspezifikation
		der PiK-Ebene
		...

- Voraussetzung für Arbeitsteilung

## **Zweck jeder Modellierung**

- Kommunikation: z.B. „Auftraggeber“, „Auftragnehmer“
- Prüfung eines “physischen” Systems vor dessen Bau
- Visualisierung zur Kommunikation bei der Erstellung:  
des Entwicklers mit sich selbst, der Entwickler  
untereinander
- Festlegungen auf Übersichtsniveau/Grobniveau  
für nachfolgende Realisierung
- Reduktion der Komplexität:  
Konzentration auf wesentliche Aspekte, auf  
Detailbeschreibung wird verzichtet (Abstraktion)
- Qualitätssicherung der Ergebnisse

- Komplexe Sachverhalte auf Übersichtsebene durch Graphen dargestellt, meist durch Diagramme repräsentiert (grafische Repräsentation)

## Diagramme in der ST: SA-Diagramme (RE)

# Architekturdiagramme (PiG)

# Flußdiagramme (PiK)

## Netzpläne (PO)

# Überdeckungsgraphen (QS)

# Datenflußgraphen (Optimierung)

## Firmenstruktur (PO)

...

- Graphenmodellierung
  - elementare (atomare) Objekte festlegen: Knoten  
verschiedene Objekte: Einteilung in Arten (Sorten, Klassen)  
durch Knotenmarkierung gekennzeichnet  
Werte durch Attribute
  - Beziehung durch gerichtete Kanten:  
verschiedene Arten von Beziehungen durch Markierungen: Sorten, Attribute
  - Konsistenzbedingungen: Unterscheidung zulässiger von unzulässigen Graphen
  - damit Einführung einer Klasse von knoten- und kantenmarkierten attributierten Graphen
  - große Wahlfreiheit bei der Modellierung
 

Niveau der Betrachtung	}	für Auswahl der Graphenklasse
welche Aspekte modelliert man		
viele Möglichkeiten der Modellierung eines Sachverhalts in einer gegebenen Graphenklasse		

# Beschreibung durch markierte Graphen

- Definition
  - $\Sigma_V$  Markierungsalphabet für Knoten
  - $\Sigma_E$  Markierungsalphabet für Kanten
  - markierter gerichteter Graph  $G = (V, E, S, T, M_V, M_E)$
  - $V$  Knoten (Vertices)
  - $E$  Kanten (Edges)
  - $S: E \rightarrow V$  (Source)
  - $T: E \rightarrow V$  (Target)
  - $M_V: V \rightarrow \Sigma_V$
  - $M_E: E \rightarrow \Sigma_E$
  - $M \in \Sigma_V$  repräsentiert Klasse von Objekten
  - $M \in \Sigma_E$  repräsentiert Klasse von Beziehungen

} oder über Relation für jede Kantenmarkierung

Kanten- und Knotenmarkierungen gegebenenfalls strukturiert: eine Markierung ist ein zusammengesetzter Typ, jeder markierte Knoten, jede markierte Kante hat auch Wert dieses Typs.

- Modellierungsproblematik:
  - Welche Knotenmarkierungen werden ausgewählt?
  - Welche Kantenmarkierungen werden ausgewählt?
  - Welche Struktur haben Typen für Knoten, Kanten?
  - Welche Objekte sind atomar, welche zusammengesetzt?
  - Verbindungsstrukturen der Graphen
  - Bei attributierten Graphen: Was steckt man in die Graphstruktur, was in Werte (Attribute)?
- vernetzte Graphen für Zusammenhänge: was muß von einem Graphen intern, von außen, für Zusammenhänge gesehen werden
- hierarchische Graphen für Abstraktionsstufen

# Prinzipien der Modellierung: Übersicht

- Ergebnis (Produktmodellierungsprinzipien)
  - Abstraktion
  - Strukturierung
    - Hierarchiebildung
    - Modularisierung
    - Lokalität
    - Mehrfachverwendung
    - Redundanz
    - Festhalten von Ähnlichkeiten
  - geringste Verwunderung
- Vorgehen (Prozeßmodellierungsprinzipien)
  - Beachtung des Zusammenhangs mit anderen Dokumenten
  - Lebenszyklusdokumente in Zusammenhang mit Projektorganisation, Dokumentation, Qualitätssicherung
  - Rücksichtnahme auf allgemeine Erkenntnisse, Normen, Standards

# Produktmodellierungsprinzipien

## Abstraktion

- Definition: Verallgemeinerung, absehen vom Speziellen, Konkreten
- Beispiele:
  - Benutzermaschine im Gegensatz zur Basismaschine
  - Programmiersprachenmaschine - Assemblermaschine
  - Kontrollstrukturen - Nachbildung mit Sprüngen, bedingten Sprüngen
  - Datenstrukturierung - Speicherverwaltungsfunktion
  - Datenabstraktion - Betrachten der Feinstruktur eines Objekts
  - abstrakte Syntax - konkrete Syntax
  - Konstrukte einer Klasse von Programmiersprachen - spezielle Konstrukte einer Programmiersprache
  - Datenbankfunktionen - Zugriffsmechanismen
  - virtuelles Gerät - spezielles Ein-/Ausgabegerät
  - generische Komponente - konkrete Komponente
  - abstrakte Klasse - Unterklasse
- Abstraktion bedingt/setzt voraus:
  - Erkennen der Detailstruktur
  - Erkennen/Gewichten wesentlicher Merkmale
  - Ordnen/Klassifizieren von Merkmalen
  - Erkennung allgemeiner Strukturen

# Strukturierung

- Definition: reduzierte Darstellung, die Charakter offenbart, losgelöst von Details, beinhaltet die wesentlichen Merkmale
- Beispiele:
  - Strukturierte Programmierung (Sequenz, Fallunterscheidung, Wiederholung)
  - Datenstrukturen
    - Datentypkonstruktoren:  
Felder, Verbunde, Mengen, Zeiger, Aufzählungen, Unterbereiche, ...
    - übliche Datenstrukturen:  
lineare Liste, verkettete Liste, Bäume, ...
  - Blockstruktur (Gültigkeit, Sichtbarkeit)
  - ...

# Strukturierung, insbesondere Hierarchiebildung

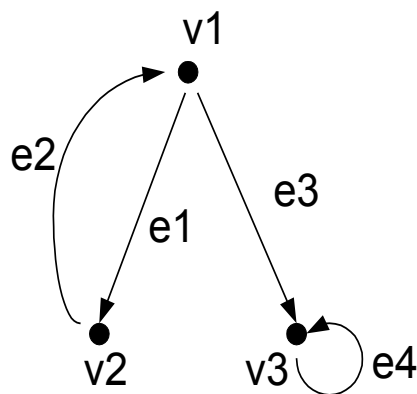
- Beispiele:
  - besteht aus, ist Teil von
  - ist Hilfsmittel zur Realisierung eines anderen Bausteins
  - ist Teilproblem von
  - A kann auf B zugreifen
  - A ordnet B Ressourcen zu
  - A gibt Arbeit an B ab
  - B ist Spezialisierung von A

- Hierarchiestrukturen:

nicht unbedingt sinnvoll für die Softwaretechnik  
(aus /Bal 82/, ergänzt)

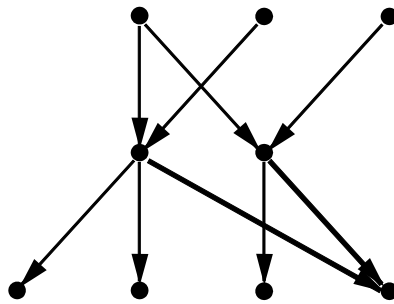
- Beispiel:

Durch gerichtete Graphen können beliebige Strukturen  
(Chaos) dargestellt werden:

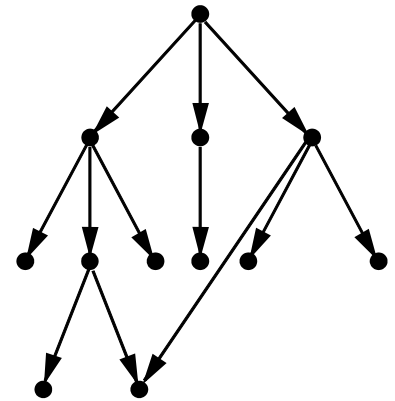




– Beispiele:

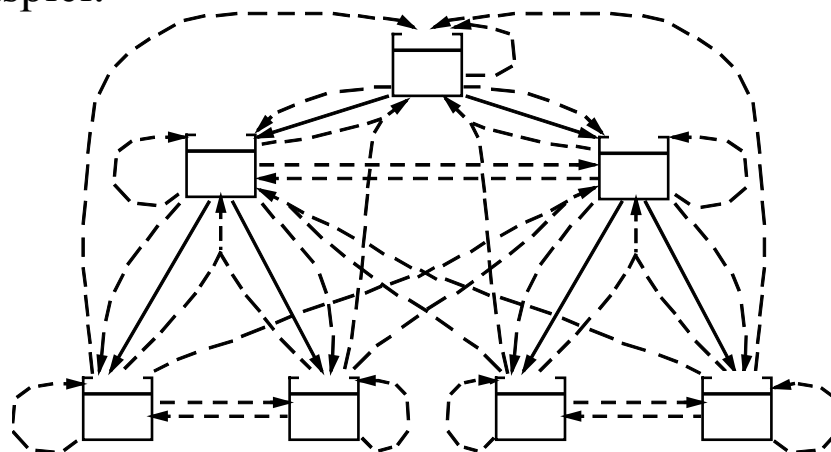


Beschreibung der dynamischen Aufrufstruktur in PSL (UTILIZE-Relation)



Beispiel einer allgemeinen Systemstruktur (Teilfunktionsrelation)

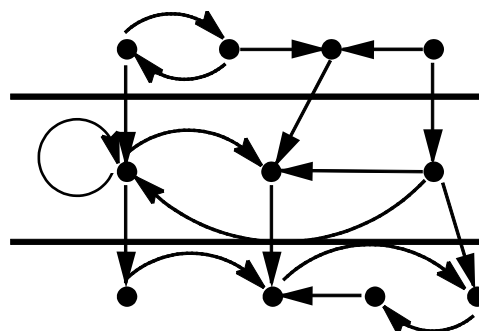
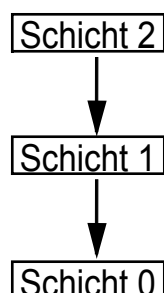
– Beispiel:



→ lokales Enthaltensein  
 --- potentielle lokale Benutzbarkeit

– Beispiel:

grafische Darstellung von Schichten mit linearer Ordnung



# Strukturierung; insbesondere Modularisierung

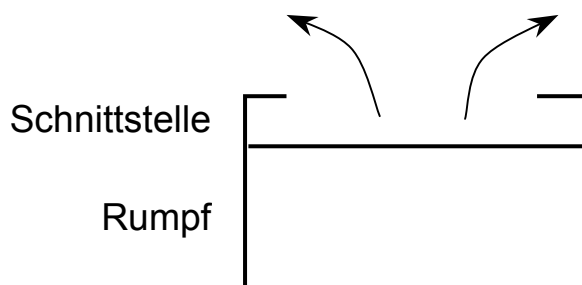
- Definition: Schnittstelle

Detailverbergung Innenleben: schwarzer Kasten

- Beispiele:

- Prozedur, Funktion, Modul, Teilsystem, in Architektur
- technisches Gerät
- Abschnitt in Buch, Aufsatz
- Kasten in Aktionsdiagramm
- Baustein auf Platine
- Komponente eines Rechnersystems

Notation im PiG:



## **Strukturierung, insbesondere Lokalität**

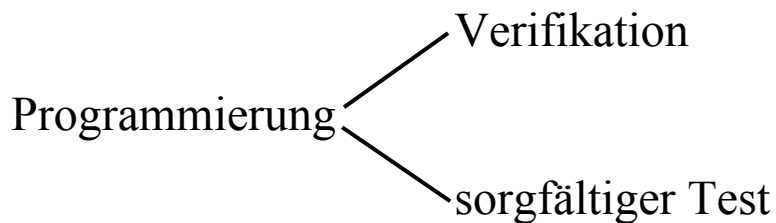
- Definition: Information für ein Teilproblem ist an einer Stelle verfügbar
- Beispiele:
  - Benutzertaschenbuch seitenorientiert: dort jeweils alle Informationen
  - Diagramm höchstens eine Seite
  - lokal deklarierte Objekte nur in dem entsprechenden Gültigkeitsbereich gültig und sichtbar
- Lokalität im Sinne von:
  - Unabhängigkeit unter Verwendung von Redundanz
  - überschaubare Größenordnung
  - Unabhängigkeit, Freiheit, Arbeitsteilung
  - Schutz vor unerlaubtem Zugriff

## Prinzip der Mehrfachverwendung

- Definition: Wiederverwendung von Überlegungen und deren Ergebnissen, gegebenenfalls mit leichter Anpassung
- Beispiele:
  - eigene Überlegungen wiederverwendet
  - Abkupfern oder "Standard" übernehmen
  - mehrmalige Verwendung eines Bausteins
  - mehrmalige Verwendung eines (z.B. Architektur-) Rahmens
  - mehrmalige Verwendung einer generischen Schablone, Generatorbausteins
  - mehrmalige Verwendung eines Vorgehens  
(→ Vorgehenspr., hier dessen Beschreibung)

## Prinzip der Redundanz

- Betrachtung verschiedener Sichten desselben Sachverhalts  
z.B. SADT  
z.B. RE: Prozeß-, Informations-, Kontrollsicht
- Betrachtung verschiedener zusammengehöriger Sachverhalte: Anforderungsdefinition, Architektur, Implementierung
- Betrachtung desselben Sachverhalts mit verschiedenen Formalismen



- statische Festlegung vorab  
Deklarationsgebot in Programmiersprache  
Importe vor statischer Benutzung

## Prinzip des Festhaltens von Ähnlichkeiten

- Abspalten von Gemeinsamkeiten } über mehrere  
Festlegung nur des Speziellen } Stufen
- Beispiele:
  - Entitytypen
  - Klassenbibliotheken in objektorientierten Sprachen
- Formuliert durch:
  - Generalisierung/Spezialisierung
  - Generizität
  - Mustervorgaben (z.B. Rahmenarchitektur)
  - Standards

# Prinzip der geringsten Verwunderung

- Definition: ?

