Zielsetzung

- - Konzepte für Softwarearchitekturen / Architekturbeschreibungssprache

- - Grundlage für Nachdenken über Struktur von Softwaresystemen

- - Voraussetzung für
  Wartungsüberlegungen
  Aussagen über Qualitätseigenschaften
  Wiederverwendbarkeit

- - Programmiersprachenunabhängigkeit

Gliederung

1. Der Kontext: Softwaretechnik- -Grundlagen

2. Das Problem: Modellieren auf Entwurfsebene

3. Ein erstes Beispiel: Ohne Vorüberlegung

4. Die Notation: Ein einfaches Modulkonzept

5. Zur Vertiefung: Teilarchitekturüberlegungen und Modulkonzept- -Erweiterungen

6. Vorbereitung für den Einsatz: Übertragung in Programmiersprachen

7. Einübung durch Beispiele: Einige Softwarearchitekturen

8. Allgemeine Hinweise: Strategien zur Adaptabilität und Wiederverwendbarkeit

9. Noch nicht gelöst: Offene Probleme und Weiterführung

-- Größenordung des Problems der Softwareerstellung, insb. Wartungsproblem

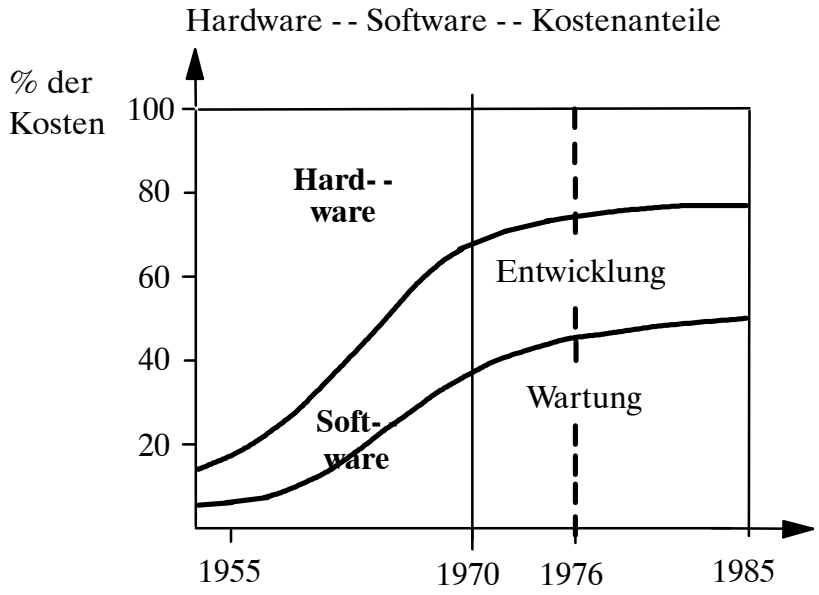


Fig. 1.1: Softwarekosten und Wartungsproblem

-- Ist Graphik noch richtig ?
Ja !

-- Antwort Softwaretechnik (1968/69 Konf. der Nato in Garmisch, Rom):
ingenieurmäßiger Ansatz
Gegensatz zur Kunst

-- Definitionen
F.L. Bauer, 73:
"... ökonomisch Software zu erhalten, die effizient und zuverlässig auf realen Maschinen läuft."
Hesse et al.:
"Softwaretechnik ist Fachgebiet der Informatik, das sich mit der Bereitstellung und system. Verwendung von Methoden und Werkzeugen für die Herstellung und Anwendung von Software beschäftigt."

-- Thesen zur Softwaretechnik

- Herstellung großer Software unterscheidet sich qualitativ von der kleinen Software.
- Probleme erst dann, wenn
  viele Personen f. Entwickl./Weiterentwickl.
  mehr als eine Version des Programmsystems
- Geregelte Zusammenarbeit in Programmierer--mannschaft
  Planung/Überwachung/Verteilung d. Arbeit
  Lösung der Arbeitsschritte und
  ihres Zusammenspiels
  Sicherung der Qualität
  Dokumentation der Ergebnisse
  Information über Schritte und Ergebnisse.

- Auf allen Ebenen: Bewältigung der Komplexität: Modellierungsproblematik.

- - Definition Lebenszyklus
  Strukturierung des Zeitraums
  ”Zyklus” wegen: Rückgriffen
  neue Software ersetzt abgestorbene

- - Lebenszyklusmodell
  spez. Form: Phasenmodelle
  Einzelaktivitäten: Phasen
  Ergebnisse: Softwaredokumente

  Grund für Aufteilung:
  kleinere Komplexität
  Überprüfbarkeit

- - Definition Softwaredokument

1.2 Teilaktivitäten und Ergebnisse einzelner Phasen

-- Problemanalyse (problem analysis, system analysis):
Problem und seine Umgebungsbedingungen vollständig, eindeutig und präzise beschreiben, insb.

- Bedienerprofil (Art, Anzahl, Eigensch. der Bediener)
- Systemumgebung (Hardware, vorgef. Software)
- Funktionalität
- weitere Parameter, z.B. Effizienz, Schutz, Sicherheit
- Bedieneroberfläche

-- Ergebnis Anforderungsdefinition (requirements def., -- specif.):

- Allgemeines: Zweck, Bedienerprofil, Systemumgeb.
- gewünschte Funktionen
- korrekte/falsche Eingaben und zugehörige Ausg./Systemreakt.
- Bedieneroberflächengestaltung
- Festlegung Effizienzparameter
- Festlegung Schutz--, Sicherheitsaspekte
- Dokumentationsanforderungen
- Qualitätssicherungsanforderungen

evtl. Be-- diener-- hand-- buch

-- Durchführbarkeitsstudie (zu Problemanalyse gerechnet)

technische (überhaupt, unter geg.Bed.)
personelle
ökonomische
Zeitplan, Personalaufwand, Kosten/Nutzen, Risiken

Durch-- führ-- barkeit

- - Unternehmerische Entscheidung über die Durchführung oder Revision der Aufgabenstellung:
  Vertrag

- - Bereits bei Erstellung der Anforderungsdefinition
  Diskussion über zukünftige Erweiterungen
  z.B. durch Brainstorming

-- Entwurf (Design, Architekturmodellierung):
Modell des Innenlebens des Systems auf grober Ebene:
Zerlegung in Module (Exporte),
Festlegung der Beziehungen (Importe),
unter statischen Gesichtspunkten (was, nicht wie)
verträgl. mit Anforderungsdef. und implementierbar

-- Ergebnis ist Entwurfsspezifikation (Spezifikation, Softwarearchitektur):
ist statische Festlegung
in einer Sprache fixiert
formal, semiformal, informell
Grundlage für Überprüf. gegen Anforderungsdef.
Grundlage der Überprüf. der Implementierung
einzelner Module

-- Handicap: nimmt breiten Raum ein, erst spät Programmcodezeilen

- - Implementierung
  Bausteine ausprogrammieren
  in der Regel von verschiedenen Personen
  ausgehend von entspr. Teil der Entwurfsspezifikation
  (Export und Import)
  heißt Daten- - und Ablaufstrukturen festlegen
  bei niedriger Programmierspr. (FORTRAN, Ass.):
  abstrakte Implementation in Pseudocode
  Modultest

- - Ergebnis sind ausgetestete, einzelne Modulimplemen- -
  tationen
  leicht verständlich, überprüfbar, änderbar, nicht
  optimiert
  Zusammenspiel der einzelnen Module noch ungeklärt

- - Integration/Funktions- -/Leistungsüberprüfung:
  Idealfall:
    Entwurfsspez. geg. Anforderungsdef. konsistent
    Korrektheit jedes Moduls bewiesen (verifiziert)
    Integration, Funktionsüberprüf. größtent. fertig

  Praxis:
    Integration von Modulen zu größeren Einheiten
    diese wieder zu größeren Einh. bis Gesamtsystem
    Überprüfung durch Test
    entdeckt dabei Implementierungs- -, Entwurfs- - und
      Problemanalysefehler
    Leistungsmessung nach funkt. ”Korrektheit”
    ggf. Optimierungen, um Leistungsparameter der
      Anforderungsdef. zu erfüllen.

- - Ergebnis:
  lauffähiges, überprüftes, dokumentiertes Software- -
    system

- Installation und Abnahme:
  Übertragung eines Softwaresystems in seine reale Umgebung (evtl. andere Hardware)
  Abnahme durch Auftraggeber

- Ergebnis
  abgenommenes Softwaresystem

- Wartung (Pflege):
  Bedeutung (vgl. Graphik Böhm 60 %)
  Begründung: ungenaue Anforderungen, keine saubere Softwarearchitektur, zu wenig über Änderungen nachgedacht, Wartung durch Erker

- Ergebnis:
  verändertes Softwaresystem
  nach einigen Änderungen keine klare Struktur mehr (Spaghettiteller)

1.3 Diskussion von Lebenszyklusmodellen

- - Rückgriffe
  auf vorangehende Phasen, dabei Modifikation der entspr. Softwaredokumente
  unvermeidbar: durch sorgfältige Modellierung Anzahl und Weite verminderbar

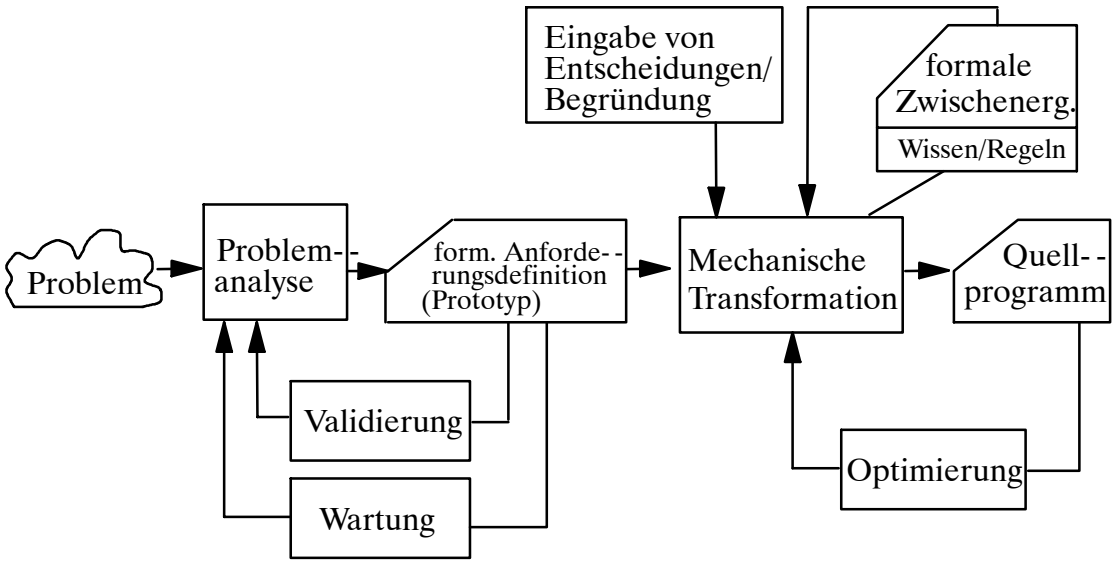
- - Beispiele für Rückgriffe
  eine Phase zurück
  mehrere Phasen zurück

- - bisheriges Phasenmodell vereinfacht:
  Phasen keine monolithischen Blöcke
  Rückgriffe
  Vorgriffe
  Wartung ist keine Phase
  praxisnahes Modell zu kompliziert als Gesprächs- - grundlage

- - Phasenmodellvarianten:

Auftrennen oder Zusammenlegen von Phasen
eingebettete Systeme: erweitertes Phasenmodell
für Gesamtsystem
Rapid Prototyping (später i.a. weggeworfen)
partizipative Modelle:
Bediener arbeiten bei Anfoderungsdef. mit
rapid prototype führt zu Anregungen/Einspr.
durch Bediener
Vorbereitung des org. Systems für späteren
Einsatz des Softwaresystems
Benutzung der Software erscheint als Phase
Berücksichtigung der Feinstruktur von Phasen (s.u.)

- - alternative Lebenszyklusmodelle kontinuierlich:
bisher diskreter (ingenieurmäßiger) Ansatz



Ansatz: (Programmer's Apprentice):
Aufheben des Wissens des Entwicklungsproz.
ausführbare Anforderungsdef.

Realisierung:
automat. Übersetzung
interaktive Eingabe von Entwurfsentscheidungen
System für Transformationen aus
anfangs einzugeben, später automatisch, da
System "lernt"

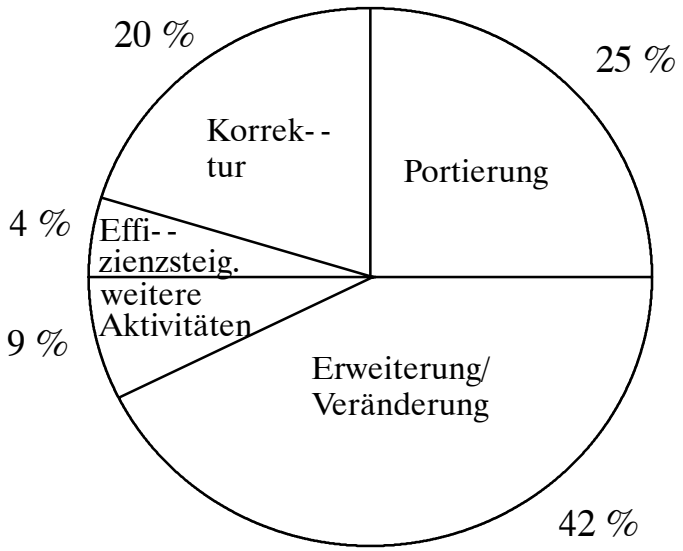
analog Optimierung

1.4 Zum Problem der Wartung

-- Begriff ”Wartung”: Software verschleißt nicht
Software ist langlebig und ”beliebig” änderbar

-- Veränderungen in der Wartung:
Erweiterungen des Systems (Bediener, Wartungspers., Oper.)
Veränderungen der Systemfunktionalität
abgemagerte Systeme
andere Bedieneroberfläche
Übertragung auf andere Basismaschinen
(Prozessor, Compiler, Betriebssyst., Dateiverw., DB--System)
Fehlerbeseitigung
Effizienzsteigerung

-- Aufteilung in Problemklassen der Wartung



- spezifische Probleme der Wartung:
  Aufbau eines Programmsytems, schwer zu verstehen,
  modif., überprüfen: Softwarearch./Entwurfsentsch.
  nicht festgehalten
  Wartungsgeschichte implizit in besteh. System
  muß Firmen- -, Abteilungskontext, Historie, Denkweise
  kennen
  Inkonsistenzen zwischen einz. Softwaredokumenten
  techn. Dokumentation unvollständig/inkonsistent

- Begründung der Probleme:
  Vielzahl techn. Fehler
  org. Fehler: Entwickler erhalten zu wenig Zeit und Ress.
  Wartung unbeliebt

- Lösung durch präventive Wartung:
  eher Ziel
  Forderungen:
  zuk. Wartungsüberlegungen gleich mitberücksicht.
  bei Wartung zuk. Wartungsüberlegungen mitberücks.
  System stets auf neuestem Stand halten (Funkt.,
  Bedieneroberfläche, Softwarearchitektur etc.)

- wichtiger Schritt: Nachdenken über Systemerweiterung
  bei Erstellung der Anforderungsdefinition
  bei Erstellung der Softwarearch.

1.5 Zusammenfassung der Aktivitäten in Arbeitsbereiche

-- Phasenmodelle
zeitl. Verlauf des Entwicklungs- -/Wartungsprozesses
Abhängigkeit bezügl. ”Ist- -Ergebnis von”, ”wird benötigt für”

-- Einteilung in Arbeitsbereiche
wo wird auf gleichem Niveau modelliert
Zusammenfassung von Tätigkeiten, die zeitl. verstreut liegen
Voraussetzung für Klärung von Zusammenhängen in und zwischen Arbeitsbereichen

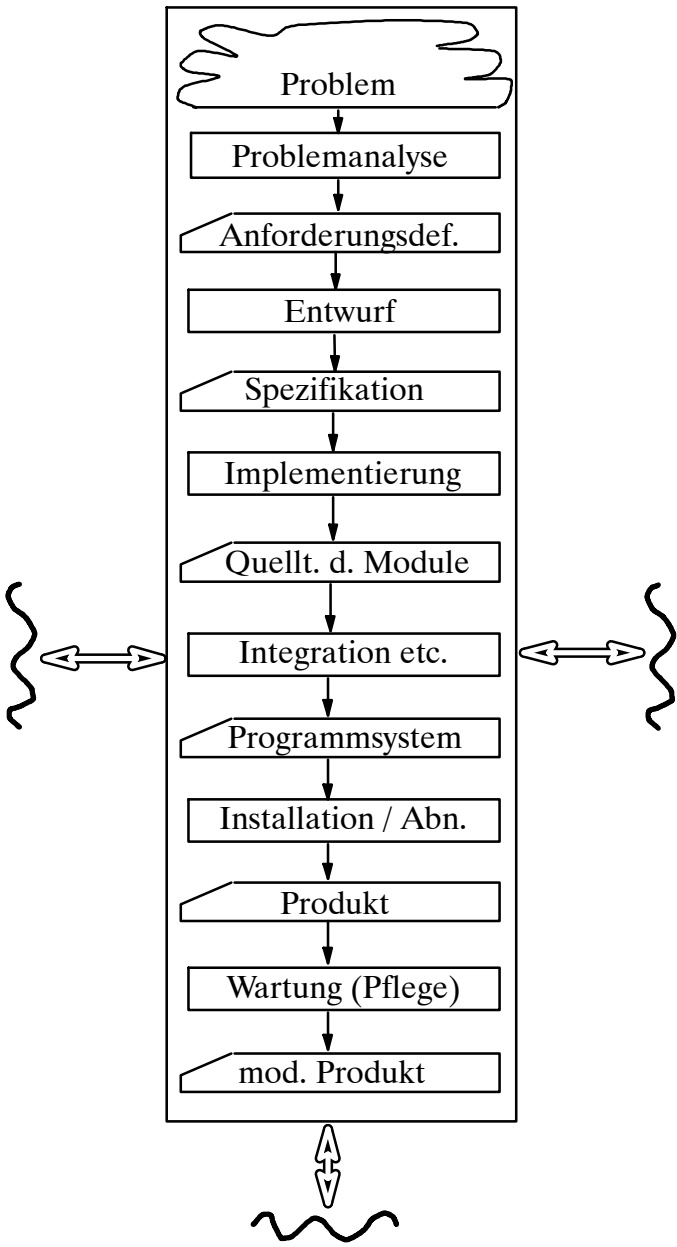
-- drei Arbeitsbereiche, zunächst grob:

- Definieren/Veränderungen der Anforderungen: Außenverh. des Systems
  Angewandte Hilfsmittel: Anforderungstechnik (Requirements Engineering)

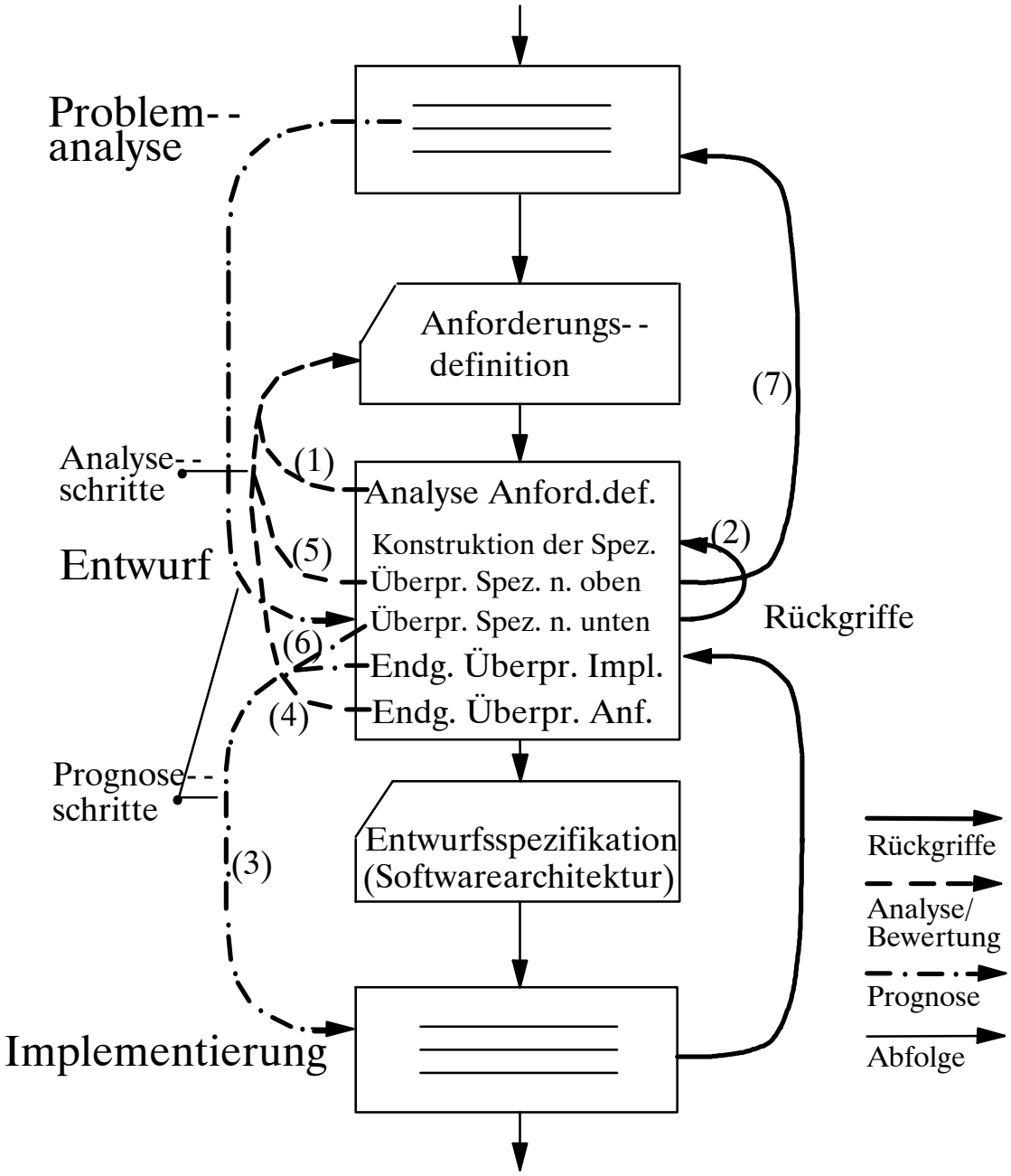
- Programmieren im Großen oder Entwurf: Festlegung/Veränderungen der Struktur des Syst. auf grober Ebene gem. Anforderungsdef.
  Angewandte Hilfsmittel: Entwurfstechnik

- Programmieren im Kleinen: Realisierung/Veränd. einz. Module
  Angewandte Hilfsmittel: Programmiertechnik

-- Wo tauchen Tätigkeiten dieser Arbeitsbereiche auf?



-- Definition des Begriffs Programmieren im Großen



- jede Lebenszyklusphase
  Analyse
  inkrementelle Konstruktion, Überprüfung, Prognose
  abschl. Überprüfung nach oben
  abschl. Überprüfung nach unten

- Definitionen:
  Requirements Engineering
  Programmieren im Großen

  - Analyse der Anforderungsdefinition unter Aspekten des Entwurfs
  - stückweiser Entwurf eines Softwaresystems aus Modulen und Teilsystemen
  - stückweise Übrprüfung der entstehenden Entwurfsspezifikations-komponenten: intern, gegen die Anforderungsdefinition und auf Implementierbarkeit, Integrierbarkeit sowie Wartbarkeit
  - abschließende Überprüfung gegen die Anforderungsdefinition
  - abschließende Überprüfung auf Realisierbarkeit
  - Übertragung der Entwurfsspezifikation in eine Programmiersprache: Codieren im Großen
  - Integration und Funktionsüberprüfung von Modulen und Teilsystemen bzw. des Gesamtsystems anhand der Softwarearchitektur
  - Leistungsüberprüfung von Modulen, Teilsystemen des Gesamtsystems
  - Installation des Gesamtsystems aus den Komponenten der Architektur
  - Veränderung der Architektur bei Rückgriffen, insbesondere in der Wartung, durch Neuangehen aller obigen Schritte

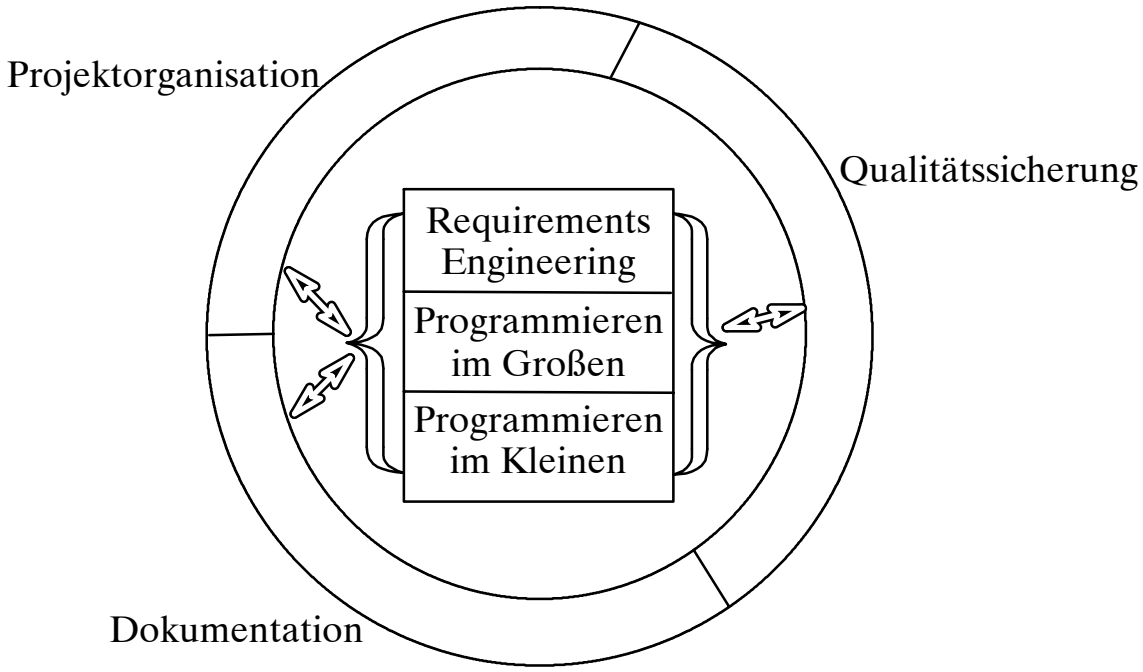
  Programmieren im Kleinen

- weiterer Arbeitsbereich Dokumentation
  - Entwicklungsdokumentation (techn. Dokumentation)
    - Entscheidungen begründen
    - Struktur erklären
  - Bedienerdokumentation
  - Projekthandbuch: invariante Teile des Projekts

- weiterer Arbeitsbereich Qualitätssicherung
  - formal (Verifikation)
  - experimentell (Test)
  - manuell (Inspektion, Walkthrough)

- weiterer Arbeitsbereich Projektorganisation
  - Einteilung:
    - Projektplanung
    - Projektführung (Projektmanagement)
    - Projektüberwachung
  - andere Einteilung
    - Management of the Tasks (Progr. in the Many)
    - Management of the Process
    - Management of the Products

-- Einteilung in Arbeitsbereiche



-- Rollenzuteilung

-- Verzahnung

-- mehrdeutige Begriffe:
Spezifikation
Implementierung
Realisierung

1.6 Eigenschaften von Programmsystemen

- - Erstellung/Wartung:
  Anforderungsdefinition → *Programmsystem in höh. Programmiersprache*
  ?
  Wie überhaupt?
  Welche unterschiedl. Möglichkeiten?
  Auswahl nach welchen Kriterien?

- - Viele Möglichkeiten: Softwarearch.; 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 Anforderungsdef. festgelegt: Bedieneroberfläche, Effizienzparameter,...
  oft nicht oder nur teilweise festgelegt: Wahlmöglichkeit

- Zuverlässigkeit
  - Korrektheit: Entwurfsspez. entspr. Anforderungsdef.
    Implementationen entspr. Entwurfsspez.
  - Robustheit: gegen falsche Eingabedaten (Anzahl, Aufbau, Wertebereich, Abhängigk.)
  - Ausfallsicherheit: Hardware--, Softwarefehler (reproduzierbar, sporadisch)

- Bedienerfreundlichkeit (Außenverhalten)
  - Verständlichkeit
  - Angemessenheit
  - vernünftiges Fehlerverhalten
  - Uniformität
  - der Bediener--funkt.
  - Prinzip d.ger. Verwunderung

- Flexibilität
  - Adaptabilität (bei Veränd. des Außenverhaltens)
  - Portabilität (bei Veränderungen der Basismaschine)

- Lesbarkeit/Einfachheit
  - Forderung an Softwarearchitektur
  - Forderung an Programmiersprache
  - Programmierdisziplin

-- Effizienz
    mech. Effizienz: Laufzeitverh., Speicherplatzbedarf
        Messen
        Ausrechnen (Schranken, oft größenordnungsmäßig:
                z.B. obere Schranke f. schlechtesten Fall)
    Tradeoff Laufzeit <--> Datenspeicherplatz
    Vernachlässigt:
Programmspeicher
                Übersetzungs- -/Neuübersetzungsaufw
                Programmerstellungsaufwand
    Fazit: Beschränkt sich auf dyn. Eigenschaften des Produkts
        Effizienzeigenschaften des Prozesses unberücksichtigt

-- Zielkonflikte
    Korrektheit stets einzuhalten
    Robustheit
    Ausfallsicherheit
    Verständlichkeit <--> mech. Effizienz
    vernünftiges Fehlerverhalten
    Adaptabilität
    Portabilität
    Lesbarkeit/Einfachheit

-- neben Forderung an ganze Programmsysteme, Forderungen an seine Teile (Module, Teilsysteme)
    Wiederverwendbarkeit
    Kombinierbarkeit
    Generierbarkeit

- - alle obigen Ziele mit Ausnahme der mech. Effizienz haben mit Veranstaltung zu tun
  - Korrektheit (→ nächstes Kap.)
  - Lesbarkeit/Einfachheit (→ alle weiteren Kap.)
  - Robustheit, Ausfallsicherh., Benutzerfreundlichkeit,
    - indirekt: werden oft übersehen und nachträgl. hinzugefügt, bedingen Eigensch. d. Software- - architektur
  - Flexibilität (→ alle weiteren Kap.)
  - Effizienz des Prozesses (Strukturierung, Flexibilität)
  - Wiederverwendbarkeit
  - Kombinierbarkeit
  - Generierbarkeit

  setzen Denken in Software- - architekturen voraus

1.7 Die Modellierungsproblematik: Allgemeines

-- Modellieren auf RE --
PiG --
PiK -- Niveau
PO --
QS --
DOK --

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

-- geeignete Modellierung

Voraussetzung für spez. Qualitätsmerkmale
z.B. für Softwaresysteme d. letzten Abschnitts

Voraussetzung auch für allg. Eigenschaften von Softwaresystemen

Verständlichkeit		Vollständigkeit
Überprüfbarkeit	fußen auf	Widerspruchsfreih.
Veränderbarkeit		Minimalität

Voraussetzung für Zusammenhang verschiedener Modellierungsebenen

Vollständigk.Anf.def.
z.B. Vollständigkeit Vollständigk.Entw.spez.
Vollständigk.PiK--Ebene

Voraussetzung für Arbeitsteilung

-- Komplexe Sachverhalte auf Übersichtsebene durch Graphen dargestellt, meist durch Diagramme repräsentiert
Diagramme in der ST: SA--Diagramme (RE)
Architekturdiagr. (PiG)
Flußdiagramme (PiK)
Netzpläne (PO)
Überdeckungsgraphen (QS)
...

-- 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 Bez. durch Markierungen
Konsistenzbedingungen:Unterscheidung zulässiger von unzulässigen Graphen
damit Einführung einer Klasse von knoten-- und kantenmarkierten, attr. Graphen

-- große Wahlfreiheit bei der Modellierung
Niveau der Betrachtung für Auswahl der
welche Aspekte modelliert man Graphenklasse
viele Möglichkeiten der Modellierung eines Sach--verhalts in einer gegebenen Graphenklasse

- - Prinzipien der Modellierung
    - Ergebnis
        - Abstraktion
        - Strukturierung
            - Modularisierung
            - Hierarchiebildung
            - Lokalität
            - Mehrfachverwendung
            - Redundanz
        - geringsten Verwunderung
    - Vorgehen
        - Beachtung d. Zusammenhangs mit anderen DOk.
        - Lebenszyklusdok. in Zusammenhang mit PO, DOK, QS
        - Rücksichtnahme auf allg. Erkenntnisse, Normen, Standards

1.8 Allgemeine Begriffe der Softwaretechnik

-- Dokumente in formaler Sprache (Notation) notiert
formale Sprache/Kunstsprache <--> natürl. Sprache
weites Verständnis von Sprache: Diagrammsprachen, Graphsprachen

-- Problembereiche (formaler) Sprachen

- Syntax:
  Welche Zeichenfolgen, Diagramme, Graphen, sind korrekt aufgebaut
  kontextfreie Syntax: Aufbau "an einer Stelle"
  aus welchen Textfragmenten, in welcher Reihenfolge
  wie ist Teildiagramm aufgebaut
  kontextsensitive Syntax: Querbez. zwischen Sprachelementen
  Beziehung definierender zu angewandtem Auftreten von Textfragmenten
  globale Zusammenhänge, Ausscheiden verbotener Situationen

- Semantik:
  Bedeutung einzelner Sprachelemente
  Bedeutung d. Zusammenh. von Sprachelementen

- Pragmatik:
  Verhältnis der Sprache zur Umwelt
  mech. Pragmatik: Auftragbarkeit, Werkzeugunterstützung
  menschl. Pragmatik: Wie gut f. Modellierung, f. best. Anwendungsbereiche
  ökonom. Pragmatik: Wert der Dokumente in dieser Sprache etc.

-- Begriffe der Softwaretechnik

- Prinzip:
  Grundsatz, den man dem Handeln zugrundelegt
  fachgebietsübergreifend
  Beisp. Modellierungsprinzip

- Technik:
  Hilfen, um gegeb. Ziel, schneller, sicherer, effizienter zu erreichen
  nichtautomatisiert: Methoden, Verfahren, Lehr-- und Lernmaterial
  (teil)automatisiert: Werkzeuge, Geräte, Dienst-- programme

- 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:
  automatisierte Unterstützung von Verfahren, Methoden, Notationen, Prinzipien

- Notation:
  Sprache, in der Softwaredokument erstellt wird
  Sprache unterstützt ”Methode”, die Prinzipien folgt

1.9 Werkzeuge der Softwareentwicklung

-- Bedeutung der Unterstützung erst spät erkannt
klassische PS wenig
interpreteror. PS etwas mehr Unterstützung
uns interessiert Unterstützung bei Erst./Veränd. großer Programmsysteme

-- Sprachimplementation einer höheren PS:
Editor, Compiler, Laufzeitpaket, Binder, Lader, Sprachstandard

-- Programmiersystem
Sprachimplementation plus Trace, Dump
ggf. mehrere Sprachen
verschiedene Compilervarianten

-- Seit 1980 Bedeutung von Werkzeugunterstützung erkannt
Qualitäts-- insb. Zuverlässigkeitserhöhung
Entlastung von Details/Produktivitätssteigerung
ausgehend von PiK zu RE, PiG, PO, QS, DOK
Softwareentwicklungs--Umgebungen
Softwareproduktions--Umgebungen

-- Werkzeuge in Softwareentwicklungs--Umgebungen

strukturbezogen/syntaxorientiert (k.f. u. k.s.)
integriert
inkrementell
interaktiv/sofort

-- Werkzeuge für das PiG

- -- 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
- -- strukturbezogener 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
- -- Anzeige wiederverwendbarer vordefinierter Module und Teilsysteme
- -- Werkzeug zur Ersetzung einer Teilarchitektur durch eine andere mit gleichem "Außenverhalten"
- -- Werkzeug zur Überprüfung der Konsistenz einer Architektur mit der Anforderungsdefinition
- -- Erzeugung von Modulrahmen für eine bestimmte Programmiersprache (Codieren im Großen)
- -- Werkzeug zur getrennten Übersetzung
- -- Werkzeug zur Qualitätssicherung, z.B. Testwerkzeug für sog. Blackbox--Test--Methoden
- -- 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

- -- Werkzeug zur Messung eines Programmsystems oder teilweise fertiggestellten Programmsystems, z.B. für Laufzeit oder Speicherplatz

- - Werkzeug zur Versionskontrolle (Bausteine einer Architektur existieren in verschiedenen Zuständen, diese nennt man Revisionen; von Teilarchitekturen gibt es unterschiedliche Realisierungen, diese nennt man Varianten)
- - Werkzeug zur Konfigurationskontrolle (Zusammenbau einer Gesamtarchitektur aus bestimmten Varianten und Revisionen von Modulen und Teilsystemen)
  ...

- - Werkzeugklassen
  - Editoren
  - Analysatoren
  - Transformatoren
  - Instrumentatoren
  - Ausführer
  - Werkzeuge für Erstellung von Repräsentationen

  Verzahnungswerkzeuge (z.B. für PiG)
  - Analyse RE
  - Mitbetrachtung PiK
  - abschließender Abgleich RE - - PiG
  - analog für PO, QS, DOK

- - gegenw. Probleme von Softwareentwicklungs- -Umgeb.:
  - Weiterentwicklung von Methoden und Notationen
  - Konsistenz zwischen verschiedenen Dokumenten
  - Inkrementalität dokumentintern u. dokumentübergr.
  - Austauschen und Kombinieren versch. Methoden
  - Wiederverwendung
  - Rapid Prototyping
  - Standardisierung von Werkzeugen
  - ...

1.10 Zum Stand der Softwaretechnik

- - Stand der Softwareerstellung: auch heute kaum wiss. durchdrungen, im folgenden einige Gründe, Vergleich mit anderen Ingenieurwissenschaften
  - Hektik der Entwicklung
  - Breite der Anwendungen
  - Probleme der Software werden nicht verstanden

  - Immaterialität von Software:

Erfahrungen mit Software können nur durch Beschreibungen derselben oder durch das Ausprobieren eines Produkts gewonnen werden. Software kann man nicht durch Sinneswahrnehmung erfahren.

Software hat eine unglaubliche "Wandelbarkeit". Dies ist ein Vorteil, aber auch ein gravierender Nachteil. So manches Programmsystem ist in der Wartungsphase von einem Küstenmotorschiff zu einem Mammuttanker geworden.

Unterschiede der Realisierung kann man bei Software kaum feststellen, man muß hierzu schon ein Fachmann des Anwendungsgebietes sein. Qualitätsmerkmale sind also schwer überprüfbar. Der Software ist von außen nicht anzusehen, ob sie solide entwickelt oder zusammengeschustert worden ist.

Mit der Schwierigkeit, eine Realisierung einzuschätzen, ist die Schwierigkeit verbunden, den Wert von Software zu bestimmen.

Software altert nicht und unterliegt keinem Verschleiß. Dadurch ist man nicht gezwungen, Software, die dem Stand der Technik nicht mehr entspricht, wegzuwerfen. Andererseits verhindert langandauernde Wartung, daß ein Softwaresystem funktionsfähig bleibt.

Die Zerstörung der Struktur von Software durch Wartung ist von außen schwer feststellbar. Dadurch wird auch wenig Nachdruck auf Wartung gelegt, die die Struktur des Produkts erhält. Dies ist einer der wesentlichen Gründe dafür, daß ein großer Teil der Software- -Entwickler mittlerweile mit Wartung beschäftigt ist.

Softwaretechniker gehen ausschließlich mit geistigen Produkten um, d.h. der Kernpunkt der Softwareentwicklung liegt in der Modellierungsproblematik. Wegen der Breite der Anwendungen, für die Software entwickelt wird, ist dies die Modellierungs- - oder Strukturierungsproblematik schlechthin. Für diese wird man in keiner Wissenschaft tiefschürfende Ergebnisse finden.