Jeder kennt Verletzung!

Warum so? Das ist doch unverständlich!

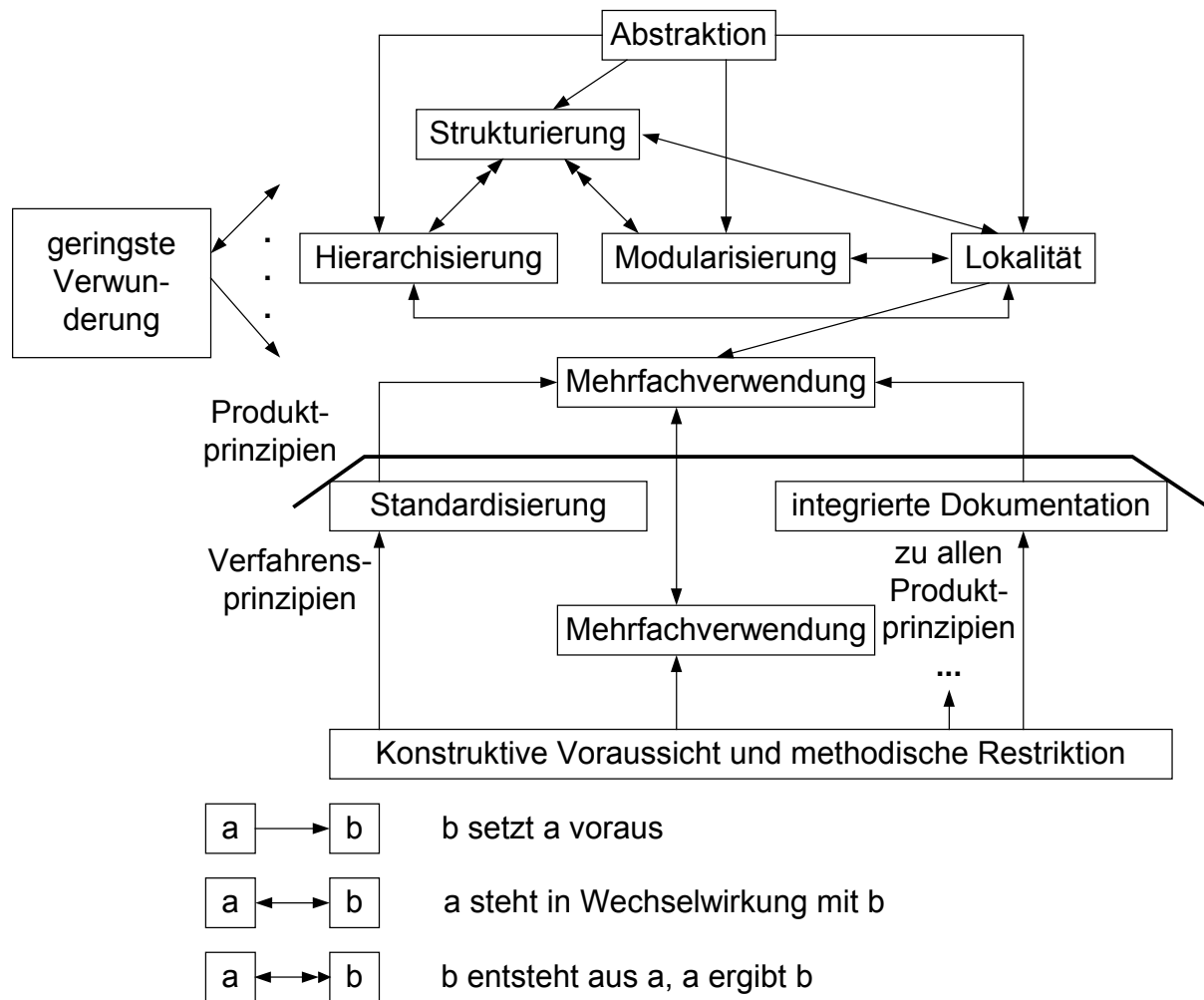
unangemessen!

Es ist doch naheliegend, es so zu machen.

- Verletzung:

- Bedienerchnittstellengestaltung
  - unverständliche Funktionen
  - unangemessene Funktionen
  - unverständliche Meldungen
  - unverständliche erwartete Eingaben
  - uneinheitliche Gestaltung
- ungegliederte, sehr stark vernetzte Entwürfe
- Spagettiprogramme
- falsche Verwendung von Rekursion

## Zusammenhang Modellierungsprinzipien



## Fachsystematisches Netz der Prinzipien

aus /Bal 82/, modifiziert



# Vorgehensprinzipien

- Prinzip der Integration der Entwicklung/Wartung mit der Dokumentation
  - Benutzerdokumentation bereits in Anforderungsdefinition
  - technische Dokumentation begleitend
- Prinzip der Integration der Entwicklung/Wartung mit Qualitätssicherung
  - Überlegungen bereits in Anforderungsdefinition festgelegt
  - Qualitätssicherung zyklusbegleitend
- Prinzip der Integration der Entwicklung/Wartung mit der Projektorganisation

# Prinzip der Mehrfachverwendung

- "Programmsysteme aus Fertigteilen",  
"Programmsystemsynthese"  
andere Formen: s.o.
- Probleme:
  - Was ist vorhanden?
  - Produktarchiv nötig
  - Einarbeitungsaufwand und mangelnde Dokumentation
  - aufwendige Umstellungsarbeiten
  - kein finanzielles Interesse des Auftragnehmers
  - kein persönliches Interesse des Entwicklers
- Produktarchiv
  - Auskunftsdienste
    - Übersicht
    - Klassifizierung
    - Kurzerläuterungen
  - Stöbern (Browsing)
  - Verwalten: Aufnahme, Änderung, Löschung
- Vorteile:
  - Vermeidung von Mehraufwand: Zeit und Kosten
  - Anregungen durch Produktarchiv
  - Prototyping erleichtert

## **Prinzip der Standardisierung/Normung**

- Definition: Festlegung eines Standards/einer Norm, an den/die sich alle halten oder halten müssen
  - nötig bei starker Arbeitsteilung
  - erleichtert Einarbeitung
  - Voraussetzung für Qualitätskontrolle oder Wartung durch andere Personen
- Beispiele:
  - Lebenszyklusmodell
  - Gliederungsschema für ein Softwaredokument, z.B. Pflichtenheft
  - Einigung auf bestimmte "Methoden"
  - Dokumentationsrichtlinien
  - Einrückschema bei Quelltext
  - DIN-Normen
  - firmenspezifische Standards
  - weitverbreitete Werkzeuge
  - (firmen-, abteilungs-, anwendungs-)spezifische Vorgehensweisen

# Allg. Begriffe der Softwaretechnik

## Sprachen

- Dokumente in formaler Sprache (Notation) notiert  
formale Sprache/Kunstsprache  $\leftrightarrow$  natürliche Sprache  
weites Verständnis von Sprache: Diagrammsprachen,  
Graphsprachen
- Problembereiche (formaler) Sprachen (Semiotik)
  - Syntax: Welche Zeichenfolgen, Diagramme, Graphen  
sind korrekt aufgebaut?  
lexikalische Syntax  
kontextfreie Syntax: Aufbau "an einer Stelle"  
aus welchen Textfragmenten, in welcher Reihenfolge  
wie ist Teildiagramm aufgebaut  
kontextsensitive Syntax: Querbeziehungen zwischen  
Sprachelementen  
Beziehungen definierender zu angewandtem Auftreten  
von Textfragmenten  
globale Zusammenhänge  
Ausscheiden verbotener Situationen
  - Semantik:  
Bedeutung einzelner Sprachelemente  
Bedeutung des Zusammenhangs von Sprachelementen  
dynamische Semantik
  - Pragmatik: Verhältnis der Sprache zur Umwelt  
mechanische Pragmatik: Auftragbarkeit,  
Werkzeugunterstützung  
menschliche Pragmatik: Wie gut für Modellierung, für  
bestimmte Anwendungsgebiete  
ökonomische Pragmatik: Wert der Dokumente in dieser  
Sprache etc.

# Begriffe der Softwaretechnik

- Prinzip:  
Grundsatz, den man dem Handeln zugrundelegt  
fachgebietsübergreifend,  
Beispiel Modellierungsprinzip
- Technik:  
Hilfen, um gegebenes Ziel schneller, sicherer, effizienter zu erreichen  
nichtautomatisiert: Prinzipien, Konzepte, Sprachen, Methoden, Verfahren, Lehr- und Lernmaterial, Erfahrungen, Muster  
(teil)automatisiert: Werkzeuge, Geräte, Komponenten, Rahmen, Dienstprogramme
- Methode:  
planmäßig angewandte, begründete, zielgerichtete Vorgehensweise zur Erreichung eines Ziels im Rahmen bestimmter Prinzipien, mit Einsatz von Techniken
- Verfahren:  
ausführbare Vorschrift zum gezielten Einsatz einer Methode
- Werkzeug:  
(teil)automatisierte Unterstützung von Verfahren, Methoden, Notationen, Prinzipien
- Notation:  
Sprache, in der Softwaredokument erstellt wird  
Sprache unterstützt "Methode", die Prinzipien folgt

# Paradigmen der Softwareerstellung

Definition: durchgängige Anwendung eines Konzepts  
Ziel anders

Entwicklungsprozeß anders  
insbesondere anderes "Phasenmodell"

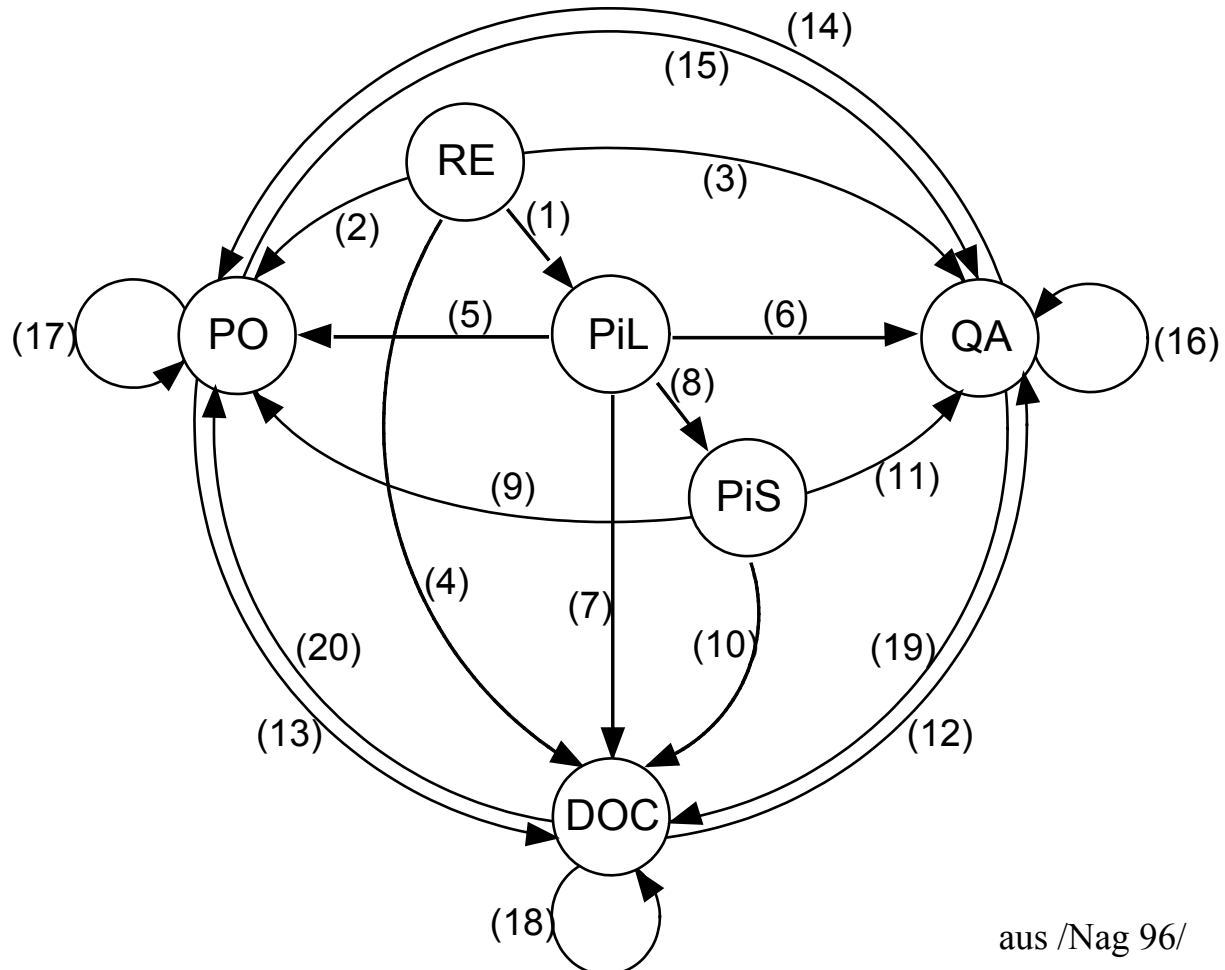
Paradigmen:

- Klassisches, ingenieurmäßiges Paradigma (s. Vorlesung):  
diskretes Modell, manuelle Erstellung der Dokumente
- Programmieren ist lediglich Spezifizieren  
Spezifikationssprache, automatische Übersetzung  
(mehrdeutiger Begriff "Spezifikation")
- Programmieren ist schichtenweise abstrakte  
Implementierung  
Entwurf und Implementierung (abstrakt) verzahnt
- Programmieren ist Transformieren  
Transformation vom Softwareentwickler, eventuell  
unterschiedliche Sprachen
- Programmieren ist Zusammensetzen von Funktionen  
allgemeiner: applikative Programmierung
- Programmieren ist Anwendung von Algebra  
abstrakte Datentypen, algebraische Spezifikation
- Programmierung ist das Zusammenspiel lose gekoppelter  
Klassen: objektorientierte Programmierung
- Programmierung ist das Zusammenspiel von Prozessen, die  
untereinander kommunizieren: prozeßorientierte Progr.
- Programmierung ist Anwendung von Logik  
Prolog, Verifikation
- Programmierung ist Angeben von Regeln  
Expertensysteme, künstliche Intelligenz

# Konfigurationen und Prozesse

## Die Gesamtkonfiguration eines technischen Entwicklungsprozesses

- Arbeitsbereiche und ihre Abhängigkeiten (Wiederholung)



- pro Arbeitsbereich:
  - ein Dokument (z.B. eine Architektur bei kleinerem System)
  - hierarchisch angeordnete Dokumente (z.B. Architektur mit unabhängigen Teilsystemen)
  - mehrere Dokumente "ohne" Zusammenhang (z.B. Bedienerdokumentation und technische Dokumentation, Module zu Architektur)
  - mehrere zueinander in Bezug stehende Dokumente (z.B. für verschiedene Sichten im RE)

# Produkte auf verschiedenen Ebenen

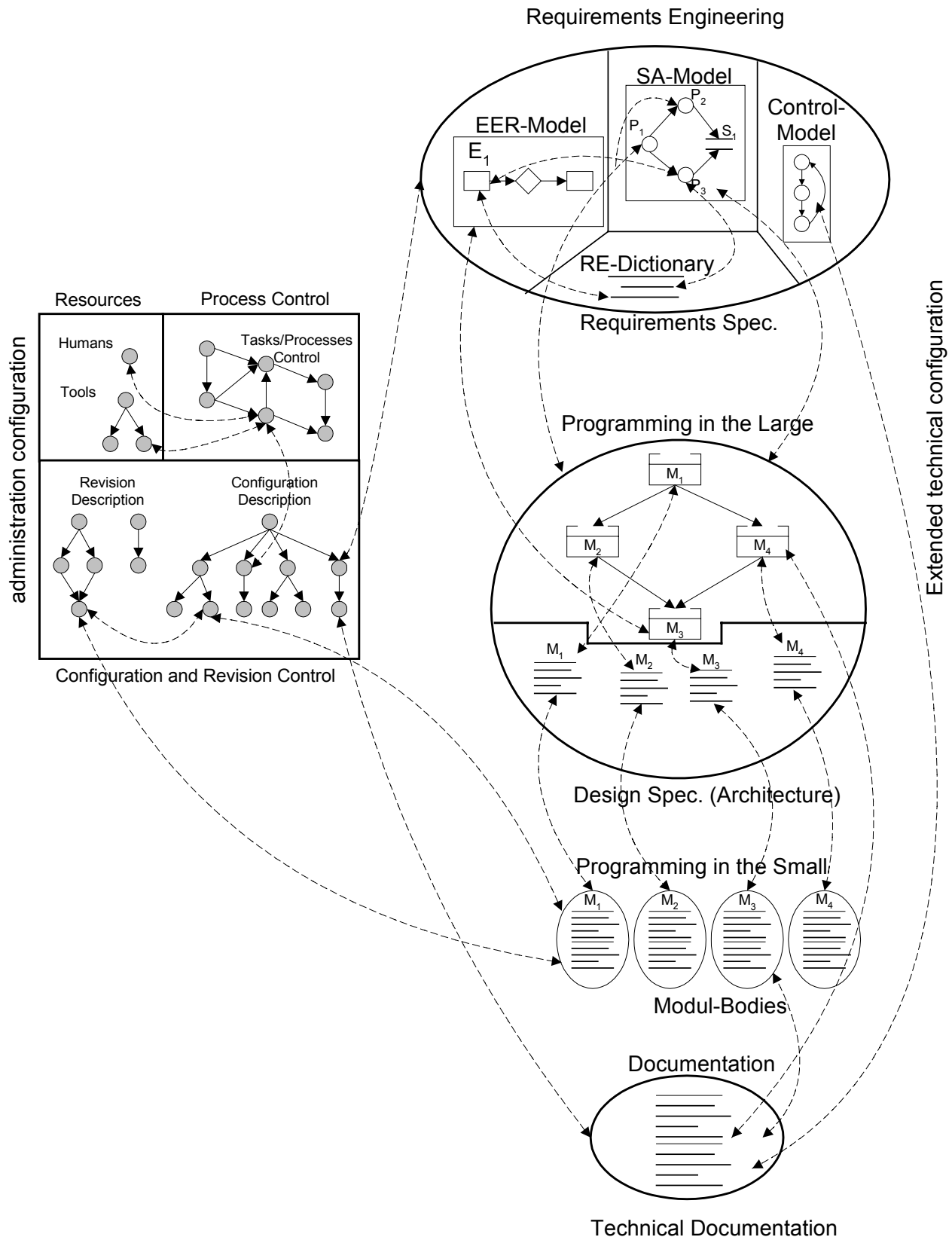


Fig. 1.15 An overall configuration for a software system according to the IPSSEN scenario

aus /Nag 96/



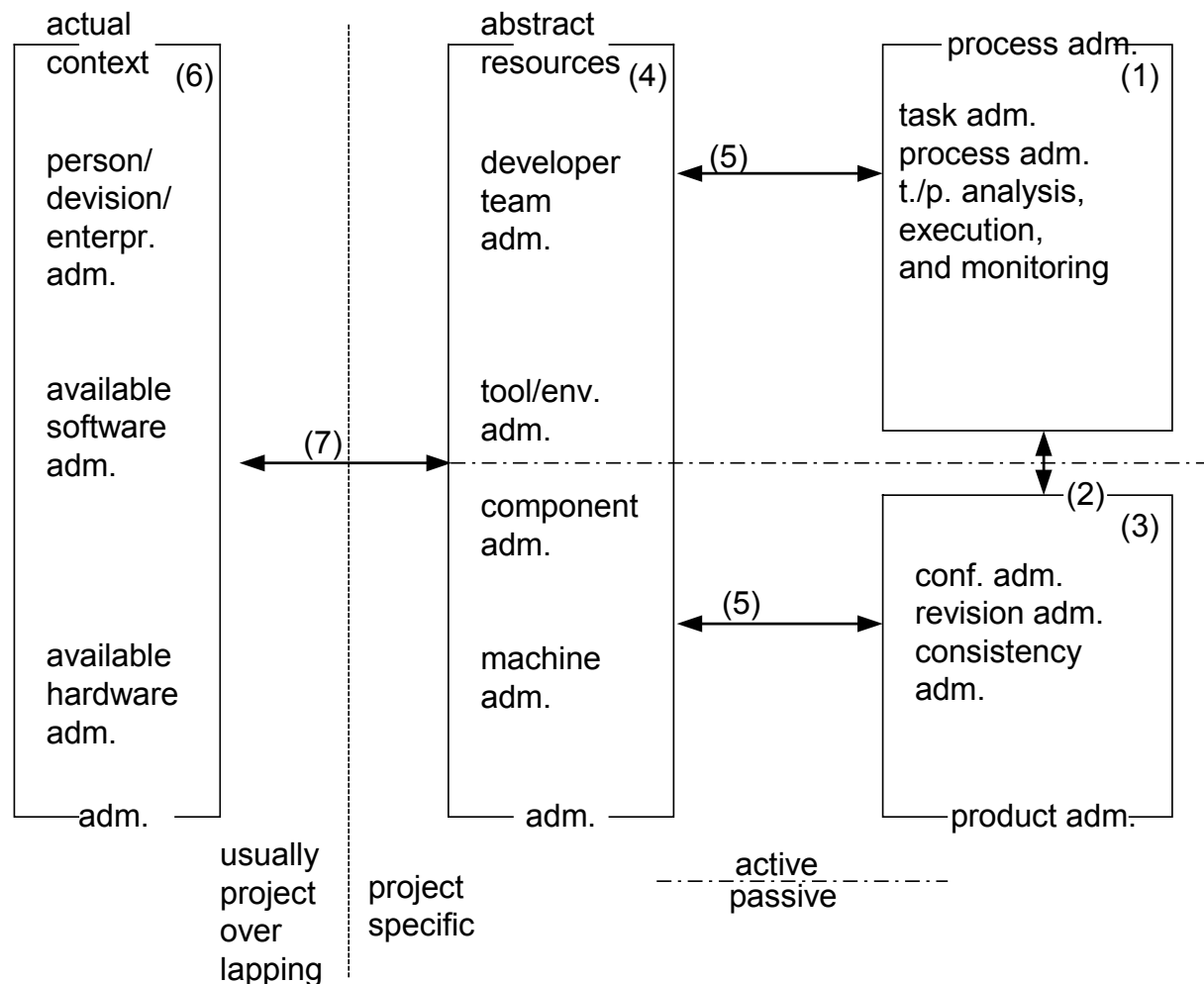
- "Produkt"begriff:
  - endgültiges Produkt (z.B. Quelltext des Systems)
  - technische Konfiguration (zusätzlich Architektur, Anforderungsspezifikation)
  - erweiterte technische Konfiguration (zusätzlich Qualitätssicherung, Dokumentationsdokumente)
  - administrative Konfiguration

⇒ Gesamtkonfiguration
- für   Erstellung  
           Wartung  
           Wiederverwendung
- Beziehungen:
  - feingranulare technische Beziehungen
  - feingranulare administrative Beziehungen
  - Beziehungen zwischen Administration und technischer Konfiguration

- Gesamtkonfiguration:
    - technische Ebene: erweiterte technische Konfiguration
    - Ergebnisse der Entwickler:
      - Dokumente mit Internstruktur und Querbeziehungen
    - Organisationsebene: administrative Konfiguration
    - Ergebnisse der Projektorganisation:
      - grobgranular: Dokumente, Prozesse nicht wie, sondern was ("Platzhalter"information)
      - intern feingranular
- } beide in Wechselwirkung
- administrative Konfiguration (eine mögliche Einteilung, "verallgemeinerter Workflow")
    - Prozeßverwaltung
    - Produktverwaltung (Konfigurations-, Versionsverwaltung)
    - Ressourcenverwaltung
    - Kontextverwaltung
  - allgemeingültig, falls geeignet parametrisier- und anpaßbar
    - Softwareentwicklung,
    - CIM (Entwicklungsbereich),
    - ...