  - Mangelndes Übereinkommen
  - Schwierige Randbedingungen

- Resümee
  - Hektik verhindert tiefes Durchdringen
  - Breite mach allg. Aussagen schwer
  - Softwareprobleme auch heute noch unverstanden
  - Rad tagtäglich neu erfunden
  - Randbed. verstellen Blick auf das Wesentliche

  These: Kommen nur voran bei
  - Betrachtung best. Problemklassen
  - Struktur der Software erkennen und geeign. beschreiben
  - Standardbausteine, Wege zur Erzeugung gewinnen

1.11 Zusammenfassung

- allg. Abschnitte
  - Softwarekrise
  - Stand der ST
  - Begriffe der ST

- Grundlage für das folg. Phasenmodell
  - idealisiert: Rückgriffe, Vorgriffe, Feinstruktur der Phasen

- zusätzlich Einteilung in Arbeitsbereiche
  - RE, PiG, PiK. PO, QS, DOK
  - Rollen, Notationen, Methoden, Werkzeuge, Verzahnung

2. DAS PROBLEM: MODELLIEREN AUF ENT- - WURFSEBENE

2.1 Zur Korrektheit von Programmsystemen

- Gedankenspiel zur Korrektheit von Programmsystemen nach Dijkstra

  Wahrscheinlichkeit der Korrektheit eines Programm-systems nach Dijkstra

  p Wahrscheinlichkeit, daß einz. Komponente korrekt

  $P=p^N$ in etwa für System aus N Komponenten

  N=10, p=0.99
  - $\Rightarrow$ P=0.9

  N=10, p=0.9
  - $\Rightarrow$ P=0.35

  N=100, p=0.99
  - $\Rightarrow$ P=0.37

  N=100, p=0.9
  - $\Rightarrow$ P=0.000027 !!!

  Schlußfolgerung: kein großes Programmsystem ist korrekt !

- obige Annahme in Formel zu optimistisch:

  Wahrscheinlichkeit der Korrektheit der Module verschieden

  Anteil für Verbindungen zwischen Modulen kommt hinzu

-- Was heißt hier ”falsch”:
    statisch, prinzipiell falsch, irgendwann eintretend
    wieso laufen manche Systeme einigermaßen zuverlässig:
    einige Programmpfade werden nie, andere oft durchlaufen, letztere ausgetestet

-- Wird sich die Situation durch geeignete Modellierung auf Entwurfsebene ändern ?
    nicht dramatisch
    was bringt die Modellierung dann:
        Fehlerlokalisierung, Fehlererkennung, Fehlerbehebung mit vertretbarem Aufwand

-- Was waren oben Fehler ?
    Fehler im Kleinen (bei Nichtvorh. PiG Modellierung im ganzen)
    Fehler im Sinne falsch verstand. Anforderungsdef.
        bei der Erst. u. Wartung ebenfalls leichter durch geeignete Architekturüberlegungen

-- Fazit:
    Beseitigung von Fehlern gleich welcher Art hat immer mit Anpaßbarkeit von Softwaresystemen zu tun: Gegenstand der Veranstaltung

2.2 Die Festlegung der Entwurfs- - spezifikation

- - Wiederholung: Aufgabe d. Entwurfs = Beschreibung des Innenlebens eines Softwaresystems auf einer groben Ebene
  statisch: Modulgeflecht, Softwarearchitektur, Bauplan

- - Festlegung in einer Sprache
  informell, formal; Graphik, Text
  Aspekte: Syntax, Semantik, Pragmatik
  Graphik: Module: Art, Name als Knoten
  Beziehungen: Art als Kanten
  zur Übersicht
  Text: Module Export und Import
  zur detaillierten Festlegung

- Syntax der Entwurfsspezifikation
  - Kontextfreie Syntax: Aufbau eines PiG--Dok. an einer Stelle
    - Welche Textelemente (Arten), wie aufgebaut
    - Welche Graphikelemente (Arten)
  - Kontextsensitive Syntax: Zusammenwirken verschiedener Teile
    - Übereinstimmung verschiedener Textteile
    - Aussondern verbotener Strukturen

- Semantik der Entwurfsspezifikation
  - Verschieden v. Semantik eines fertigen Programms: keine dyn. Semantik, da Rümpfe noch offen
  - Zusammenwirken der Module als statische Angaben drücken wir innerhalb der Syntax aus
  - Begriff Softwarearchitektur bevorzugt
  - Sprache und Architektur haben viel mit Semantik des Anwendungsbereichs und Semantik des beschriebenen Programmsystems zu tun
- Semantik der Module
  - kann formal definiert werden
    - alg. Spezifikationen
    - Vor--/Nachbedingungen
    - hier nicht betrachtet

-- Pragmatik
    Implementierbarkeit einer Architektur
    Übertragbarkeit in eine PS
    Lesbarkeit einer Architektur
    Arbeitsteilung, Überprüfbarkeit, Kostenermittlung
    Auftragbarkeit eines Architekturdiagramms

-- Hatten bereits versch. Bedeutungen von Spezifikation
    (RE, PiG, math. Spez.)
  Auf Architekturmodellierungsniveau
    Entwurfsspezifikation (Architektur (Syntax),
      Funktion (Sem.), z.B. Machbarkeit (Pragmatik)
    Architektur allein
    Festlegung eines Moduls der Architektur (Syntax,
      Sem., Pragmatik)
    Syntaktischer Anteil der Spezifikation eines Moduls

-- Zielsetzung d. Veranst.: Einführung einer formalen
  Sprache f. Softwarearchitekturen (Text + graph. Form)
    Bedeutung

RE,	PiG,	PiK
z.B. SA	?	z.B. übl. PS
o.ä.	Modulkonzept von PS	

-- Architekturbeschreibung hier
    Detailbeschreibung aller Module (Export, Import)
    enthält alle Informationen
    Überblicksdarstellung: Architekturdiagramm
    zusätzl. Design Rationale (DOK)
    Skizze der bet. Rümpfe (PiK)

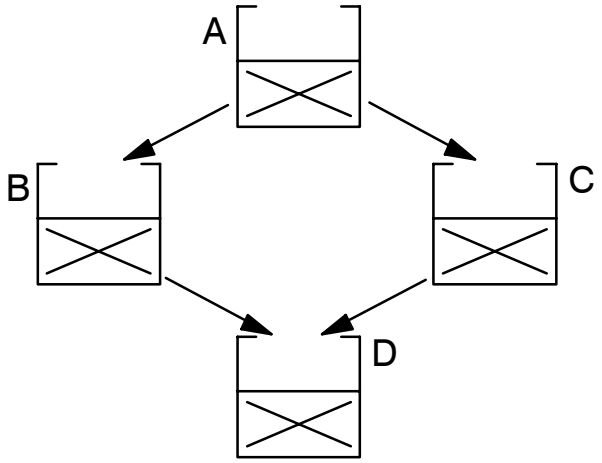
-- Bedeutung einer solchen Notation
    Quintessenz eines Programmsystems:
    Dokument zur Beurteilung der Struktureigensch.
    Qualitätseigenschaften damit bewertbar
    Wiederverwendbarkeit

-- Titel der Veranstaltung
    gibt keine Methode (automatisch) zur Erstellung
    von Softwarearchitekturen: geistig schwierige Aufg.
    neben geeign. Sprache: viele Hinweise, Teilarch.,
    Standardstrukturen

-- im folg. Beschreibung von Architekturen und Teilarch.:
    Momentaufnahmen (Ergebnisse) des Entwurfs/
    der Wartung
    Weniger mit Prozeß selbst
        d.h. Entstehen einer Architektur
        RE -->PiG -- Übergang
        PiG --> PiK -- Übergang

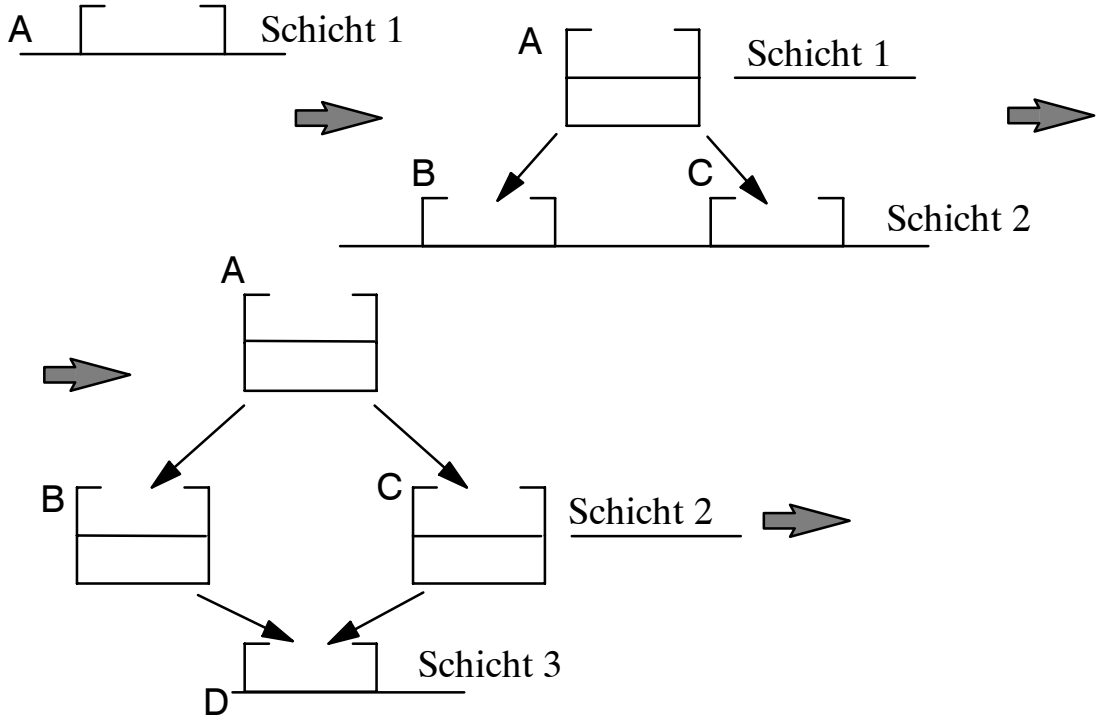
2.3 Das Architekturparadigma

-- Softwarearchitektur ist idealisierte Vorstellung
(Architekturparadigma, diskretes Paradigma)
alle Niveaus einer Architektur bedacht
alle Module aufgeführt
ohne Modulrümpfe zu betrachten



-- Probleme
schwer, über alle Niveaus nachzudenken
tiefe Niveaus vage
Erkennen der für die Realisierung eines Moduls
nötigen Ressourcen ohne daß dieser real. wird
Module sind schwer zu realisieren: Notwendigkeit
Herausziehen von Teilen, (Moduln)
Module zu trivial: eingebettet, Verschmelzen auf
Architekturebene
=> PiK müßte bereits durchgeführt sein
=> führt zu vielen Rückgriffen

-- kontinuierliches Paradigma: PiG und PiK verzahnt
Spezifikation und Implementation in einem schichtenweise
Gesamtprogramm entsteht durch Schichten



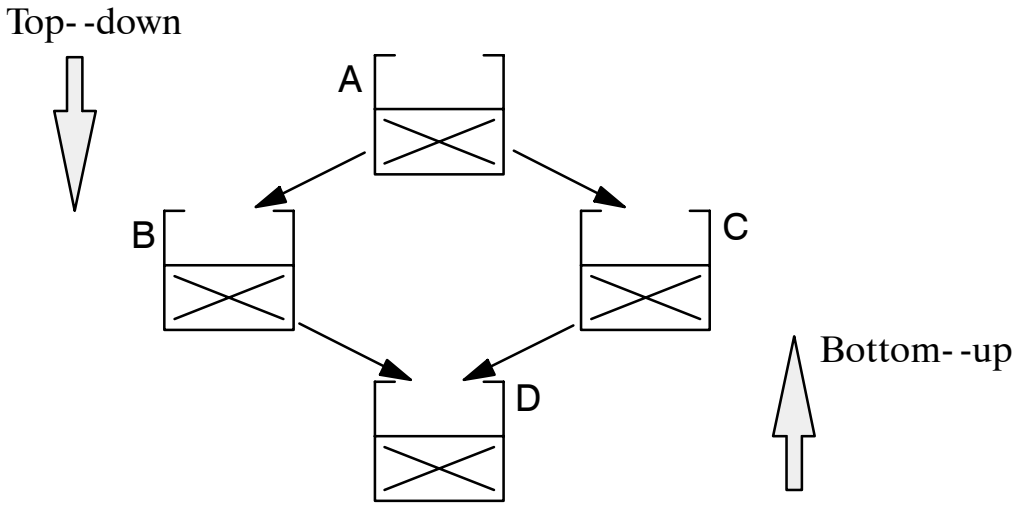
- - Architekturparadigma unvermeidlich, denn bei PiG- - PiK- -Verzahnung:
    gäbe es keine Arbeitsteilung (Entw., Impl.,...,)
    entsteht vielleicht gar keine Architektur
    PiG- -PiK- -Ebenen gehen durcheinander
    Erkennen von Gemeinsamkeiten schwierig
    Überprüfung der Architektur vor Realisierung
        nicht möglich

- - Name Programmieren im Großen
    wird nicht programmiert
    Programmieren ist aber mit zu bedenken

- - Architektur ist Abstraktion
    Herausfaktorisieren aller Modulrümpfe
    versch. von der Abstraktion durch bestimmte Schicht
        eines Softwaresystems

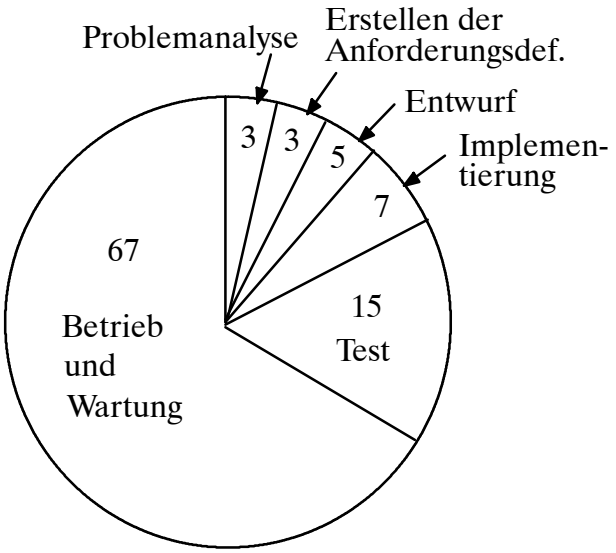
- - große Systeme: Architekturerstellung arbeitsteilig
    Gesamtarchitektur bis auf Module/Teilsysteme
    Teilarchitekturen für Teilsysteme
    obige Idailisierung zwei- - oder mehrfach
        Gesamtarchitektur ohne Modulrealisierungen
        Gesamtarchitektur ohne Teilsystemarchitekt.

- - Entwicklungsstrategien (nach Architekturparadigma Entwurfsstrategien):
    - top- -down
    - bottom- -up
    - jo- -jo

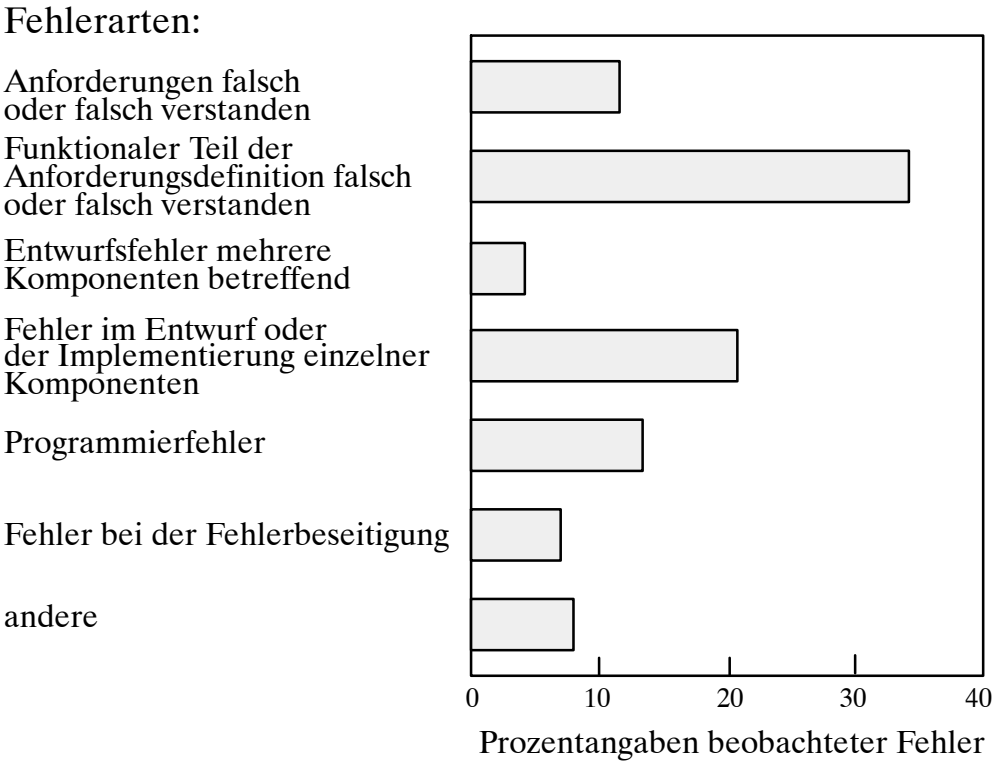


2.4 Zur Bedeutung des Programmierens im Großen

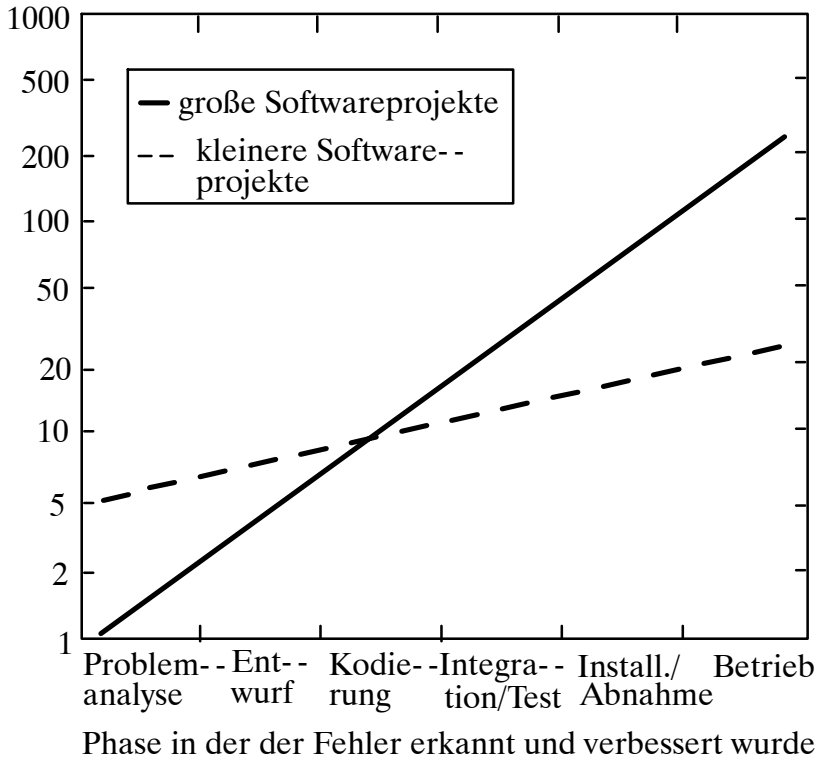
-- Zum Aufwand in einzelnen Phasen



-- Arten von Fehlern


Fehlerarten:
Anforderungen falsch oder falsch verstanden
Funktionaler Teil der Anforderungsdefinition falsch oder falsch verstanden
Entwurfsfehler mehrere Komponenten betreffend
Fehler im Entwurf oder der Implementierung einzelner Komponenten
Programmierfehler
Fehler bei der Fehlerbeseitigung
andere
0
10
20
30
40
Prozentangaben beobachteter Fehler

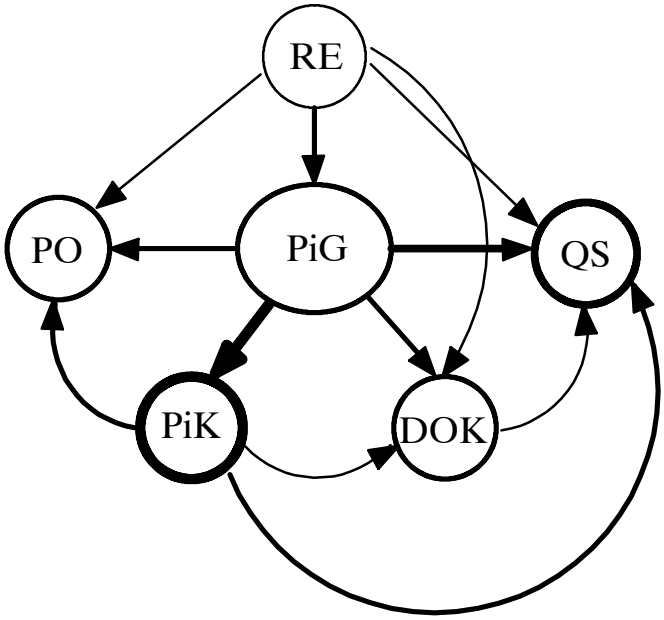
- relative Kosten von Fehlern



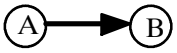
- Schlußfolgerung:
  Aufwandserhöhung
  Notationen einführen für RE
  Erfahrungen im Umgang sammeln und PiG
  für Erstellung/Wartung

-- Verzahnung der Arbeitsbereiche
Beisp. Änderung eines Softwaresystms

-- PiG im Zentrum



Erklärung:



Strichstärke für den Pfeil gibt ungefähr den Anteil des Arbeitsaufwands in B an, der durch Festle-- gungen in A bestimmt ist.

Strichstärke des Kreises gibt ungefähr den abso-- luten Aufwand von A bei der Softwareerstel-- lung an

=> Softwarearchitektur ist die wichtigste
Struktur eines Softwaresystems
Änderungen übersehen
Struktureigenschaften erkennen
Qualitätseigenschaften beurteilen
Standardlösungen identifizieren
Wiederverwendbarkeit von Komponenten

- - Abgrenzung PiG - - Projektorg. insb. Projektmanagement (Programming in the Many)
  - Versions- - und Konfigurationskontrolle
  - Verantwortlichkeits- - und Zugriffskontrolle
  - Freigabekontrolle
  - Nachrichten- - und Dokumentverteilungskontrolle
  - Erfolgskontrolle

  - bezieht sich auf Architektureinheiten
  - rechnen es nicht zum PiG

2.6 Zum Einfluß des Softwareerstellungs--Paradigmas

-- hier klassische Vorstellung
    ingenieurmäßiges Paradigma (phasenorient., probl.--bereichsorient., Architekturparadigma)
    beeinflußt von Linie klass. Spr. (v.Neumann--Spr.); neue Entw. Ada, Modula--2; anwendbar auf ältere: FTN, Cobol, Ass.)

-- nichtklass. Paradigmen
    appl. Programmierung
    funkt. Programmierung
    log. Programmierung
    objektorient. Programmierung

-- Charakterisierung: kontinuierl., keine unterschiedl. Problember., keine Arch. vorab
        aus Spez. manuelle Transf.
        Verw. v. Transf. (vorg. v. Programmierer)
        automatische Übersetzung

-- Expertensysteme / wissesnbasierte Systeme

2.7 Zusammenfassung

- Notwendigkeit des Festhaltens einer Architektur
  - Architekturdiagramm
  - detaillierte Modulbeschreibungen
  - Design Rationale
  - Modulrumpf Specs

- Bedeutung des PiG
  - Strukturfesthalten v. d. Erstellen d. Programmsyst.
  - Architektur verstehen als Quintessenz
  - Wartungsüberlegungen auf Architektur
  - Qualitätsbeurteilung
  - Wiederverwendbarkeit

- im folg. diskr. Paradigma
  - RE
  - PiG
  - PiK
  - ...

3. EIN ERSTES BEISPIEL: OHNE VORÜBERLEGUNG

3.3 Charakterisierung der Lösung

- Struktur der Lösung:
  - lauter funktionale Module
  - Kanten ”Unterauftrag weitergeben”
  - Architektur ist Baum
    - Gründe
      - haben einiges übersehen
      - Top--down--Entwicklung
      - denken in funkt. Abstraktion
  - Baum ist teilweise geordnet
  - Struktur ergibt sich nahezu 1--1 aus Aufgabenstell.
    (Bedieneroberfläche bei interakt. Beispiel)
  - stützen uns auf Hilfsmittel, die nicht in der Arch.
    erscheinen, z.B. Datei
  - globale Daten, Datenflüsse

- bereits jetzt erkennbare Fehler:
  - Gemeinsamkeiten nicht erkannt: z.B. Syntaxanalyse
    eines Telegramms
  - Bausteine haben sehr unterschiedliche Komplexität
    - Gesamtdatenermittlung Dreizeiler
    - Einzelauswertung aufwendig, insb. wenn Syntax--
      analyse vorgen. wird

3.4 Zusammenfassung

- - kleines Batch- -Problem, ohne Vorüberleg. auf RE- -, PiG- -Niveau entworfen
  Architektur hat bestimmte Charakteristika u. Fehler: Ist Baum, ausschließl. funkt. Komponenten, globale Daten, Datenflüsse

4. DIE NOTATION: EIN EINFACHES MODULKONZEPT

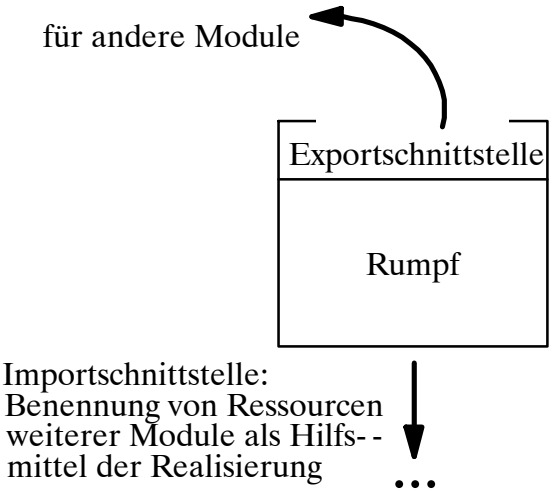
4.1 Module als Bausteine einer Architektur

-- Klärung des Begriffs ”Modul”
schillernde Semantik

-- Diskussion ”Was ist ein Modul?”

-- Modulcharakteristika

- ”log. Einheit” in Gesamtzusammenhang, ein Satz zur Beschreibung
- M. ist abstr. Maschine oder Hilfsmittel
- M. repr. Entwurfsentscheidung, Arch. ist Gesamtheit aller Entwurfs--entscheidungen
- M. ist Einheit aus Daten und Operationen
- M. hat ”bestimmte Komplexität (Spezifikationsseiten, Quelltextseiten)
- stellt Ressourcen nach außen zur Verfügung für andere Module: Exportschnittstelle. Ress. einfach und orthogonal



- Interna (Rumpf) verkapselt
- Bei Implementierung Ab--stützen auf andere M.e: Importschnittstelle. Verbin--dung zu and. M.en.
- neben(seiten)effektfrei: Verwendung/Änderung der Parameter d. Export--schnittstelle, Importschnitt--stelle
- ersetzbar durch M. mit gleicher Exportschnitt--stelle: keinen Einfluß auf Semantik, aber Pragmatik
- Korrektheit des M.s ohne Kenntnis seiner Verwendung nachweisbar
- ”Korrektheit” der Entwurfsspezifikation ohne Kenntnis der Imple--mentationen der M.e ”nachweisbar” (formale Anforderungsdef.)
- M. unabhängig entwickelbar: Arbeitseinheit
- M. getrennt übersetzbar
- Einheit der Wiederverwendung

- - Module sind log. Einheiten der Architektur- - modellierung
  keine PiK- -Einheiten
    UPe/Makros
    sequentiell, nebenläufig, reentrant
    solche Details erst später
  keine Einheiten der Programmiersprache

- - Modul aus versch. Teilen zusammengesetzt und mit anderen in Verbindung
    wird an versch. Stellen verwandt:
      hierfür Exportschnittstelle
    verwendet Ressourcen anderer Module:
      hierfür Importschnittstelle
    Realisierung:
      hierfür Rumpf und Importe

- - Unterscheidung (Export)Schnittstelle und Rumpf

Programmieren- -im- -Großen- -
Anteil des Moduls

```
abstract data object module ITEM_STACK is - -- -****Export- -Schnittstelle***- -- -
   ...                                                              - -- -
   procedure PUSH (X: in ITEM_TYPE);         - -- -Zugriffsoperationen einer- -- -
   procedure POP;                            - -- -FIFO- -Datenstrukt. beste- - -- -
   function READ_TOP return ITEM_TYPE;       - -- -hend aus Veraenderungen- -- -
   function IS_EMPTY return BOOLEAN;         - -- -und Abfragen. Die Bedeu- -
- -- -
   function IS_FULL return BOOLEAN;          - -- -tung der einzelnen Opera- -
- -- -
   ST_UNDERFLOW, ST_OVERFLOW: exception; - -- -tionen ist die folgende:- -- -
   - -- -PUSH legt ein Element auf dem Keller ab. POP loescht das oberste Ele- -- -- -
   - -- -ment. Mit READ_TOP kann das oberste Element abgefragt werden, mit - -- -
   - -- -IS_EMPTY, ob der Keller leer ist, mit IS_FULL, ob er voll ist.      - -- -
   - -- -ST_UNDERFLOW und ST_OVERFLOW sind Ausnahmen (Fehleranzeigen).
- -- -
   - -- -Die erstere wird erweckt, wenn versucht wird, mit POP das "oberste Ele- -- -- -
   - -- -ment" eines leeren Kellers zu loeschen, die zweite, wenn mit Hilfe von - -- -
   - -- -PUSH ein weiteres ELement auf einem vollen Keller abgelegt wird.      - -- -
end                                                              ITEM_STACK;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -
- -- -= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
- -- -
module body ITEM_STACK is - -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- - Rumpf des Moduls
- -- -- -- -- -- -- -
   SIZE: INTEGER := 100;                                         - -- -
   SPACE: array (1..SIZE) of ITEM_TYPE;                          - -- -
   INDEX: INTEGER range 0..SIZE := 0;
- -- -
   procedure PUSH (X: in ITEM_TYPE) is begin ... end;            - -- -
   ...                                                           - -- -
   function IS_FULL return BOOLEAN is begin ... end;             - -- -
   ...                                                           - -- -
end ITEM_STACK;  - -- -****************************************- -- -
```

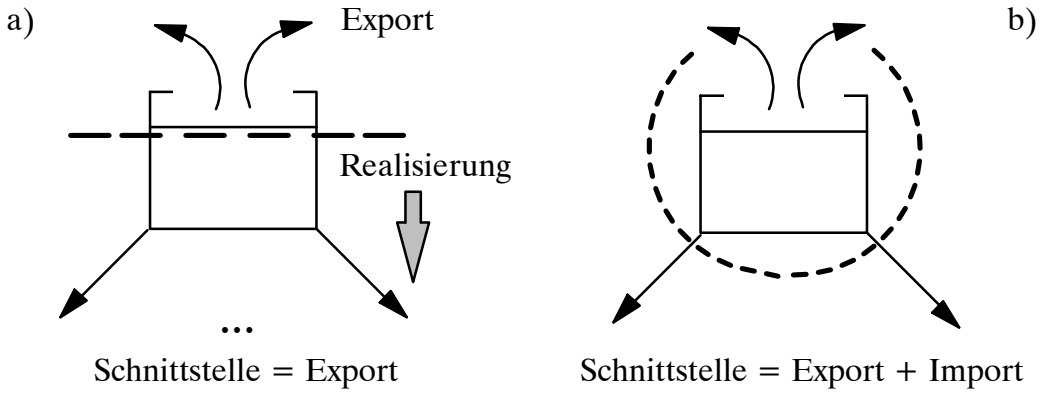
Programmieren- -im- -Kleinen- -
Anteil des Moduls

-- Übereinstimmung Schnittstelle -- Rumpf

-- Schnittstelle sichtbar, Rumpf verborgen

-- Sinn des Verbergen des Rumpfs
Information--Hiding--Prinzip:
Einhalten des Abstraktionsschritts Rumpf --> Schnittstelle
Grundlage des Architekturparadigmas
Verwender will Rumpf i.a. nicht sehen
Grundlage für Austauschbarkeit und Wieder--verwendbarkeit

-- Sprachgebrauch "Schnittstelle"



-- Module in Architekturdiagramm: Modulsymbol, Art, Name
Module in textueller Detailbeschreibung:
zusätzlich Exportschnittstelle, Importschnittstelle

4.2 Modularten und Funktionsmodule

-- Abschnitt
    Klassifikation Modularten
    abschließende Einführung funktionaler Module

-- Module
    funktionale Abstraktion:
        funktionaler Modul/Funktionsmodul
    Datenabstraktion
        abstr. Datenobjektmodul, abstr. Datentypmodul

-- funktionale Abstraktion
    Module haben Transformationsverhalten
        Eingabedaten in Ausgabendaten transformieren
        funktionale Module ”aktionsorientiert”
        E/A--Daten erscheinen in der Schnittstelle
        oft mehr als eine Operation an der Schnittstelle
        kein Gedächtnis
    Formalisierung
        durch Vor-- und Nachbedingungen bei jeder Op.

-- Datenabstraktion
    Module folgen Datenabstraktions-- oder Daten--
    verkapselungsprinzip
        Strukturen haben Gedächtnis, d.h. inneren Zust.
        Datenabstraktionsmodul ”passiv”
    Formalisierung
        algebraische Gleichungen: $POP \circ PUSH \equiv ID$
        Vor-- und Nachbedingungen

- - Datenabstraktion aus der Theor. Informatik: formale algebr. Spez.
  hier die softwaretechnische Bedeutung der Datenabstraktion

- - beide Abstraktionsprinzipien zur Architektur- - modellierung wichtig
    ohne DA: nicht anpaßbar
    ohne funkt. Abstraktion: unübersichtlich

- - Erläuterung kurz
    wegen Vertrautheit mit funkt. Abstraktion
    jedoch mehr als eine Funktion an der Schnittstelle

- - Beispiel eines funktionalen Moduls

```
functional module ZEICHNE_FUNKTION is
- -- -**********************- -- -
   - -- -Eingabedaten jeweils in der Parameterliste, Ausgabedatum ist das - -- -
   - -- -erstellte Plotterfile - -- -
   procedure POLYGON_LIN(X,Y: in FELD; X_TEXT, Y_TEXT, UE_TEXT: - -- -
                              in STRING); - -- -
   procedure INTPOL_LIN(X,Y: in FELD; X_TEXT, Y_TEXT, UE_TEXT: - -- -
                              in STRING); - -- -
   procedure APPROX_LIN(X,Y: in FELD; X_TEXT, Y_TEXT, UE_TEXT: - -- -
                              in STRING); - -- -
   - -- - alle weiteren Prozeduren mit der gleichen Parameterliste - -- -
   procedure POLYGON_HLOG(...); - -- -
   procedure INTPOL_HLOG(...); - -- -
   procedure APPROX_HLOG(...); - -- -
   procedure POLYGON_DLOG(...); - -- -
   procedure INTPOL_DLOG(...); - -- -
   procedure APPROX_DLOG(...); - -- -
   ... - -- -
   - -- - Angabe der Semantik von ZEICHNE_FUNKTION: - -- -
   - -- - globale Vorbedingung, d.h. Vorbedingung für alle Operationen: - -- -
   - -- - Seien X, Y Parameter eines reellen Feldtyps FELD, d.h. - -- -
   - -- - Xi, Yi ⟨ REEL, i = 1, ..., FELDGROESSE - -- -
   - -- - X_TEXT, Y_TEXT, UE_TEXT ⟨ STRING, d.h. Zeichenketten- -- -
   - -- - globale Nachbedingungen: - -- -
   - -- - Es erfolgt die Ausgabe mit Hilfe eines Plotters auf ein - -- -
   - -- - Zeichenblatt wobei für die Abzissen- - und Ordinatenwerte - -
- -
   - -- - geeignet skaliert wird, und X_TEXT an der X- -Achse, - -- -
   - -- - Y_TEXT an der Y- -Achse und UE_TEXT als Bildueber- - - -- -
   - -- - schrift erscheint. - -- -
   - -- - Nachbedingungen für einzelne Operationen:
- -- -
   - -- - POLYGON_LIN verbindet die eingegebenen Punkte durch einen Polygon
- -- -
   - -- - zug und traegt X- - und Y- -Achse linear auf. - -- -
   - -- - INTPOL_LIN verbindet die eingegebenen Punkte durch eine glatte
- -- -
   - -- - Spline- -Kurve und traegt X- - und Y- -Achse linear auf.
- -- -
      ... - -- -
end ZEICHNE_FUNKTION;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -

module body ZEICHNE_FUNKTION is
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
   ... - -- -
begin - -- -
   ... - -- -
```

- In Beispiel
  - X, Y Eingabedatenstrukturen
  - Ausgabedatenstruktur ist Zeichenblatt
    - nur als Kommentar der Schnittstelle
    - würde in einer fertigen Architektur als Modul erscheinen

- Weitere Beispiele funkt. Module
  (gehäuft in Batch- - oder Transformationsproblemen)
  - Compiler, Compilerphasen
    - mehrere Eingabe- - bzw. Ausgabedatenstrukt.
  - math. Funktion
  - Hauptprogramm
  - Steuerung eines Dialogs

- Typische Anwendung funkt. Module:
  - Steuerungs- - oder Koordinationsaufgabe
  - Transformationsprobleme
  - Auswertungen
  - Hilfsdienste auf einer oder versch. Datenstrukturen

- Erläuterung kurz, Anwendung evtl. schwierig
  - insb. Unterscheidung funkt./Datenabstraktion
  - Anwendung stets zusammen mit Datenabstraktion

-- Worin besteht bei funkt. Modul die funkt. Abstraktion?

- Realisierung der Op (mit E/A--Char.) der Schnittst.

im Rumpf

mit lok. Prozeduren
mit anderen Prozeduren

- Realisierung --> Schnittstelle
  Details verborgen: Information Hiding
- Details:
  Innenleben des Rumpfes
  oft Teilarchitektur unter funkt. Modul

-- Sonderfall: eine Operation an der Schnittstelle
heißt nicht, daß einfach zu realisieren
leicht abbildbar auf Prozeduren
bleiben bei der text. Schreibweise

4.3 Das Datenabstraktionsprinzip und Datenobjektmodule

-- Abschnitt:
Datenabstraktionsprinzip und seine softwaretechn. Bedeutung
Einführung abstrakter Datenobjektmodule

-- Idee der Datenabstraktion
keine direkten Zugriffe auf Datenstrukturen
Zugriffsop. mit Datenstruktur unauflösliche Einheit
ausschließlich über Zugriffsop.: log. Schnittstelle
Details der Realisierung verborgen
Realisierung auswechselbar
Anwendung des Prinzips des Information Hiding

-- Name ”Datenabstraktion”, ”abstr. Datenobjekt” etc.
Wie ”abstrakt” ist ein ado?
Schnittstelle abstrakter als Realisierung
Auftreten gehäuft in unt. Teilen einer Architektur
Abstraktion von einer Vielzahl von Realisierungen

-- Sprechweisen
Datenabstraktionsprinzip
abstraktes Datenobjekt (ado)
abstrakter Datenobjektmodul
abstrakter Datentyp (adt)
abstrakter Datentypmodul

-- Datenobjektmodul: Beispiel
Rumpf (PiK) betrachten

Schnittstelle

```
abstract data object module AUSKUNFTEI is ---*********************---
   ---Das folgende Lexikon stellt Operationen zum Ablegen (STORE), zum  ---
   ---Aendern (CHANGE) und zum Auffinden (FIND) von Eintraegen zu                --
--
   ---Personen zur Verfuegung.                                          ---
   ---Die Semantik der einzelnen Zugriffsoperationen ist die folgende: ...  ---
   procedure FIND (KENNZ: in STRING_K; GES_INFO: out STRING_I);                 --
--
   procedure STORE (KENNZ: in STRING_K; INFO: in STRING_I);             ---
   procedure CHANGE (KENNZ_ALT, KENNZ_NEU: in STRING_K;                 ---
                     INFO: in STRING_I);                                ---
   function IS_EL_OF (KENNZ: in STRING_K) return BOOLEAN;               ---
   function IS_SPACE return BOOLEAN;                                    ---
   THERE_IS_NO_ENTRY, ALREADY_THERE, MEMORY_FULL: exception;            ---
end                                                                  AUSKUNFTEI;
-----------------------------------------------------------------------------
---    ---
```

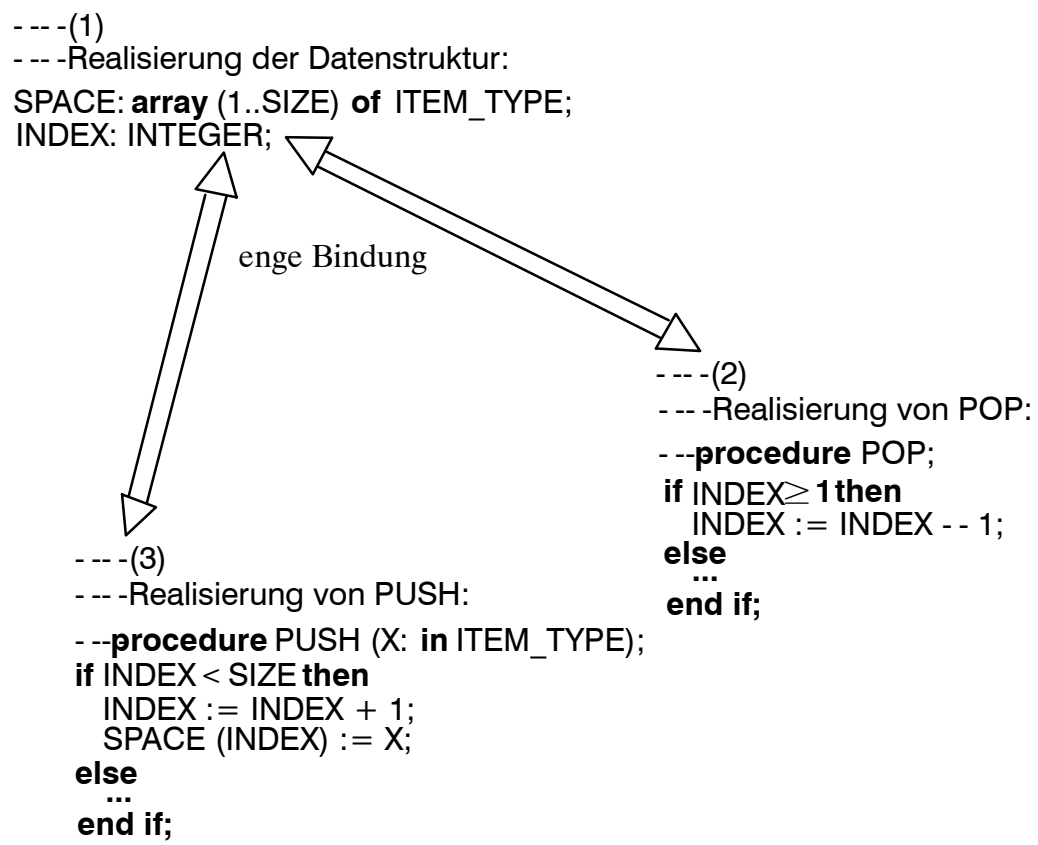
Rumpf

```
module body AUSKUNFTEI
is---------------------------------------------------------------------------------------------------
-----
   ---Deklarationen für die Haldenstruktur:                                      ---
   type SCHLUESSELWERT is INTEGER range 1..100_000;                                 ---
   type BAUM_LE; ---LE steht fuer Listenelement                                     ---
   type Z_BAUM_LE is access BAUM_LE;                                                ---
   type BAUM_LE is                                                                  ---
     record                                                                         ---
       KEY: SCHLUESSELWERT;                                                         ---
       INFO: STRING_I;                                                              ---
       LINKER_SOHN: Z_BAUM_LE := null;                                              ---
       RECHTER_SOHN: Z_BAUM_LE := null;                                             ---
     end record;                                                                    ---
   ZEIGER_AUF_WURZEL, GEF_KN: Z_BAUM_LE; ---ggf. weitere Zeiger                     ---
   ENTH: BOOLEAN;                                            ---zur Beschleuni-
gung ---
   ...                                                                          ---
   ---lokale Prozeduren:                                                            ---
   function PRIMAERSCHLUESSEL(KENNZEICHNUNG: in_STRING_K)
---
     return SCHLUESSELWERT is begin ... end;                                        ---
   procedure SUCHE_IN_BAUM(GES_SCHLUESSEL: in SCHLUESSELWERT;--
--
     ANF_KNOTEN: in Z_BAUM_LE := ZEIGER_AUF_WURZEL;                                 ---
     ERFOLG: out BOOLEAN; ENDKNOTEN: out Z_BAUM_LE) is                              ---
   begin ... end;                                                                   ---
   procedure AKTUALISIERE_BAUM(AKT_SCHLUESSEL:                                      ---
     in SCHLUESSELWERT; AKT_INFO: in STRING_I) is                                   ---
   begin ... end;                                                                   ---
   ...                                                                          ---
   ---Realisierung der Schnittstellenoperationen:                                   ---
   procedure FIND(KENNZ: in STRING_K; GES_INFO: out STRING_I) is                    ---
     AKT_SCHLUESSEL:SCHLUESSELWERT:=PRIMAERSCHLUES-
SEL(KENNZ); ---
   begin                                                                            ---
     SUCHE_IN_BAUM(AKT_SCHLUESSEL,                                                  ---
                       ERFOLG=>ENTH,ENDKNOTEN=>GEF_KN);                             ---
     if not ENTH then                                                               ---
       raise THERE_IS_NO_ENTRY;                                                     ---
     else                                                                           ---
       GES_INFO: = LINKSBUENDIG (GEF_KN.INFO);---Fkt. sei geeignet def.
---
     end if;                                                                        ---
   end;                                                                             ---
   ...                                                                          ---
begin --- Anweisungsteil von AUSKUNFTEI fuer Initialisierung                        ---
   ...                                                                          ---
end body AUSKUNFTEI;
```

- - Zusammenfassung ado- -Modul:
  - Schnittstelle: Zugriffsoperationen
    - Veränderungsop.
    - Abfrageop. Systematik s.u.
  - Rumpf: Realisierung Abstr.
    - Realisierung d. Datenstr. voneinand. schritt
    - Realisierung d. Zugriffsop. abhängig

- - softwaretechn. Bedeutung der Datenabstraktion:
  Lokalisierung von Gefahren, hier Zeiger
  - aliasing
  - inaccessable objects
  - dangling references

- - Vorbereitung Adaptabilität
  Beispiel: Mißachtung der Datenabstraktion



- softwaretechn. Bedeutung der DA: Adaptabilität
  - lose Kopplung: auf das log. Notwendige reduziert
  - Änderung der Realisierung bleibt lokal:
    - modullokal oder architekturlokal
  - Änderungen an Datenstrukturen treten häufig auf
  - bei Mißachtung der DA treten weitreichende
    - Architekturänderungen auf

- nochmal zur Adaptabilität: versch. Realisierungsmögl.
  - Laufzeitkeller
  - statisch adressierter Speicherbereich
  - dabei seq., einfach oder mehrfach verkettete Liste,
    - bin. Baum etc.
  - permanente Speicherung, etwa B- - oder B*- -Baum
  - auf Knoten im Netz über remote procedure call

- softwaretechn. Bedeutung der DA: Vereinheitlichung
  - keine Unterscheidung zwischen Programmen und
    - Daten
  - jedes komplexe Datum erscheint als Modul
    - (ado oder adt) in der Architektur
  - Architekturdokumente enthalten die gesamte
    - Struktur des Programmsystems

- - Schnittstellensytematik v. Datenabstraktions- - (hier ado)- -Modulen
  - Veränderungsoperationen
  - Abfragen
  - Sicherheitsabfragen
  - Ausnahmen

	(a)	(b)
Stelle der Anwen- - dung der Zugriffs- - operation	**if** IS_SPACE **and** **not** IS_EL_OF(...) **then** STORE (...,...) **end if** ;	STORE (...,...);
Ausführung der Zugriffsoperation	regulärer Ablauf	ggf. Ausnahme- - erweckung von MEMORY_FULL oder ALREADY_THERE

- - Unterscheidung log. Abfragen - - Sicherheitsabfragen schwierig
  - Beisp. IS_EL_OF
  - Sicherheitsabfragen in Abfrageop. enthalten
  - Ausnahme für Sicherheitsabfrage

- - Alternative zu Sicherheitsabfragen: Return- -Parameter

- - - veränderte Schnittstelle:

```
procedure STORE (KENNZ: in STRING_K; GING_GUT: out BOOLEAN;
                 INFO: in STRING_I);
```

- - - Stelle der Anwendung:

```
STORE (AKT_KENNZ, IN_ORDNUNG, AKT_INFO);
if IN_ORDNUNG then
  - - - weiter
  ...
else
  - - - tue etwas anderes
  ...
end if;
```

unsicher: wenn vergessen wurde, den Return- -Parameter abzufragen, geht etwas schief, ohne daß dies jemand merkt

- - Beispiel Karten- -Kästen- -Beispiel
  DA vergessen:
    Kasten- -Namensliste <- - genauer
    Karten- -Namensliste

```
abstract data object module KK_Namensliste is - -- -*******************- -- -
   - -- -Dient der Handhabung einer Namensliste (abg. NL) von                - -- -
   - -- -Karteikaesten (abg. KK). Die Semantik ist die folgende: ...         - -- -
   procedure oeffne_KK_NL;                                                  - -- -
   procedure schliesse_KK_NL;                                               - -- -
   function ist_KK_NL_leer return BOOLEAN;                                  - -- -
   function ist_noch_Platz_in_KK_NL return BOOLEAN;                         - -- -
   function ist_KK_Name_vorh (Name: in KN_Typ) return BOOLEAN;              - -- -
   procedure einfuege_KK_Name (Name: in KN_Typ);                            - -- -
   procedure loesche_KK_Name (Name: in KN_Typ);                             - -- -
   procedure positioniere_Anfang_KK_NL;                                     - -- -
   procedure gib_akt_KK_Namen_aus (Name: out KN_Typ);                                  - -
- -
   procedure naechster_KK_Name;                                             - -- -
   procedure vorausgeh_KK_Name;                                             - -- -
   function ist_KK_Name_Nachf_vorh return BOOLEAN;                          - -- -
   function ist_KK_Name_Vorg_vorh return BOOLEAN;                           - -- -
   KK_NL_leer, kein_Platz_fuer_KK_NL, KK_Name_ex_nicht,                     - -- -
   KK_Name_Nachf_ex_nicht, KK_Name_Vorg_ex_nicht: exception                 - -- -
end                                                                                KK_Namensliste;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -
```

  Gruppierung der Operationen
    Operationen auf der gesamten Liste
    Handhabung einzelner Elemente
    Bewegung auf der Liste
    Ausnahme in d. Reihenfolge der Sicherheitsabfragen

-- bisher Kollektionsanwendungen
weitere DA--Anwendung: komplex aufgebauter Eintrag

```
abstract data object module KARTE is ---*************************----
   ---Die Parametertypen KEY_T, NAME_T etc. muessen bekannt sein.                    --
--
   procedure initialisiere_Karteninhalt;                                            ----
   procedure loesche_Karteninhalt;                                                  ----
   procedure eintrage_Schluessel (KEY: in KEY_T);                                   ----
   function lies_Schluessel return KEY_T;                                           ----
   procedure eintrage_Name (NAME: in NAME_T);                                       ----
   function lies_Name return NAME_T;                                                ----
   ...                                                                              ----
   ---Je eine Schreib-- und eine Leseoperation fuer jede                            ----
   ---Komponente des kompliziert aufgebauten Eintrags                               ----
   function ist_konsistent (KEY: in KEY_T) return FALL_T;                           ----
end                                                                                    KARTE;
----------------------------------------------------------------------------------------------------
----------          ----
```

-- Charakterisierung Eintragsanwendung
Konsistenzprüfungsoperationen vom Anwender
(”männlich” inkonsistent zu ”schwanger”)
Diskriminante männlich, weiblich vom Anwender:
Konsistenzprüfung intern
Sicherheitsabfragen/Ausnahmen hier selten

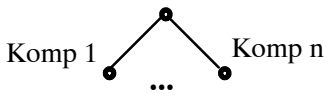
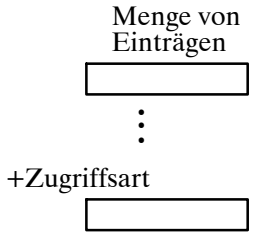
-- Eintrag hat viele Komponenten:
breite Schnittstelle
eine Lese-- und eine Schreiboperation:

```
abstract data object module COMPLEX_RECORD is ----****************----
   procedure READ(COMP: in COMP_DENOTATION; INFO:                            --
--
                        out INFO_TYPE);                                  ----
   procedure STORE(COMP: in COMP_DENOTATION; INFO:                        ----
                        in INFO_TYPE);                                   ----
   ...                                                                   ----
end                                                            COMPLEX_RECORD;
------------------------------------------------------------------------     --
--
```

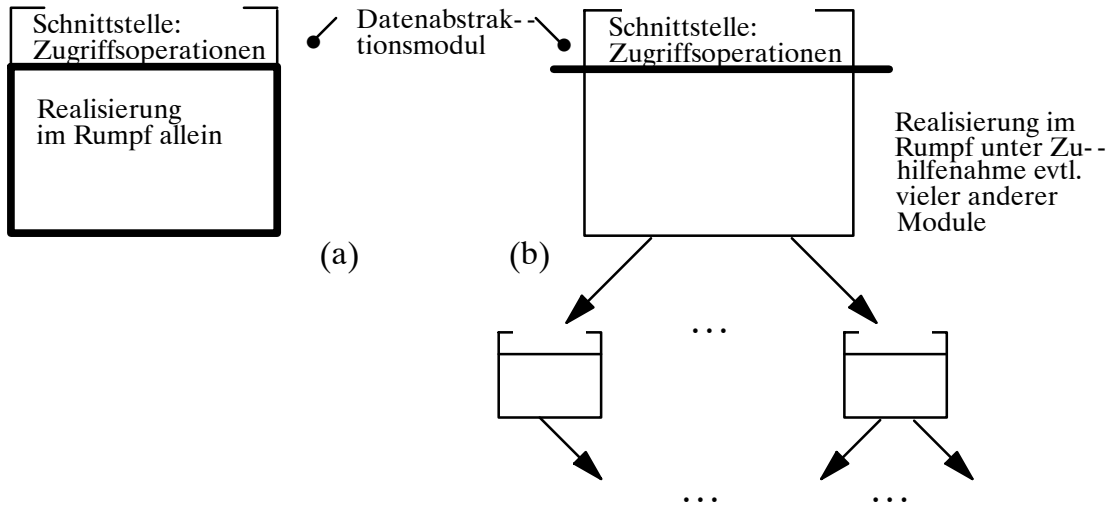
-- Diskussion:
Schreibaufwand für Schnittstelle kleiner
unspezifischere Festlegung an den Stellen der Ver--
wendung (Sicherheitsverlust)
unspezifischer Parametertyp (z.B. STRING oder
varianter Verbund (Sicherheitsverlust))
schwieriger für Implementator und für Verwender:
Konversionsproblematik

andere Lösungen:
Gruppierung der Schnittstelle nach log. Gesichts--
punkten und Aufteilung auf Module
je eine Lese-- und Schreibop. für Gruppe von
Daten (z.B. Adresse)

-- Datenabstraktion in beiden Anwendungsklassen

Anwendungs--klasse der Datenabstraktion	Abstraktion: logische Sicht	Realisierung: physische Sicht
komplexer Einzeleintrag	Gesamtobjekt  Lese-- und Schreibzugriff auf die einzelnen "logischen" Komponenten	Komponentenreihenfolge hintereinander, verstreut Realisierung der Reihen--folge durch sequentielle Ablage, über Indizes oder Zeiger verkettet Einzelkomponenten ver--dichtet, einzelne Kompo--nente muß erst berechnet werden o.ä. zusätzliche Komponenten für interne Zwecke, z.B. Statistik
Kollektion von Einträgen	Ablegen, Auffinden, Ändern und Löschen von Elementen der Kollektion  Festlegung der Zugriffsart (FIFO, LIFO, RANDOM etc.)	Wie wird die Menge/geordnete Menge realisiert: sequentiell, über Indizes oder Zeiger verkettet, char. Speicherung etc. Wo wird die Menge abgelegt: im statischen Speicherbereich, im Laufzeitkeller, auf der Halde, auf dem Sekundärspeicher, auf einen Knoten im Netz o.ä. Wie wird die Zugriffsart realisiert: über Berechnung, Einrichtung von Zugriffspfaden Zusätzliche Komponenten für interne Zwecke, etc.

-- Datenabstraktion und Information Hiding



4.4 Datentypmodule und sonstige Module

- - braucht mehr als ein abstraktes Datenobjekt
  Quelltextvervielfältigung oder
  Schablonenmechanismus: abstr. Datentypmodule

- - Unterschied abstrakter (opaker, verkaps.)
  und transparenter (offener) Datentyp

- - - -(a) Verwendung transparenter oder normaler Datentypen und Datenobjekte:

```
type SPACE_T is array (1..N) of ITEM_TYPE; - - - - (1)
SPACE_1 : SPACE_T; - - - - (2)
...
if ... then ... ; SPACE_1(INDEX) := X else ... end if; - - - - (3)
```

- - - -(b) Verwendung opaker oder abstrakter Datentypen und Datenobjekte:

```
STACK_1: STACK_TYPE; - - - - (4)
...
if not IS_EMPTY (STACK_1) then - - - - (5)
    POP (STACK_1);
else ...
end if;
```

-- Keller als abstrakter Datentypmodul

```
                                                    Programmieren--im--Großen--Teil
abstract data type module ITEM_STACK_STENCIL is ***************------
    ...                                                                  ------
    type  ITEM_STACK_TYPE is private;                                    ------
    procedure INITIALIZE (ST: in out  ITEM_STACK_TYPE);                  ------
    procedure PUSH (EL: in ITEM_TYPE; ST: in out ITEM_STACK_TYPE);       ------
    procedure POP (ST: in out ITEM_STACK_TYPE);                          ------
    function READ_TOP (ST: in ITEM_STACK_TYPE) return ITEM_TYPE;         ------
    function IS_EMPTY (ST: in ITEM_STACK_TYPE) return BOOLEAN;           ------
    function IS_FULL (ST: in ITEM_STACK_TYPE) return BOOLEAN;            ------
    ST_UNDERFLOW, ST_OVERFLOW, ST_NOT_INITIALIZED: exception;            ------
    ------ Semantikbeschreibung: ...                                     ------
end                                                  ITEM_STACK_STENCIL;
-------------------------------------------------------------------------------
--
module                body                ITEM_STACK_STENCIL                is
-------------------------------------------------------------------------------
    ------Festlegung der Repraesentation des Datentyps,                  ------
    ------(in Ada im physischen Teil der Schnittstelle):                ------
    SIZE: constant INTEGER := 100;                                       ------
    type  SPACE_T is array (1..SIZE) of ITEM_TYPE;                       ------
    type ITEM_STACK_TYPE is                                              ------
       record                                                            ------
          SPACE: SPACE_T;                                                ------
          INDEX: INTEGER range 0..SIZE := 0;                             ------
       end record;                                                       ------
    ...                                                                  ------
    ------ Realisierung der Zugriffsoperationen (abhaengig von der       ------
    ------ gewaehlten Repraesentation des Datentyps)          :
------
       ...                                                               ------
end ITEM_STACK_STENCIL; ------**********************************
------
```

Programmieren--im--Kleinen--Teil

```
_____________________________
...
    ------Anzeige, daß Modul ITEM_STACK_STENCIL in einem anderen Modul
    ------verwendet werden soll (Import). Dies wird spaeter behandelt.
                                                    Programmieren--im--Großen--Teil
_____________________________
...
    ------Verwendung im Rumpf dieses anderen Moduls:
       ST_1: ITEM_STACK_TYPE;                                  Programmieren--im--
       EI_1 : ITEM_TYPE;                                          Kleinen--Teil
       if not IS_FULL (ST_1) then PUSH (EL_1, ST_1) else ... end if;
```

-- Aufgabe eines adt--Moduls:
    Deklaration eines abstrakten Datentyps:
        Typbezeichner und Zugriffsoperation
    Realisierung des abstrakten Datentyps:
        Deklaration der Datenstrukturen,
        Realisierung der Operationen

-- ado--Modul <--> adt--Modul
    Gemeinsamkeiten s. letzter Absatz
    Unterschied: ado--Modul ist ein ado
                 adt--Modul Schablone für ados
    ado von ado--Modul ist (als Gedächtnis) in der
        Architektur verankert
    Erzeugen von ados aus adt--Modul im Rumpf
        eines Moduls als PiK--Einheit
    Verwendung eines adt--Moduls
        zur Erzeugung von ados
        zur Veränderung/zum Abfragen eines ander--
            weitig erzeugten ados

- bisherige Datentypmodule nicht verwendbar, wenn
  zugrundeliegende Programmiersprache keine Typ--deklarationen kennt (FORTRAN, Cobol, Assembler, etc.)
  Anzahl der erzeugten ados zur Programm--erstellungszeit nicht bestimmbar

- andere Art von adt--Modul
  mit Erzeugungsoperation (und Löschop.)
  evtl. mit Initialisierungsop. zusammen (falls ado nicht mehrfach verwendet werden soll)
  evtl. Dekl. eines Typbezeichners für Bezeichner auf ados an der Schnittstelle
  Verwendung solcher adt--Module im Anweisungs--teil des Rumpfs des verwendenden Moduls

- - Beispiel adt- -Modul mit Erzeugungsoperation

```
abstract data type module ITEM_STACK_STENCIL is - -- -**************
- -- -
    ...                                                              - -- -
    type STACK_DENOTER_TYPE is private;                              - -- -
    procedure CREATE_AND_INIT(ST: out STACK_DENOTER_TYPE);           - -- -
    procedure DELETE(ST: in STACK_DENOTER_TYPE);                     - -- -
    procedure PUSH(EL: in ITEM_TYPE; ST: in STACK_DENOTER_TYPE);     - -- -
    procedure POP(ST: in STACK_DENOTER_TYPE);                        - -- -
    function READ_TOP(ST: in STACK_DENOTER_TYPE)                     - -- -
        return ITEM_TYPE;                                            - -- -
    function ST_IS_EMPTY(ST: in STACK_DENOTER_TYPE)                  - -- -
        return BOOLEAN;                                              - -- -
    function ST_IS_FULL(ST: in STACK_DENOTER_TYPE) return BOOLEAN; - -
- -
    function ANOTHER_ST_POSSIBLE return BOOLEAN;                     - -- -
    ST_UNDERFLOW, ST_OVERFLOW, NO_ST_AVAILABLE, ST_IS_NIL:           - -- -
        exception;                                                   - -- -
    - -- - Semantikbeschreibung:
- -- -
    - -- - ...
- -- -
end                                                  ITEM_STACK_STENCIL;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -
module              body              ITEM_STACK_STENCIL              is
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
    - -- - Beschreibung des Bezeichnertyps:
- -- -
    type STACK_DENOTER_TYPE is INTEGER;                              - -- -
    ...                                                              - -- -
    - -- - Realisierung der Zugriffsoperationen:                     - -- -
    ...                                                              - -- -
end ITEM_STACK_STENCIL; - -- -**********************************
- -- -

_____________________________
...
- -- - fuer einen anderen Modul:
- -- - Importteil (Programmieren- -im- -Gossen- -Teil), kommt spaeter.

_____________________________
...
- -- - Verwendung im Rumpf dieses anderen Moduls (Programmieren im Kleinen):
   ACT_ST: STACK_DENOTER_TYPE;    - -- -ACT_ST kann bel. Keller bezeichnen
   EL1: ITEM_TYPE;
   ...
   if ANOTHER_ST_POSSIBLE then
        CREATE_AND_INIT (ACT_ST);    - -- - im Anweisungsteil erzeugt, Wert von
```

-- Muß Verwender eines adt--Moduls wissen, ob dieser Typbezeichner oder Erzeugungsop. exportiert?
Ja, unterschiedl. Verwendung beim Schaffen von ados

Variablensemantik	<-->	Zeigersemantik
Obj. auf Laufzeitkeller		Obj. auf "Halde"
werden v. d. Sprachimpl.		Verwender erzeugt
angelegt, gelöscht		und löscht sie
Zuweisung: Kopie		"eigenständ." Objekte
		Zuweisung: weitere Bez.

-- Semantik der unterschiedl. Verwendung an der Schnittstelle
- aus Syntax ablesbar
- Kommentar
- Realisierung anders

-- Aussagen über DA und ST des letzten Abschnitts zu wiederholen
- Gefahr lokalisieren
- lose Bindung von Modulen durch DA
- große Spannweite unterschiedl. Realisierungen
- Adaptabilität
- Anwendungen: Einträge, Kollektionen
- Schnittstellengestaltung
- Abkürzungen (Return--Code, allg. Lese-- und Schreibop. bei Eintragsanwend.) unsicherer
- Information Hiding zwischen Schnittstelle und Realisierung

- - Semantikspez. von adt- -Modulen durch alg. Spezifikation

```
spec ITEM_STACK_TYPE
        - -- -gewisse Importe von anderen Spezifikationen sind noetig, z.B. von ITEM
        sorts ITEM_STACK_TYPE
        operations
                        - -- -Festlegung der Exportschnittstelle:
                        NEW:  ——> ITEM_STACK_TYPE
                        PUSH: ITEM x ITEM_STACK_TYPE  ⟼——> ITEM_STACK_TYPE
                        POP: ITEM_STACK_TYPE  ⟼——> ITEM_STACK_TYPE
                        READ_TOP: ITEM_STACK_TYPE  ⟼——> ITEM
                        IS_EMPTY:  ITEM_STACK_TYPE  ——> BOOLEAN
                        IS_FULL:  ITEM_STACK_TYPE  ——> BOOLEAN
        preconditions
                        - -- -Vorbedingungen für die Operationen, damit die algebraische
                        - -- -Spezifikation gueltig ist:
                        pre POP(ST: ITEM_STACK_TYPE) = not IS_EMPTY(ST)
                        pre READ_TOP(ST:  ITEM_STACK_TYPE) = not IS_EMPTY(ST)
                        pre PUSH(EL: ITEM; ST: ITEM_STACK_TYPE) = not IS_FULL(ST)
        equations
                        for all EL: ITEM, ST: ITEM_STACK_TYPE:
                        IS_EMPTY(NEW())
                        not IS_EMPTY(PUSH(EL, ST))
                        not  IS_FULL(NEW())
                        not IS_FULL(POP(ST))
                        READ_TOP(PUSH(EL, ST)) = EL
                        POP(PUSH(EL, ST) = ST
end spec ITEM_STACK_TYPE
```

Signatur: Sorts- -, Operations- -Teil
partielle Funktionen, totale Funktionen
Unterschied Funktion PUSH zu Zugriffsop. PUSH
NEW entspricht Erzeugungsoperation
Vorbedingungen, unter denen alg. Spez. gilt
alg. Gleichungen (Axiome):
sem. Zusammenhang der Zugriffsoperationen

-- formale Semantikfestlegung
für funkt. Module wichtige Ergänzung zu
DA--Module hier vorgest. Sprachen

-- weitere Modularten
Zusammenf. v. transp. Typen: Typkollektionsmodule
nach Anwendungsbereichen (z.B. Physik)
nach versch. log. Verwendung in Programmsyst.
Anschluß an Programmiersprache
Op. f. Typumwandlungen, --Konversionen
solche Typen zu einfach, somit keine Architektureinheit
ggf. Hinzunahme von Konstanten

spielen für Architekturüberlegungen keine große Rolle
machen Architekturdiagramm unübersichtlich

Sehen keine Module mit Objekten an der Schnitt-stelle vor

Zwischenformen funktionaler und DA--Module
Zufallszahlengenerator, Tastatur, Maus?

-- bisherige Anwendungsklassen versch. Modularten

Modulart	funktionale Module	abstr. Datenobjekt-- module	abstr. Datentypmodule
Verwen-- dungs-- zweck	● Steuer-- oder Koordinationsaufgabe ● Transformations-- aufgabe ● komplizierte Auswertung Hilfsdienste auf versch. Datenstruk-- turen (funkt. Zwischenschicht zwischen Datenabstr. Schichten)	einzelner, komplex aufgebauter Eintrag ● einzelne Kollektion mit best. Zugriffs-- operationen	● Handhabung von komplex aufgebau-- ten Einzeleinträgen ● Handhabung von Kollektionen mit best. Zugriffsopera-- tionen

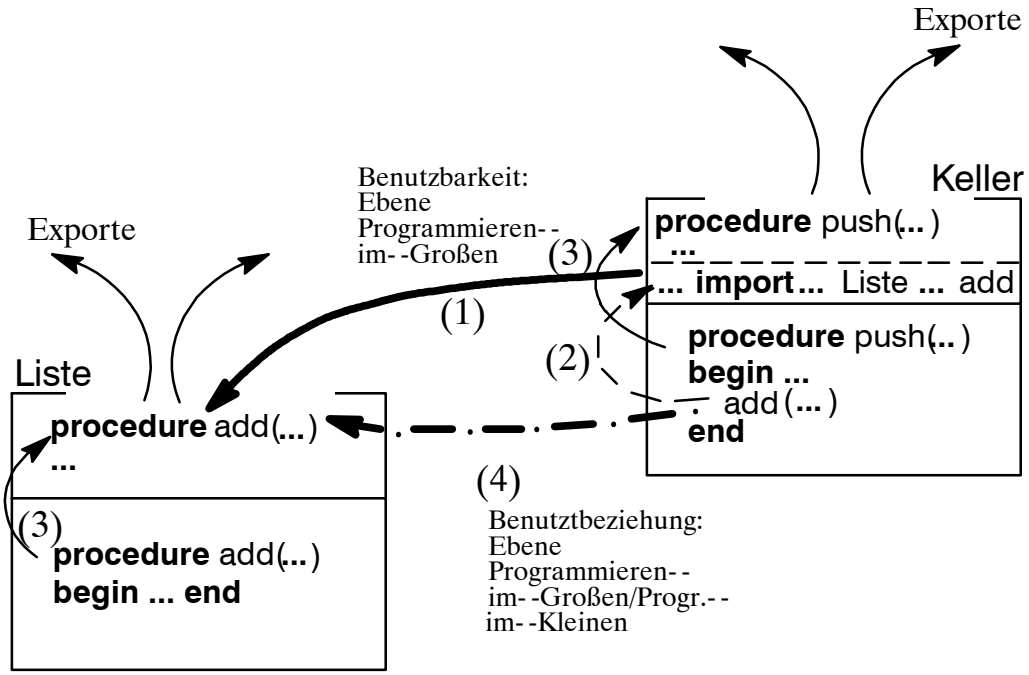
4.5 Beziehungen zwischen Modulen

- - Modulbeziehungen: z.B. Importschnittstelle, gibt auch weitere verschiedene Arten für unterschiedliche Beziehungen betrachten logische Beziehungen und nicht z.B. Beziehungen auf Programmiersprachen- -Einheiten, wie bei der getrennten Übersetzung

- - Auf welcher Ebene Modulbeziehungen für Zusammenhang von Modulen
  Benutzbarkeits- - oder Importebene
  in der Spezifikation
  statische Benutzt- - oder Benutzungsebene
  statisch im Rumpf
  dynamische Benutzung:
  dynamisch im Rumpf

- - Was kann benutzbar gemacht werden?
  Hängt von der Art des exportierenden Moduls ab:
  Funktionsmodul: Nutzung der Funktionen
  ado- -Modul: Nutzung der Zugriffsoperation
  adt- -Modul: Nutzung Typ/ Erzeugungsoperation
  und Zugriffsoperationen

- - Benutzbarkeitsebene für das Programmieren im Großen (Zusammenspiel mit dem Programmieren im Kleinen)



Benutzbarkeitsebene einzige auf PiG- -Niveau

Benutztebene allein auf PiK- -Niveau

Redundanz durch Zusammenspiel PiG- -PiK

Arbeitsteilung

Sicherheit

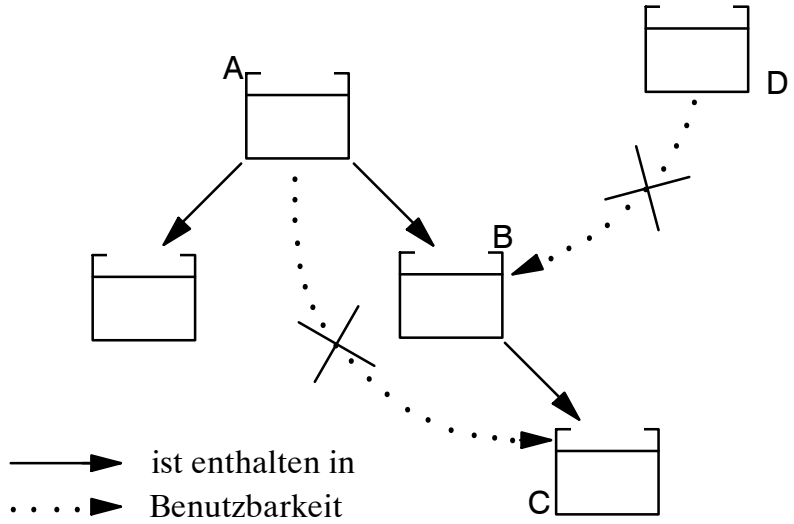
4.6 Die lokale Benutzbarkeit

- - erste Art von Benutzbarkeit
  weitere kommen

- - Unterscheidung von der ”Logik” der Verwendung
  auf Architekturebene
  nicht durch Programmiersprache, auf die wir abbilden
  Konzepte haben aber Ursprung in Programmiersprachen:
  hier Schachtelungsprinzip/ Blockstruktur,
  Gültigkeit, Sichtbarkeit

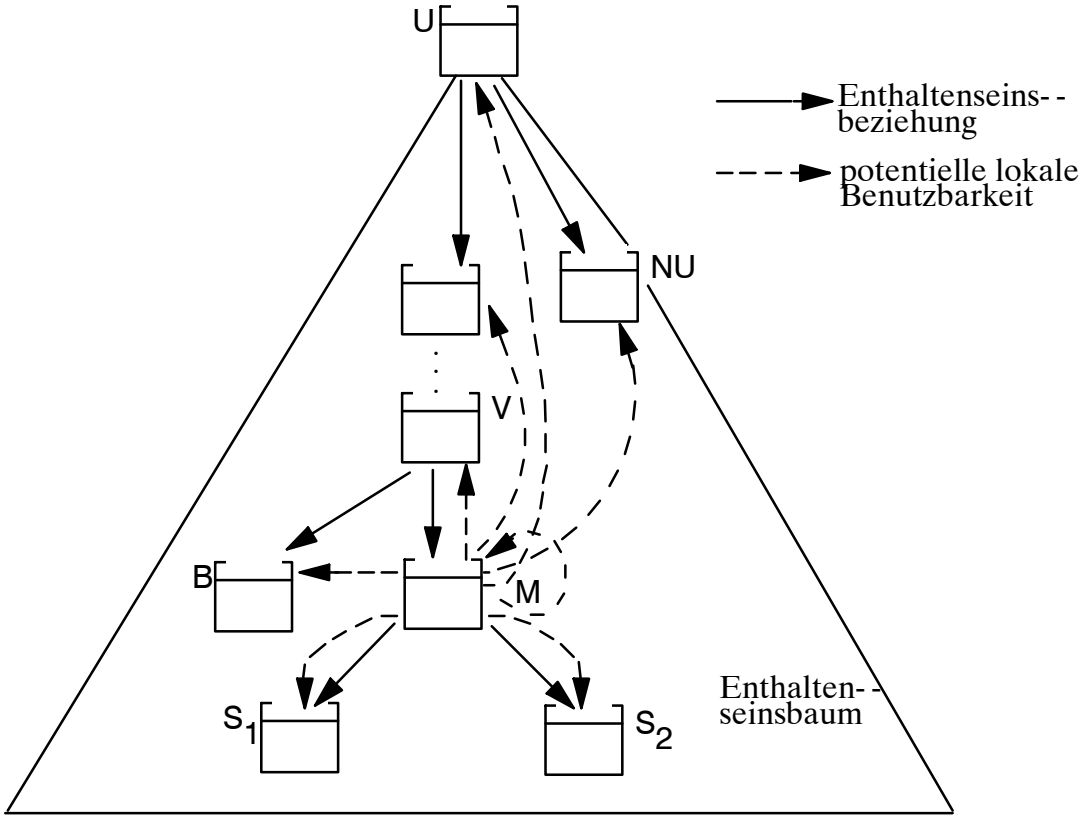
- - vorläufige Definition der lokalen Benutzbarkeit:
  ”Ein Modul ist in einem anderen enthalten und damit
  prinzipiell nur in einem lokalen Kontext benutzbar.
  Es muß explizit festgelegt werden, wo er benutzbar
  sein soll.”

- - Was möchte man modellieren?

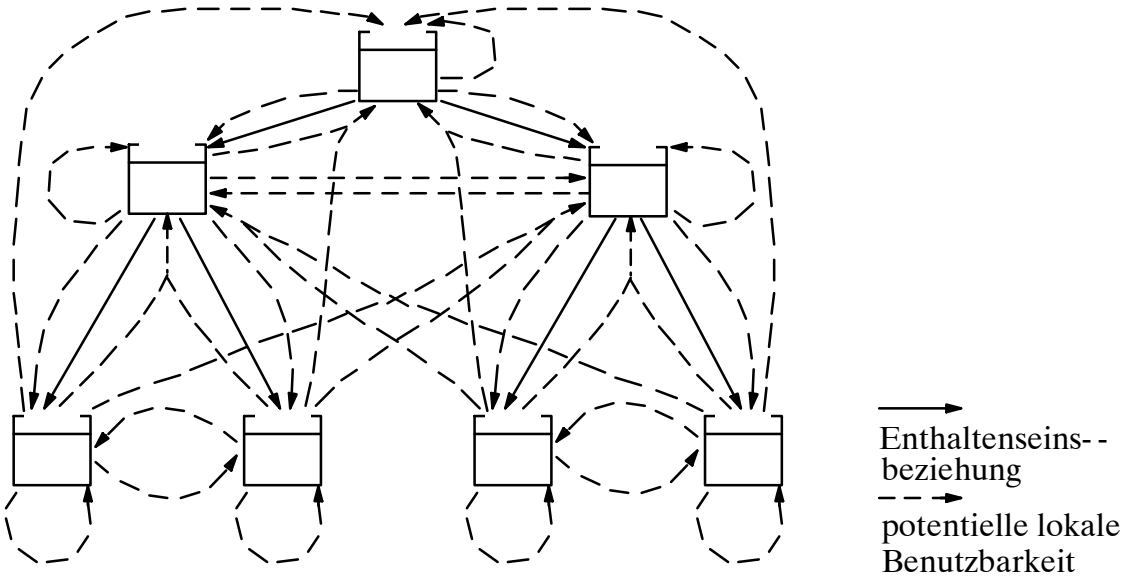


- - Enthaltenseinsbeziehung als logische Beziehung
  auf Architekturebene
  B ist in A enthalten:
  B ist zur Realisierung von A nötig
  Realisierung von A u.a. dürfen von B Gebrauch machen
  z.B. darf B im Rumpf von A statisch benutzt werden
  B ist nur an einer "Stelle" von Wichtigkeit,
  lokale Bedeutung
  drücken Enthaltenseinsbeziehung aus
  durch Pfeil in Graphik
  nicht durch Ineinanderschachtelung i. Modulbeschreibung

- - Benutzbarkeit in bestimmten Kontext:
  inverse Beziehung: potentielle lokale Benutzbarkeit

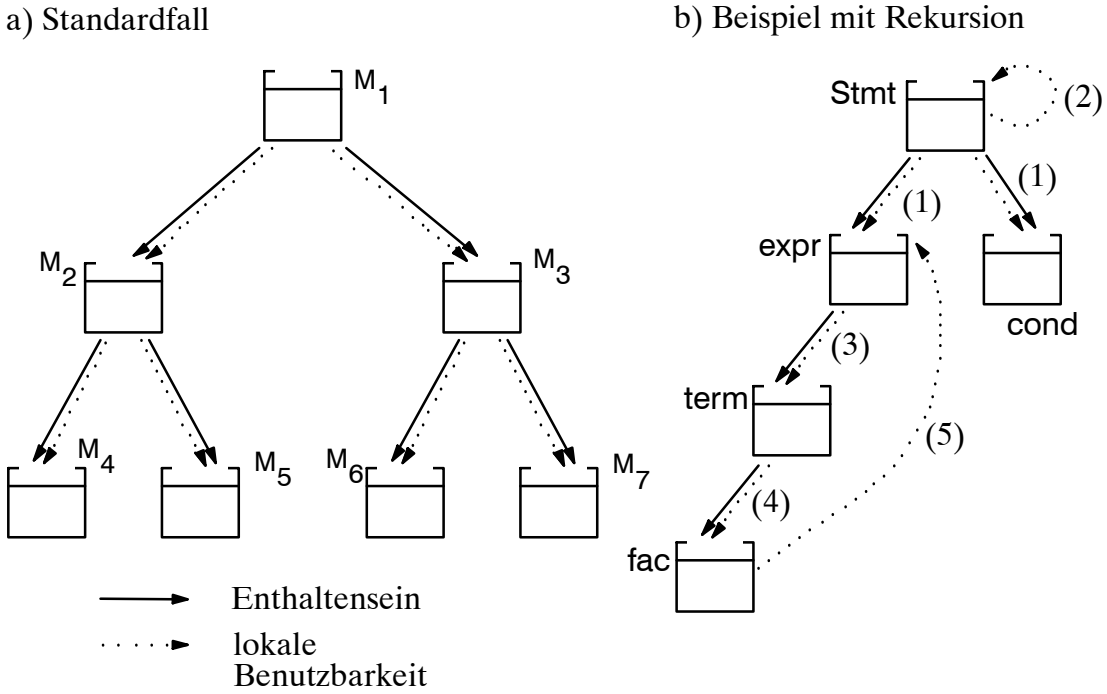


- - Beispiel mit 7 Modulen: Chaos!