# Die administrative Konfiguration

- Motivation: keine Internstruktur von Dokumenten, keine Festlegung der Prozesse, allgemeine Verwendung, Parametrisierung
- Einteilung der administrativen Konfiguration



aus /Nag 96/

- (1) Aufgaben/Prozeßverwaltung:  
Aufgaben/Prozeßnetze aufstellen, Aktoren für Prozesse, Attribute für Zeit (Anfang, Ende, Dauer), Attribute für Zustand (bereit, aktiv, suspendiert), schrittweise Ausführung, Ausführung beobachten (<sup>3</sup> Dynamik), Analysen auf „vollständigen“ Netzen
- (2) Integration mit Produktadministration:  
Ein-/Ausgaben sind Revisionen von Dokumenten/Teilkonfigurationen, umgekehrt für jeden Bestandteil einer Konfiguration/ Abhängigkeitsbeziehung muß es Aufgabe geben

- (3) Produktadministration:  
(Teil-)Konfigurations-, Revisionsverwaltung: Konfiguration ist grobgranulare Produktbeschreibung (Bestandteile, Abhängigkeiten, Gesamtzustand), Komponenten und Konfigurationen in Versionen durch Nachfolgebeziehung geordnet, Nachfolgebeziehung nicht "semantisch"
  - (4) Verwaltung abstrakter Ressourcen  
aktive Ressourcen (Entwickler, Werkzeuge), passive Ressourcen (benötigte Komponenten für Produkt oder für Unterstützung aktiver Ressourcen); Rollen, Teams von Rollen (abstrakte Gruppen); high-level Werkzeuge, primitive Werkzeuge; Begriff abstrakt: Eigenschaften und Fähigkeiten werden vorausgesetzt; Akteure Menschen durch Werkzeuge unterstützt, manchmal Werkzeuge, als Ressourcen prinzipiell der gesamte Kontext modellierbar (Hardware bis zu Schreibtisch, Telefon)
  - (5) Integration Aufgaben/Produkt- mit abstrakter Ressourcenverwaltung:  
Zuordnung Rollen, Gruppen zu Aufgaben/Prozessen, Konfigurationskomponenten zu abstrakten Komponenten
  - (6) aktueller Kontext:  
Beschreibung Firma, aktuelle Ressourcen wie Personen, Hardware/Software, aktuelle Ressourcen haben Möglichkeiten und Restriktionen
  - (7) Integration abstrakter und konkreter Ressourcen:  
Verfügbarkeit von aktuellen Ressourcen, Auslastung aktueller Ressourcen: Betrachtung verschiedener Projekte in einem Kontext
- Enge Integration von (1) - (7) durch Operationen wie Aufgabe, Revisionen  $A^j$ ,  $B^j$  Eingabe aus Konfiguration  $C^m$ , Revisionen in Revisionsgeschichte eingebettet, Rolle verantwortlich für Aufgabe, durch abstraktes Werkzeug  $T^h$  unterstützt, Rolle durch Person  $Pe^l$  zu Zeitpunkt  $[t, t + k]$  wahrgenommen, Einplanung eines konkreten Werkzeugs
  - "Semantische" Operationen auf administrativer Ebene: Qualitäts-, Dokumentationskontrolle als Vorsorge

- Klärung der Begriffe zur Projektorganisation aus der Softwaretechnik:
  - Projektplanung<sup>(1)</sup> im Sinne der Organisation der Arbeitsteiligkeit, Schätzung:  
 Aufbau/Änderung der administrativen Konfiguration, Eingabe von Schätzwerten  
 Durchführung, Management<sup>(2)</sup>:  
 auf Netzen Tokenspiel, Konfigurationsverwaltung, Ressourcenverwaltung  
 Überwachung: Instrumentierung, Monitoring des Geschehens, Beobachtung der Werte und Vorgaben
  - Management der Prozesse: Aufgabenverwaltung  
 Management der Produkte: s.o.  
 Management des Projekts: ?  
 betrachten auch Integration  
 betrachten auch Ressourcen- und Kontextverwaltung
  - wir unterscheiden strikt zwischen der administrativen und der technischen Ebene: führt zur Klärung und Präzisierung
- Wir werden später sehen, daß obige Phaseneinteilung nicht durchzuhalten ist
  - 1) zu anderen Semantiken von Planung s. später
  - 2) Durchführung umfaßt auch technische Aspekte

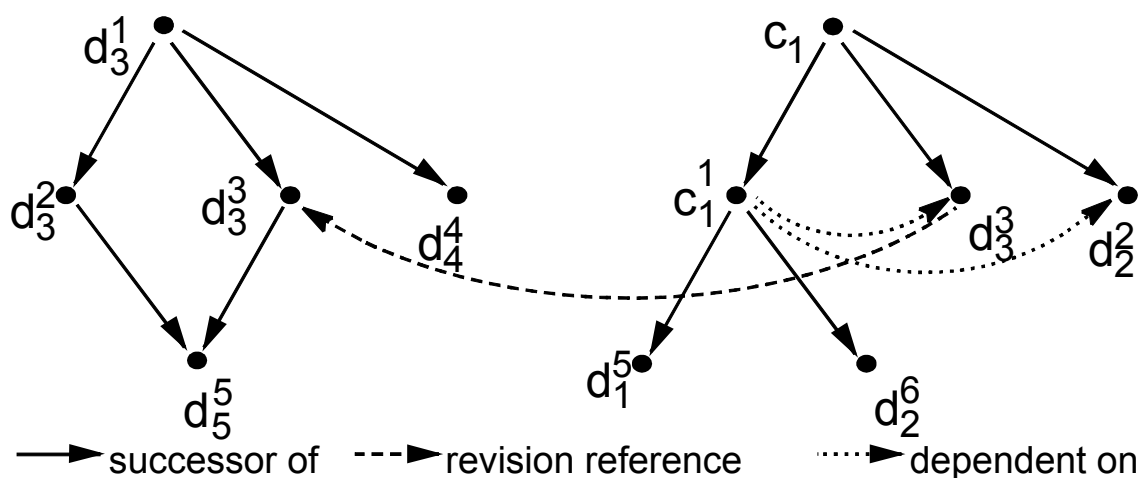
# Modelle und Gestalt der Gesamtkonfiguration

- Festlegung der Modelle
  - technische Modelle für Requirements Engineering, Architekturmodellierung, Implementierung, Qualitätssicherung, Dokumentation:  
zugehörige Sprachen, Methoden, Standards festlegen  
Integrationsbeziehungen festlegen
  - administrative Modelle (für obige Einteilung)  
Aufgaben-/Prozeßmodell  
Konfigurations-, Revisions(beschreibungs)modell  
Ressourcenmodell  
Kontextmodell  
Integrationsbeziehungen festlegen
  - Wechselwirkungen zwischen beiden Ebenen festlegen
- Beispiel Konfigurations-/Revisionsmodell

Zustände von Revisionen, Konfigurationsbestandteilen festlegen

Beziehungen festlegen

Revisionsgeschichte für Dokument  $d_1$



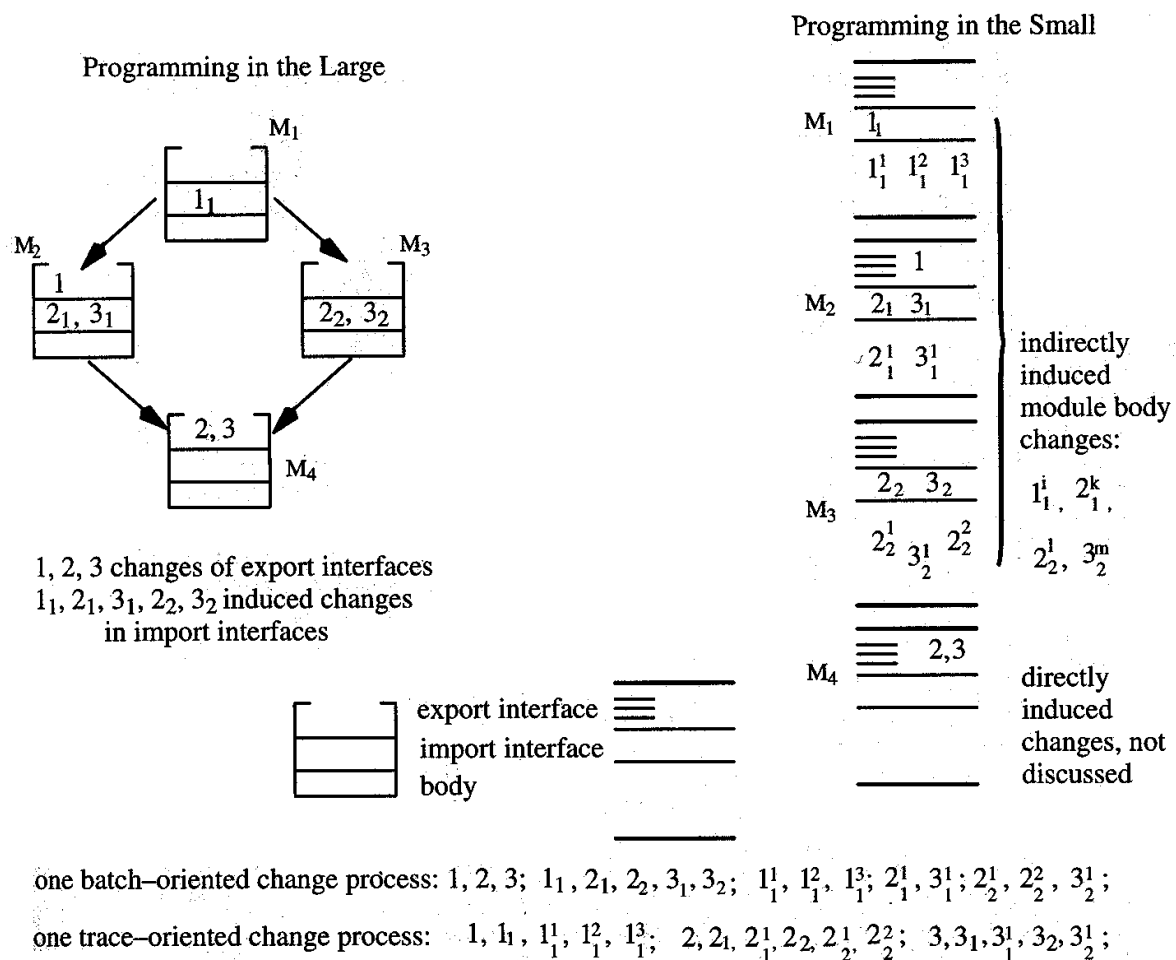
## Prozesse (verschiedener Art, auf unterschiedlichen Ebenen)

- bisher nur Ergebnisse der Tätigkeit von Entwicklern, von Managern betrachtet:
  - Wie wird ein Teilsystementwurf gemacht?
  - Wie wird ein Netz erstellt/modifiziert?
- Somit Unterscheidung Prozesse und Beschreibungen, die Prozeßabläufe darstellen
  - Beispiel Aufgabenverwaltung:
    - Prozeß eines Managers erstellt/modifiziert Netz, dieses Netz ist ein Produkt des Prozesses und koordiniert Entwicklerkooperation
- Klassifikation von Prozessen:
  - Granularität: gröbstgranular (Lebenszyklusmodell), grobgranular (Workflow-Ebene), feingranular (technische Ebene), Arbeitsplatz des Entwicklers
  - Ebene, was: technischer Prozeß (z.B. Entwicklerprozeß), administrativer Prozeß (Managementprozeß)
  - Komplexität: einfache/atomare (z.B. Modulimplementierung aus Managementsicht), komplexe (z.B. Teilsystementwicklung aus Managementsicht)
  - Formalisierung: vage/chaotisch (ohne Festlegung), strukturiert (detailliert festgelegt)
  - Dynamik: zur Projektlaufzeit (zum Teil) bestimmt, statisch vorab festlegbar
  - ausgeführt durch Mensch/Werkzeug: nichtautomatisiert (durch Menschen), nichtautomatisiert (durch Werkzeuge unterstützt), halbautomatisch (Werkzeug verlangt Entscheidungen), automatisch (nur durch Werkzeug)

- Vorbereitung/Ausführung: Festlegung von Struktur (z.B. Parametrisierung von Netzen), Nutzung der Strukturfestlegung für bestimmten Prozeß (Netzerstellung in bestimmten Anwendungsbereich)
- Weite/Integration: technischer oder administrativer Prozeß, technischer und administrativer Prozeß, Zusammenspiel von Abteilungen, Firmen (übergreifende Prozesse)
- Dauer: Teilprojekt, Gesamtprojekt, Projektfamilie
- Einmaligkeit/Wiederholung: Prototyping, erstmalige Erstellung eines Produkts, mehrmalige Erstellung nach "Schema F", Wiederverwendungsprozeß
- Zusammenhang (Ketten): Schaffung Werkzeug/ Umgebung, Parametrisierung Umgebung, Nutzung Werkzeug für Produkterstellung, Nutzung des Produkts in übergeordnetem Produkt, Zielprozeß, für das System erstellt wurde
- Isoliertheit/Gleichzeitigkeit: ein Prozeß, verschiedene Prozesse gleichzeitig und verzahnt (zur Auslastung einer Firma)
- für alle diese unterschiedlichen Prozesse und die Erstellung ihrer Produkte ist Werkzeugunterstützung denkbar  
Globalziel: integrierte Gesamtumgebung
- Prozeßunterstützung
  - direkt: durch Beobachten von Prozessen, Festhalten und Bewerten, Verbessern, Nutzen von Vorlagen
  - indirekt: durch Sprachen, Methoden, Werkzeuge, die Qualität der Ergebnisse verbessern bzw. Prozeß erleichtern



- Beispiel: komplexe Prozesse auf technischer Ebene



**Fig. 1.17.** Tight integration and different change processes shown for one simple maintenance example

aus /Nag 96/

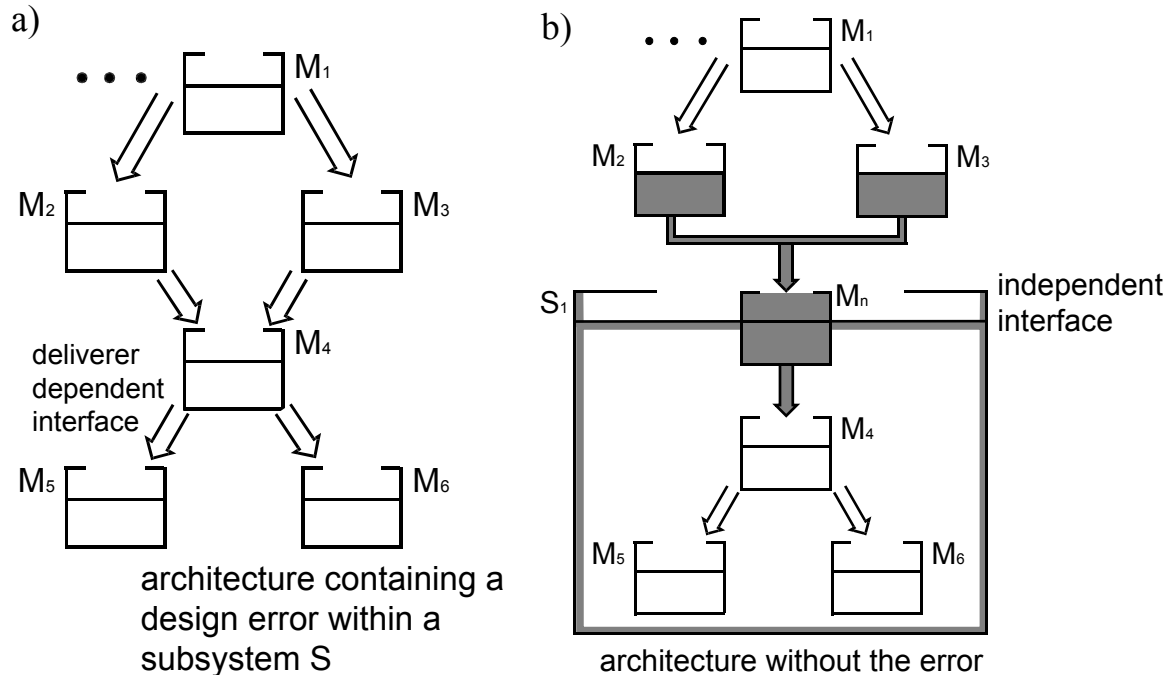
verschiedene Prozesse sind möglich zur Erzielung eines bestimmten Ergebnisses  
 Zusammenspiel unterschiedlicher Entwickler, Umgebungen  
 Koordination nötig

# Dynamik/ statische Bestimmtheit auf Konfigurationen

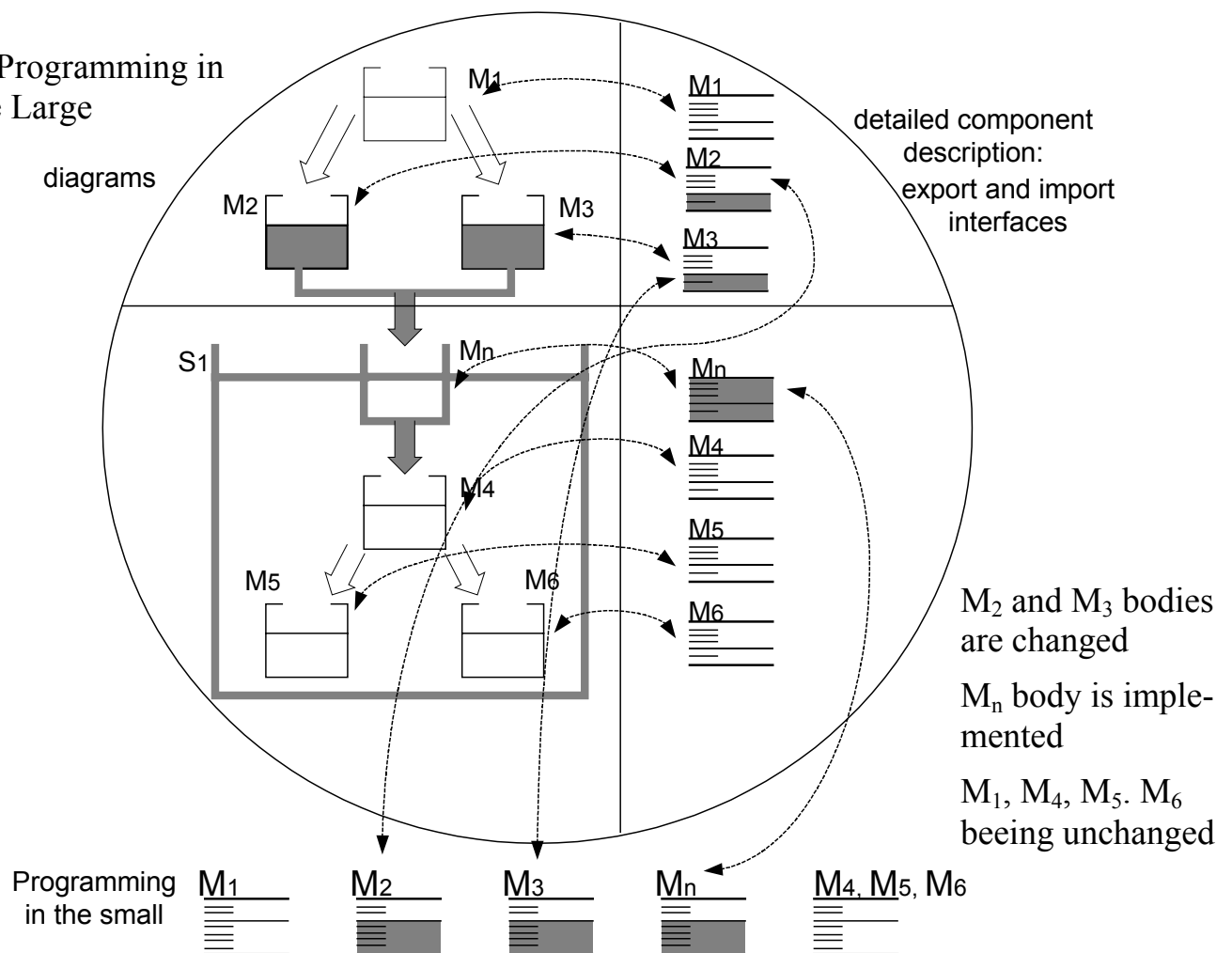
## Dynamikprobleme

- Beschränken uns zur Klärung auf die administrative Ebene und dort auf den Aufgaben/Prozeßteil (dieser ist Produkt, das zur Koordination von Entwicklern herangezogen wird)
- Dynamikprobleme (während eines Projekts)
  - Evolutionsdynamik
  - Rückgriffsdynamik
  - Veränderung aufgrund von Managementdaten  
Prozeß läuft aus dem Ruder, neues Werkzeug ersetzt menschliche Arbeit/ implizites Wissen
  - Zusammensetzung eines Schemas aus Stücken  
z.B. Teilsystementwicklung mit/ohne Prototyping
  - Auswahl zwischen verschiedenen Mustern  
z.B. Implementierung mit Black- und/oder Whitebox-Test
  - Einführung von Mustern auf Objektebene
  - Auswahl/Einführung von Wiederverwendungsmustern
  - ...

- ein einfaches Beispiel für Rückgriffsdynamik



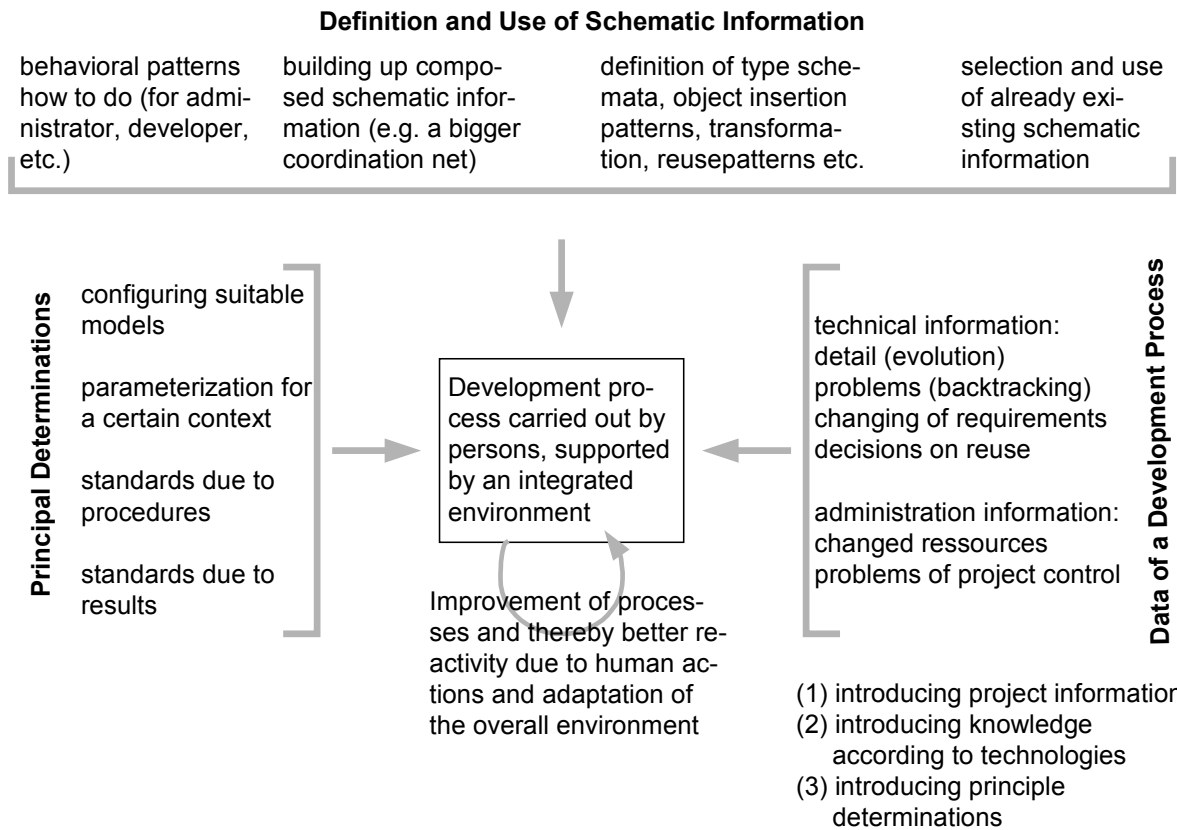
c) Programming in the Large



c) technical configuration after the backtracking process (only interdocument increment-to-increment relations necessary for the process are shown)