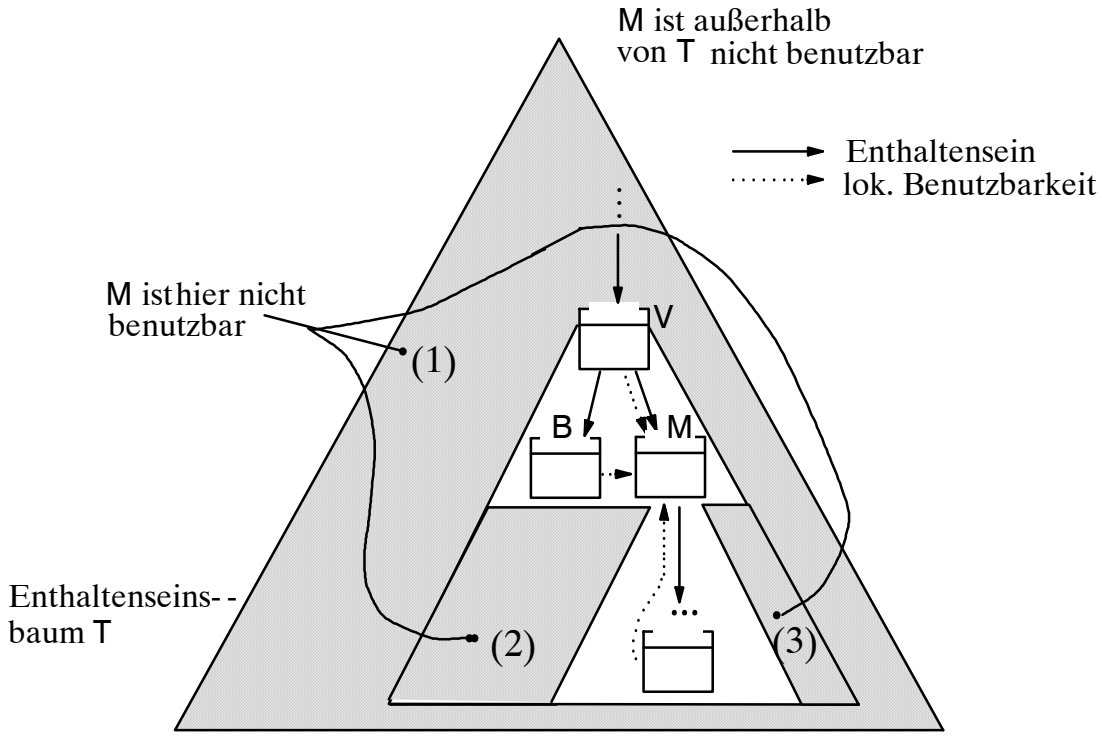
- - explizite Festlegung der Benutzbarkeit durch
  lokale Benutzbarkeitsbeziehung als spezielle
  Importbeziehung
  verträglich mit potentieller Benutzbarkeit (und
  damit auch mit Enthaltenseinsbeziehung)
  Graphik: Kante bestimmte Art
  textuelle Detailfestlegung Modul: lokale Importklausel

- - Standardsituationen



-- Was haben wir für die Architekturmodellierung erreicht?
lokale Bedeutung ausgedrückt durch Enthaltenseinbeziehung
dadurch Einschränkung der prinzipiellen Benutzbarkeit
explizite Festlegung im Rahmen dieser Möglichkeiten
damit festgelegt
welche Module M (lokal) verwenden darf
und welche M (lokal) verwenden dürfen

-- lokale Benutzbarkeit und Sicherheit



(1) -- (4): keine Benutzung, nicht einmal Benutzbarkeit

-- Modul mit darunterhängendem Enthaltenseinsbaum:
Teilsystem

-- lokale Benutzbarkeit <--> Lokalitätsprinzip blockstrukt. Sprachen
keinen Gültigkeitsbereich, sondern explizite Festlegung
gleicher (Modul--)Name in Enthaltenseinsbaum nicht vorgesehen
Schnittstellenressourcen sind stets verschieden benannt
keine Abbildung auf Ineinanderschachtelung

4.7 Die allgemeine Benutzbarkeit

- - zweite Art von Benutzbarkeitsbeziehung
  Ursprung Programmiersprachenkonzept: Importklausel
  logische Beziehung für Architekturmodellierung

- - Warum genügt die lokale Benutzbarkeit allein nicht?
  (prinzipiell geht es: Algol, Pascal etc.)

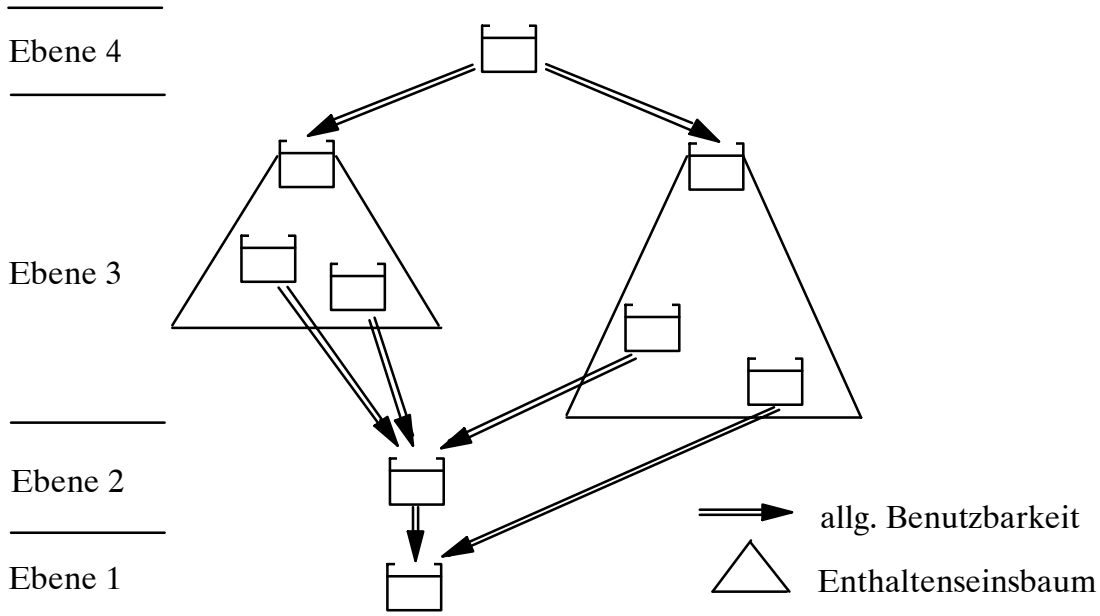
Wohin hängen?
Nicht verständlich
Kann auch an anderer Stelle benutzbar gemacht werden
Kann auch in anderem Enthaltenseinsbaum benötigt werden

- - Lösung

in Graphik andere Kanten
in Textbeschreibung andere Importklausel

- - vorläufige Definition allgemeiner Benutzbarkeit:
”Ein Modul (später auch Teilsystem) ist ein allgemeines Hilfsmittel in einem Softwaresystem, das zur Realisierung anderer Module benötigt wird. Es ist überall dort benutzbar, wo dies explizit festgelegt wurde.”

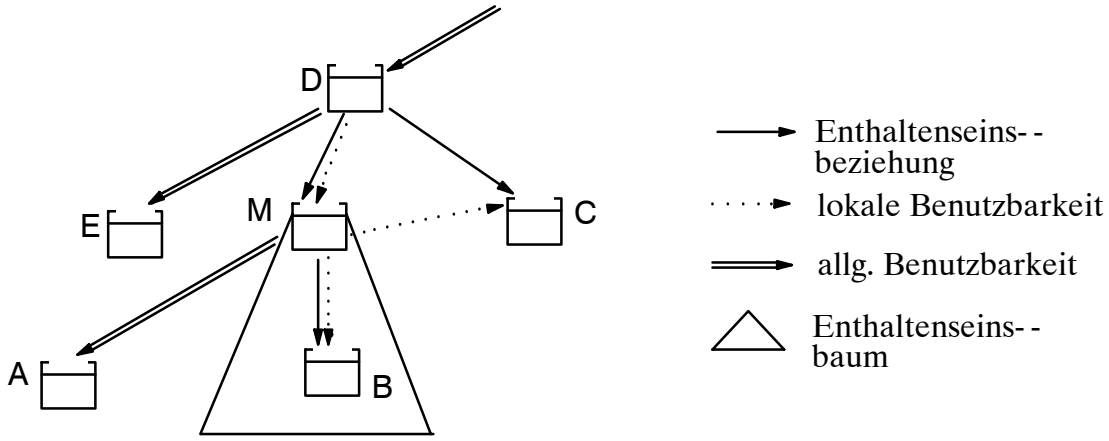
- - verwendeter Modul logisch tiefer als Verwender,
allgemeine Benutzbarkeit ist Hierarchie:
Schichteneinteilung
keine strenge Hierarche



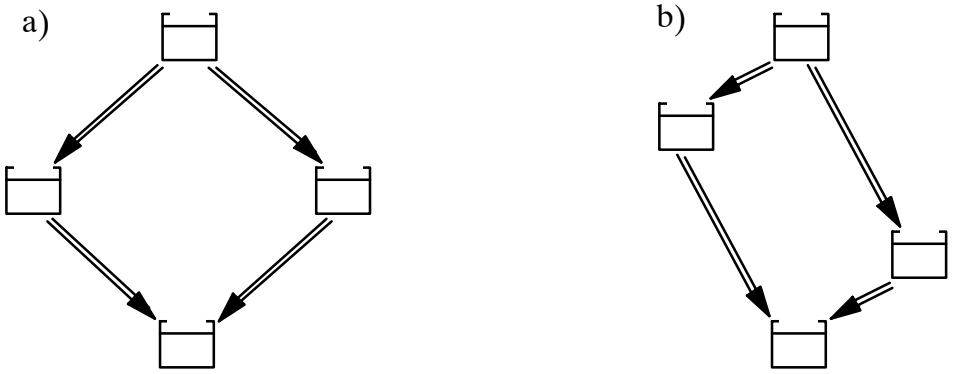
- - allgemein verwendbarer Modul
mit entsprechender Absicht entworfen
oder später entsprechend verallgemeinert
oft eher Datenabstraktions- -Module als funkt. Module

- - Einhängen eines Moduls (Teilsystems)
braucht Realisierung nicht zu kennen
Information Hiding auf Architekturebene
muß wissen, ob allgemeine oder lokale Verwendung

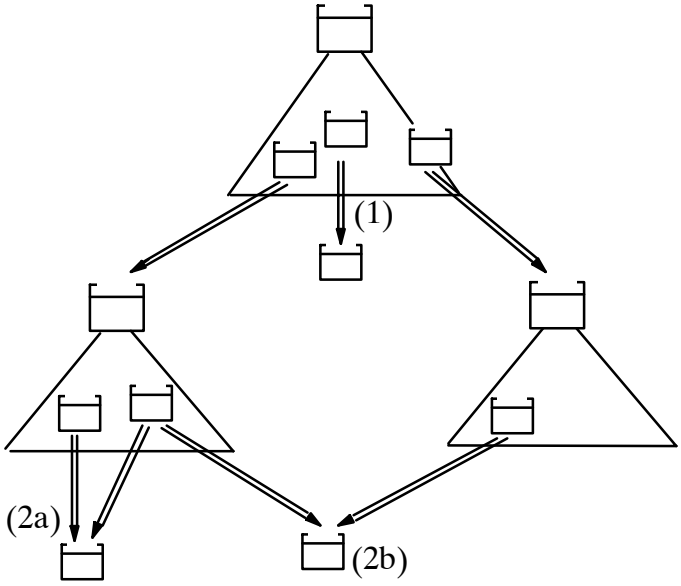
- - Zusammenspiel Modulbeziehungen in Bezug auf Exporte/ Importe



- - bisher kennengelernte Modulbeziehungen liefern nicht notwendigerweise logische Schichtung



- - verschiedene Arten allgemeiner Benutzbarkeit
  einzige Stelle
  innerhalb eines Teilsystems mehrfach
  verschiedene Teilsysteme
  in verschiedenen Softwaresystemen dazugebunden
  Prozeß für verschiedene Softwaresysteme

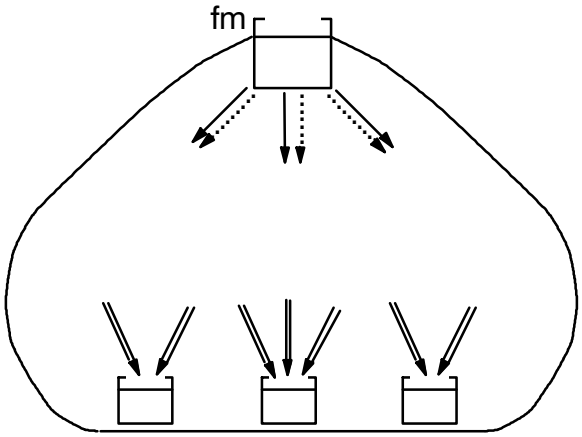


- - Genügt allgemeine Benutzbarkeit allein?
  (prinzipiell ja: FORTRAN, Assembler etc.)
  verzichtet auf
    Baustein hat nur Bedeutung für spez. Situation
    braucht ihn nicht zu kennen, wenn man Teilsystem
      betrachtet, in das er eingebettet ist — Inform.
    verboten ihn von außen zu benutzen — Hiding
    allg. verfügbare Hilfsmittel nicht mit Kleinkram
      verschmutzt

- - lokale und allgemeine Benutzbarkeit
  und zugrundeliegende Strukturbeziehungen

- - Modulbeziehungen/ Modularten und Entwurfstrategien
  Modulbeziehungen
    Top- -down: Enthaltenseinsbeziehung/ lok. Benutzbarkeit
      Gefahr allg. Bausteine nicht zu erkennen: gravierende Architekturveränderungen später
    Bottom- -up: allg. Benutzbarkeit
  Modularten:
    Top- -down: funktionale Module
      Gefahr Datenabstraktion zu übersehen
    Bottom- -up: Datenabstraktions- -Module

- - Modulart/ Modulbeziehungen und Architektur- -Schichten
  Modularten
    funktionale Module: oben
    Datenabstraktions- -Module: unten
  Modulbeziehungen
    Lokalitätsprinzip: oben
    allg. Benutzbarkeit: unten



verstärktes Auftreten von Funktionsmodulen und lok. Benutzbarkeit

verstärktes Auftreten von Datenabstraktionsmodulen und allg. Benutzbarkeit

4.8 Zur Darstellung von Architekturen

- - Architekturdiagramme: zur Übersicht
  textuelle Beschreibung für Module: für alle Details
  Entwurfs- -Begründungspapier: zur Begründung u. Erklärung
  Spezifikation der Modulrümpfe: für Programmieren i. Kleinen
  und dynam. Zusammenspiel

- - Architekturdiagramm:
  Überblicksdokument
  zum Verstehen des Gesamtsystems, zum Diskutieren,
  später zur Diskussion allein, wenn die Modulspezifikationen
  verinnerlicht sind
  geübter Entwerfer: Entwurf mehr oder minder auf
  Architekturdiagrammebene, Details später
  große Architekturen (Progr. im Größten)
  ebenfalls Architekturdiagramm, das Teilsysteme enthält
  (siehe unten)

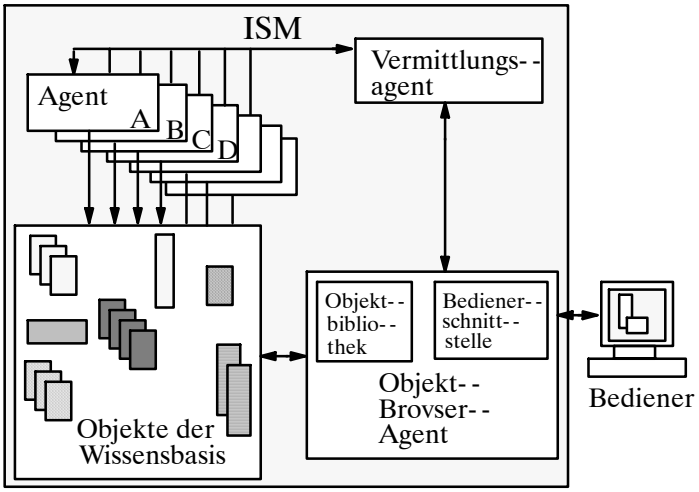
  insges. Quintessenz des fertigen Programmsystems

- - Architekturdiagramme
  Modulsymbole, Artkennzeichung für Module
  Kanten verschiedener Arten für Beziehungen
  Konsistenzbedingungen

=> Klasse geordneter, markierter Graphen in graphischer
  Repräsentation

-- anderes Verständnis von ”Architektur” in der Literatur

Kästen:
Komplexe Verbindungen:
haben etwas miteinander zu tun



Die ISM Architektur

-- Forderung an Gestalt von Architekturdiagrammen
einfach
übersichtlich versuchen zu präzisieren
augenfällig

-- zur Anzahl und Größe von Modulen

- - möglichst wenig Querbeziehungen (Kanten)
  richtige Modulanzahl, nicht 1
  viele DA- -Module
  wegen Kosten von Kanten

- - möglichst viele ”unabhängige” Teildiagramme (Teilsysteme)
  d.h. wenig Kanten zum Rest
  bei größerem Projekt zu Teilsystemen gezwungen
  Verständlichkeit der Gesamtarchitektur

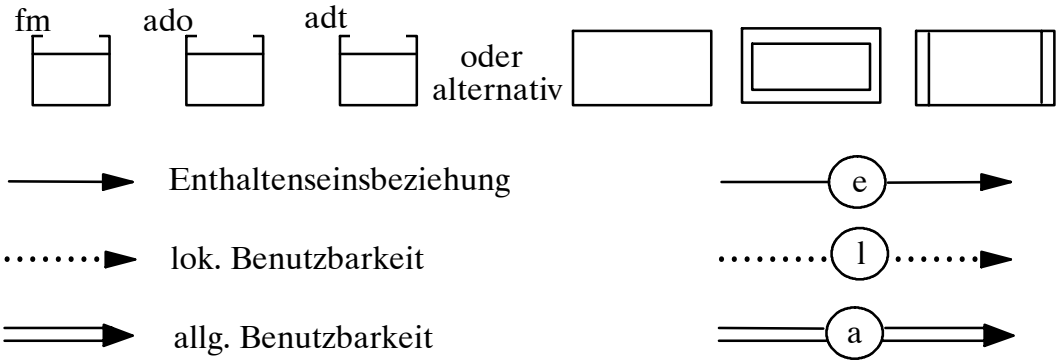
- - Fazit: vage Forderungen, es fehlt geeignetes
  Komplexitätsmaß

- - ”Methodikregeln” im Umgang mit Sprache

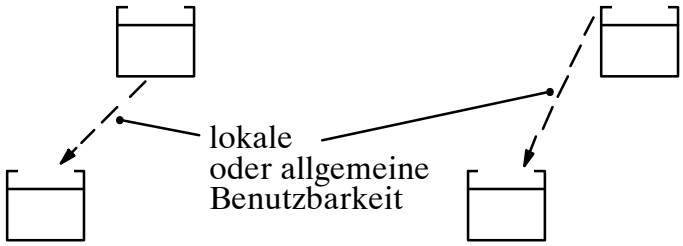
  - Enthaltenseinsbaum nicht zu tief, allg. Hilfsmittel herausziehen
  - Modul/Teilsysteme: nicht zu viele einlaufende Benutzbarkeitskanten, insb. nicht von sehr unterschiedlichen Niveaus aus; Zwischenniveau fehlt
  - nur eine einlaufende Kante: ist Modul wirklich ein allg. Baustein?
  - Schnittstelle nicht zu groß: Verteilung auf verschiedene Module/Teilsysteme

  etc.

- - Diagrammsprachelemente



- - Import für Realisierung oder Schnittstelle



- - Textnotation für Modul

module_text_notation ::= module_kind **module** *module*_name **is**
structure_clause
*module_globa*l_imports
export_interface
module_semantics
end *module*_name;
module body *module*_name **is**
*module_body*_imports
programming_in_the_small_part
end *module*_name;

module_kind ::= **functional** | **abstract data object** | **abstract data type**

structure_clause ::= **is contained in** *module*_name; | ...

imports ::= Z import_kind **import from** *module*_name **using** *interface_ressources*_name_list; X

import_kind ::= **local** | **general** | ...

export_interface ::= *Ada_proc_or_func_or_limited_private_type_source*_text

module_semantics ::= *semantic_description*_text

programming_in_the_small_part ::= *Ada_source*_text

comment ::= *natural_language*_text

name_list ::= *Ada*_identifier_list

4.9 Konsistenzbedingungen für Architekturen

- - Konsistenzbedingungen: kontextsensitive Syntax der
  vorgegebenen Sprachen und ihre Beziehungen
  zu anderen Sprachen

  etwas anderes als ”Methodikregeln”

- - Einteilung 1:
  unterscheiden zulässige und unzulässige Architekturen
  verweisen auf unvollständige Stellen
  stellen Warnungen dar

- - Einteilung 2:
  auf Architekturdiagrammebene allein
  zusätzlich Detailspezifikationen betrachten
  zusätzlich Modulrumpf betrachten

- - für Erklärung brauchen wir
  Erläuterung Benutzbarkeitsebene
  Erläuterung zur Gestalt von Modulen, insb. DA- -Modulen
  Erläuterung lokale, allgemeine Benutzbarkeit

- - Bedingungen auf Architekturebene

- - Regeln aus Zusammenspiel Export bzw. Export/Import

-- Übergang zum PiK
bei exportierendem oder importierendem Modul

4.10 Zusammenfassung

- - zwei Architekturbeschreibungssprachen eingeführt
  für Architekturdiagramme, Detailspezifikationen
  (zusätzlich Design Rationale, Minispecs)

- - zugrundeliegendes Modulkonzept
  Arten von Modulen: fm- -, ado- -, adt- -Module
  lokale Benutzbarkeit und Enthaltenseinsbeziehung,
  allg. Benutzbarbeit
  Konsistenzbedingungen

- - haben lediglich Denkrahmen geschaffen,
  Anwendung nichttrivial (Aufwand, intellekt. Anstrengung)

- - uns fehlt noch Vertrautheit im Umgang
  siehe weitere Kapitel: Teilarchitekturen, ganze Architekturen,
  Studium von Klassen von Anwendungssystemen nötig

5. ZUR VERTIEFUNG: TEILARCHITEKTUR-ÜBERLEGUNGEN UND MODULKONZEPT- - ERWEITERUNGEN

- - Modulkonzept anwenden
  In welchen Zusammenhängen treten bestimmte Modularten auf?
  Standard- -Situationen für Teilarchitekturen

- - sehen, daß wir weitere Konzepte brauchen
  Teilsysteme
  Generizität
  Objektorientierung

5.1 Datenabstraktion versus funktionale Abstraktion

-- Zuordnung Modul funktionale Abstr. oder Datenabstr.
Wann und wo treten Mißverständnisse auf?

-- Von welcher Art ist ein Modul
Gedächtnis --> ado
Schablone für Gedächtnisse --> adt
E/A--Verhalten --> fm

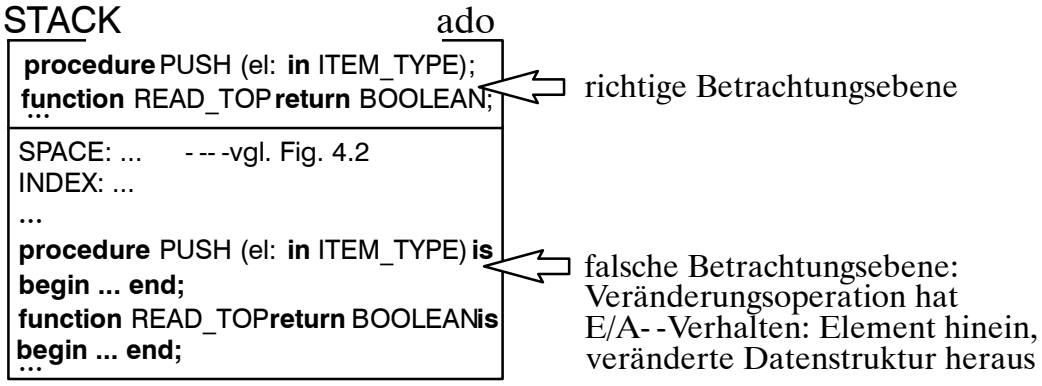
-- Zuordnung nicht im Nachhinein
Mischcharaktermodule, die Probleme bzgl. Ver--
ständnis und Beseitigung verursachen
verleitet zu Entwurfsfehlern (Äpfel und Birnen)

-- Frage
einfach bei adto--Modulen
fm Unterscheidung
ado Module schwierig

Frage: eigenständiger Zugriffsoperation(en
fm oder eines ado--Moduls

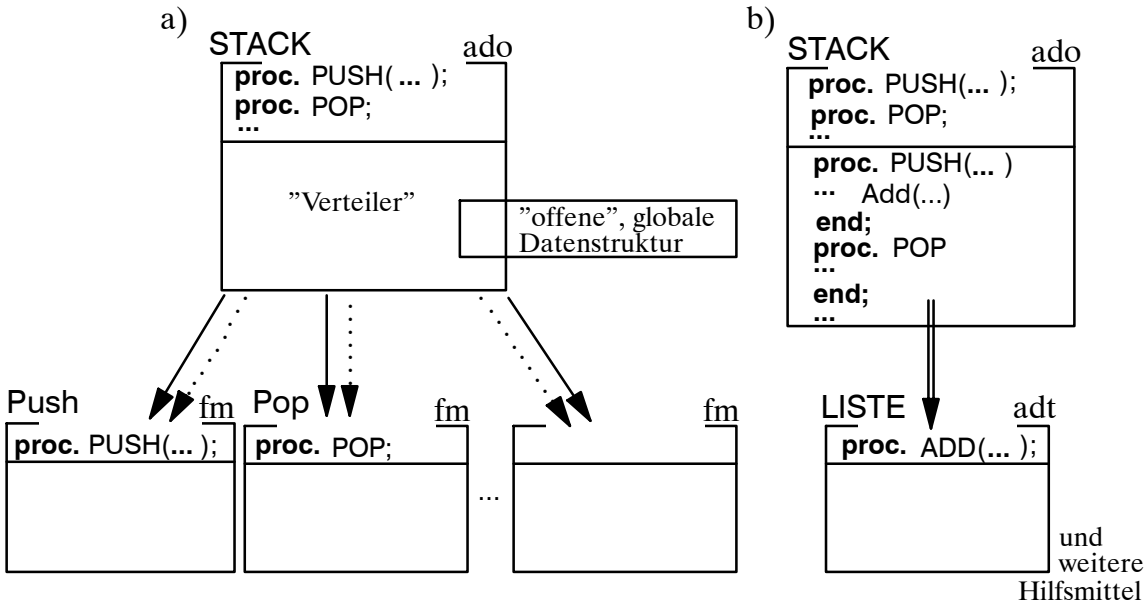
-- Gleichheit von Eingabe-- und Ausgabedatenstruktur
ist kein Indiz für Zugriffsoperationen
(vgl. Compiler--Optimierungsphase
Verschiedenheit von Eingabe--/Ausgabedatenstruktur
”mit Gewalt” beseitigbar (Op. auf E x A)
dann keine sinnvolle Zugriffsoperation

- - Regel 1: Betrachte für Unterscheidung Schnittstelle und nicht Rumpf
  Bsp. Datenobjektmodule



- - Regel 2: Betrachte Modul und nicht Gesamtsystem
  Bsp. funkt. Modul
  Steuerung des Dialogs in einem interakt. System

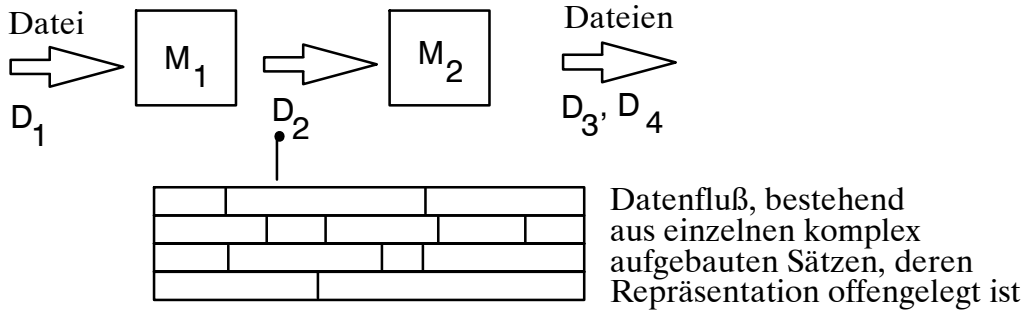
- - falsche Entwurfsentscheidung: Zugriffsop. sind funkt. Module kann in folg. Form nicht gemacht werden



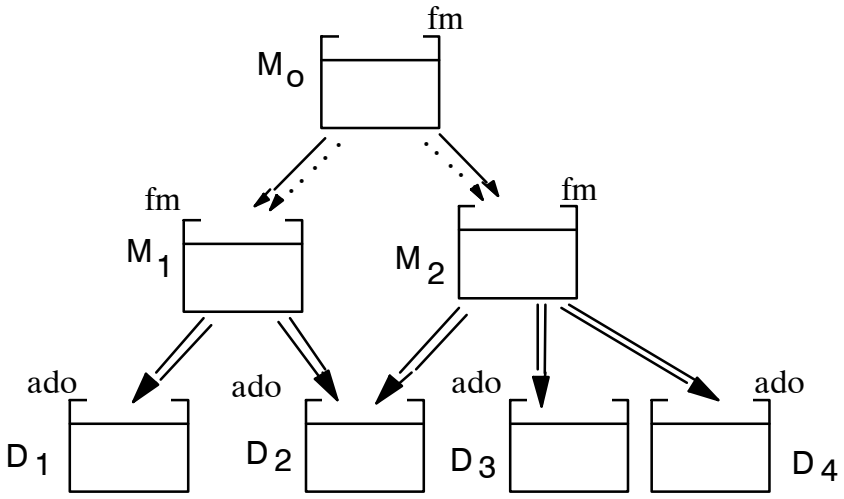
- - Fragen "Funktionale Module für die Realisierung von DA- -Modulen"
"Wie schneidet man?"

5.2 Zusammenspiel zwischen funkt. Modulen und DA- -Modulen

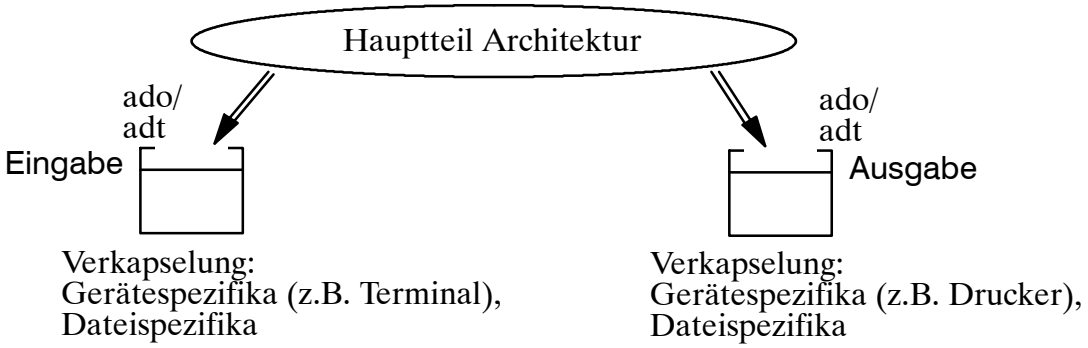
- - Herkömmliche Systementwicklung:
  Programme + Daten
  Handhabung offener Daten
  Detailwissen unnötigerweise verschmiert



- - Situation bei Beachtung von DA
  keine komplexen Datenströme mehr
  jede komplexe Datenstruktur als Modul
  Was wird verkapselt?
  Zusammenspiel fm- -ado, bzw. adt



- - allg. Aussage
  Eingabe Verkapselung von Eintrags- -,Datei- -,
  Ausgabe Gerätedetails
  ist Anwendung für Datenabstraktion



Gerätespezifika: virtuelles Terminal
virtueller Drucker

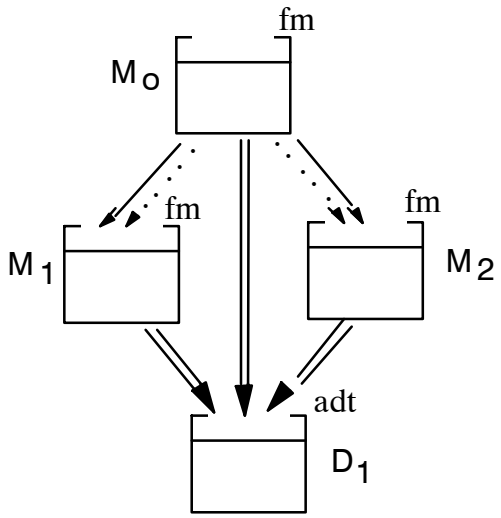
- - Mißverständnis bei Verwendung der Begriffe
  Eingabe/Ausgabe
  bisher Geräte, Datenstrukturen etc.
  d.h. Behälter, aus denen Daten bezogen werden

  Ein- -/Ausgabe als Aktivität von funkt. Modul
  mit Datenabstraktionsmodul zusammen

-- Unterscheidung funkt. und DA--Modul erneut
Bemerkungen und Beispiele

-- Bem. 1: fm aktions-- oder transformationsorientiert
DA--Module passiv oder zustandsbewahrend
heißt nicht, fm werden von sich aus aktiv, werden ebenfalls i.a. importiert und benutzt
heißt nicht, DA--Module bedienen sich bei Realis. keiner Hilfsmittel: sowohl für fm als auch DA--Module möglich

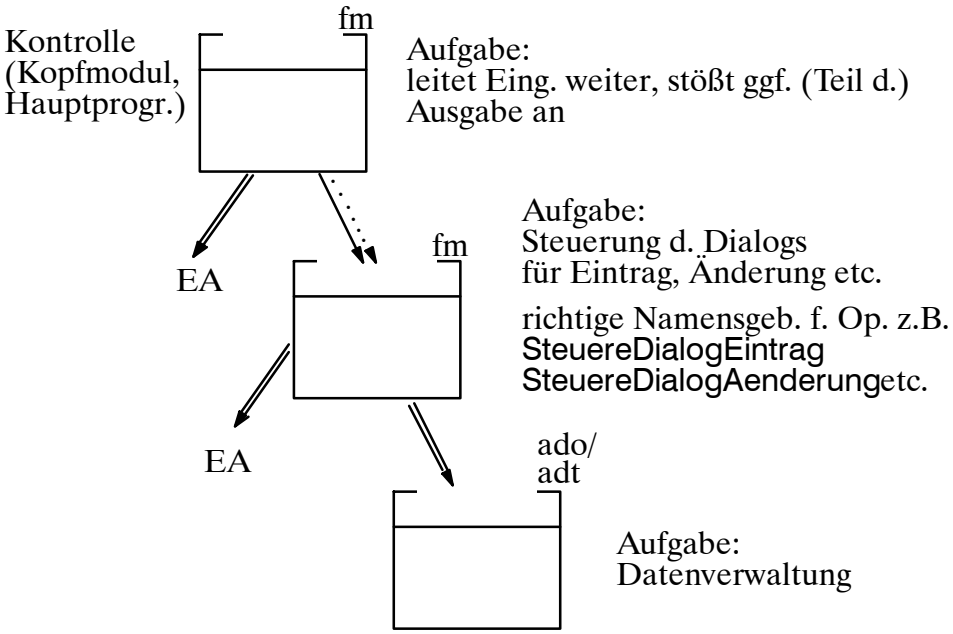
-- Bem. 2: Zusammenspiel fm und adt



ado im Rumpf von M_o
zur Verarbeitung über Parameter an M_1, M_2
auf ado Zugriffsop. anwendbar in M_1, M_2, deshalb Importe
Gedächtnis taucht also nicht mehr als Modul auf

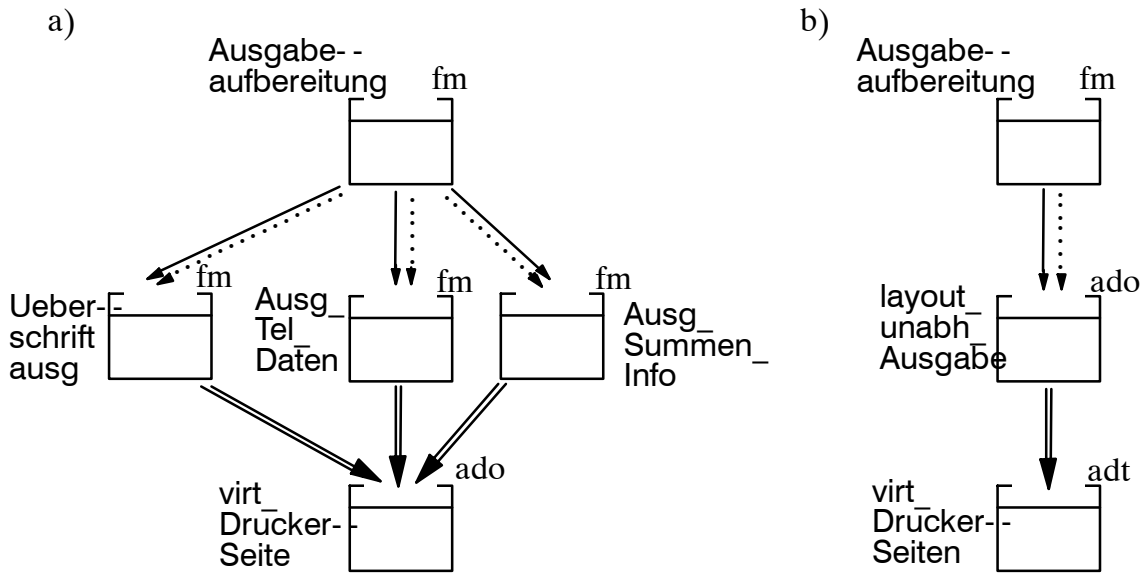
-- Bem. 3: Vermischen funkt. und Datenabstraktion
Kollektion zur Abspeicherung
mit allen Einträgen soll etwas gemacht werden
sogen. Iteratoren an der Schnittstelle des
Kollektionsmoduls: falsch!