aus /Nag 96/

# Zusammenfassung Dynamik



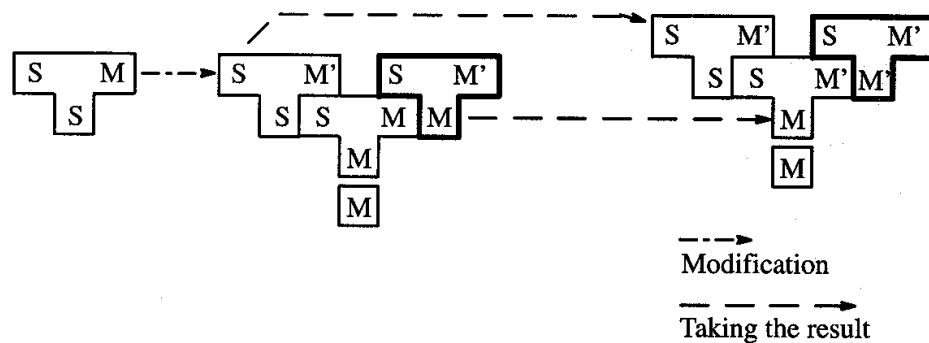
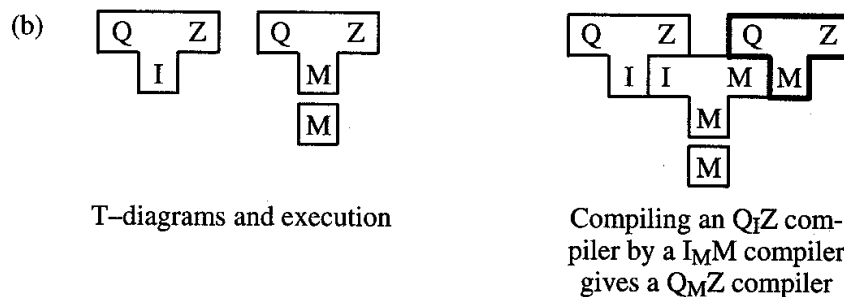
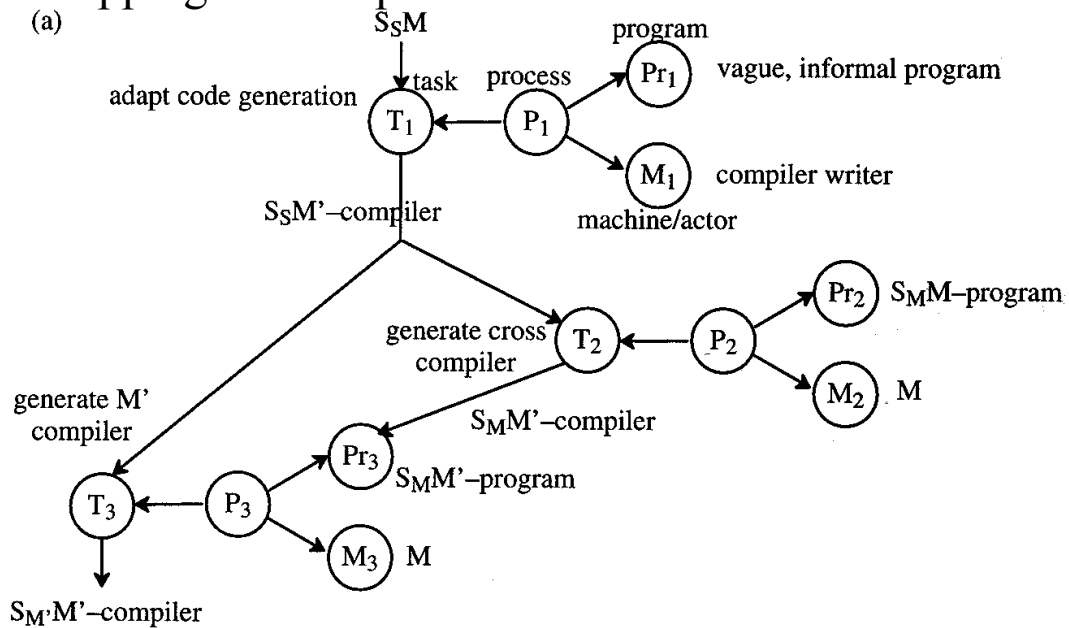
**Fig. 5.1.** Influences on a development process and reactivity of the integrated overall development environment

aus /Nag 96/

- von Projekt zu Projekt
- zur Projektlaufzeit

# statische Konfigurationen und Wissen

## • Bootstrapping im Compilerbau

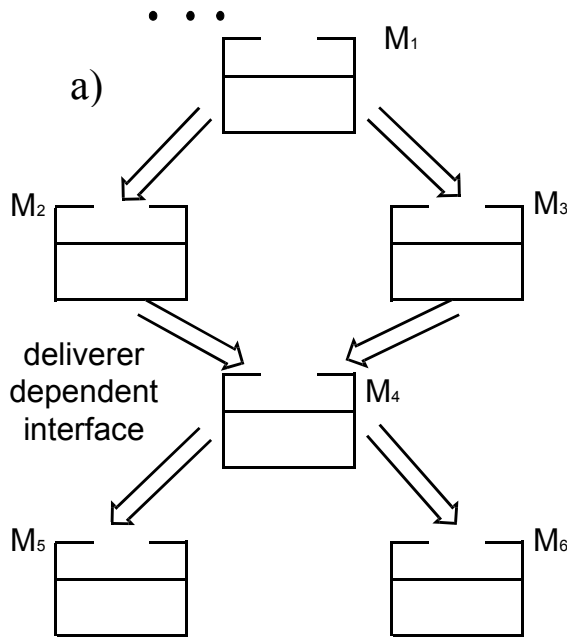


**Fig. 1.7.** A task/process net for the task of porting a compiler using bootstrapping:  
(a) task/process net, (b) explanation by T-diagrams /9. Wir 86/

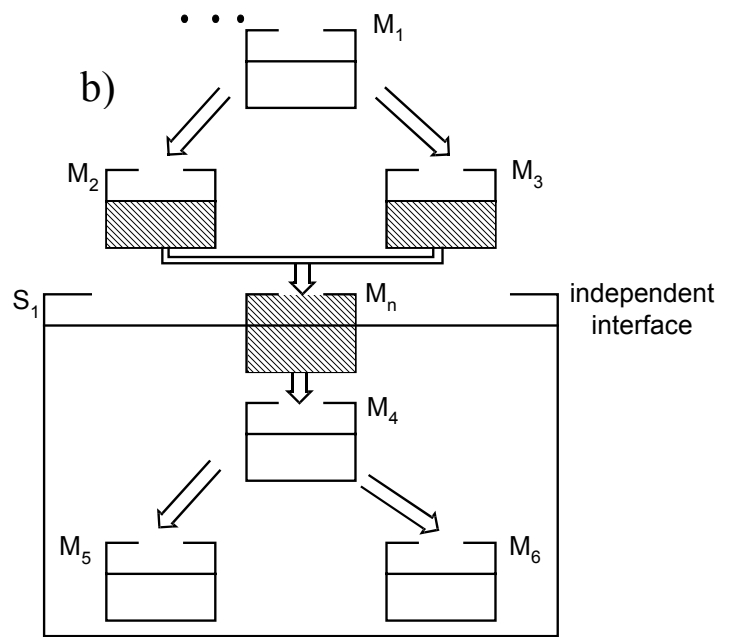
- analog Neuentwicklung Mehrphasen-Compiler in Compiler-Softwarehaus
- vergleiche Plädoyer für Anwendungs-, Strukturklassen-technik

- Entwicklung eines Mehrphasencompilers als Wiederverwendungsprozeß

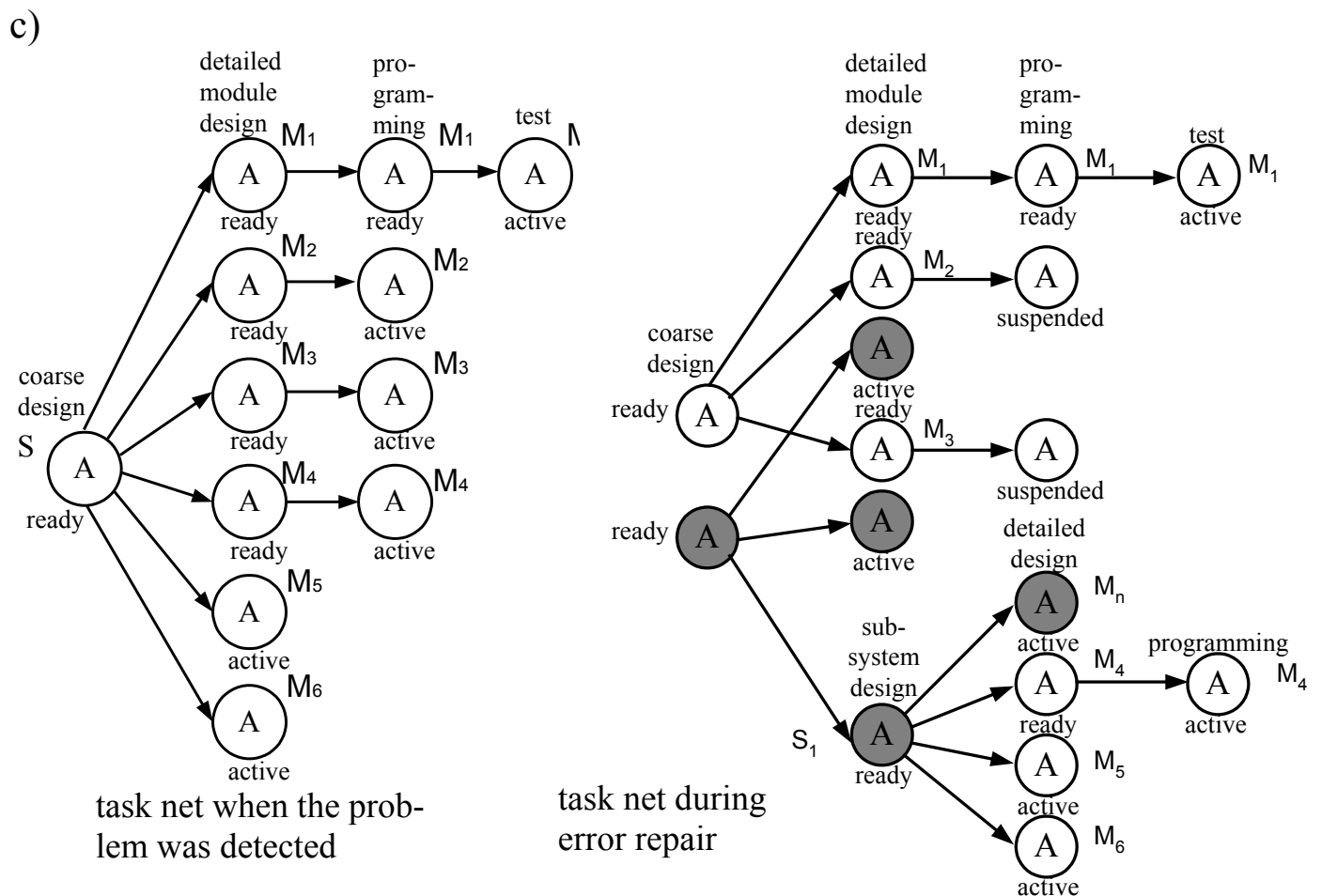
# Einschub Rückgriffe



architecture containing a design error within a subsystem S

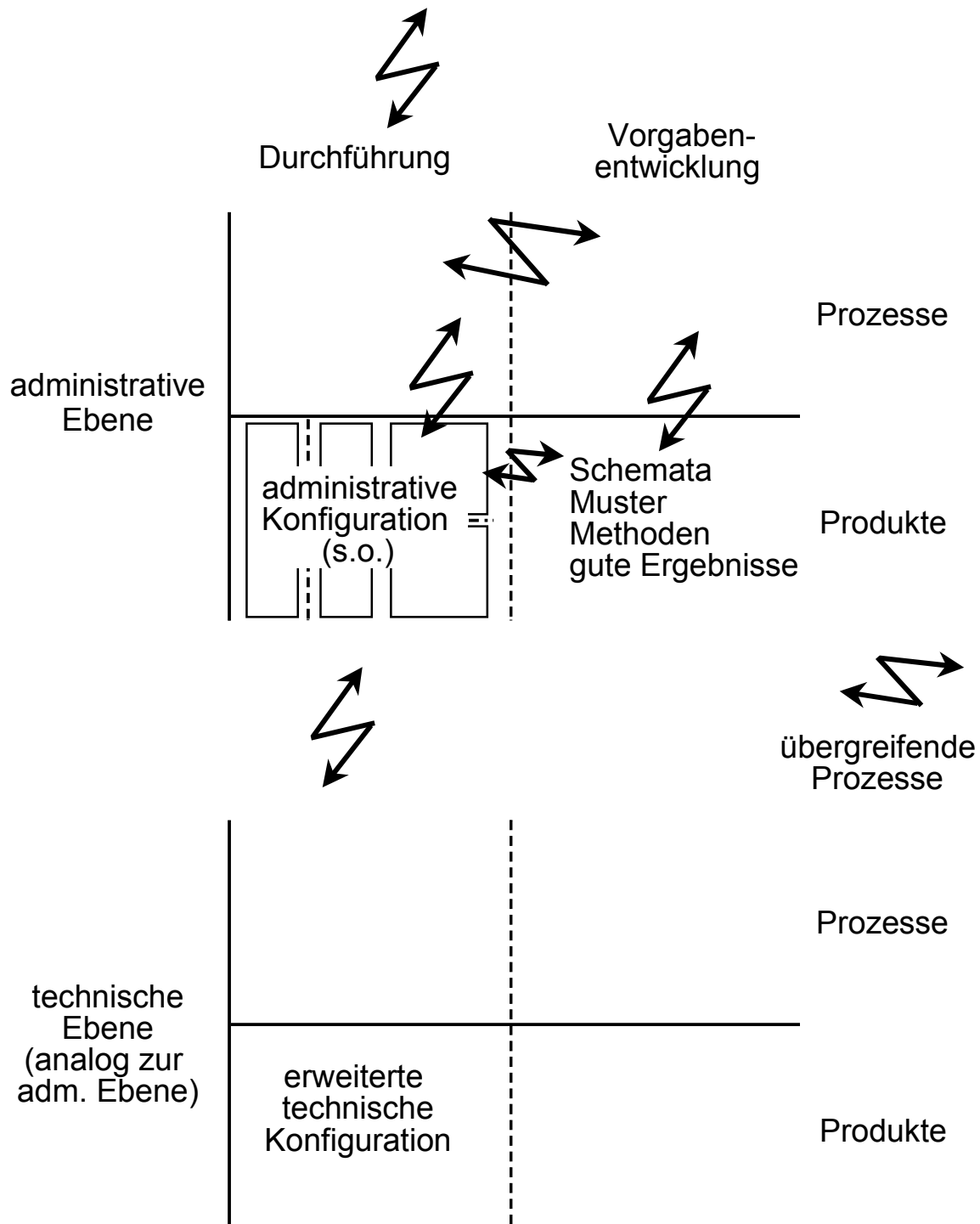


Architecture without the error



# Ein Gesamtmodell für Konfigurationen/ Prozesse (Ausschnitt)

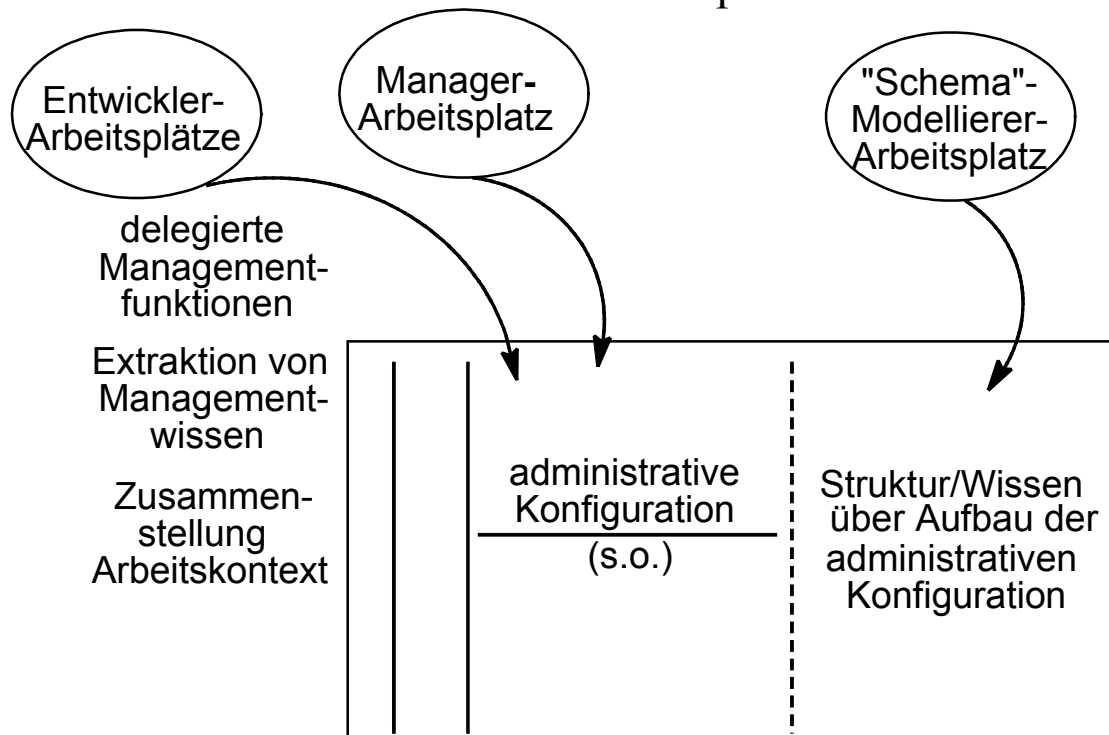
## Gesamtmodell





# Grobgranulare Steuerung durch Administrationskomponente

- Bestandteile der Administrationskomponente



- Administrationskomponente

- Edieren
  - Analysieren
  - Ausführen
  - Instrumentieren/Monitoring
  - +
  - Veränderung des Strukturwissens
  - +
  - Wechselwirkung mit technischer Ebene
  - +
  - Zusammenhang bei übergreifender Kooperation zwischen administrativen Komponenten verschiedener Teilprojekte
- eng verschränkt

- Dynamikproblem tritt auf der gesamten Entwicklungskonfiguration auf!

# Unterschiedliche Planung (Festlegung für Koordination)

## **I. Techn. Ebene (feingranulare Ebene)** **unter Nutzung festgelegter Notationen etc. (s. III)**

- 1) Requirements Engineering: Planung im Sinne von Festlegung der Funktionalität des zukünftigen Systems
- 2) Prototyping/Simulation: Planung im Sinne von Erstellung eines Modells des Systems bzw. der Simulation seines Verhaltens, eventuell unter Einbeziehung des Verhaltens des Zielsystems
- 3) Architekturmodellierung: Planung im Sinne von Strukturierung des Bauplans des zukünftigen Systems (Sonderqualität, s.u.)
- 4) Verwaltung: Planung im Sinne der Handhabung bzw. der Festlegung der Handhabung von Versionen, Varianten, Konfigurationen (erweiterte technische Konfiguration), Konsistenzen innerhalb einer Konfiguration
- 5) Wiederverwendung: Planung im Sinne der Nutzung Komponenten, Rahmenwerke, prozeßunterstützende Werkzeuge (Generatoren)
- 6) Festlegung Qualitätssicherung: Qualitätssicherungsplan oder Testplan: Testdatenfestlegung, Testbeendigung etc.

## **II. Adm. Ebene (grobgranulare Ebene)**

**unter Nutzung festgelegter Notationen etc. (s. III)**

- 1) Schätzung: Planung im Sinne der Schätzung des Umfangs (z.B. in LOC), des Personalaufwands (in PM), der Kosten (in DM), der Zeiten (Anfangs-, Enddatum, Zeitdauer)
- 2) Projektplanung und -management projektspezifisch: Planung im Sinne von Aufgaben des Prozesses und ihrer Abhängigkeit festlegen, der Komponenten des Produkts und ihrer Abhängigkeitsbeziehung, Festlegung von Verantwortlichkeiten, hierfür nötige Hilfsmittel, Zeiten voraussichtlicher Fertigstellung, zugehörige Aufwände
- 3) Projektmanagement projektübergreifend (versch. Projekte): Planung im Sinne der gleichmäßigen Auslastung von Mitarbeitern, Werkzeugen, Hilfsmitteln
- 4) Planung teilprojektübergreifend (in einem Projekt): von zusammengesetzten oder firmenübergreifenden Projekten, Projektfamilien
- 5) Projektbegleitende Organisation (Projektadministration): Planung im Sinne der Festlegung von Projektbesprechungen, Schulungen für Mitarbeiter, Urlaubsplanung, Kontenbelastung etc.

### **III. Vorabfestlegung von**

**Konzepten, Sprachen, Modellen, Methoden, Richtlinien etc.  
(auf adm. Ebene wie auf techn. Ebene)**

- 1) grobe Vorgehensweise: Planung im Sinne von Lebenszyklusfestlegung, Gesamtproduktfestlegung (Gröbstgranulare Ebene)
- 2) einzelne Teilmodelle: Planung im Sinne der Festlegung von Prozeß- und Produktmodellen für Arbeitsbereiche (grobgranulare Ebene) und ihre Integration
- 3) Prozesse, Produkte: was überhaupt (z.B. “so wird RE durchgeführt”, diese Dokumente entstehen, hängen so und so zusammen etc.)
- 4) Muster: Planung im Sinne der Festlegung von Typmustern (Teilsystemimplementierung), Objektmustern (Modulimplementierung), Objektmustern (für gut befundener Aufgabenplan)
- 5) Festlegung technischer Modelle (Sprachen) und ihre Integration, z.B. SA, EER, CTR für RE, OOD für PiL, C++ für PiS, Review Anforderungsspezifikation, Architektur, Whitebox-Test PiS für QS
- 6) entspr. Vorgehensweisen für techn. Modelle und ihre Integration (Prozeßmodelle)
- 7) Festlegung von Richtlinien, Normen, Standards für alle obigen Punkte

# Werkzeuge der Softwareentwicklung

## Klassische Werkzeuge

- Niveau der Unterstützung:  
klassische Progr.-sprachen } wenig  
interpreterorientierte Progr.-spr. } etwas mehr > Unterstützung,  
uns interessiert Unterstützung bei Erstellung/Veränderung  
großer Programmsysteme
- Sprachimplementation einer höheren Programmiersprache:  
Editor, Compiler, Laufzeitpaket, Binder, Lader, Sprach-  
standard
- Programmiersystem:  
Sprachimplementation plus Trace, Dump,  
gegebenenfalls mehrere Sprachen,  
verschiedene Compilervarianten
- Seit 1980 Bedeutung von Werkzeugunterstützung erkannt  
Qualitäts-, insbesondere } Produktivitäts-  
Zuverlässigkeitserhöhung } steigerung  
Entlastung von Details }  
ausgehend von Programmieren im Kleinen zu Require-  
ments Engineering, Programmieren im Großen, Pro-  
jektorganisation, Qualitätssicherung, Dokumentation  
Softwareentwicklungsumgebungen  
Softwareproduktionsumgebungen

# Softwareentwicklungsumgebungen

- Werkzeuge in SW-Entwicklungsumgebungen (vgl. IPSEN):
  - strukturbezogen/syntaxorientiert (kontextfrei und kontextsensitiv)
  - integriert
  - inkrementell
  - interaktiv/sofort
- Werkzeuge für das PiG (analog PiK, ...):
  - Browser (Werkzeug zum "Schmökern") für das Lesen der Anforderungsdefinition, z.B. um die für den Entwurf eines Teilsystems relevanten Teile anzuzeigen oder ein Werkzeug, das die Information aus der Anforderungsdefinition für einen Teil der Entwurfsaufgabe gezielt zusammenstellt
  - strukturbezogener Editor zur Eingabe und zur Veränderung von Softwarearchitekturen, wobei die Softwarearchitekturen einer bestimmten Syntax genügen müssen
  - strukturbez. Editor, der darüber hinaus eine bestimmte "Entwurfsmethode" unterstützt und damit eine bestimmte Vorgehensweise oder das Einhalten bestimmter Strukturierungsprinzipien etc.
  - Analysen einer Softwarearchitektur auf Vollständigkeit, Konsistenz, Minimalität, bezogen auf die Syntax und Semantik von Architekturdokumenten
  - Analysen, die des weiteren noch die Strukturierungsprinzipien einer Methode berücksichtigen
  - Suche/Anzeige wiederverwendbarer vordef. Module und Teilsysteme
  - Werkzeug zur Ersetzung einer Teilarchitektur durch eine andere mit gleichem "Außenverhalten"
  - Werkzeug zur Überprüfung der Konsistenz einer Architektur mit der Anforderungsdefinition oder Ableitung der Architektur aus der Anforderungsdefinition
  - Erzeugung von Modulrahmen für eine bestimmte Programmiersprache (Codieren im Großen)
  - Werkzeug zur getrennten Übersetzung
  - Werkzeug zur Qualitätssicherung, z.B. Testwerkzeug für sog. Black-box-Test-Methoden, auf Architektursprache oder -methode abgestimmt

- Werkzeug zur Ausführung eines noch unvollständigen Programmsystems (einige Module sind fertig, andere teilweise ausprogrammiert, andere noch nicht angefangen)
- Werkzeug zur Unterstützung der Integration, z.B. für die Reihenfolgebestimmung zu integrierender Module auf Architektursprache/-methoden abgestimmt
- Werkzeug zur Messung eines Programmsystems oder teilweise fertiggestellten Programmsystems, z.B. für Laufzeit oder Speicherplatz, architekturbezogen
- Werkzeug zur Versionskontrolle (Bausteine einer Architektur existieren in verschiedenen Zuständen (Revisionen); von Teilarchitekturen gibt es unterschiedliche Realisierungen, diese nennt man Varianten), architekturbezogen
- Werkzeug zur Konfigurationskontrolle (Zusammenbau einer Gesamtarchitektur aus bestimmten Varianten und Revisionen von Modulen und Teilsystemen), architekturbezogen
- Werkzeugklassen
  - Editoren
  - Analysatoren
  - Transformatoren
  - Instrumentatoren
  - Ausführer, Monitoren
  - Werkzeuge für Erstellung von Repräsentationen
  - !- Verzahnungswerkzeuge (z.B. für PiG)
  - Transformatoren, Konsistenzprüfer, Browser
- gegenw. Probleme von Softwareentwicklungsumgebungen:
  - Weiterentwicklung von Methoden und Notationen
  - Konsistenz zwischen verschiedenen Dokumenten
  - Inkrementalität dokumentintern und -übergreifend
  - Austauschen und Kombinieren verschiedener Methoden
  - Wiederverwendung
  - Rapid Prototyping
  - Standardisierung von Werkzeugen
  - Unterstützung Projektadministration und Verzahnung