-- Bsp. 1: Steuerungsmodul eines interaktiven Systems



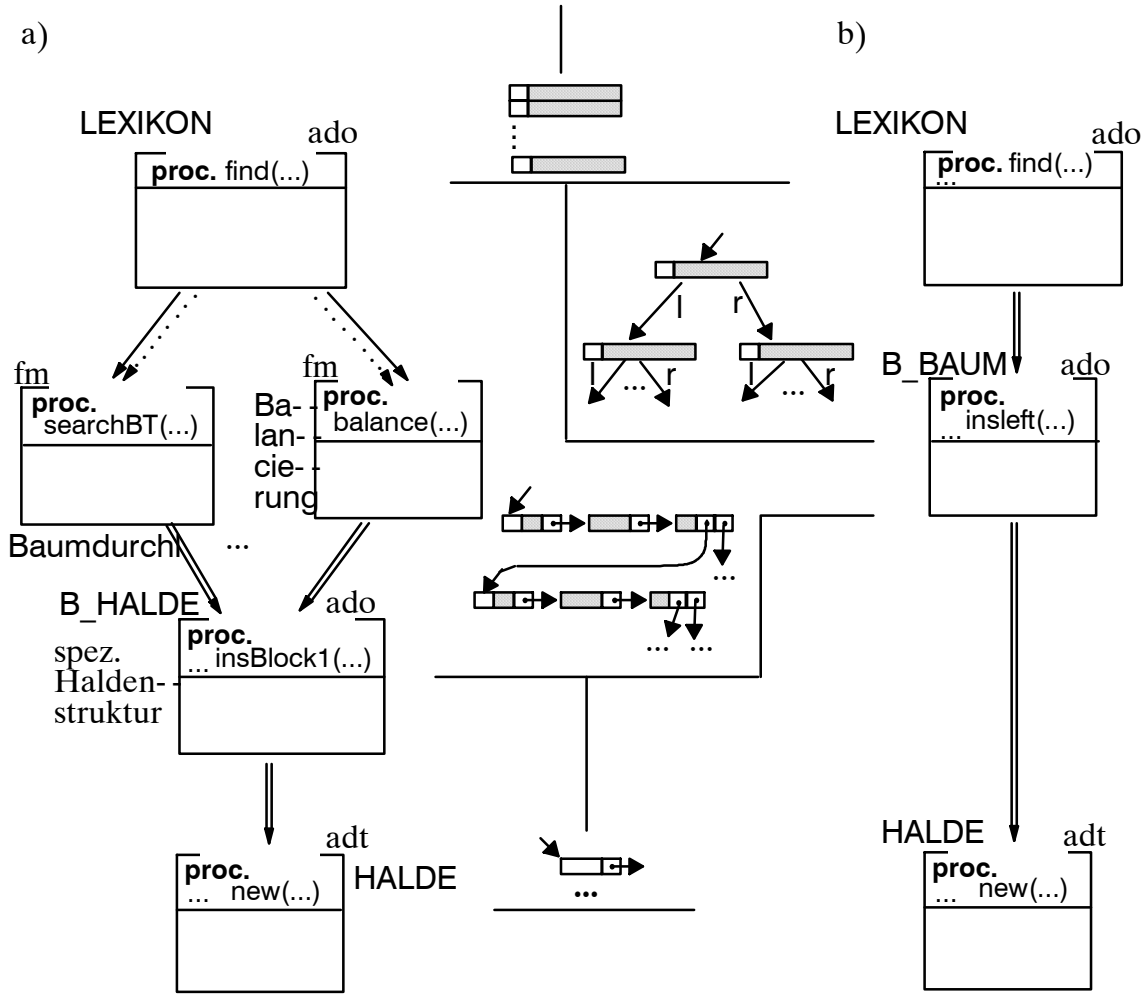
-- Bsp. 2: Steuerung einer Maschine mit kompl. inneren Zustand: Reaktion und Folgezustand davon ab--hängig (endl. Automat, Kellerautomat etc.)
Trennung Entwurfsentscheidung
Gedächtnisaufbewahrung, Zustandsübergang, Steuerung
Vorteil: Übergangsverhalten austauschbar

- - Bsp. 3: Ausgabeaufbereitung für Druckausgabe



Layout- -Druckaufbereitung ist DA- -Anwendung!

- - Bsp. 4: Wie entwirft man unterhalb eines DA- -Moduls?
  rechts: DA- -Schicht auf DA- -Schicht
  links: funkt. Zwischenschicht auf ”globalen” Daten



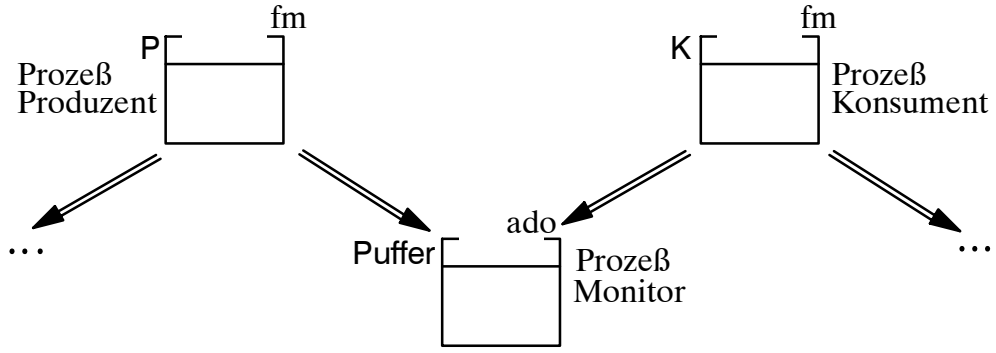
rechte Lösung hier besser: Adaptabilität

Geeign. DA- -Zwischenniveau muß aber vorhanden sein!

- - Rest des Abschnitts: nebenläufiges Problem
  gleiches Modulkonzept wie bei seq. Problemen?
  Zusammenspiel von Modulen gleich?

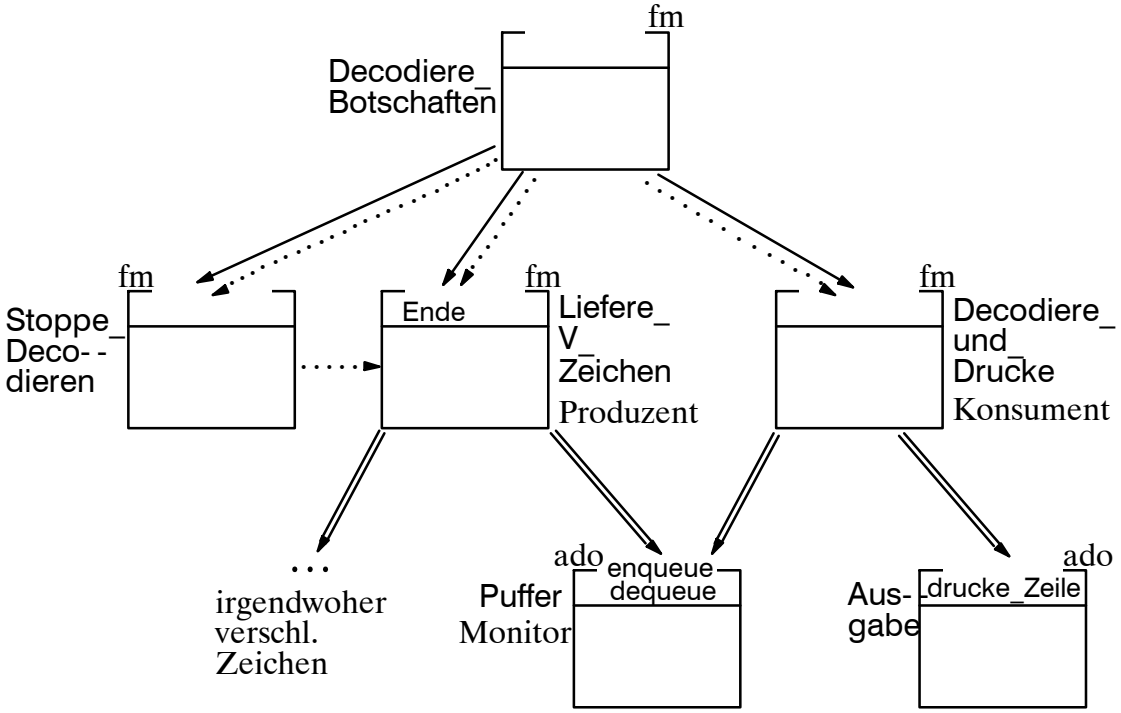
  Produzenten- -Konsumenten- -Problem

  Begriffe: Prozess, Nebenläufigkeit, expl./impl.
  Aktivierung, Anzahl der akt. Prozesse,
  Beendigung impl./expl., Synchronisation,
  gegens. Ausschluß, Informationsaustausch,
  Ada- -Rendezvous



Monitor: pass. Prozeß zur Entkopplung von
Produzent und Konsument

Beispielsystem



Unterschied zu seq. Systemen

Nebenläufigkeit

andere Benutzung v. Modulen: Aktivierung, Synchr.

Ereignissteuerung

Hoffnung: Entwurf i.w. ohne diese Überlegungen,

Nebenläufigkeitsaspekte dann hinzu

5.3 Teilsysteme einer Gesamtarchitektur

-- Motivation der Einführung von Teilsystemen
    Abschotten von Interna einer Teilarchitektur
    Gruppe von Modulen zusammen benutzbar
    Grobentwurf
    Einheiten der Wiederverwendbarkeit
    PO--Einheiten (Teilprojekt, Teilprodukt, Meilen--
        stein)
    Teststummel, Treiber: Zuordnung zu Modulgruppen

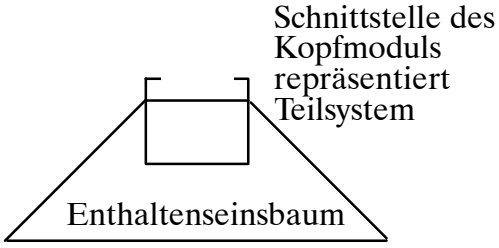
-- somit Modellierungseinheit auf Architekturebene,
    größer als Modul
    Verstehen einer bestimmten Architektur
    Erstellung und Wartung einer Architektur
    Wiederverwendbarkeit
    Grundlage für Arbeitseinheit

-- Forderungen:
    Zusammengefaßte Module müssen log. zusammen--
        gehören
    Teilsystem muß Unabhängigkeit und Eigenständig--
        keit besitzen

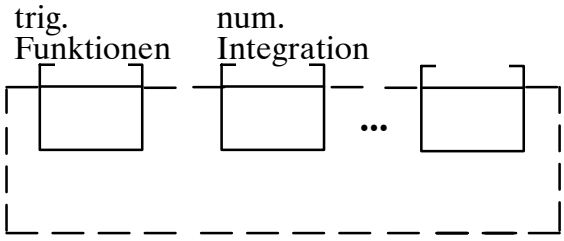
-- ansonsten
    bel. Zusammenfassung von Modulen durch beliebige
        der hier eingeführten Beziehungen verbunden
    => Teilsystembegriff ist neues und zus. Konstrukt

-- zwei Beispiele

a) Enthaltenseinsbaum



b) Bibliothek vordef. Module



-- grobe Definition:

”Ein Teilsystem ist eine log. Zusammenfassung von Modulen zu einem unabhängigen Ganzen, mit der Angabe, welche (Ressourcen welcher Schnittstelle welcher Module) außen benutzbar sein sollen.”
Exportschnittstelle des Teilsystems
Rumpf des Teilsystems

-- Liste typ. Anwendungen und Klassifikation:

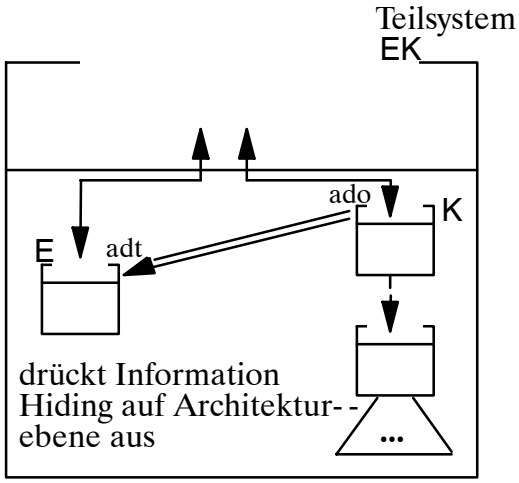
einzelner Modul (selten sinnvoll)
Unterbaum eines Enthaltenseinsbaums
vollständiger Enthaltenseinsbaum
Ansammlung einzelner Module
allg. Fall: Ansammlung von Modulen unterein--
ander evtl. über allg. Benutzbarkeit verbunden
etc.

-- Aus Forderung Unabhängigkeit Forderung für Einschränkungen der Modulbeziehungen im Rumpf und nach außen

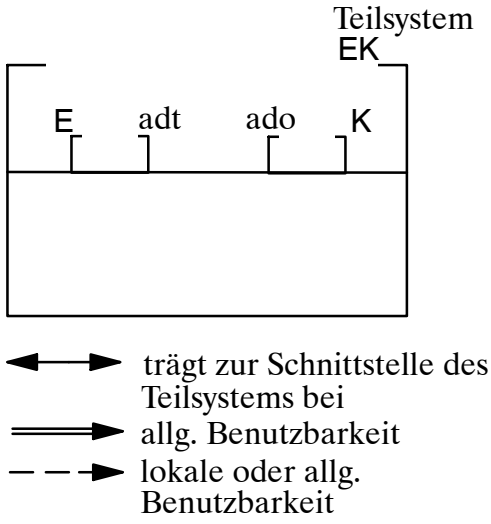
keine lokale Benutzbarkeit nach außen
nach Möglichkeit keine allg. Benutzbarkeit innen

-- graph. Notation für Teilsysteme

a) für den Entwerfer



b) für den Verwender



- - textuelle Detaildarstellung für Teilsysteme

für den Verwender dieses Teilsystems
zu erstellen vom Entwerfer des Teilsystems

```
subsystem EK is
    - - - -Exporte des Teilsystems gebildet aus Ressourcen der Schnittstellen der
    - - - -exportierenden Module
    export from abstract data type module E is
        type ET is private;
        procedure  opi (F0:ET, ...);
        ...
        - - - -Die Semantik der Schnittstellenoperationen ist die folgende:
        - - - - ...
    end E;
    export from abstract data object module K is
        procedure  opj (...);
        ...
        - - - - Die Semantik der Schnittstellenoperationen ist die folgende:
        - - - - ...
    end K;
end EK;
```

auszugestalten vom Entwerfer dieses Teilsystems

```
subsystem body EK is
    general import from Sub1.M1, Sub1.M2, Sub2.M3
        using ...  ;
    - - - - jetzt werden alle Module des Rumpfs in der üblichen Notation
    - - - - von Fig. 4.36 einschließlich ihrer Beziehungen aufgeführt
        ...
end EK;
```

- - Entwurf mehrstufig

Übersichtsarchitektur (aus Modulen u. Teilsyst.)
Teilsysteme zu entwerfen (aus Modulen, Teilsyst.)

meist genügt zweistufige Vorgehensweise:

Modellieren der Grobarch. (”Progr. im Größten”)
Modellieren der Teilarch. (Progr. im Großen)

-- Analogie Modul -- Teilsystem

Exportschnittstelle: Auswahl von Teilen d. Rumpfs
Rumpf: versch. Strukturen erlaubt (lokal, global)
Importschnittst.: Ressourcen anderer Architektur--bausteine

-- Verschiedenheit Module -- Teilsysteme

- Teilsystem Entwicklung gehört dem gleichen Arbeitsbereich an (PiG), Modulimpl., dem PiK,
- Verschiedene Betrachtungsebenen in einem Arbeitsbereich (PiG)

Grobarchitektur
Teilsystemarchitekturen

-- Teilsysteme als Architekturmodellierungseinheiten bei Entwurf/Wartung

Teilarch. einer Gesamtarch. nachträgl. zu Teilsyst. gemacht: Strukturierung
Teilsystem wird entworfen zu (teilw. verborgener Teilarchitektur): Entwicklung
Teilsystem als fertiges in Architektur eingehängt: Wiederverwendung
Menge vorgegebener Bausteine zu Teilsyst. gemacht: Strukturierung
als Teilsyst. vorgegebener Baustein abgeflacht als Teilarch. in die Gesamtarch. eingehängt, i.a. nach Modifikation: Wiederverwendung

-- Architekturdokumente enthalten i.a. sowohl Module als auch Teilsysteme. Die Unterscheidung ist wichtig wegen

Verständlichkeit (Unterschiede der Abstraktions--niveaus, Versehen der Importe eines Teilsystems)

Entwerfer weiß Unterschied bei

vorgegeb. Teilarch., die zu Teilsyst. gemacht wurde

Basisbaustein (Kenntnis nicht so wichtig)

neu zu entw. Teilsyst. (muß auf der Grobarch. bedacht worden sein, z.B. für Importe); gem. Schnittstelle gibt dies sofort zu erkennen)

Muß Unterschied kennen bei Eintragung von Beziehungen,

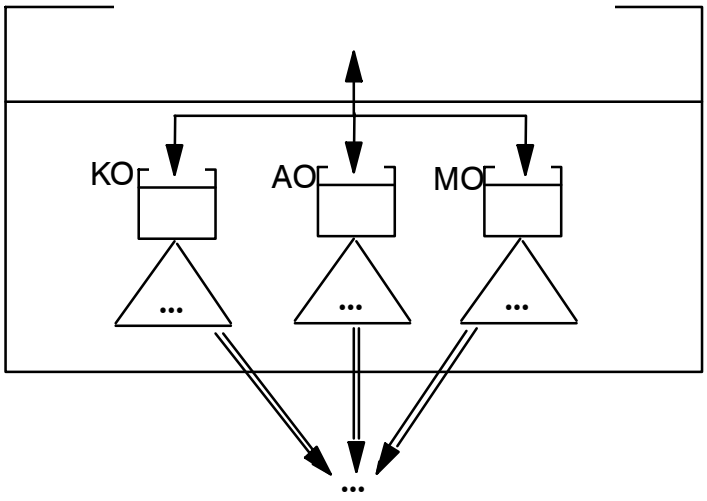
da Konsistenz der Gesamtstruktur von der internen Struktur abhängt.

-- Teilsysteme als log., abgeschl., unabh. Einheiten Grund--
lage für
Arbeitsteilung (Entwurf, Integration, Wartung, Dokumentation, PO)
Teilprojekte: PO--Probleme in besser überschaub. Größenordnung

-- Art eines Teilsystems
nur Schnittstellenmodule betrachten
falls alle von gl. Art (fm, ado, adt) klar
falls verschieden: ?, oft bestimmender Modul

-- Art
eines Moduls eindeutig
eines Teilsystems oft eindeutig
eines Gesamtsystems (Batchsystem klar, interakt. System ?)

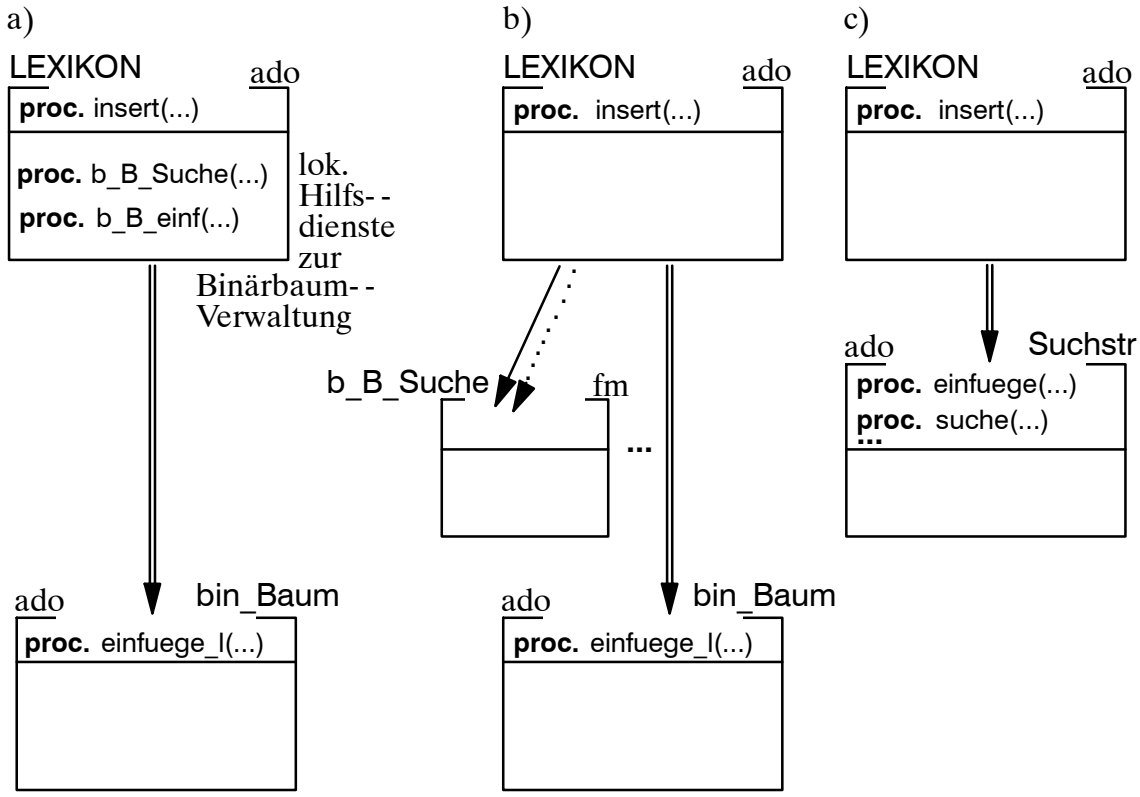
-- Teilsysteme zur Strukturierung breiter Schnittstellen



5.4 Schichtenbildung durch mehrstufige Datenabstraktion

- - Teilarchitektur unter DA- -Modul
  - Real. im Rumpf allein
  - Abstützen auf weiteren DA- -Modulen
  - Abstützen bel. viele Stufen nach unten
  - Zwischenniveaus funkt. Module

- - Zusammenspiel direkt übereinanderliegender DA- - Schichten



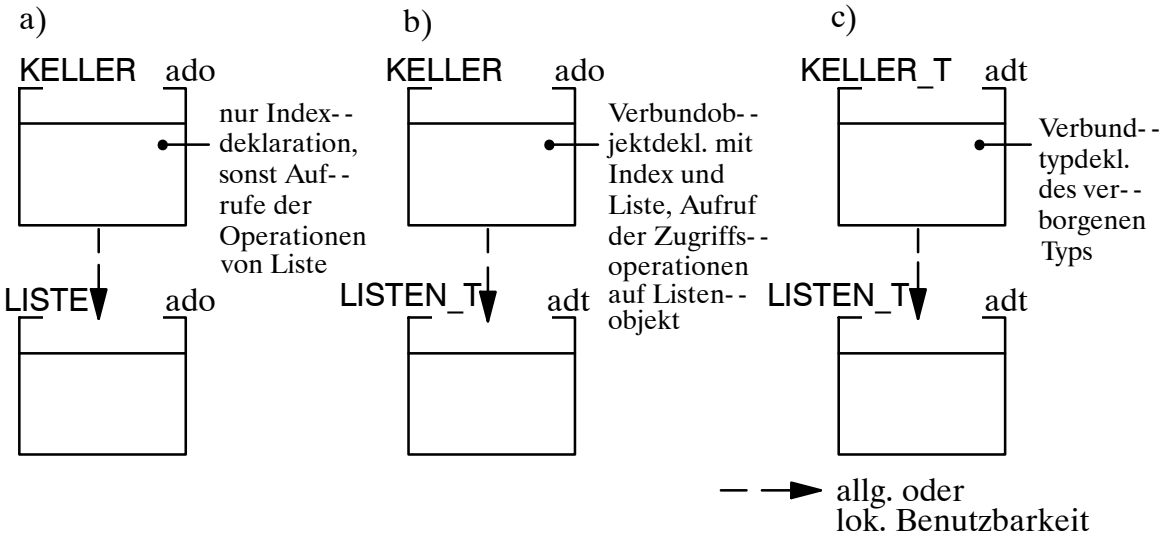
Sicht des Rumpfes ist Sicht der nächsten DA- -Schnitt- - stelle (auch bei dazwischenliegenden fm- -Modulen)

Hilfsdienste im Rumpf, eigenständiger fm, in der Schnittstelle des nächsten DA- -Moduls

bei DA- -Schichten: Niveauuntersch. d. Schnittst. nötig.

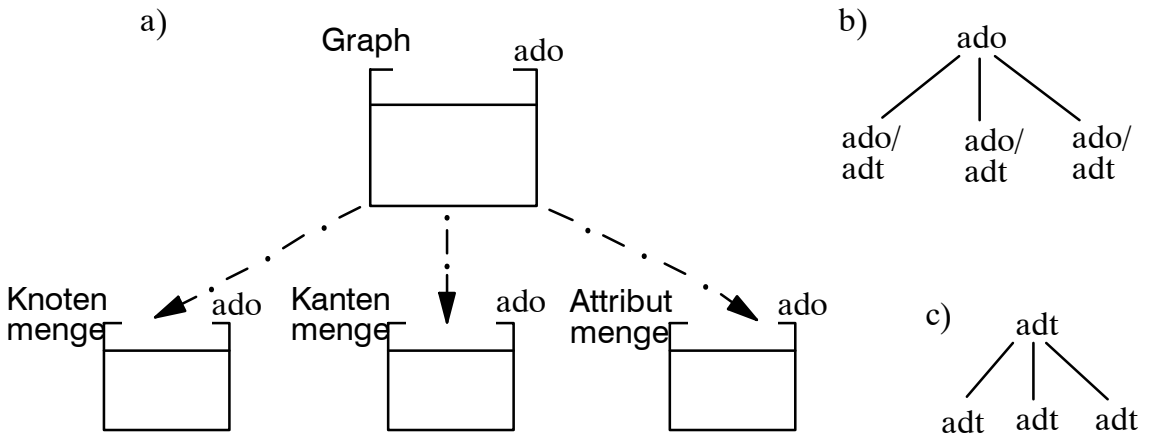
- - Woraus besteht Gedächtnis eines ado- -Moduls?
    - im Rumpf vollständig abgehandelt (Keller oder mit seq. Realisierung)
    - DA- -Module übereinanderliegend, Gedächtnis aufgeteilt
      Bsp. Keller - - Liste, Lexikon - - Baum

- - Arten von DA- -Modulen, die sich aufeinander ab- - stützen



mehrstufig analog
adt - - ado unzulässig bei gl. DA- -Anwendung

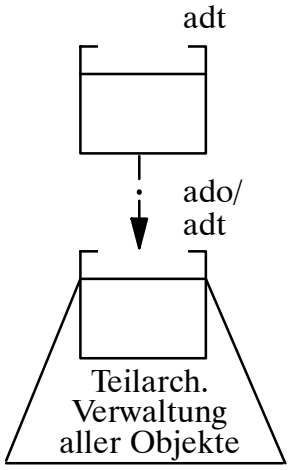
-- Abstützen auf mehrere DA--Module



-- Wo liegen die Objekte eines adt--Moduls?

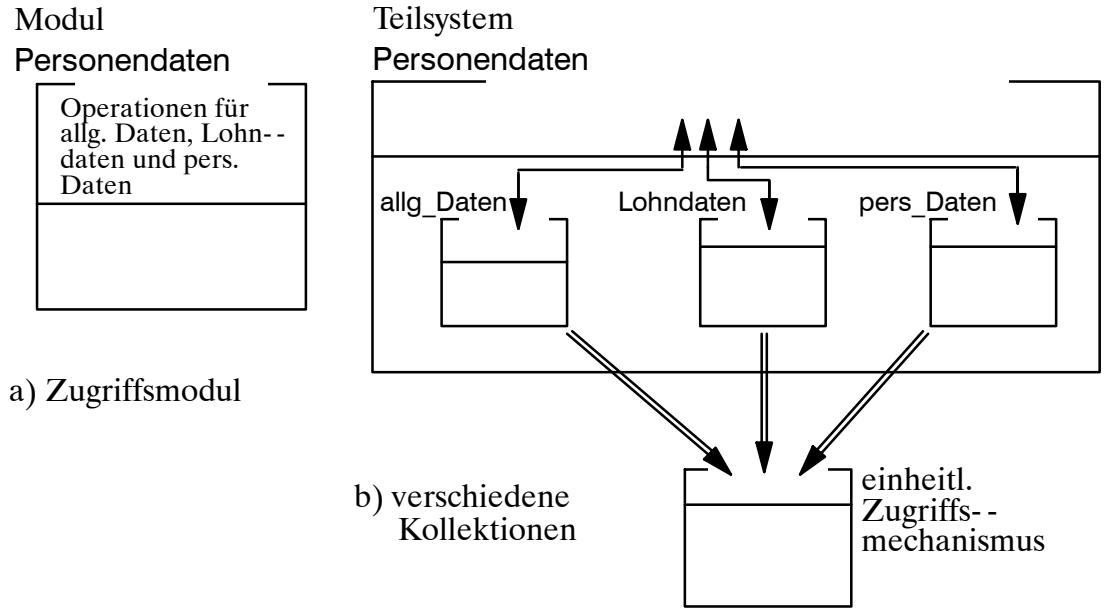
- Variablensemantik: Typ exportiert, Objekte im Rumpf üb. Dekl. erzeugt; Laufzeitkeller, PS macht Verwaltung
- Zeigersemantik: Erzeugungsop., Objekte über Anweisungsteil erzeugt; Halde mit Zeigern, PS mach Verwaltung

keine Zeiger in PS:
Haldenverwaltung selbst



Fazit: Unter adt--Modul stets Teilarchitektur, von PS ggfl. geschenkt.

-- Zusammenfassungen von Kollektionen von Objekten Bsp. Personalverwaltung



5.5 Mehrfache Datenabstraktion: Einträge und Kollektionen

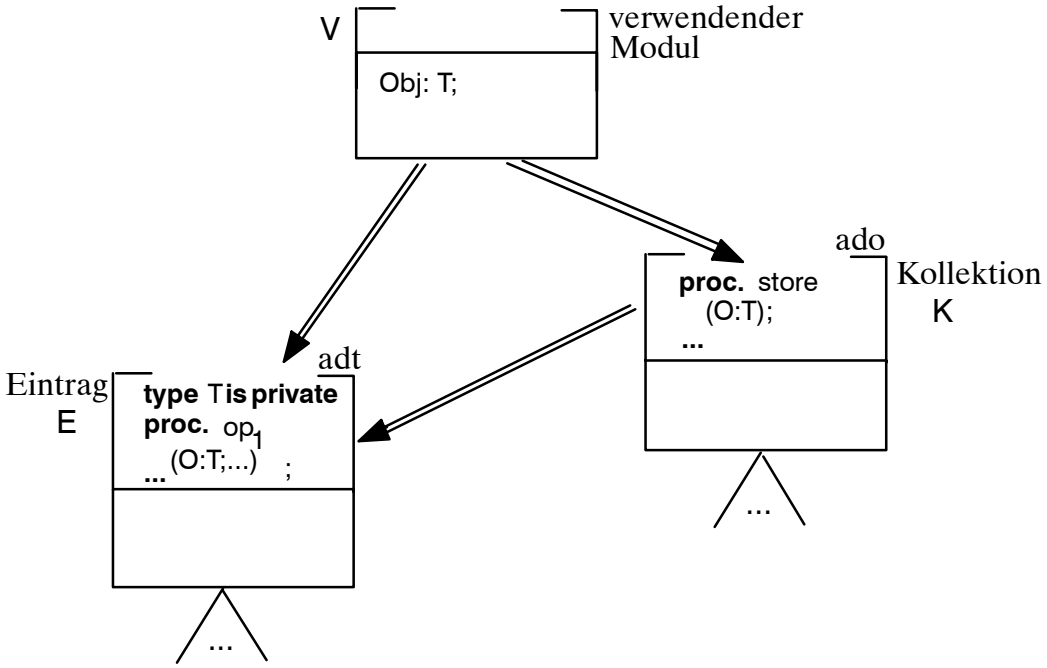
- - Eintrags- -Kollektions- -Situation:
  zweifache Datenabstraktion
  wie wirken DA- -Module zusammen
  Anwendung des Teilsystembegriffs

- - einfache Lösung: Kollektionsmodule, Einträge implizit

```
abstract data object module Kartenkasten is
    procedure Karte_ablegen (...);          - -- -Saemtliche benoetigten Komponenten
    procedure Karte_loeschen (...);- -- -eines Eintrags werden als Para- -
    procedure Karte_aendern (...);          - -- -meter aufgefuehrt
    function ist_Karte_vorhanden (...) return BOOLEAN;
    function ist_noch_Platz return BOOLEAN;
    ...
    Karte_nicht_da, kein_Platz_mehr : exception;
    - -- - Die Bedeutung der Schnittstellenoperationen ist die
    - -- - folgende: ...
end Kartenkasten;
```

- - Voraussetzungen
  Verwender geht nur mit einem Eintrag um
  Verwender geht nicht mit Eintrag als ganzem um

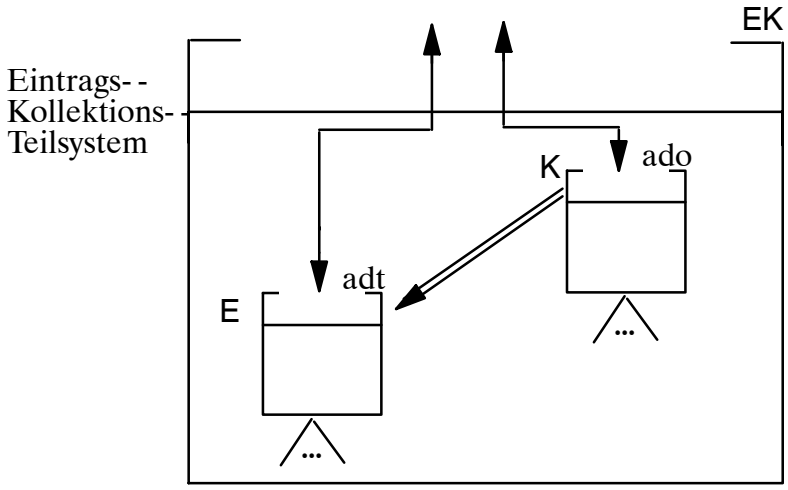
- - sauberer:
  Trennen der beiden DA- -Entwurfsentscheidungen



  Voraussetzungen von oben entfallen

- - Zusammenspiel V, E, K
  - Unterscheidung ”Art” des adt- -Moduls E
    - Variablensemantik
      - ein oder mehrere Objekte deklariert, nach Veränderung (Werte) in K abgelegt
    - Zeigersemantik
      - Bezeichner auf Objekt(e) deklariert, Objekt(e) im Anweisungsteil erzeugt und nach Veränderung abgelegt
  - Benutzbarkeit K - - E nötig

- - Teilsystem für Eintrags- -Kollektions- -Problem



Ist ”ado- -Teilsystem”

- - unsaubere vereinfachte Eintrags- -Kollektions- -Lösung

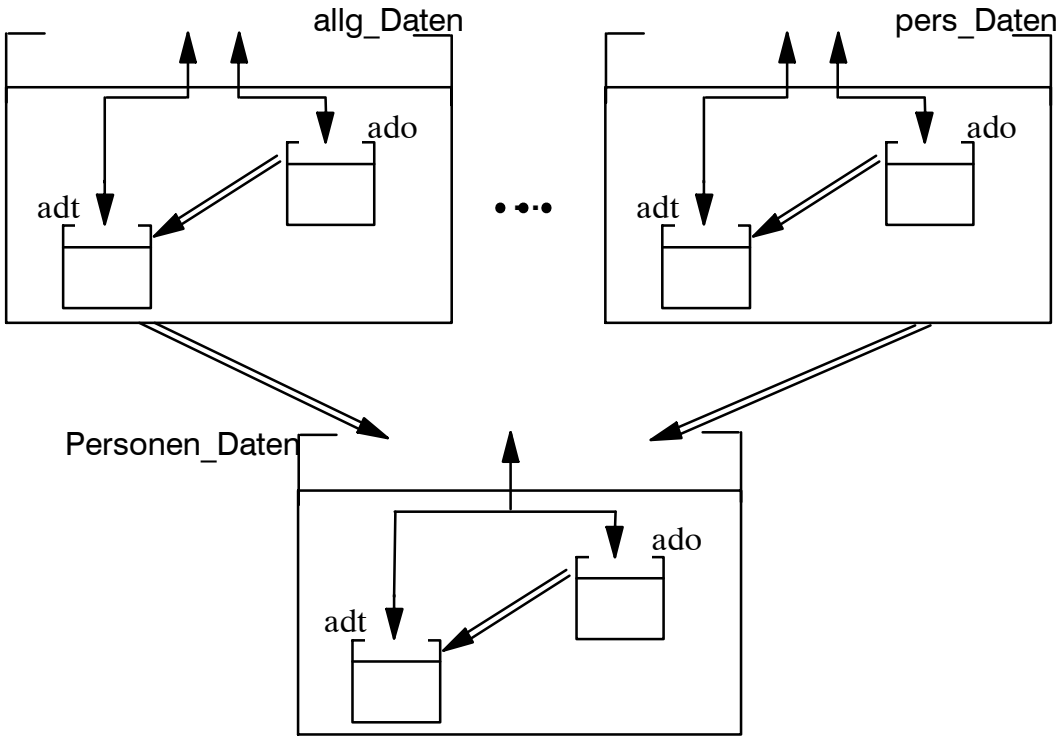
```
”undetermined” module Kartenkasten is
        type Karte is
                record
                        KEY: KEY_T
                        NAME: NAME_T
                        ...
                end;
        procedure Karte_ablegen (K: in Karte);
        procedure Karte_loeschen (K: in Karte);
        ...
end KartenKasten;
```

Mischcharaktermodul
DA- -Verletzung

- - Anwendung auf Karteikastenproblem

```
- -- -data type
subsystem Kartenkaesten is
   general import from type collection module Komponententypen using all;
   - -- -Das Teilsystem handhabt eine Eintrags- -Kollektions- -Situation. Da mehrere
   - -- -Kartenkaesten gehandhabt werden sollen, wurde fuer den Kollektionsmodul
   - -- -ein Datentypmodul gewaehlt. Die Semantik der Operationen auf einem Ein- -
   - -- -trag und auf einer Kollektion sind unten angegeben. Fuer die Kompo- -
   - -- -nententypen einzelner Karten liege deren Beschreibung durch den Kollek- -
   - -- -tionsmodul Komponententypen definiert vor.

   export from abstract data type module Karten is
      type Karten_T is private;
      procedure initialisiere_Karteninhalt(Karte: out Karten_T);
      procedure loesche_Karteninhalt(Karte: in out Karten_T);
      ...
      - -- -Die Semantik der Operationen auf Objekten des opaken Eintragstyps
      - -- -Karten_T ist die folgende: ...
   end Karten;

   export from abstract data type module Kaesten is
      type Kaesten_T is private;
      procedure Karte_ablegen(Karte: in Karten_T; Kasten: in out Kaesten_T);
      procedure Karte_loeschen(Karte: in Karten_T; Kasten: in out Kaesten_T);
      ...
      - -- -Die Semantik der Operationen des opaken Kollektionstyps Kaesten_T ist
      - -- -die folgende: ...
   end Kaesten;
end Kartenkaesten;

subsystem body Kartenkaesten is
   - -- -Hier tauchen die zur Realisierung des Teilsystems benoetigten Importe auf
   - -- -sowie die Module des Teilsystems, naemlich Karten, Kaesten und weitere,
   - -- -die ggfs. zur Realisierung von Karten und Kaesten noetig sind und zwar in
   - -- -der textuellen Notation von Fig. 4.36. Die textuelle Architekturnotation fuer
   - -- -diese Module enthaelt auch die Beziehungen dieser Module untereinander.
   ...
end Kartenkaesten;
```

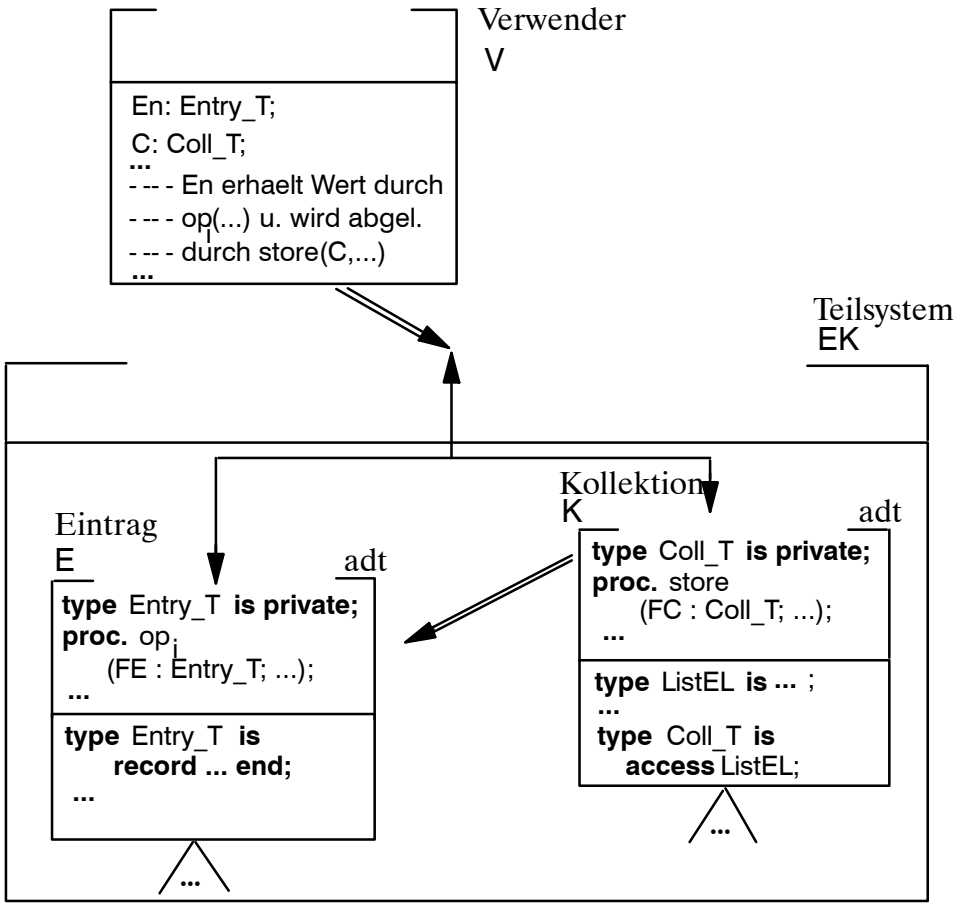
-- Personendatenbeispiel erneut



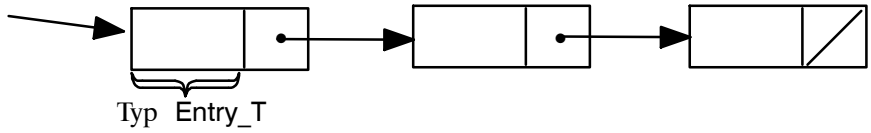
- - im folgenden Zusammenspiel E, V, K
  bei Variablensemantik
  Zeigersemantik für Einträge
  Ist die unterschiedliche Handhabung für die
  Architekturmodellierung bedeutsam?
  Auswirkung auf PiK- -Ebene
  Erläuterung der Situation
  Kollektion ist adt
  Kollektion verkettet realisiert

- - Variablensemantik für Einträge

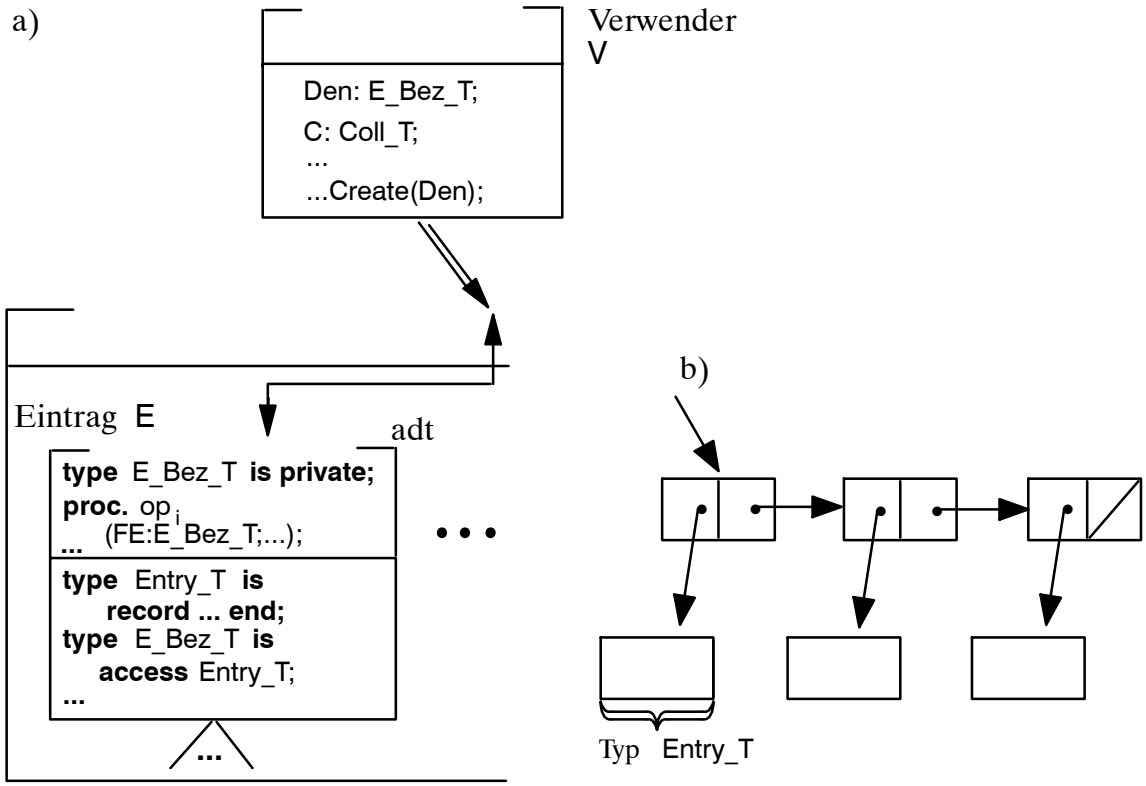
a)



b)



-- Zeigersemantik für Einträge



- - Unterschiede
    - Variablensemantik: Kopieren
      Zeigersemantik: versch. Bez. auf das gl. Objekt
    - Änderung eines Eintrags
        Änderung einer Kopie
        Änderung aller ”Einträge” (Aliasing)
    - Vergleich von Objekten
        Vergleich von Werten
        Vergleich von Zeigern

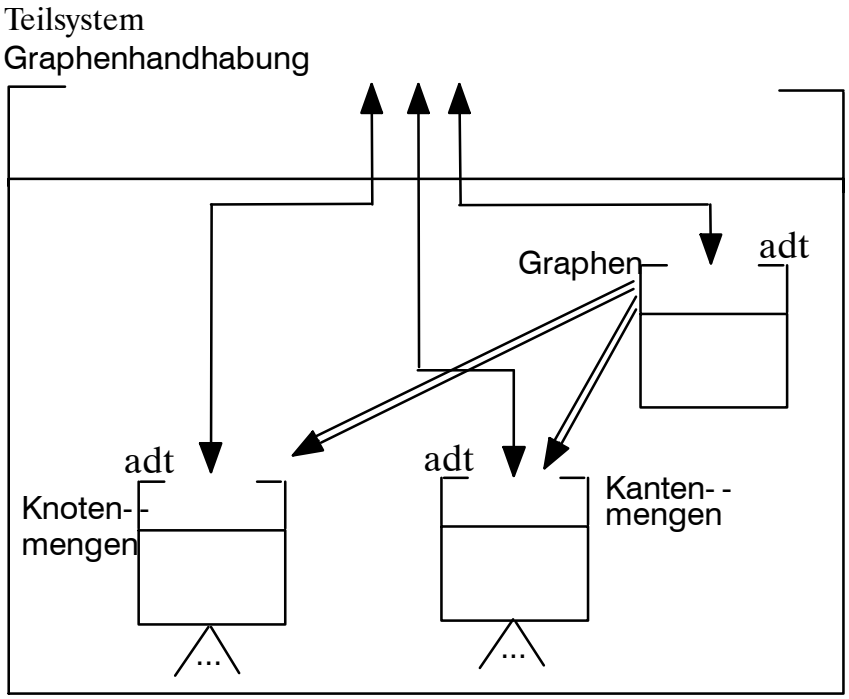
- - Unterschied sollte auf PiG- -Ebene sichtbar sein
    Schnittstelle Eintragsmodul
    ggfl. Haldenarchitektur unter Eintragsmodul

5.6 Teilarchitekturen für Einträge und Kollektionen

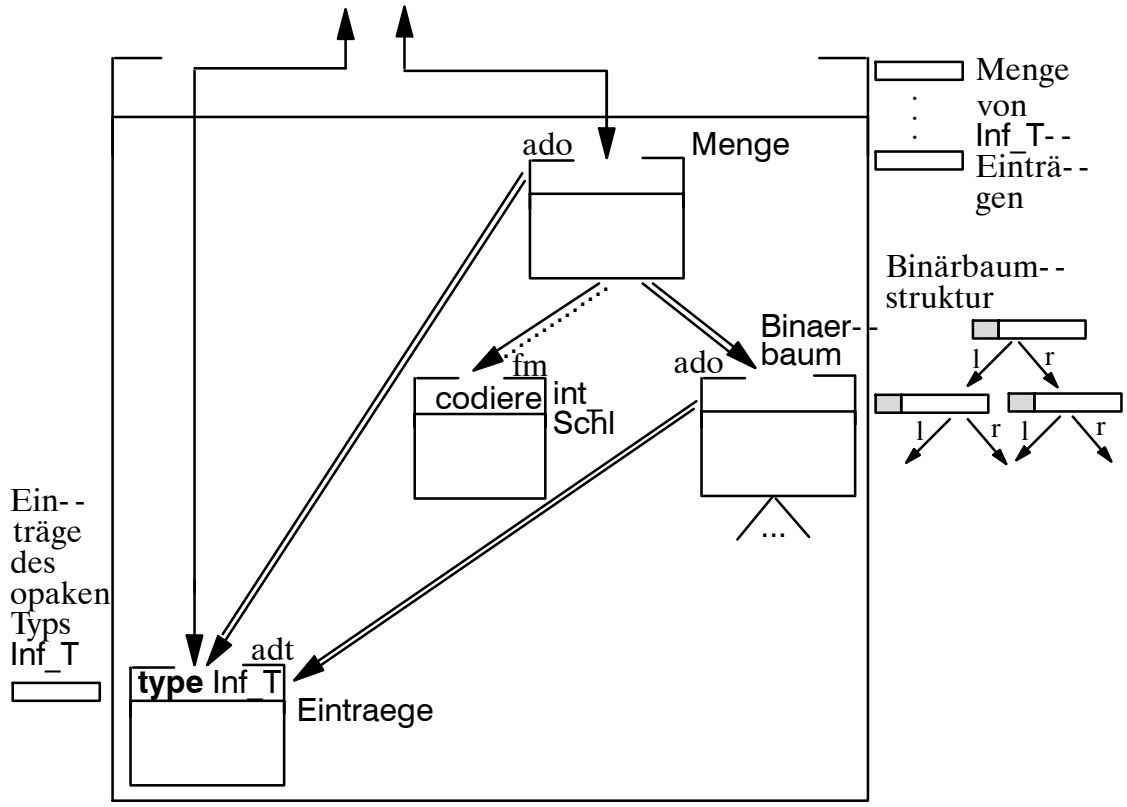
-- kompliziertere Eintrags-- u. Kollektions--Architekturen

mehrere Einträge oder Kollektionen
Realisierung derselben über mehrere Niveaus

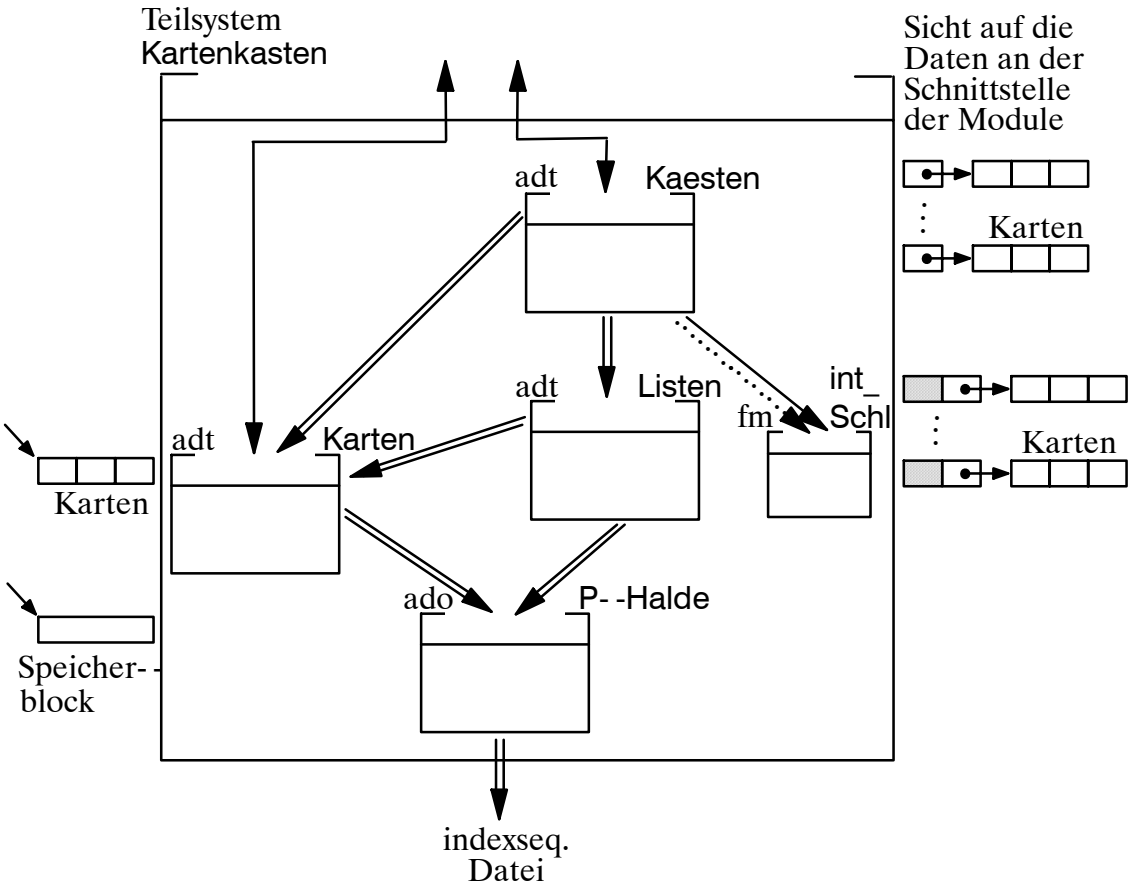
-- Teilsystem für Graphen, Bsp. 1



-- Teilsystem für Menge, Bsp. 2



-- Kästenbeispiel mit Zeigersemantik, Bsp. 3



5.7 Generische Module und Teilsysteme

- - Wiederverwendbarkeit setzt Anpaßbarkeit voraus
  - Anpassen des einzufügenden Bausteins
  - Anpassen des Kontexts, in den er einzubinden ist

- - Bisher für Wiederverwendbarkeit
  - Untersch. Schnittst. - - Rumpf v. Modulen
  - funkt. Abstraktion bzw. Datenabstraktion Inf.
  - Real. - -> Schnittstelle Hiding
  - allg./lokale Benutzbarkeit
  - Teilsysteme
  - ...

- - Ada- -Beispiel für Generizitätsidee

```
generic- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
generischer Teil - -- -- -- -- -- -- - -- -
      SIZE: NATURAL;                    - -- -SIZE und TYP_ITEM werden erst- -- -
      type TYP_ITEM is private;         - -- -bei der gen. Ausprägung festgelegt- -- -
package G_ITEM_KELLER is  - -- -*****************Schnittstelle********- -- -
      procedure PUSH(X: in TYP_ITEM);   - -- -Zugriffsoperationen bestehen- -- -
      procedure POP;                    - -- -aus Veraenderungen
- -- -
      function READ TOP return TYP_ITEM;  - -- -und Abfragen incl. Sicher- -- -- -
      function IS_EMPTY return BOOLEAN;   - -- -heitsabfragen. Die        - -- -
      function IS_FULL return BOOLEAN;    - -- -Bedeutung der einzelnen  - -- -
      ST_UNDERFLOW, ST_OVERFLOW: exception; - -- - Operationen ist: ...  - -- -
end                                                                 G_ITEM_KELLER;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -  - -- -

package body G_ITEM_KELLER is  - -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -  Rumpf
- -- -- -- -- -- -- -- -- -- -- -- -- -          - -- -
      SPACE: array(1..SIZE) of TYP_ITEM;                                  - -- -
      INDEX: INTEGER range 0..SIZE;                                       - -- -
      procedure PUSH(X: in TYP_ITEM) is begin ... end;                    - -- -
      ...                                                                 - -- -
      function IS_FULL return BOOLEAN is begin ... end;                   - -- -
end G_ITEM_KELLER;  - -- -*************************************
- -- -
```

---Erzeugung eines generischen Exemplars:

```
package INTEGER_KELLER is
    new G_ITEM_KELLER(SIZE => 200, TYP_ITEM => INTEGER);
```

---Kann jetzt mit INTEGER_KELLER.PUSH(...) etc. modifiziert werden.

---Man beachte: Das Exemplar INTEGER_KELLER "ist" selbst der Keller.

-- Idee der Generizität
  Offenlassen von Details der Real., Definition
  Schablone definieren
  Detailfestlegung durch Exemplarerzeugung
  Ist Abstraktion für Menge von Bausteinen

-- Welche Details werden offengelassen (gen. Parameter)?
- Festl. von Typen aus Klasse von Typen
- Festl. von Funktionen aus Klasse von Funkt.
- Dimensionierung

-- Generizitätsidee Fortsetzung der bish. Überlegungen
  kompl. Bausteine (hier gen. Bausteine) nur einmal entwerfen und implementieren
  Herausfakorisieren der Details (gen. Parameter) vielfach verwendbar
  Verwendung (Detailfestl. durch gen. Exemplar--erzeugung) einfach

-- Generizität hat andere Qualität
  gen. Baustein kein Architekturbaustein im bish. Sinne (Menge v. Architekturbausteinen)
  erst durch Exemplarerzeugung entsteht Baustein
  Exemplarerzeugung einfach hinzuschreiben:
  Ist das dann eine PiG--Einheit?
  Generizität paßt eher zum Prozeß der Entwicklung

einer Architektur als zum Festhalten eines Ergebnisses

-- gen. Bausteine können nicht über die (allg.) Benutz--
barkeit in Arch. eingehängt werden. Benutzbarkeit
hat andere Semantik
Nicht Import von Ressourcen, sondern
Nutzung einer Schablone, um pass. Architek-
tur--
baustein zu erzeugen

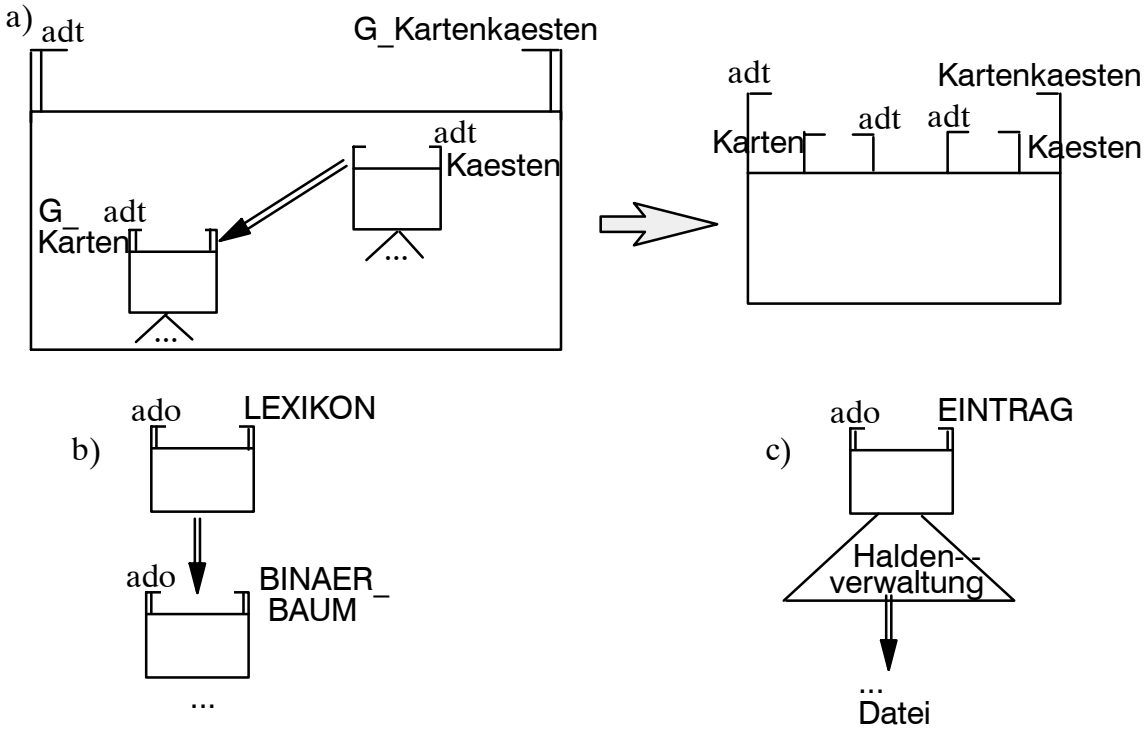
-- Einordnung, ob zu den bisherigen Konzepten paßt,
hängt von zugrundel. Programmiersprache ab:
Makromechanismus in PS: kontextsensitive
Überprüfung zur Compilezeit
Makromechanismus durch Werkzeug:
Überprüfung auf expandiertem Text

Laufzeitmechanismus in Programmierspr.

da die meisten Programmierspr. keine Typ--
oder
Prozedurparameter kennen, bleiben wir bei
Ada--Auffassung des Makromechanismus:
Betrachten gen. Exemplare als Architektur--
bausteine, aber nicht die gen. Bausteine selbst

-- zwei Erzeugungsmechanismen für abstr. Datenobjekte
adt--Modul --> ado--Erzeugung (Laufzeit)
gen. ado--Modul --> gen. Exemplarerzeug.
(Compilezeit)
auch Frage, ob ado als Modul in der Architektur
verankert sein soll

-- gen. Mechanismus nicht nur für Module (in Ada), sondern bel. Teilarchitekturen



- - textuelle Notation für ein gen. Teilsystem

```
generic
   type Schl_T, Namen_T, ... is private;
   - -- -evtl. Operationen fuer diese opaken Typen
- -- -data type
subsystem G_Kartenkaesten is
   ...
   export from abstract data type module Karten is
      type Karten_T is private
      procedure initialisiere_Karteninhalt(Karte: out Karten_T);
      procedure loesche_Karteninhalt(Karte: in out Karten_T);
      procedure setze_Schluessel(Karte: in out Karten_T; Schl: in Schl_T);
      procedure liefere_Schluessel(Karte: in Karten_T) return Schl_T;
      procedure setze_Name(Karte: in out Karten_T; Name: in Namen_T);
      procedure liefere_Name(Karte: in Karten_T) return Namen_T;
      ...
   end Karten;
   export from abstract data type module Kaesten is
      type Kaesten_T is private;
      procedure Karte_ablegen(Karte: in Karten_T;
                              Kaesten: in out Kaesten_T);
      procedure Karte_loeschen(Karte: in Karten_T;
                              Kaesten: in out Kaesten_T);
      ...
   end Kaesten;
end G_Kartenkaesten;

susbsystem body G_Kartenkaesten is
   ...
end G_Kartenkaesten;
```

5.8 Objektorientierte Programmier- - sprachenkonstrukte

- - Zielsetzung
  - Ideenwelt OOP einführen
  - behandeln dabei auch PiK: Ausflug
  - Unterscheidung PiG- -PiK in OOP nicht gemacht
  - Smalltalk als Beispielsprache

- - Unterscheidung Objektbasiertheit- -Objektorientiertheit

- - Smalltalk - - Begriffe
  - Objekt --> ado
  - Methode --> Zugriffsoperation
  - Botschaft --> Aufruf einer Zugriffsop.
  - Klasse --> adt- -Modul

classname	Finanzsituationen
superclass	Object
instance variable names	verfuegbaresGeld einnahmen ausgaben

instance methods

Veränderungsoperationen

```
erhaelt: betrag von: quelle
    einnahmen at: quelle
                put: (self insgesamtErhaltenVon: quelle) + betrag.
    verfuegbaresGeld ⇐ verfuegbaresGeld + betrag
```

```
gibAus: betrag fuer: zweck
    ausgaben  at: zweck
                put: (self insgesamtAusgegFuer: zweck) + betrag.
    verfuegbaresGeld ⇐ verfuegbaresGeld - - betrag
```

Anfragen

```
verfuegbarIst
    ↑ verfuegbaresGeld
```

```
insgesamtErhaltenVon: quelle
    (einnahmen includesKey: quelle)
        ifTrue: [↑ einnahmen at: quelle]
        ifFalse: [↑ 0]
```

```
insgesamtAusgegFuer: zweck
    (ausgaben includesKey: zweck)
        ifTrue: [↑ ausgaben at: zweck]
        ifFalse: [↑ 0]
```

Initialisierung

```
anfangsSituation: betrag
    verfuegbaresGeld ⇐ betrag.
    einnahmen ⇐ Dictionary new.
    ausgaben ⇐ Dictionary new
```

a) Smalltalk Klasse

b) Schnittstelle des entspr. abstrakten Datentyps

```
abstract data type module Finanzsituationen
   type Fin_Sit_T is private;
   ---Veraenderungsoperationen
   procedure erhaelt_von(Fin_Sit: in out Fin_Sit_T; betrag: in betrags_T;
            quelle: in ang_T);
   procedure gib_aus_fuer(Fin_Sit: in out Fin_Sit_T; betrag: in betrags_T;
            zweck: in ang_T);
   ---Anfragen
   function verfuegbar_ist(Fin_Sit: in Fin_Sit_T) return betrags_T;
   function insgesamt_erhalten_von(Fin_Sit: in Fin_Sit_T; quelle: in ang_T)
            return betrags_T;
   function insgesamt_ausgeg_fuer(Fin_Sit: in Fin_Sit_T; zweck: in ang_T)
            return betrags_T;
   ---Initialisierung
   procedure Anfangssituation(Fin_Sit: in out Fin_Sit_T; zweck: in ang_T);
   ---Beschreibung der Semantik der Schnittstelle:
        ...
end Finanzsituationen;
```

- - Zusammenfassung:
  - nur diese Art von Modulen vorhanden:
    - adt- -Modul mit Zeigersemantik
  - keine ado- -Module
  - keine funkt. Module
  - untypisierte Programmiersprache

- - Programmieren im Kleinen
  - Ausprogrammieren der Methoden
  - hauptsächl. durch Anweisungen:
    - Ausdrücke oder Zuweisungen
  - außerhalb v. Methoden: direkt ausführb. Ausdrücke

```
erhaelt: betrag von: quelle
   einnahmen at: quelle
                put: (self insgesamtErhaltenVon: quelle) + betrag.
   verfuegbaresGeld ⇐ verfuegbaresGeld + betrag
```

- - Botschaftsaustausch: Zweiwege- -Kommunikation
  - Programmstelle (Aufruf): Sender
  - Empfänger: abstr. Datenobjekt
  - Heraussuchen der ”passenden” Methode in einer Klasse zur Laufzeit
  - Ausführung der Methode d.h. Rumpf
  - Zurücksenden an Sender: Ergebniswert, veränderter Empfänger

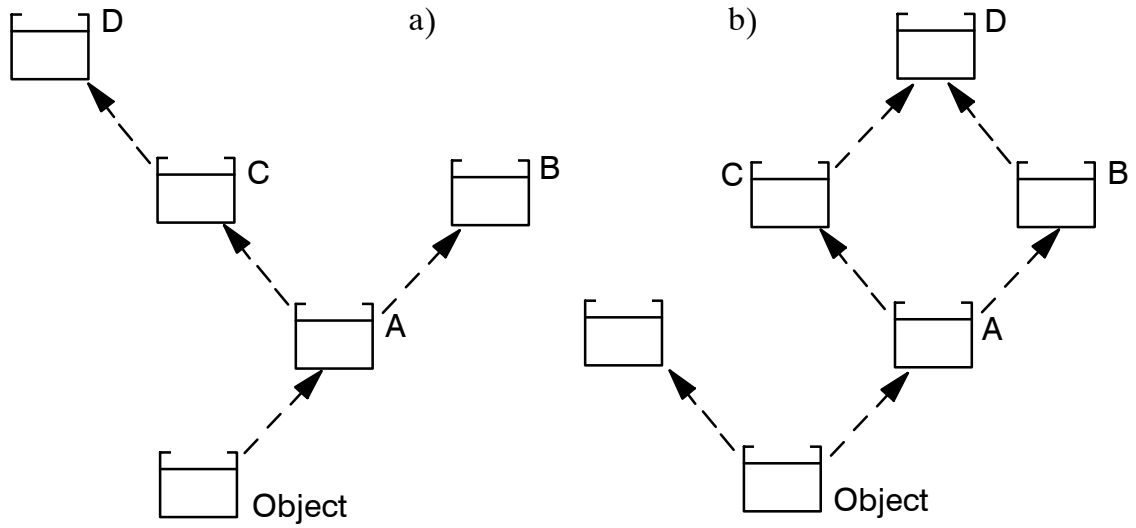
```
ottosFinanzsituation verfuegbarIst.
ottosFinanzsituation erhaelt: 10000 von: ”Oma”.
ottosFinanzsituation insgesamtAusgegFuer: ”Auto”.
ottosFinanzsituation erhaelt:(OmasFinanzsituation verfuegbarIst) von: ”Oma”.
```

- - Botschaftsaustausch
  - zur Def. der Semantik
  - zur Impl. von Smalltalk
  - zentraler universeller Mechanismus
    - arithm. Ausdrücke
    - (Zuweisung)
    - (Zurückliefern eines Werts)
    - Kontrollstrukturen (durch lazy evaluation)
      - sind Methoden vordef. Klassen

- - Ansprechen des Objekts auf das Methode angewandt wird durch self

```
gibAus: betrag fuer: zweck
        | bisherigeAusgaben |
bisherigeAusgaben ← self insgesamtAusgegFuer: zweck.
    ausgaben at: zweck
      put: bisherigeAusgaben + betrag.
    verfuegbaresGeld ← verfuegbaresGeld - - betrag
```

-- Unterklasse -- Oberklasse

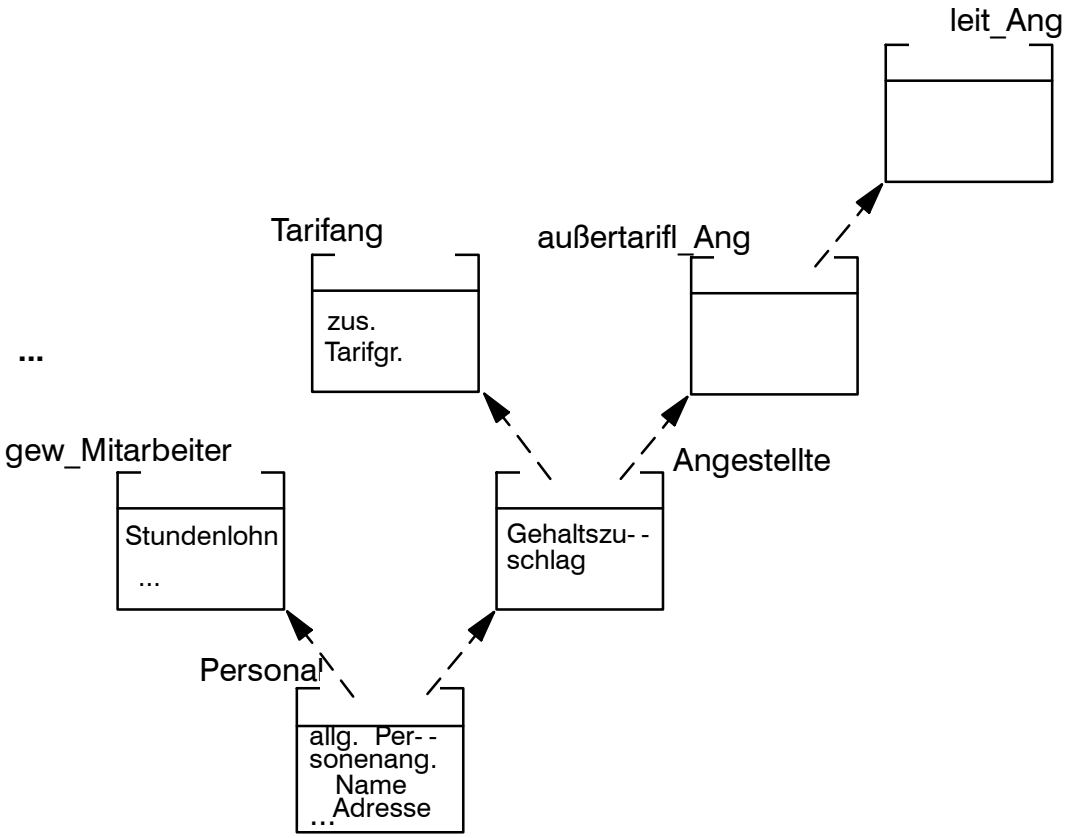


Semantik

Unterklasse B erbt alle Eigensch. der Oberklasse A
sind für Objekte der Unterklasse verfügbar, soweit nicht redefiniert
Objekte von B haben zusätzl. noch die in A fest-gelegten Eigenschaften
B ist Spezialisierung von A, A Generalisierung von B
Vererbung sollte nur Schnittstelle betreffen:
Meth. von A auch für Objekte von B anwendbar
Vererbungsbeziehung transitiv: Vererbungskette

Einfachvererbung -- Mehrfachvererbung

- - Beispiel Unterklasse, Vererbung



- - Top- -down
  Bottom- -up Auftragung der Bäume

  In diesem Sinne ist eine Spezialisierung abstrakter
  nämlich weiter entfernt von Basismaschine

  (Faßt man abstrakt als allgemeingültig auf, so ist
  die Oberklasse abstrakter)

- Jedes Objekt ist Objekt genau einer Klasse:
  - anwendbare Methoden die der Klasse plus
    - Vererbungspfad
  - Exemplarvariable der Klassendef. plus impl.
    - Exemplarvariable aller Oberklassen

- Erklärung des Vererbungsmechanismus durch Impl.:
  - Suche auf Vererbungsstruktur zur Laufzeit
    - bei Klasse gefunden
    - Übergang zur Oberklasse usw.
    - doesNotUnderstand: ...

- mehrere Methoden mit gleichem Botschaftsmuster
  - in verschiedenen Pfaden
  - auf einem Pfad

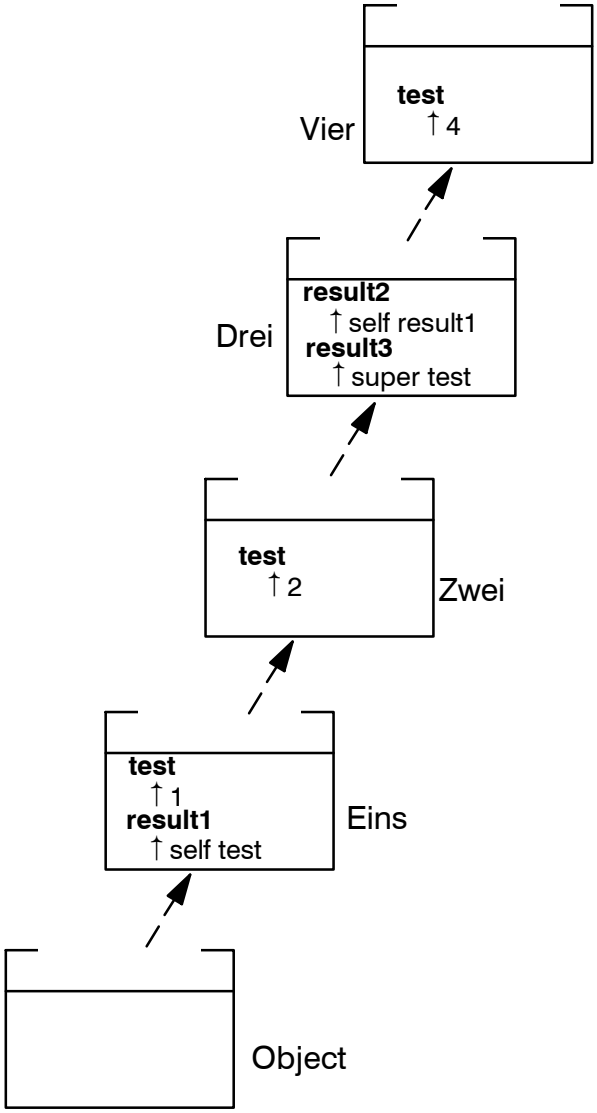
-- self und super

a)

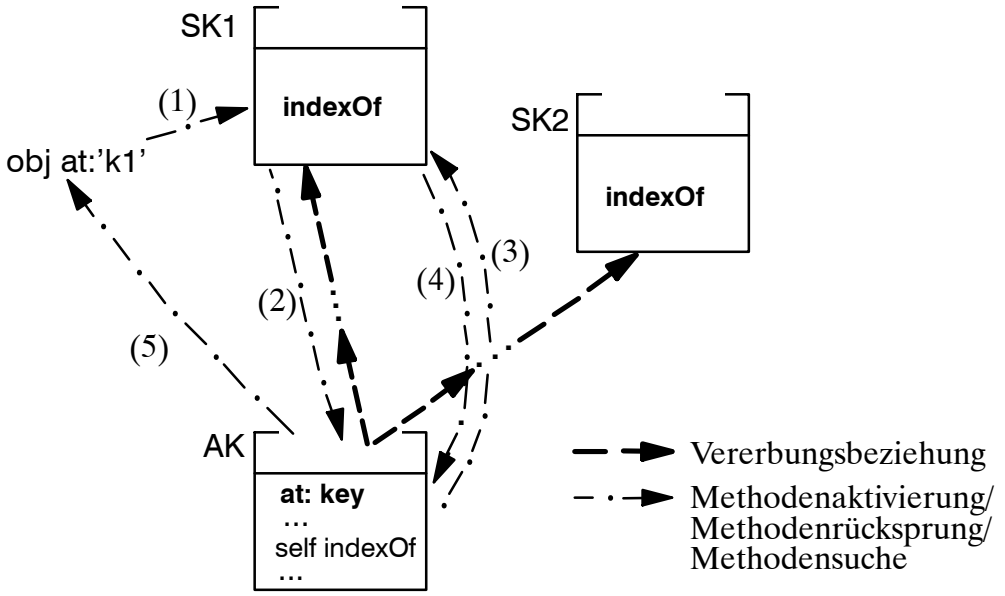
```
beispiel1 ← Eins new.
beispiel2 ← Zwei new.
beispiel1 test.        1
beispiel1 result1.     1
beispiel2 test.        2
beispiel2 result1.     2

beispiel3 ←Drei new.
beispiel4 ← Vier new.
beispiel3 test.        2
beispiel4 result1.     4
beispiel3 result2.     2
beispiel4 result2.     4
```

b)

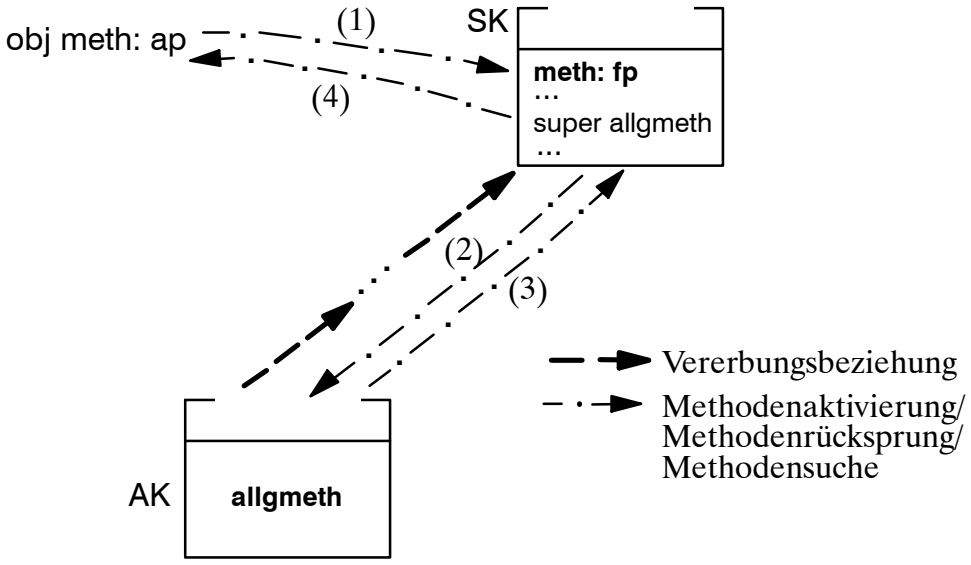
```
beispiel3 result3.     2
beispiel4 result3.     2
```



-- typ. Verwendung: spez. Methoden



-- typ. Verwendung: allg. Methoden



-- Oberklassen
    normal: mit Objekten
    ohne Objekte: abstrakte Oberklassen

-- Smalltalk auf Flexibilität ausgelegt
    Variable: Zeiger, untypisiert
    Methode: untypisiert
        keine Untersch. nach Parametertyprofil
    Botschaftsaustausch zur Laufzeit
    Blöcke verschickbar u. an beliebiger Stelle ausgef.