- Werkzeuge für das Requirements Engineering (Istanalyse, Sollkonzept)
  - Handhabung von Prüflisten: allgemein für bestimmte Anwendungsbereiche
  - strukturbezogener Editor zur Erstellung der Anforderungsdefinition (insbesondere verschiedene Sichten, Integration der Sichten)
  - strukturbezogener Editor folgt einer bestimmten Anforderungsspezifikationsmethode
  - Analysen zur Ermittlung der Konsistenz einer Anforderungsdefinition
  - Werkzeuge zum Erstellen eines schnellen Prototyps/zur Simulation der Anforderungsdefinition
  - Durchführbarkeitsstudie <sup>3</sup> Projektorganisation
  
- Werkzeuge zur Qualitätssicherung
  - Erzeugung von Testtreibern, Teststummeln
  - Unterstützung von Teststrategien: Top-Down, Bottom-Up (allgemein inkrementelles Testen)
  - Blackbox-Test: Erzeugung der Testdaten
  - Whitebox-Test: Prüfung von Überdeckungen
  - Unterstützung der Testdurchführung: Testdatenarchivierung, Testdatenvergleich
  - Testplanung/Testdurchführung: Aufwand, Personal, Zeit → Projektorganisation
  - Reviews: Checklisten, allgemein, anwendungsbezogen, software-dokumentbezogen
  - Ablaufunterstützung für menschliche Begutachtung: Review, Walkthrough, Peer Rating etc.
  - Leistungs-, Last-, Installationstest
  - Verifikationshilfsmittel



- Werkzeuge zur Projektorganisation

- Projektplanung:

- Kostenschätzung, Personalschätzung, Zeitschätzung  
unter Heranziehung von Erfahrungen ähnlicher Projekte  
unter Benutzung von Schätzformeln mit Attributen  
unter Berücksichtigung der internen Struktur  
allgemeine Hilfsmittel (z.B. Netzpläne)

- betriebswirtschaftliche Projektdurchführung:

- betriebswirtschaftliche Seite eines Softwareprojekts  
Personalführung/Personalförderung  
Schaffung der Arbeitsumgebung (Hard-/Software, Personal)

- Projektüberwachung:

- Feststellung einer Verzögerung kritischer Teile  
Veränderung eines Netzplans  
Abgleich Projektfortschritt mit Projektplan

- Projektmanagement:

- Festlegung der Struktur eines Softwareprojektes: Phasenmodell,  
Arbeitsbereiche, Qualitätssicherung, Projektorganisation, Doku-  
mentation  
Verantwortlichkeitskontrolle, Zugriffskontrolle  
Nachrichtenkontrolle, Verteilungskontrolle  
Freigabekontrolle

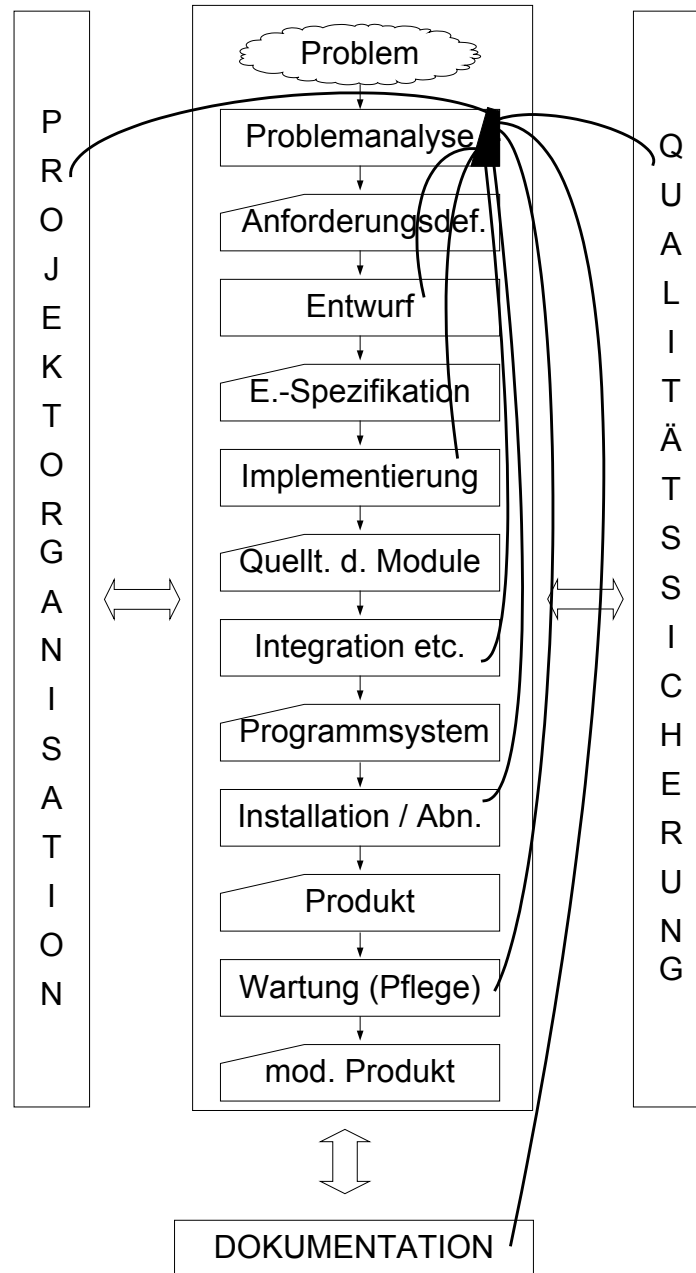
- Werkzeuge zur Dokumentation

- (strukturbezogener) Editor für Erstellung von Benutzerdokumen-  
tation

- (strukturbezogener) Editor für Erstellung der technischen Doku-  
mentation

- Checklisten für Benutzer-/technische Dokumentation

- integrierte Gesamtumgebung für Entwicklungsprozeß



- für Entwicklung, Wartung, Wiederverwendung
- diskretes Paradigma und enge Integration, inkrementelle Werkzeuge:  
  - geben viel Unterstützung
  - erlauben verschiedenartige Prozesse

# Übersicht über die weitere Vorlesung

## Weitere Kapitel

- Problemanalyse und Anforderungsspezifikation
- Entwurf und Spezifikation
- (Programmieren im Kleinen)
- Projektorganisation
- Dokumentation
- Qualitätssicherung, insbesondere Test

## Weiterführende Veranstaltungen

- Softwaretechnik-Projektpraktikum
- Einführung in die Softwaretechnik-Programmiersprache Ada
- Softwareentwicklungsumgebungen
- Systematisches Programmieren im Großen (Architekturmodellierung)
- Operationelle Spezifikation (Graphersetzungssysteme)
- [ • Requirements Engineering, LS Inf. V ]
- Projektorganisation
- Visuelle Programmierung

## Aufgaben zu Kap. 3:

1. In der Vorlesung wurden unterschiedliche Facetten des Begriffs “Planung” diskutiert. Geben Sie zu jeder der dort auftretenden Charakterisierung ein Beispiel (kursive Skizze genügt).
2. Fassen Sie zusammen: Was heißt SW-Entwicklung/-wartung? Welche Beziehungen gibt es zwischen den Teilen des Produkts? Was können wir über den Prozeß aussagen?
3. Bei der Durchführung eines Wartungsprozesses für ein SW-System sind verschiedenartigste Prozesse möglich: (a) Wir haben keine klare Vorstellung von der Struktur des Systems: (a1) Konsequenzen von Änderungen nicht klar (Was ist zu tun?), (a2) Änderungsprozesse sind danach auf verschiedenste Weise durchführbar (Wie wird die Änderung durchgeführt, batchartig, traceartig?). (b) Wir haben eine klare Struktur des Systems vor Augen. (b1) Änderungen sind größtenteils vorab planbar (b2) trotzdem sind vielerlei Änderungsprozesse möglich. (c) Es gibt Strukturwissen für die Anwendung (vgl. folgende Aufgaben)
4. Tragen Sie den Entwicklungsprozeß A eines üblichen Mehrphasencompilers als Prozeßnetz auf. Dabei sollen die Bausteine für die entsprechenden Datenstrukturen (Texteingabestrom, Tokenliste etc.) ebenfalls entwickelt werden. Die Compilerphasen sind handcodiert. Beim nächsten Compilerprojekt B werden die zugrundeliegenden Datenstrukturen als Ergebnisse verwendet. Wie sieht der Prozeß mit handcodierter Entwicklung aus? Wie sieht die Änderung des Aufgabennetzes aus beim Übergang von A zu B?
5. Im nächsten Schritt werden die Hilfsmittel für den Prozeß der Mehrphasencompilerentwicklung bestellt. Nehmen wir

an, die entsprechenden Spezifikations-Umgebungen (für lexikalische Analyse, kontextfreie Syntaxanalyse etc.) und Generatoren (Scannergenerator, Parsergenerator usw.) sind jetzt vorhanden. Wie sieht jetzt der Prozeß C zur Entwicklung eines Mehrphasencompilers aus? Was hat sich über B geändert?

# Literatur zu Kap. 3:

- /Bab 86/ W.A. Babich: Software Configuration Management, Addison Wesley, 1986
- /BHS 80/ E. H. Bersoff, V. D. Henderson, S. G. Siegel: Software Configuration Management - An Investment in Product Integrity, Prentice-Hall, 1980
- /BKM 84/ R. Budde, K. Kuhlenkamp, L. Mathiassen/H. Züllighoven: Approches to Prototyping, Springer, 1984
- /Boe 84/ B. Boehm: Verifying and Validating Software Requirements and Design Specifications, IEEE Software 1, 1, 1984
- /BP 92/ W. Bischofberger, G. Pomberger: Prototyping-oriented Software Development - Concepts and Tools, Springer, 1992
- /CSM 89/ Conference on Software Maintenance 1989, Proceedings, IEEE Comp. Soc. Press, 1989
- /Dow 86/ M. Dowson (Ed.): Proc. 3rd Int. Software Process Workshop, IEEE Comp. Soc. Press, 1986
- /Fei 91/ P. Feiler (Ed.): Proc. 3rd Int. Workshop on Software Configuration Management (SCM3), ACM Press, 1991
- /FKN 94/ A. Finkelstein, J. Kramer, B.A. Nuseibeh (Eds.): Software Process Modelling and Technology, Taunton: Research Studies Press, 1994
- /Fre 87/ P. Freeman (Ed.): Software Reusability, Washington: IEEE Comp. Soc. Press, 1987
- /FW 96/ A. Fuggetta, A. Wolf (Eds.): Software Process, John Wiley & Sons, 1996
- /Nag 96/ M. Nagl (Ed.): Building Tightly-Integrated Software Development Environments: The IPSEN Approach, LNCS 1170, Springer, 1996, insb. 1.1, 5.1 und 6
- /Per 90/ D.E. Perry (Ed.): Proc. 5th Int. Software Process Workshop, Los Alamitos: IEEE Comp. Soc. Press, 1990
- /RW 86/ C. Rich, R. C. Waters: Artificial Intelligence and Software Engineering, Morgan Kaufman, 1986
- /Tha 88/ R. Thayer: Software Engineering Project Management, IEEE Comp. Soc. Press, 1988
- /Tra 88/ W. Tracz (Ed.): Software Reuse - Emerging Technology, IEEE Comp. Soc. Press, 1988
- /Tul 89/ C. Tully (Ed.): Proc. 4th Int. Software Process Workshop, ACM Software Engineering Notes 14, 4, 1989

- /WD 85/ J. C. Wileden, M. Dowson (Eds.): Proc. 2nd Int. Software Process Workshop, IEEE Comp. Soc. Press, 1985
- /Win 88/ J. F. H. Winkler (Ed.): Proc. 1st Int. Workshop on Software Version and Configuration Control (SCM1), Bericht 30 German Chapter ACM, Teubner-Verlag, 1988
- /WT 89/ J. F. H. Winkler/W. Tichy (Eds.): Proc. 2nd Int. Workshop on Software Configuration Management (SCM2), ACM Software Engineering Notes 17, 7, 1989
- /Zel 79/ M. Zelkowitz et al.: Principles of Software Engineering and Design, Prentice Hall, 1979