-- geht zu Lasten der Sicherheit und Effizienz
    Compilezeitabprüfungen fehlen
    Sprachkonstrukte führen zu unübersichtl. Progr.
    Programm unsicher und schwer zu warten
    Zeiger, Botschaftsaustausch: ineffizient

-- Programmiersprachenentwurf

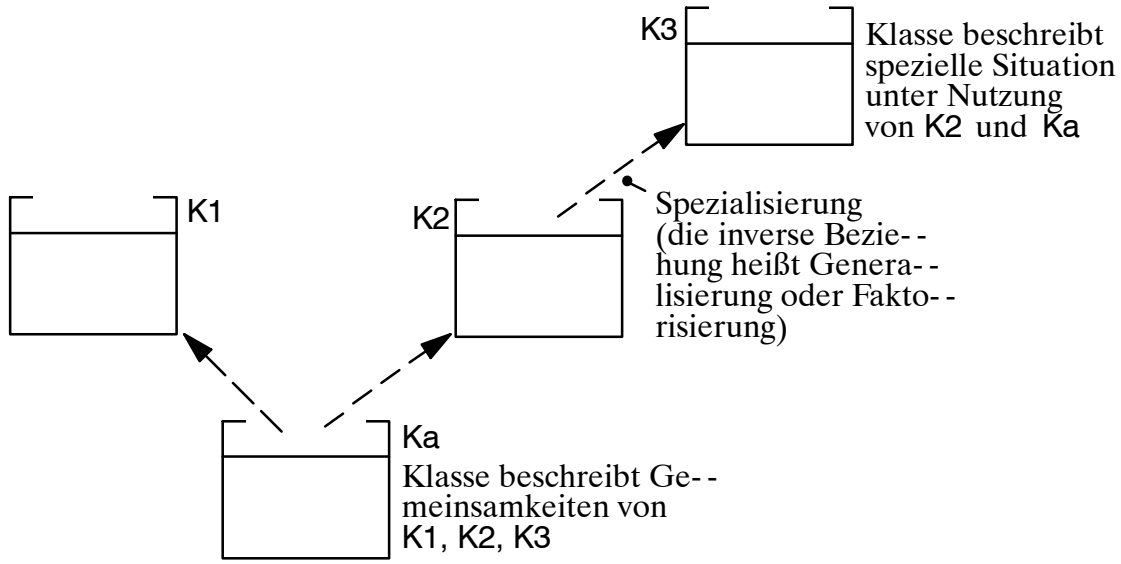
Flexibilität		Sicherheit
Erweiterbarkeit	<==>	Übersichtlichkeit
Einfachheit		Effizienz
Uniformität		
Smalltalk		*Ada*

-- objektorientierte Ansätze
    Einfachvererbung, Mehrfachvererbung
    von welcher Klasse (Typ) ist ein Objekt
    Spezialisierung
        Wertebereichseinschränkung
        Operation eingeschränkt, umbenannt usw.

5.9 Objektorientierte Architektur- - modellierung

- - Zielsetzung
  objektorientierte Konzepte zur Architekturmod.
  Verbindung zu den anderen Konzepten

- - Zusammenf. d. objektorient. Strukturierungshilfsmittel
  Oberklasse - - Unterklasse: Spezialisierung
  Vererbungs - - Teilhierarchie (Faktorisierung, Generalisierung)
  spez. Methoden für spez. Objekte
  allg. Methoden für spez. Objekte
  spez. Methode von allg. induziert
  allg. Methode durch spez. induziert



- - objektorientierte Programmerstellung
    Bottom- -up- -Entwicklung
    neue Klassen hinzu
    Vererbungsstruktur wächst von unten nach oben

- - reichhaltiger Satz vordef. Klassen (Vererbungs- -
  hierarchie vorgegeben
  z.B. in Smalltalk
    Zahlen
    Datenstrukturen/Dateistrukturen
    Prozeßverwaltung
    Graphik
    Kontrollstrukturen

- - Komplexität d. Sprache - - Komplexität d. vordef. Klassen
  Sprache einfach, unif. Standard komplex
  Sprache komlex Standard (etwas) einfacher

- - objektorient. Sprachen u. Entw. großer Softwaresysteme
    Klarheit verschaffen über vorhandene Vererbungs- -
      strukturen nicht einfach
    Hinzufügung neuer Vererbungs- -Teilhierarchien

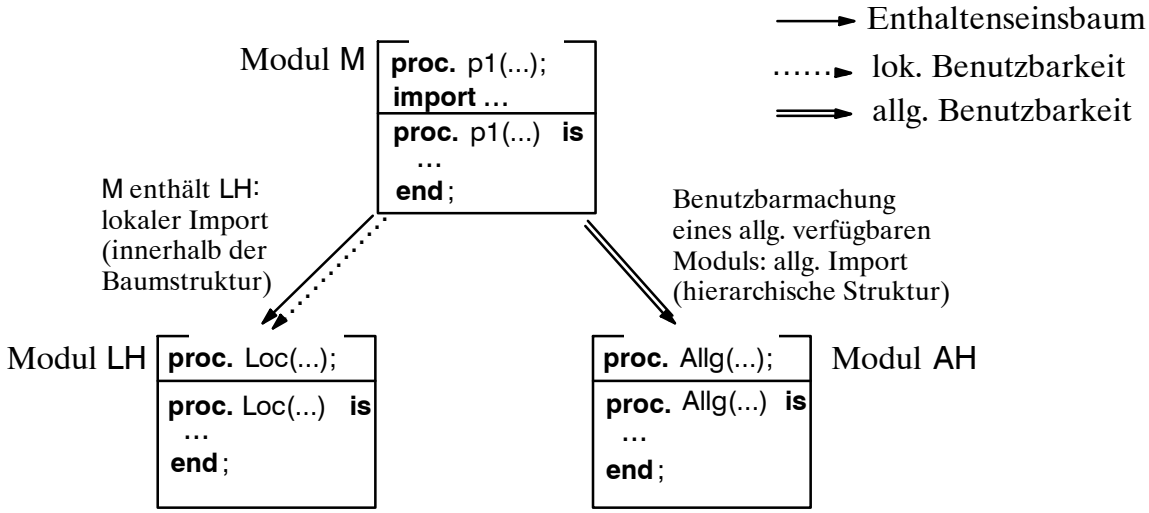
- - objektorientierte Sprachen und Wiederverwendbarkeit
    vordef. Vererbungsstruktur offen
    neue Vererbungsteilhierarchien offen
      Anwendungsbereich
      Firmenkontext
      Bausteine für Klasse von Systemen

- - Offenheit macht Probleme
    Kennenlernen der verwendbaren Klassen
    alle Klassen der Änderung zugänglich

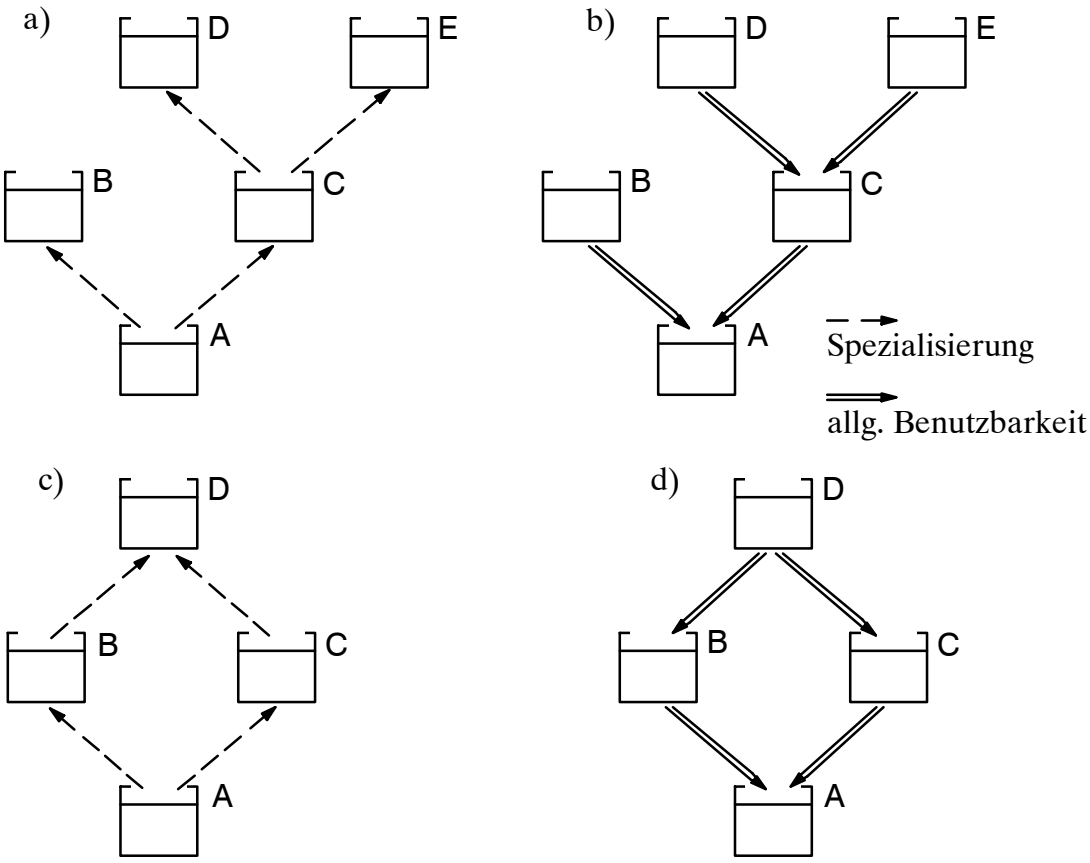
neue Teilhierarchien nicht für jeden interessant
(Umweltverschmutzung)

- - Abgrenzung zu bisherigen Konzepten
nur Datentypmodule
nur Vererbungsbeziehung

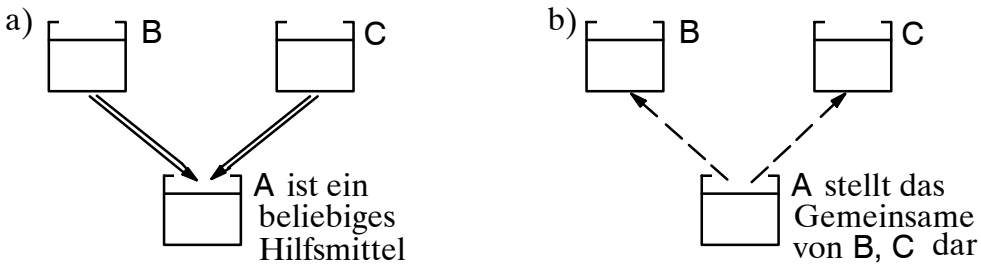
Vererbung hat nichts mit Enthaltensein zu tun



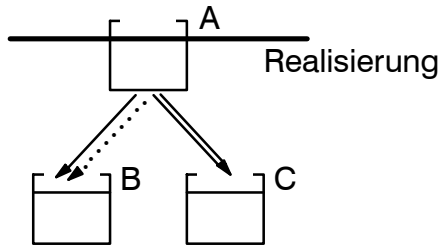
-- Vererbung und allg. Benutzbarkeit

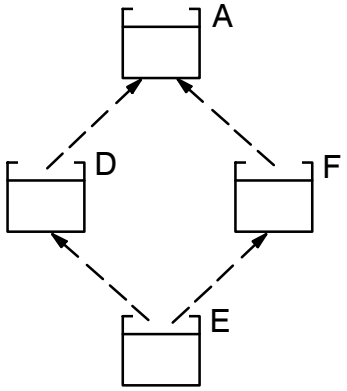


a), b) Einfachvererbung, c), d) Mehrfachvererbung

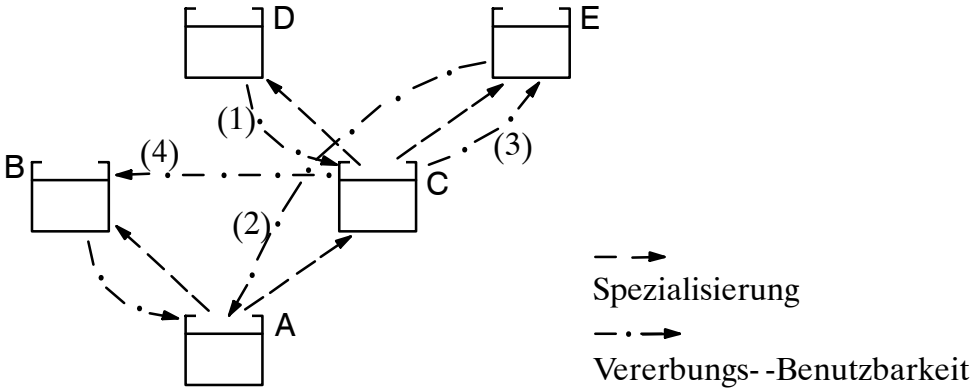


-- Objektorientierung und Datenabstraktion





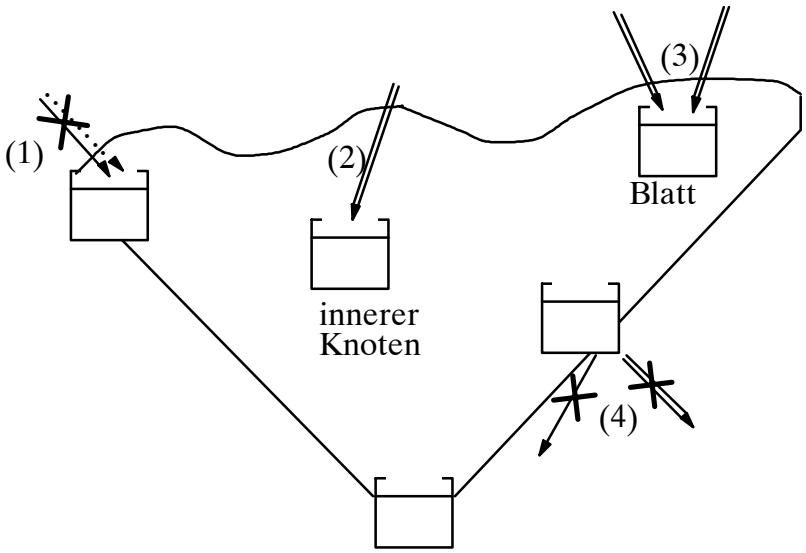
-- Vererbung als Strukturbez. unseres Modulkonzepts



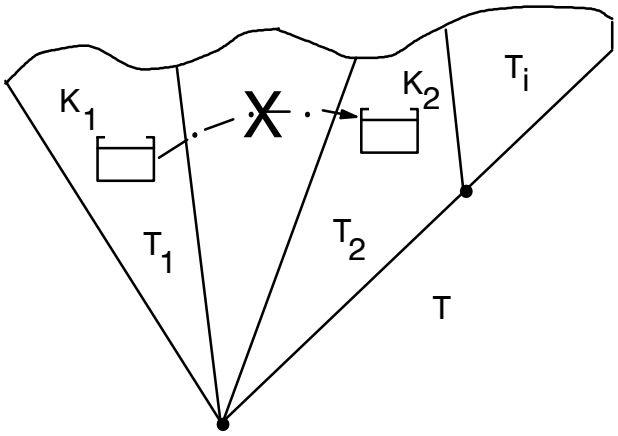
explizite Festlegung der Benutzbarkeit
Vererbungs- -Benutzbarkeit: Kanal
Vielzahl von Vererbungs- -Benutzbarkeit spiegelt innere Komplexität wider!

-- Geben Flexibilität auf, gewinnen Sicherheit:
   Löschen einer Klasse, keine hängenden Methoden
   Sieht, welche Klassen man löschen kann
   Einfügen von Klassen: Vorabfestlegung der zu nutzenden Ressourcen: Methodensuche ist Mechanismus zur Spezifikationszeit
   Wird Mechanismus der spez. Nebenrechnung angewandt: Sieht sofort, wo solche spez. Methoden vorzusehen sind

-- Verankerung von Vererbungshierarchien in Software-Architekturen



-- neue Kandidaten für Teilsysteme



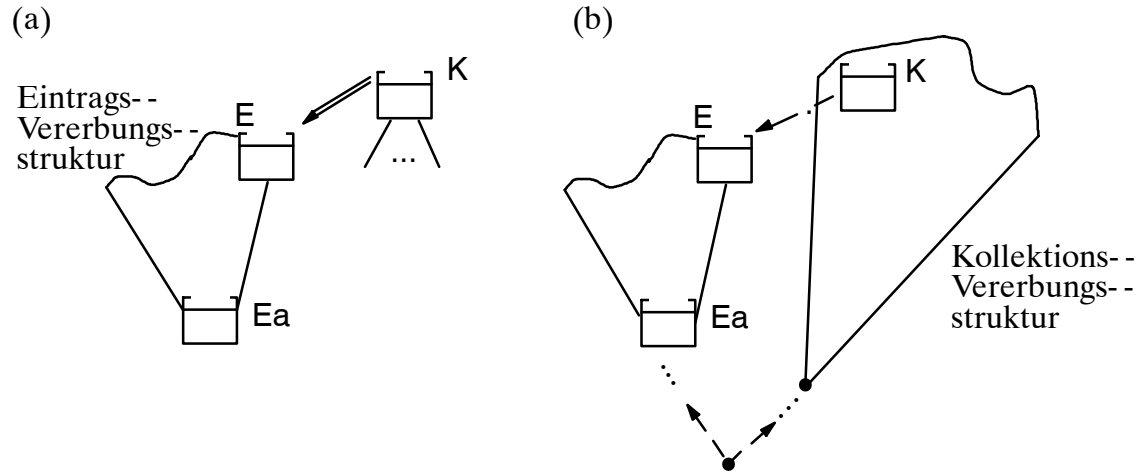
Vererbungs--Teilhierarchie T_1, T_2, T_i
gesamte Vererbungshierarchie T

-- Wo in Softwarearchitektur ist objektorientiert zu modellieren?

DA--Fortsetzung $\S$ Betr. von DA--Anwendungen

Einträge
Kollektionen
Geräte
Layout

- - neue Modellierung der Eintrags- -Kollektions- -Sit.



- - Objektorientierung und Generizität
  G.: Parametrisierung bel. Module, Teilsysteme
  O.: Ähnlichkeiten zwischen Modulen

  ein mögl. Zusammenspiel:
  Eintrags- -Vererbungsstruktur ist generisch
  z.B. Eigenschaft des Schlüssels
  Anzahl der Einträge in der Kollektion ist generisch
  etc.

5.10 Zusammenfassung

- - Anwendungen des einf. Modulkonzepts für Teil- - architekturen:
    - Unterscheidung funkt. Abstr. und Datenabstr.
    - Zusammenspiel fm und ado- -, adt- -Module
    - Ein- -Aussgabe
        - Geräteverkapselung
        - Layoutverkapselung
    - Teilarchitekturen unterhalb von DA- -Modulen
    - Eintragskollektions- -Situationen und ihre Arch.
    - Anwendung auf Nebenläufigkeit

- - Erweiterungen
    - Teilsysteme
    - Generizität
    - Objektorientierung

- Zuordnung zu Modellierungsprinzipien:
  - Abstr.: Module, Schichten in Softwarearch., Teilsysteme
  - Strukturierung:
    - Hierarchiebildung
      - Enthaltens., allg. Benutzbarkeit, Vererbung
    - Modularisierung
      - Module, Teilsysteme
    - Lokalität
      - Schnittstelle--Rumpf, funkt. Abstr. und DA, Enthaltenseinsbäume, Schnittstelle--Rumpf v. Teilsyst., (Generalisierung), Teilarch. unterh. d. Schnittst. von Modulteilsyst.
  - Mehrfachverwendung
    - Module, Teilsyst., gen. Module/Teilsyst., Vererbungshierarchie
  - Redundanz
    - Schnittstelle--Rumpf, Export--Import bei allen Benutzbarkeitsbez., erlaubt kontextsens. Prüfungen
  - Prinzip der geringsten Verwunderung:

- Verbesserung
  - Wartungsproblematik
    - Änderungen lokal in Architektur durch Anw. der DA und durch Struktur--/Importbez.
  - Wiederverwendbarkeit
    - Wiederverwendbarkeit als Baustein
    - Wiederverwendung einer gesamten Arch. nach Modifikation

- - Welche weiteren Konzepte auf Architekturebene sinnvoll?
    1. weitere Arten von Modulen:

    2. weitere Arten von Beziehungen:

    3. weitere Konsistenzbeziehungen

6. VORBEREITUNG FÜR DEN EINSATZ: ÜBERTRAGUNG IN PRO-GRAMMIERSPRACHEN

- Abbildung Modulkonzept und sprachl. Ausprägung auf gängige Programmiersprachen

- Übertragung vollständig durch Werkzeuge als Teil einer Entwurfs--Umgebung
  - strukturbezogene Editoren (Sprache, Methode)
  - Analysatoren
  - Browser
  - Integrationswerkzeuge
  - etc.
  - --> Softwareentwicklungs--Umgebung

- Übertragung durch einen Unparser
  nicht isoliertes Werkzeug
  erzeugt syntaktisch (k.f. und k.s.) richtige Vorgaben

- Verzichten hier auf Vorstellung von Werkzeugen
  - zeigen, daß es auch ohne Werkzeuge geht
  - lernen Modulkonzept unter neue Perspektive
    - (autom. Transf.) kennen: Vertiefung

- Modulkonzept einfach (Kap. 4) detailliert
  Erweiterungen kursiv

6.1 Probleme der Übertragung

-- Probleme aufzeigen
Lösungen angeben
hängen vom Niveau der Programmiersprache ab
(Ada versus FORTRAN)

-- Hat Programmiersprache Modulkonstrukt?
Zusammenf. versch. Ressourcen an der Schnittstelle,
Abschottung des Rumpfs
Sonst
geistig zusammenfassen
Zusammenfass. durch text. Reihenf. u. Kommentar
Disziplin

-- Hat Programmiersprache Konstrukte auf der Ebene der Beziehungen zwischen Modulen (z.B. Importklauseln)?
sonst Simulation wie eben

-- Analog Konsistenzbedingungen

-- Übertragung nie unmöglich
jedoch mehr oder minder schwierig
Dürfen keine Konzepte auf Architekturebene verwendet
werden, die "nicht" abbildbar sind (z.B. Rekursion und FORTRAN)

- - Klärung ”Modulkonstrukt” von Programmierspachen:
  lassen Zusammenf. beliebiger Ressourcen zu
  Modularten: fm, ado, adt sind dann ”semantische” Verwendung dieser Konstrukte

- - Betrachten drei Programmiersprachenklassen und jeweils (mind.) einen Vertreter

  - Spr. ohne Modulkonstrukt mit losen Einheiten, flach
    keine Hierarchie, alles erlaubt an Vernetzung,
    Konsistenz wird nicht überwacht
    FORTRAN, Basic, Cobol, C, Assembler
    haben meist statische Speicherverwaltung

  - Blockstrukt. Programmierspr. ohne Modulkonstrukt
    Baumstrukturen durch Enthaltensein
    Gültigkeit, Sichtbarkeit, eine Einheit, globale Konsistenz
    Algol 60, Algol 68, Pascal etc.

  - Klass. neuere Programmierspr. mit Modulkonstrukt
    Ineinanderschachtelung, getrennte Übersetzung, globale Konsistenz
    Ada, Modula- -2, Turbo- -Pascal etc.

6.2 Sprachen mit unabhängigen Programm- - einheiten: Beispiele FORTRAN, C

- - FORTRAN IV als Repräsentant für Sprachen mit losen Einheiten
  nicht Module, sondern Einheiten, aus denen Modul- - rümpfe zusammengesetzt sind
  separate Übersetzung

- - Übertragung von Modulen der obigen Arten
  Übertragung von Beziehungen der obigen Arten

-- Übertragung funktionaler Module

```
C     ***************************************************
C     *                                                 *
C     *  functional module ZEICHNE_FUNKTION is          *
C     *     Eingabedaten jeweils in der Parameterliste, *
C     *     Ausgabedatum ist das erstellte Plotterfile,
*
C     *     procedure POLYLN(X,Y: in FELD;              *
C     *                      XTEXT, YTEXT,UETEXT: in STRING);  *
C     *     procedure INTLN(...);                       *
C     *     ...                                         *
C     *     Angabe der Semantik von ZEICHNE_FUNKTION:   *
C     *     ...                                         *
C     *                                   end ZEICHNE_FUNK-
TION;-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -*
C
C     *                  module body ZEICHNE_FUNKTION
is- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -*
C
      SUBROUTINE POLYLN(X,Y,XTEXT,YTEXT,UETEXT)
        REAL X(100), Y(100)
        INTEGER XTEXT(20), YTEXT(20), UETEXT(20)
        ...
        RETURN
      END
C
      SUBROUTINE INTLN(X,Y,XTEXT,YTEXT,UETEXT)
        REAL X(100), Y(100)
        INTEGER XTEXT(20), YTEXT(20), UETEXT(20)
        ...
        RETURN
      END

      ...


C     *  end ZEICHNE_FUNKTION;                        *
C     ***************************************************
```

- - Übertragung ado- -Modul

```
C      *****************************************************
*
C      *                                                   *
C      * abstract data object module INTEGER_STACK is      *
C      *   procedure PUSH(X: in INTEGER);                  *
C      *   procedure POP;                                  *
C      *   function READ_TOP return INTEGER;- -- -RDTOP     *
C      *   function IS_EMPTY return BOOLEAN;- -- -ISEMTY    *
C      *   function IS_FULL return BOOLEAN;   - -- -ISFULL  *
C      *   ...                            *
C      *   Angabe der Bedeutung der Operationen:           *
C      *   ...                            *
C      *                                   end        INTE-
GER_STACK;- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -*
C
C      *          module      body      INTEGER_STACK      is
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -*
C
       SUBROUTINE PUSH(ELEMNT)
         INTEGER STACK(100), POINTR, ELEMNT
         COMMON /STDATA/ STACK, POINTR
         ...
         RETURN
       END
C
       SUBROUTINE POP
         INTEGER STACK(100), POINTR
         COMMON /STDATA/ STACK, POINTR
         ...
         RETURN
       END
C
       INTEGER FUNCTION RDTOP
         INTEGER STACK(100), POINTR
         COMMON /STDATA/ STACK, POINTR
         ...
         RETURN
       END
C
       ...
       BLOCKDATA
         INTEGER STACK(100), POINTR
         COMMON /STDATA/ STACK, POINTR
         ...
       END
C      * end INTEGER_STACK;                                *
C      *****************************************************
*
```

- - Abbildung Modulbeziehungen
  FORTRAN unstrukt. Haufen loser Programmeinheiten,
  aus denen Modulrümpfe zusammengesetzt sind
  somit auch keine Modulbeziehungen, Enthaltensein,
  lokale Benutzbarkeit, allg. Benutzbarkeit
  $ Kommentar und Disziplin

- - lokale Benutzbarkeit

```
C     ***********************************************************
C     *  functional module USERINPUT is                         *
C     *     procedure EINGABE(...);              - - -EINGAB    *
C     *     ...                                                 *
C     *     Die Semantik der Schnittstellenoperationen ist:     *
C     *     ...                                                 *
C     *                                                         *
C     *                                          end            USERIN-
PUT;- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- - -- -- -  *
C     *                                                         *
C     *                              module          body       USERINPUT
is- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -*
C     *     local import from CHECKINPUT                        *
C     *        using CHECKSTRING, CHECKNUM;  - - -CHKSTR, CHKNUM*
C     *     Realisierung obiger Operationen:                    *
            ...

C     *  end USERINPUT;                                         *
C     ***********************************************************


C     ***********************************************************
C     *  functional module CHECKINPUT is                        *
C     *     is contained in USERINPUT;                          *
C     *     procedure  CHECKSTRING(...);         - - -CHKSTR    *
C     *     procedure  CHECKNUM(...);            - - -CHKNUM    *
C     *     ...                                                 *
C     *     Die Semantik der Schnittstellenoperationen ist:     *
C     *     ...                                                 *
C     *                                          end            USERIN-
PUT;- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- - -- -- -*
C     *                                                         *
C     *                        module          body       USERINPUT      is
```

```
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -*
C     *                                                                                 *
C     ...

C     *  end CHECKINPUT;                                                                *
C     ***********************************************************************************
```

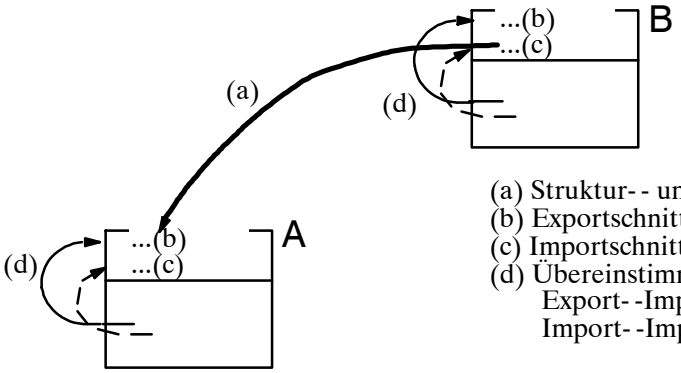
- - allg. Benutzbarkeit
  der FORTRAN- -Gedankenwelt näher
  anal. zu oben:
  general- -imports- -Klausel als Kommentar

- - Übertragung in C
  jeder ”Modul” eine Datei für sep. Übersetzung
  Übertragung durch Werkzeug (Unparser) als Teil
  einer Softwareentwicklungs- -Umgebung

```
/*
 *      DATATYPE MODULE IntStack
 *      EXPORT INTERFACE:
 *              IntStT
 *              void InitSt (St);
 *              void Push (St,El);
 *              void Pop (St);
 *              INT ReadTop (St);
 *              BOOLEAN IsEmpty (St);
 *              BOOLEAN IsFull (St);
 *      IMPORT INTERFACE
 *              FROM IntList IMPORT
 *                  type IntListT
 */
                    extern void InitLst();
                    extern void Insert ();
                    extern void Delete ();
                    extern INT Read ();
                    typdef<...> IntListT;

    typedef < ... > IntStT;

    void InitSt (St)
          IntStT *St;
    {
    }

    void Push (St,El)
          IntStT *St;
          INT El;
    {
    }

    void Pop (St)
          IntStT *St;
    {
    }

    INT ReadTop (St)
          IntStT *St;
    {
    }

    BOOLEAN IsEmpty (St)
          IntStT *St;
    {
    }

    BOOLEAN IsFull (St)
          IntStT *St;
    {
    }

/*
 *      END MODULE IntStack
 */
```

```
/*      DATATYPE MODULE IntList
 *      EXPORT INTERFACE:
 *              IntListT
 *              void InitLst (List);
 *              void Insert (List,El,Pos);
 *              void Delete (List,Pos);
 *              INT Read (List,Pos);
 */

    typedef < ... > IntListT;

    void InitLst (List)
        IntListT *List;
    {
    }

    void Insert (List,El,Pos)
        IntListT *List;
        INT El;
        INT Pos;
    {
    }

    void Delete (List,Pos)
        IntListT *List;
        INT Pos;
    {
    }

    INT Read (List,Pos)
        IntListT *List;
        INT Pos;
    {
    }

/*
 *      END MODULE IntList
 */
```

-- Diskussion: Welche Konsistenzbedingungen muß Quelltext erfüllen, der aus Transf. hervorgeht?



(a) Struktur- - und Benutzbarkeitsbez. grob
(b) Exportschnittstelle detailliert
(c) Importschnittstelle detailliert
(d) Übereinstimmung
Export- -Implementation
Import- -Implementation

Einhaltung durch Disziplin
Bedingungen für (a)
(b)
A (b) und B (c)
A (b) und B (c) insb. Art von A
PiK PiG (d)

-- Ergebnis der Diskussion alle 23 Konsistenzbedingungen aus Tab. 4.37, 4.38, 4.39 ferner, da in den hier betrachteten Sprachen keine Module existieren:

(a) Jeder eingeführte Modul ist von einer der in Kap. 4 angegebenen Arten (funktionaler Modul, abstrakter Datenobjektmodul, abstrakter Datentypmodul) bzw. in Sonderfällen einer der am Ende von Abschnitt 4.4 diskutierten Spezialformen (Typkollektion, Konstantenansammlung).

(b) Die Interna des Rumpfs werden von außerhalb nicht verändert bzw. nicht zur direkten Veränderung benutzt. Hierzu zählen:

(b1) Die lokalen Prozeduren werden nicht von außen aufgerufen.

(b2) Die Datenstrukturen von Datenobjektmodulen werden nicht von außen direkt verändert oder gelesen.

(b3) Die interne Strukturbeschreibung eines abstrakten Datentypmoduls in Form einer Typdefinition wird nicht dazu herangezogen, die Datenobjekte dieses Typs direkt zu verändern oder direkt zu lesen.

6.3 Blockstr. Sprachen ohne Module: Beispiel Pascal

- - Übertragung in Standard- -Pascal
  Probleme ähnlich:
  gedanklich zusammenfassen
  textuell hintereinanderschreiben mit Kommentar
  Disziplin üben

- - damit auch Übertragung einer Sprachklasse:
  Blockstrukturierung
  ohne Modulkonstrukt
  Lokalitätsprinzip das einzige Strukturierungs- -
  prinzip: Baum
  Repräsentanten Algol 60, 68, PL/I, etc.

- - Übertragung einfacher, da Pascal 10 Jahre jünger?

- - Pascal- -spezifische Erschwernis:
  Deklarationsreihenfolge

- - da Übertragungsproblem für alle Modularten analog
  u. bereits Vertrautheit mit Übertragung vorhanden:
  Betrachten nur adt- -Modul- -Übertragung

- -   adt- -Modul in Standard- -Pascal

irgendwo im Konstantendeklarationsteil:

```
(*  Nur fuer den internen Gebrauch im Modul STACK: *)
const STACKMAX = ... ;
•••
```

irgendwo im Typdeklarationsteil:

```
(*  Nur fuer den internen Gebrauch im Modul STACK: *)
type StackT =
                    record
                         SPACE : array D1..STACKMAXF of integer;
                         INDEX : integer
                    end;

•••
```

irgendwo im Prozedurdeklarationsteil:

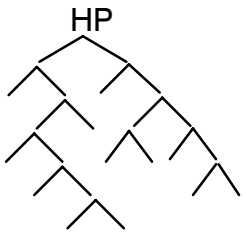
```
(******************************************************
*
               *
*    data type module INTEGER_STACK is
                    *
*         type StackT is private;
                    *
*         procedure PUSH (St: in out STACK_T; El: in INTEGER);
*
*         ...
               *
*         Semantikbeschreibung:
                    *
*         ...
               *
*    end                                                          INTE-
GER_STACK;- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -- -- -- -- -

*    module                        body                        INTEGER_STACK
is- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -*)
          procedure PUSH (var St: StackT; El: integer);
               ...
          begin
               ...
          end;
               •••

(*   end INTEGER_STACK;
******************************************************)
```

-- Übertragung Modulbeziehungen
Idee: Enthaltenseinsbeziehung auf Ineinander--schachtelung
pot. Benutzbarkeit dann durch Gültigkeits--/Sichtbarkeitsregeln
geht nicht: Pascal kennt keine Ineinanderschachtelung für Module, sondern nur für Teile, aus denen Modul--rümpfe bestehen: Prozeduren
=> Ineinanderschachtelung nur durch Kommentar
allg. Benutzbarkeit durch Kommentar

-- Wo liegen die "Module", die über Enthaltensein, lok. Benutzbarkeit, allg. Benutzbarkeit, verbunden sind?
Im obersten Block!

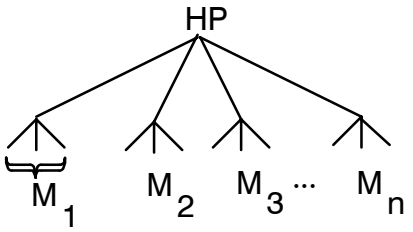
-- Struktur eines so übertragenen Programms

typische Pascal--Programmstruktur



geschachtelte Prozeduren
evtl. tiefer Baum

Struktur eines Programm--systems, das aus einem Architekturdiagramm hervorging



Objekte zu einem Modulrumpf

6.4 Sprachen mit Modulen: Beispiel Ada

- - Klasse der neueren, klass. Programmiersprachen
  Repräsentanten Ada, Modula- -2, Pascal- -Dialekt etc.

- - Bisher für Modul- -Exportschnittstellen und für Modul- -
  rümpfe Ada als Beispielsprache

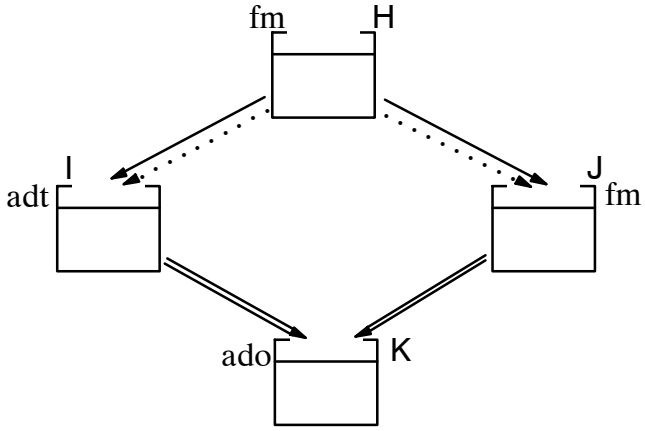
- - Modularten wieder Beschränkung auf adt- -Module

- - adt- -Modul in Ada

```
- -- -data  type ******************************************************
package ITEM_STACK_STENCIL is                                           - -- -
    type ITEM_STACK_TYPE is private;                                    - -- -
    procedure INITIALIZE (ST: in out  ITEM_STACK_TYPE);                 - -- -
    procedure PUSH (EL: in ITEM_TYPE; ST: in out ITEM_STACK_TYPE);      - -- -
    procedure POP (ST: in out ITEM_STACK_TYPE);                         - -- -
    function READ_TOP (ST: in ITEM_STACK_TYPE) return ITEM_TYPE;        - -- -
    function IS_EMPTY (ST: in ITEM_STACK_TYPE) return BOOLEAN;          - -- -
    function IS_FULL (ST: in ITEM_STACK_TYPE) return BOOLEAN;           - -- -
    ST_UNDERFLOW, ST_OVERFLOW, ST_NOT_INITIALIZED: exception;           - -- -
    - -- -Semantikbeschreibung:                                         - -- -
    - -- -...                                                           - -- -
    - -- -...                                                           - -- -
    private                                                        - -- -
      SIZE: constant INTEGER := 100;                                    - -- -
      type  SPACE_T is array (1..SIZE) of ITEM_TYPE;                    - -- -
      type ITEM_STACK_TYPE is                                           - -- -
          record                                                        - -- -
             SPACE: SPACE_T;                                            - -- -
             INDEX: INTEGER range 0..SIZE := 0;                         - -- -
      end record;                                                       - -- -
end                                                    ITEM_STACK_STENCIL;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -

package            body            ITEM_STACK_STENCIL            is
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
  - -- -Implementation der Schnittstellenoperationen:
- -- -
  ...                                                              - -- -
end ITEM_STACK_STENCIL; - -- -************************************
```

- - Übertragung der lok. Benutzbarkeit und der zugrundeliegenden Enthaltenseinsbeziehung

Enthaltenseinsbeziehung auf Ineinanderschachtelung:
im Deklarationsteil eines Modulrumpfs steht Mo- - dul- -deklaration
is- -contained- -in- -Klausel kann weggelassen werden



ganze Rümpfe schachteln: große Quelltexteinheiten
Auslagern der Rümpfe: Stummel

Rümpfe von I, J als getrennte Einheiten
Export- -Schnittstelle als Kommentar
mit is- -contained- -in- -Klausel

Gültigkeitsbereich- -/Sichtbarkeitsregelung entspricht pot. lokaler Benutzbarkeit

local- -import- -Klausel schränkt ein (als Kommentar)

```
- -- -abstract data object ************************************************************
package K is                                                                        - -- -
        ...                                                                         - -- -
end                                                                                   K;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
package                              body                                  K                                  is
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -- -- -- -- -- -- -- -
        ...                                                                         - -- -
end K;  - -- -************************************************************************

- -- -functional ***********************************************************************
package H is                                                                        - -- -
        ...                                                                         - -- -
end                                                                                   H;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
package body H is                                                                   - -- -
        - -- -local import from I using ...;                                        - -- -
        - -- -local import from J using ...;                                        - -- -
                                                                                    - -- -
        - -- -       abstract data type ****************************************     - -- -
        package I is                                                  - -- -         - -- -
            ...                                                       - -- -         - -- -
        end                                                                           I;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -            - -- -
        package body I is separate;- -- -*******************************            - -- -
-------------------
        - -- -functional ***********************************************      - -- -
        package J is                                                  - -- -         - -- -
            ...                                                       - -- -         - -- -
        end                                                                           J;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -            - -- -
        package body J is separate; - -- -*****************************            - -- -
        ...                                                                         - -- -
end H; - -- -************************************************************************

-------------------
with K; use K; separate (H)   - -- -************************************************
- -- -abstract data type package I is                                               - -- -
- -- -       is contained in H;                                                    - -- -
- -- -               ...                                                             Exportschnittstelle
als Kommentar                                - -- -
- -- -end
I;- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
package                              body                                  I                                  is
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -- -- -- -- -- -- -- -
    - -- -general import from K using ...;                                        - -- -
        ...                                                                         - -- -
end I; - -- -************************************************************************



with K; use K; separate (H); - -- -************************************************
- -- -functional package J is                                                       - -- -
- -- -       is contained in H;                                                    - -- -
```

-- allg. Benutzbarkeit (gleiches Bsp.) für Modul K
Abb. auf Bibliothekseinheit, with--Klausel
benötigte Ressourcen durch Kommentar

-- Übertragung des Modulkonzepts vollständig durch Unparser möglich

-- Welche der Konsistenzregeln müssen in Ada durch Disziplin erbracht werden?

Tab. 4.37, 4.38, 4.39 und 6.6

von 24 Bedingungen auch in Ada 16 durch Disziplin!

-- Diskussion geht von bestimmter Abbildung aus
Enthaltensein --> Ineinanderschachtelung
lok. Benutzbarkeit --> Untereinheiten
falls ebenfalls auf Bibliothekseinheiten und with--Klausel abgebildet wird, noch mehr Disziplin nötig

-- Abbildung der Erweiterungen des Modulkonzepts
Überlegungen gelten in etwa für alle vorgestellten
Sprachklassen

-- Teilsystem--Abb.
Enthaltenseinsbaum: Abb. mit Modulabb.
Enthaltenseinsbez. erledigt
lok. Benutzb.

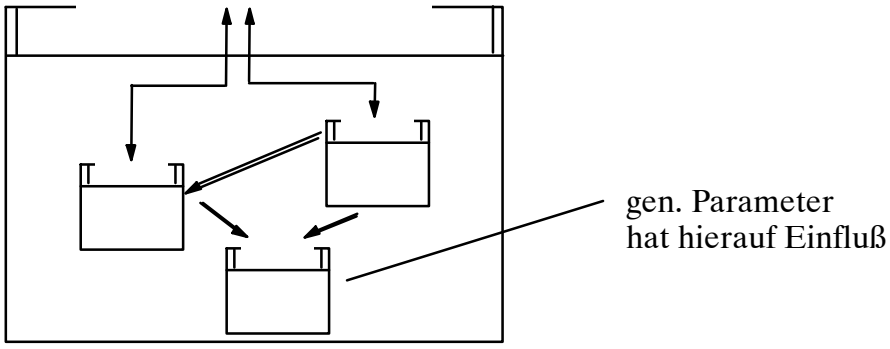
Bibliothek vordef. Module:
Abb. auf Bibl. vordef. Module
d.h. nicht auf Ada
Abb. auf Summationsmodul (s.u.)

allg. Teilsystem: Abb. auf Summationsmodul
nur ganze Schnittst. exportierbar

```
- - - -data object subsystem ****************************************
  package Entry_Coll is                                                - -- -
- -- -          data type  ***************************************- -- -
                package Entry is                                     - -- - - -- -
                        type Entry_Den_Type is private;              - -- - - -- -
                        procedure Create_and_Init (El: out Entry_Den_Type);   - -- - - -- -
                        ...                                          - -- - - -- -
                        - -- -Semantikbeschreibung: ...              - -- - - -- -
                private                                              - -- - - -- -
                        type  Entry_Den_Type is ...                  - -- - - -- -
                end Entry; - -- -*************************************- -- -
- -- -          data object  ************************************ - -- -
                package Coll is                                      - -- - - -- -
                        procedure Store (El: in Entry_Den_Type);     - -- - - -- -
                        ...                                          - -- - - -- -
                        - -- -Semantikbeschreibung: ...              - -- - - -- -
                end Coll;  - -- -************************************- -- -
  end                                                                  Entry_Coll;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -- -- -- -- -- -- -

  package                 body                 Entry_Coll                 is
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -
                ...                                                  - -- -
                package body Entry is - -- -*************************- -- -
                        - -- -Abb. der Teilarchitektur von Entry wie oben skizziert   - -- - - -- -
                        - -- -Rumpf kann ausgelagert werden          - -- - - -- -
                        ...                                          - -- - - -- -
                end Entry;  - -- -***********************************- -- -
                package body Coll is  - -- -*************************- -- -
                        - -- -Abb. der Teilarchitektur von Coll wie oben skizziert    - -- - - -- -
                        - -- -Rumpf kann ausgelagert werden          - -- - - -- -
                        ...                                          - -- - - -- -
                end Coll;  - -- -************************************- -- -
  begin                                                                - -- -
     ...                                                               - -- -
  end Entry_Coll;  - -- -*******************************************
```

-- Abb. der Generizität

kein Mechanismus vorh. für allg. gen. Teilsysteme
Generizität für Pakete möglich
manche Teilsyst. auf Summationspakete abbildbar
im Rumpf des Teilsystems dürfen keine allg. Benutzbarkeit oder Vererbung auftauchen, wobei die davon betroffenen Module generisch sind



-- Abbildung der Objektorientierung

Generalisierung mithilfe allg. Benutzbarkeit
Disziplin, daß nicht ein beliebiges allg. Hilfsmittel eingeführt wurde
Vererbungs--Benutzbarkeit durch Kommentar
nur in eingeschränktem Maße Objektorientierung (self, super, Methodensuche nicht simulierbar)
Erweiterungen der Schnittst. um die von Oberklas--sen
Spracherweiterungen: Ada ++, Classic--Ada, InnovAda

```
- -- -data type *** class ****************************************- -- -
package SET is                                                   - -- -
   - -- -is a COLLECTION;                                         - -- -
   - -- -inheritance import from COLLECTION using ...;           - -- -
   type SET_TYPE is private;                                     - -- -
   procedure INIT_S(FS: in out SET_TYPE);                        - -- -
   procedure PUT_IN(FS: in out SET_TYPE; FE: in EL_TYPE);        - -- -
   function IS_EL_OF(FS: in SET_TYPE; FE: in EL_TYPE) return BOOLEAN;- -- -
   ...                                                           - -- -
   - -- -Semantikbeschreibung:                                    - -- -
   ...                                                           - -- -
   private                                                       - -- -
      ...                                                        - -- -
end                                                                SET;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -

with          COLLECTION;          use          collection;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
package body SET is                                              - -- -
   ...                                                           - -- -
end SET; - -- -*********************************************- -- -
```

```
- -- -data type *** class  **************************************- -- -
package COLLECTION is                                            - -- -
   - -- -is a ...;                                               - -- -
   - -- -inheritance import from ...;                            - -- -
   type COLL_TYPE is private;                                    - -- -
   procedure INIT_C(FC: in out COLL_TYPE);                       - -- -
   ...                                                           - -- -
   - -- -Semantikbeschreibung:                                    - -- -
   ...                                                           - -- -
   private                                                       - -- -
      ...                                                        - -- -
end                                                        COLLECTION;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -- -- -

with                         ...;                              use...;
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
- -- -- -- -- -- -- -- -- -- -- -
package body COLLECTION is                                       - -- -
   ...                                                           - -- -
end COLLECTION; - -- -***************************************- -- -
```

6.5 Zusammenfassung

- Übertragungsdiskussion fruchtbar
  Anwendbarkeit auf div. Programmiersprachen nachgewiesen
  insb. die praktisch bedeutsamen
  Modulkonzept vertieft

- Modulkonzept
  stärker von Modulbez. und Konsistenzbed. geprägt
  Unterschied auch zu Ada!

- Sprachklassen kennengelernt

- Übertragungsprozeß durch Werkzeuge als Teil einer Softwareentwicklungs--Umgebung

- Programmiersprache, die alle für Abb. nötigen Grundkonzepte hat, wird noch vermißt

7. EINÜBUNG DURCH BEISPIELE: EINIGE SOFTWARE- - ARCHITEKTUREN

- Kap. 4 Module
  Kap. 5 Teilarchitekturen
  hier: ganze Architekturen
  in Architekturdiagrammform

- Beispiele
  - Telegrammdatenbeispiel wiederaufgreifen
    neue Lösung
    Adaptabilität bezgl. Realisierungsänderungen
    Adaptabilität bezgl. Erweiterungen

  - analog Adreßverwaltungssystem

  - Abstiegscompiler

  - Softwareentwicklungs- -Umgebung: Grobarchitektur

- Batch- -Systeme: Telegrammdatenbeispiel
  Abstiegscompiler

  interaktive Systeme: Adreßkartenbeispiel

- neu:
  - Bedeutung der Diskussion ”Was kann sich ändern?”

  - Layout als DA- -Anwendung vertiefen

7.1 Wiederaufgreifen von Beispielen: Angabe der Architektur

- Disk. ”Was kann sich ändern?”
  nach Erstellung der Anforderungsdefinition
  nach Erstellung der Architektur
  Bedeutung: Erweiterungen mitbedenken
  andere Realisierungsentscheidungen mitbedenken

- Jetzt Diskussion auf einmal
  liefert Antwort auf beide Fragen

- Bedeutung:
  liefert uns ”automatisch” alle Stellen,
  wo Datenabstraktion beachtet werden muß

- TD- -Beispiel: Was kann sich ändern?

-- Adreßkarten- -Kästen- -Beispiel: Was kann sich ändern?

- globale Änderungen

Die Einträge in den Datenbeständen (Karteikästen) besitzen eine festgelegte Struktur (siehe auch Tab. 7.4), d.h. sie sind aus Komponenten zusammengesetzt.
Die Einträge in einem Datenbestand besitzen zusätzlich ein Attribut, mehrere Attribute oder beliebig viele Attribute, nach denen selektiert werden kann.
Die Sortierung der Kästen- -Namensliste, der Karten- -Namensliste eines Kastens und die Sortierung der Karten eines Kastens erfolgt nicht automatisch, sondern auf Kommando des Benutzers.
Bei der Eingabe einer Karte erfolgt zusätzlich eine Abprüfung auf syntaktische Korrektheit einzelner Komponenten (z.B. darf ein Name nur aus Buchstaben und bestimmten Sonderzeichen bestehen). Ferner erfolgt eine Abprüfung von Werten einzelner Komponenten (z.B. $0 \leq$ Alter ≤ 150) bzw. eine Überprüfung der gegenseitigen Abhängigkeit von Werten (z.B. gedient => männlich).
Es können neue Datenbestände dadurch erzeugt werden, daß interaktiv aus einem Datenbestand ein Teildatenbestand ausgewählt wird.
Neue Datenbestände können durch Mengenoperationen (Vereinigung, Durchschnitt, Differenz) aus vorhandenen Datenbeständen gebildet werden.
Das Löschen von Einträgen in einzelnen Datenbeständen kann durch Angabe einer Löschmenge erfolgen, die Einträge aus verschiedenen Datenbeständen enthalten darf.
Datenbestände können aus einem Datenbestand oder aus mehreren Datenbeständen durch assoziative Anfragen selektiert werden. Dabei können zusätzlich Attributwerte unvollständig angegeben werden, z.B. `A*` für alle Namen, die mit `A` beginnen.
Das Selektieren von Datenbeständen geschieht über eine Anfragesprache (z.B. SQL- -ähnlich).
Der Bediener des Systems kann gleichzeitig mit mehreren Datenbeständen umgehen (z.B. für die manuelle Selektion; für assoziative Anfragen, die verschiedene Datenbestände übergreifen; für gleichzeitige Änderung mehrerer Datenbestände).
Das System erlaubt mehrere Bediener, die gleichzeitig arbeiten. Dabei kann es verschiedene Zugriffsmodelle geben: Jeder Bediener kann einen Datenbestand verändern, beliebig viele lesen; er kann mehrere verändern, die dann für alle anderen gesperrt sind; alle Bediener dürfen alle Datenbestände verändern und lesen; bei "gleichzeitiger" Veränderung eines Datenbestands müssen verschiedene Revisionen verändert werden, die später interaktiv verschmolzen werden usw.
Das Karten- -Kästen- -System besitzt zusätzliche Hilfe- -Funktionen, die den Bediener unterstützen, wenn er nicht weiterweiß.
Für alle Operationen des Systems gibt es einen allgemeinen Undo- - und Redo- -Mechanismus.
Zu Beginn seiner Arbeit muß sich der Bediener dem System bekanntmachen. Dabei wird seine Zugangsberechtigung abgeprüft.
Das System wird zu einem Personalverwaltungssystem ausgebaut:

Für jede Person gibt es allgemeine Daten (Name, Vorname, Geburtsdatum etc.).
Es gibt zusätzlich Daten, die seine Stellung und seine Aufgaben betreffen.
Es gibt zusätzlich Daten, die seine Arbeitszeit betreffen.
...
Auf allen diesen Daten zu einer Person können Eingaben bzw. Veränderungen vorgenommen werden.
Die Daten zu verschiedenen Personen können miteinander verknüpft werden (z.B. Ehemann <--> Gattin).

Das System wird zu einem Bibliotheks- -Recherchesystem ausgebaut (Angabe der Funktionalität als Übung für den Leser).

- Änderungen Bedieneroberfläche bzw. Druckausgabe
  Verwendung von Gestaltungselementen (Fenster, Masken, Menüs etc.)
  Gestaltung unterschiedlicher Oberflächen mit diesen Elementen
  Austauschbarkeit von beiden
  Analoges gilt für Druckausgabe

RE, A Die Ein- -/ Ausgabe erfolgt in Fenstern auf dem Bildschirm ggf. in bestimmten Masken. Es gibt z.B. eine Eingabemaske für einen Kastennamen und eine für die Bearbeitung von einzelnen Karten. Die Ausgabe der Kästen- -Namensliste bzw. der Karten- -Namensliste erfolgt in Fenstern.

RE, A Die Selektion einzelner Komponenten von Einträgen bei der Eingabe und bei der Veränderung erfolgt über Funktionstasten. Zusätzlich kann eine Komponente direkt mit der Maus ausgewählt werden.

RE, A Nachrichten, Warnungen und Fehler werden in zusätzlichen Fenstern angezeigt.

RE, A Die Gestaltung der Bedieneroberfläche erlaubt ein gleichzeitiges Umgehen mit mehreren Fenstern einer Art, z.B. mehrere Masken für die Einträge bei der interaktiven Selektion aus mehreren Datenbeständen.

RE, A Das Layout der Bedieneroberflächengestaltung wird geändert, d.h. der layoutmäßige Aufbau einer Kartenmaske, der Fenster für die anzuzeigenden L
sten etc.

RE, A Die bisherige Skizze der Bedieneroberfläche sieht eine Ein- -/ Ausgabe einzelner Karten (z.B. in einer speziellen Maske) vor und sonst nur die Anzeige von Listen für Kästennamen, Kartennamen bzw. aller Karten eines Kastens. Für die interaktive Selektion kann man sich auch eine Bearbeitung eines Abschnitts einer dieser Listen vorstellen, der, im Fall der letzten beiden Listen, mehrere Karten umfaßt.

RE, A Die Bedieneroberfläche sieht Fensterstapel vor.

RE, A Die Bedieneroberfläche bezieht Standardbausteine von Fenstersystemen mit ein, wie Uhr, Mail, Notizblock, Taschenrechner etc.

RE, A Assoziative Suche auf einem oder mehreren Datenbeständen wird dadurch unterstützt, daß Masken teilweise ausgefüllt werden und daß einzelne Attributwerte teilweise angegeben werden können.

RE, A Die Anfrage über eine Anfragesprache wird durch die Anzeige eines entsprechenden Anfragefensters unterstützt.

A Bezüglich der Bedieneroberfläche können unterschiedliche Systeme konfiguriert werden, oder es wird durch dynamisches Binden oder Auswahl zur Laufzeit das spezielle System ”erstellt”.

A Das zugrundeliegende Fenstersystem wird ausgetauscht.

A Das zugrundeliegende Ein- -/ Ausgabegerät wird ausgetauscht (vgl. Tab. 7.3), wodurch sich andere Möglichkeiten der Bedieneroberflächengestaltung ergeben.

RE, A Die Namensliste für Kästen bzw. für Karten und die Liste aller Karten eines Kastens ändern sich bezüglich des Layouts, und zwar einerseits bei der Anzeige auf dem Bildschirm als auch bei der Ausgabe auf einem Drucker.

RE, A Die Papierausgabe dieser Listen erfolgt mit verschiedenen Papierlistenformaten bis hin zu Adreßetiketten.

A Die Ausgabe kann alternativ auf verschiedenen Ausgabegeräten erfolgen (Laserdrucker, Typenraddrucker o.ä.), oder die Ausgabe erfolgt auf ein bestimmtes Ausgabegerät, das, je nach Konfiguration des Systems, verschieden sein kann.

- E/A- - bzw. A- -Geräte- -Austausch

A	Es wird ein anderes Terminal als in Kap. 3 angenommen, nämlich eines mit Bildschirmspeicher. Dieses eröffnet die Möglichkeit, an eine beliebige Stelle des Bildschirms zu schreiben (im Gegensatz zum Schreiben in die letzte Zeile und zu dem impliziten Rollen in Kap. 3).
RE, A	Das Ein- -/ Ausgabegerät besitzt einen hochauflösenden Bildschirm. Damit können Fenster und Masken dargestellt werden, verschiedene Fonts und Größen für Schrift als auch beliebige Graphik.
A	Die Charakteristik des Druckers ändert sich, so daß auf eine beliebige Stelle einer Seite geschrieben werden kann.
RE, A	Der Drucker ist graphikfähig und erlaubt es, beliebige Fonts, Schriftgrößen und Graphik zu verwenden.

- Änderung der Struktur der Einträge
  Zugriffsmechanismen auf Karten
  Wunsch nach veränderbarer Strukturierung
  Realisierungsdetails Datenablage

RE, A Die Struktur des Schlüssels eines Eintrags (einer Karte) ändert sich: Der Schlüssel erhält einen anderen Typ, er wird aus mehreren Komponenten zusammengesetzt o.ä.

RE, A Was der Schlüssel eines Eintrags ist, soll vom Entwerfer leicht geändert werden können, oder dies soll sogar vom Bediener bestimmbar sein.

RE, A Zu einem Eintrag sind zusätzliche Attribute als Sekundärschlüssel angebbar (für die Anwendung Literaturrecherche z.B. Schlagwörter).

RE, A Diese zusätzlichen Sekundärschlüssel unterliegen einem Klassifikationsschema (z.B. Schema der Computing Reviews für Literatur der Informatik). Dieses Schema soll leicht veränderbar sein.

A Das Informationsfeld ist bei der Aufgabenstellung von Kapitel 3 eine Folge von Zeilen. Diese kann intern als ein Feld oder eine verkettete Liste von Zeilen fester Länge abgelegt sein. Alternativ dazu kann sie verdichtet als Zeichenkette mit Zeilenendekennzeichnung abgelegt sein. Diese Realisierungen sollen austauschbar sein.

RE, A Das Informationsfeld eines Eintrags erhält eine bestimmte Struktur: Es besteht aus mehreren Komponenten je eines bestimmten Typs.

A Die Realisierung eines Eintrags ändert sich (vgl. Diskussion zu Fig. 4.13).

RE, A Von den Einträgen werden Sonderformen benötigt (vgl. Spezialisierung in Abschnitt 5.8 und in Fig. 5.38).

RE, A Die Einträge sollen jeweils nur einmal vorkommen, auch wenn sie in verschiedenen Datenbeständen enthalten sind. Eine Änderung des Eintrags in einem Datenbestand soll auch zur Änderung dieses Eintrags in anderen Datenbeständen führen (Zeigersemantik anstelle von Variablensemantik).

- Kollektionsänderungen
  Schnittstellenänderungen
  Realisierungsänderungen

RE, A Auf einer Kollektion sind assoziative Anfragen möglich, wobei gewisse Komponentenwerte vorgegeben sind. Es werden alle Einträge gesucht, die diese Komponentenwerte besitzen. Bei den Komponentenwerten sind auch Abkürzungen erlaubt, z.B. A* für alle Textkomponentenwerte bestimmter Länge, die mit A beginnen.

RE, A Zwischen den verschiedenen Datenbeständen (Kästen) können Querbeziehungen eingetragen werden (z.B. Verweis von dem Eintrag einer Ehefrau auf den Eintrag ihres Gatten).

RE, A Eine Kollektion kann Einträge verschiedener, aber ähnlicher Struktur enthalten (vgl. Fig. 5.38 für Mitarbeiter, ..., leitende Angestellte).

RE, A Das Sortieren einer Kollektion wird vom Bediener angestoßen, also nicht mehr automatisch vorgenommen.

RE, A Das Sortieren eines Datenbestandes kann auch nach einem anderen Schlüssel als nach dem Primärschlüssel erfolgen.

RE, A Für jede Kollektion wird zusätzlich "Statistikinformation" gesammelt, z.B. die Anzahl der Karten eines Kastens.

A Die Kollektionsrealisierung für die Kästennamen ist "unabhängig" von der Struktur dieser Namen. Dies soll deshalb erfolgen, weil sich die Struktur dieser Namen leicht ändern kann.

A Die Kollektionsrealisierung für einen Karteikasten soll "unabhängig" von der Struktur des Einzeleintrags sein. Die Zielsetzung ist dabei, diese Kollektionsrealisierung auch bei einer Änderung der Anwendung verwenden zu können (z.B. Personalverwaltungssystem soll als Grundlage der Realisierung eines Literaturrecherchesytems verwandt werden).

RE, A Bei den Einträgen wird von der Variablensemantik zur Zeigersemantik übergegangen, um Inkonsistenzen verschiedener Datenbestände zu vermeiden.

A Die Realisierung der Kollektionen ändert sich, z.B. von einer sequentiellen Datei oder indexsequentiellen Datei und einer eventuellen Zugriffsstruktur zu verbesserten Zugriffsstrukturen (Hashing, B- -Baum, B*- -Baum usw.).

RE, A Neben der Zugriffsstruktur für den Primärschlüssel werden auch Zugriffsstrukturen für die Sekundärschlüssel aufgebaut. Dies macht insbesondere dann einen Sinn, wenn der Bediener entsprechende assoziative Zugriffsmöglichkeiten angeboten bekommt.

A Die Veränderung der Kollektionen durch den Bediener geschieht direkt. Alternativ dazu kann eine Kollektionsstruktur aufgebaut werden, die nur für eine Sitzung besteht und die dann mit dem Datenbestand auf der Platte abgeglichen wird.

-- Brainstorming führt uns auf DA--Stellen
interner Aufbau einer Karte
spezielle Realisierung der Kollektionen
spezielle Form von E/A--Gerät, spezielle Form von Drucker
Layoutgestaltung Bedieneroberfläche bzw. Druck--ausgabe

-- Beispiel von Kapitel 3: Funktionalität (Bedien--kommandos)
prägen Architektur. Ist indirekte Folge der Mißachtung der DA

Op. zu Karten stehen
Op. zu Kollektionen jetzt zusammen
Op. zu Gesamtheit aller Kollektionen

=> Steuerungsbausteine für Dialog sind jetzt ebenfalls kompakt

-- Mißachtung der DA ergibt große Änderungsunfreund--lichkeit
(Argumentation für Beispiele, die die Funktionalität nicht erweitern)

- Realisierung einer Karte
berührt nahezu alle Module unterhalb von Eing._Kartenkomm.

- Änderung Dateiform zur Abspeicherung der Karten eines Kastens
berührt nahezu alle Module unterhalb Eing._Kartenkomm.
Analog für Realisierungsänderung der Kästen--Na--

mensliste

- Änderung E/A- -Gerät oder Drucker

- Änderung des Layouts der E/A von Nachrichten,
  Karten,
  E des Suchkriteriums

-- Änderungen sind global und schwer durchzuführen
alle Stellen der Änderung im Programm ermitteln
konsistent ändern

DA im nachhinein einführen => globale Architek--
turänderungen

-- Zielsetzung des Neuentwurfs
Realisierungsentscheidung (Karte, Kollektionen)
auf Architekturebene verkapseln
Nebeneffekt: Architektur spiegelt akt. Funktionalität
nicht mehr wider
Layoutentscheidungen verkapseln
Geräteänderungen verkapseln
zukünftige Erweiterungen des Systems
möglichst lokale Architekturänderungen

=> Änderungen sollen sich auf Architekturebene
möglichst wenig auswirken

-- Entwurfsentscheidungen

Karteikasten_System: Sitzungsverwaltung (Eröffnen der Sitzung, Abschluß); später kann hier die Benutzeridentifikation stattfinden; sowie die Verzweigung in eine Auswahl von Untersystemen (dann muß der Modul umbenannt werden).

Verwaltung_Kaesten_und_Karten: Verwaltung der Änderungen auf einem Kasten oder auf der Gesamtheit der Kästen (Öffnen, Schließen der jeweiligen Datenbestände); verzweigen in die zwei Teile je nach Kommandoeingabe; bei Erweiterungen können weitere Architekturteile darunter gehängt werden.

Verwaltung_Kastenveraenderung: Zuständig für die Handhabung der Kommandos auf einem einzelnen Kasten, insbesondere für dessen Karten.

Verwaltung_Kaestenbestand: Zuständig für die Handhabung der Kommandos auf dem Bestand der Kästen (macht keine Bearbeitung eines Kastens).

Kaesten_Namensliste: Verkapselt die spezielle Realisierung der Liste der Namen aller Kästen, z.B. daß diese direkt auf eine vorgegebene Dateiform aufsetzt. Dieser Modul ist als ein abstrakter Datentyp von vornherein auf die Handhabung mehrerer Listen durch das System ausgelegt.

bel_Karteikasten: Verkapselt die spezielle Realisierung eines Kastens (einer Kollektion von Karten). Als abstrakter Datentyp ist der Modul auf die Handhabung mehrerer Kästen durch das System ausgelegt.

bel_Karteikarte: Verkapselt die Realisierung eines einzelnen Eintrags (hier zunächst nur, wie Text und Informationsfeld abgelegt sind und wie in dem letzteren die Zusammenfassung von Zeilen realisiert ist). Bei strukturiertem Eintragsfeld (vgl. nächster Abschnitt) verkapselt dieser Modul noch weitere Realisierungsentscheidungen.

Kommando_E_A: Verkapselt die spezielle Form der Kommandoaktivierung (durch einzelne Buchstaben, langer Kommandoname, Funktionstaste, Menüselektion, bei Menüs das spezielle Layout).

Nachrichten_E_A: Verkapselt das Layout eines Nachrichtenfensters und ob bzw. wie die Nachrichten bestätigt werden.

Text_E_A: Verkapselt das Layout eines Textfensters. Ein solches brauchen wir später zur Eingabe des Namens eines Karteikastens, ferner zur Eingabe eines Schlüssels, eines Teiles des Informationsfeldes, nach dem zu suchen ist, etc.

Karten_Layout: Verkapselt das Layout einer Karte auf dem Bildschirm (und somit in der "Liste" der Karten eines Kastens, die über eine Hardcopy einzelner Karten erzeugt wurde).

Kaestenliste_Layout: Verkapselt das Layout dieser Liste auf dem Bildschirm und somit auch auf der Hardcopy. Blättern oder Rollen findet auf dieser Liste statt.

virt_Terminal: Verkapselt die speziellen E/A- -Operationen eines konkreten Terminals.

virt_Drucker: Verkapselt die speziellen E/A- -Operationen eines konkreten Druckers.

-- Zusammenspiel Module Datenablage und Anzeige

Kaesten_Namensliste	Datenablage
Kaestenliste_Layout	Anzeige

bel_Karteikarte	Datenablage
Karten_Layout	Anzeige

Es gäbe auch Entsprechung für Karteikasten, wenn wir nicht nur Hardkopies der einzelnen Karten gezo--gen hätten

Verwaltungsmodule kennen das Layout nicht!

-- Teilsysteme der Architektur

Steuerungsteil: spez.
Datenhaltung 2 Teilsyst. (Kollekt. Realisier.) (3) allg.
virtueller Bildschirm (1) benutz--
virtueller Drucker (1) bar
E/A--Behandlung: Kommandos, Nachricht., Texte (2)
Layoutverkapselungsmodule (3)

verschiedene Grade von Allgemeinheit:
(1)
(2)
(3)

-- Beispiel bestätigt

f--Module, lokale Benutzbarkeit oben
DA--Module, allgemeine Benutzbarkeit unten

- - Unterschiede Architekturen
  Schnellschußbeispiel,
  ”richtige” Architektur:

  keine Baumstruktur mehr
  viele DA- -Module
  Entwurfsentscheidungen deutlich sichtbar
  Eintrag, Kollektion, E/A- -Geräte, Dialogelemente, Layout
  Steuerungsbausteine ”konzentriert” (f- -Module)

- - Änderungen der Realisierung bleiben lokal
  (Änderung der Funktionalität nächster Abschnitt):
  Eingabe der Kommandos über Funktionstasten
  ebenso Ende Parametereingabe
  Rumpf Kommando_E_A
  Nachrichten_E_A, Text_E_A, Karten_Layout
  in Schnellschußbeispiel Rumpf jedes Moduls, der mit Kommondobearbeitung zu tun hat

- - • Anderes E/A- -Gerät, das ganzen Bildschirm spei- -chern kann
  Rumpf von Karten_Layout
  in Schnellschußbeispiel alle Module, die mit Karten- -Kommandos zu tun haben

-- • Realisierung der Kollektionen
Rümpfe der Kollektionsmodule ändern sich
in Schnellschußbeispiel wird Realisierung in vielen Modulen
angesprochen: Lese- - und Schreiboperation auf bestimmte Datei

=> nur lokale Änderungen
Modulrumpfänderungen allein
Rumpfänderungen mit darunterhängenden Teilarchitekturen

- Änderung der Realisierung von Daten: globale Programmsystemänderungen
  im Nachhinein DA hineinbringen: globale Architekturänderungen
  Änderungen in funktionalen Teilen: bei richtiger und falscher Architektur meist modullokal

- Gestaltung von Bedieneroberflächen:
  weitere wichtige DA- -Anwendung

7.2 Erweiterungen des KK- -Beispiels und zugehörige Architekturänderungen

- - Erweiterungen
  andere Kommandos, weitere Kommandos
  Veränderung der Bedieneroberfläche
  zeigen:
  neue Architektur ”stabiler”
  (a) Änderungen lokal
  (b) Änderungen lassen sich aus Architekturdiagramm
  ablesen

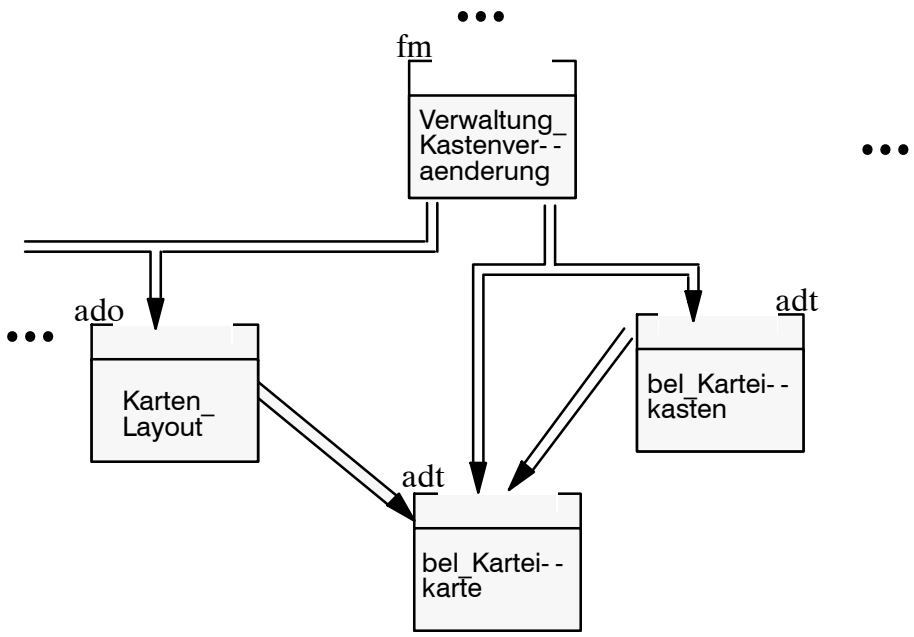
- - Erste Erweiterung:
  Strukturierung d. Informationskomponente einer Karte
  jetzt Karte ”logischer” Verbund von Komponenten
  Schnittstelle von bel_Karteikarte ändert sich
  Wohin wirkt sich das aus?

- - Änderungen: Verfolgen der Benutzbarkeits- -Kanten

  - Kollektionsmodul bel_Karteikasten
    kaum Änderung, wenn nur mit ganzen Einträgen umgegangen wird
    Soll nach Einträgen mit bestimmter Komponente gesucht werden: Schnittstelle ändert sich kaum
    Realisierung: muß Suchstruktur eingesetzt werden

  - Rumpf Verwaltung_Kastenveränderung

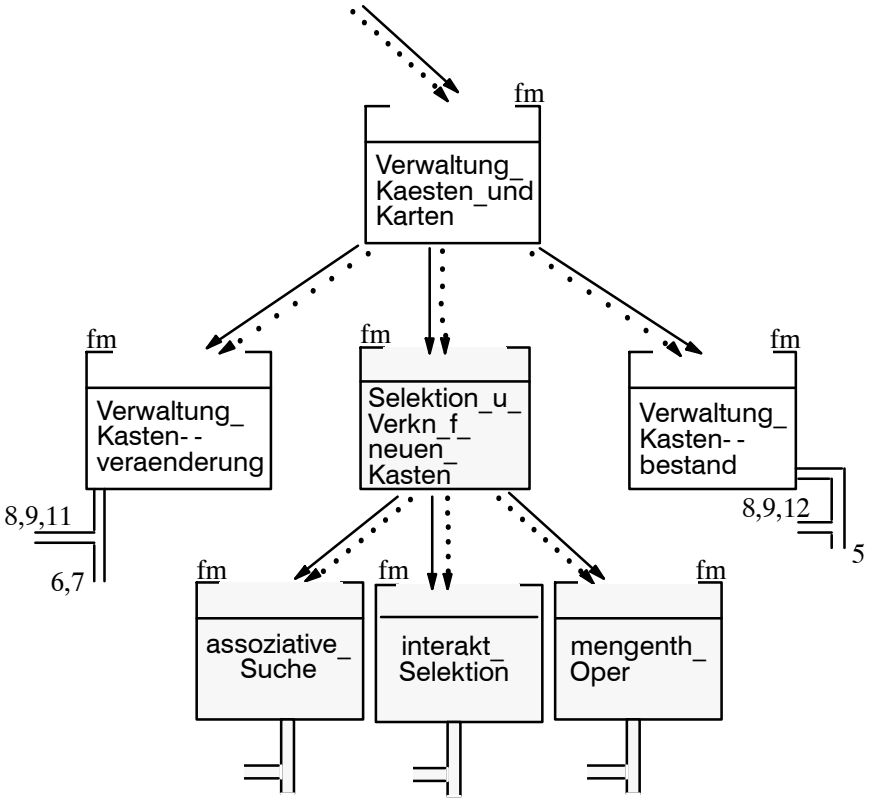
  - Schnittstelle und Rumpf von Karten_Layout:
    Benutzbarkeitskante eintragen


fm
Verwaltung_
Kastenver- -
aenderung
ado
Karten_
Layout
adt
bel_Kartei- -
kasten
adt
bel_Kartei- -
karte

-- Zweite Erweiterung:
Interaktive Selektion aus einem Kasten oder mehreren Kästen zur Erzeugung neuer Kästen
Einfache Abfragesprache: Komponenten einer Karte vorgeben
Mengentheoretische Operationen zum Verknüpfen von Teilmengen

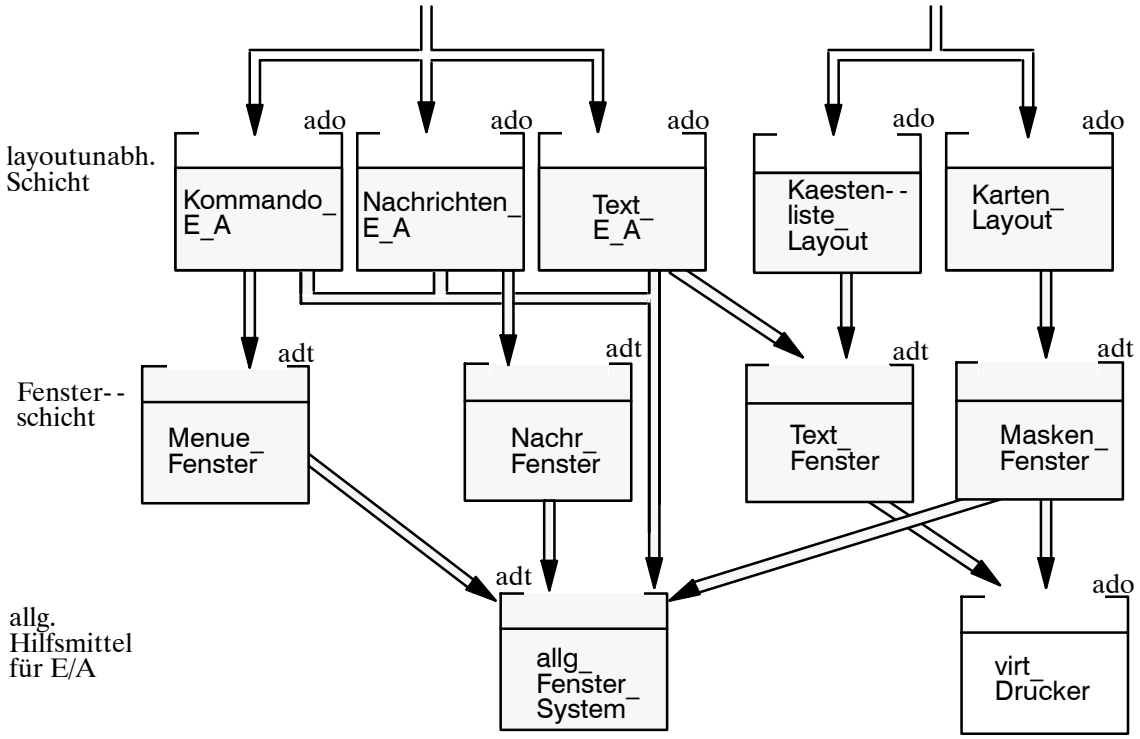
=> Zeigersemantik für Einträge wegen Konsistenzproblem

-- Änderungen
- Zeigersemantik Eintrag und Kollektionen (vgl. Kapitel 5)
- neue Verwaltungsmodule


fm
Verwaltung_
Kaesten_und
Karten
fm
Verwaltung_
Kasten- -
veraenderung
fm
Selektion_u_
Verkn_f_
neuen_
Kasten
fm
Verwaltung_
Kasten- -
bestand
8,9,11
6,7
8,9,12
5
fm
assoziative_
Suche
fm
interakt_
Selektion
fm
mengenth_
Oper

-- Dritte Erweiterung:
Fenstersystem: Menues, Masken, Mausselektion

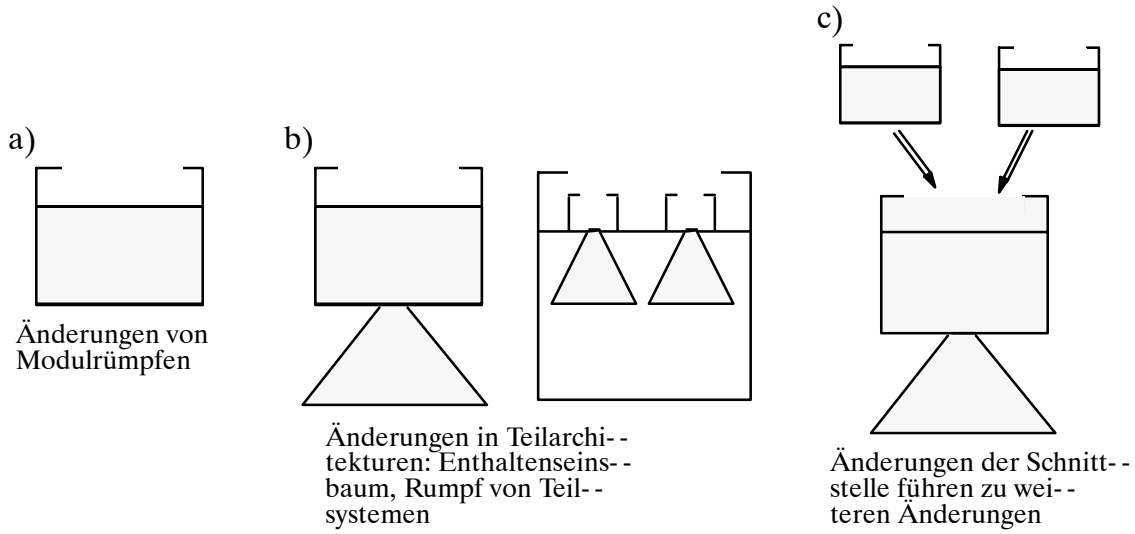
-- Änderungen
weitere Schicht für verschiedenartige Fenster mit jeweils spezieller Funktionalität



-- Jede Schicht des E/A--Teils hat bestimmte Abstraktions--
aufgabe, z.B.

Kommando_E_A
Menue_Fenster
allg_Fenstersystem
virt_Terminal

-- Programmsystem--Modifikationen



- Zusammenfassung

  neue Architektur objektbasiert
  Realisierungsänderungen und Erweiterungen der Funktionalität blieben begrenzt
  Entwurfsentscheidungen klar sichtbar; daraus auch, wo Modifikation vorzunehmen sind
  Verfolgen von Benutzbarkeitskanten lieferte weitere Änderungsstellen
  Führte dies zu Schnittstellenänderungen, so war Verfolgen zu wiederholen

7.3 Ein Transformationsproblem: Rekursiver Abstiegscompiler

-- andere Problemklasse: Batch--Systeme, wie Telefonab--rechnungssystem treten in
betriebswirtschaftlichen Anwend. (z.B. Lohn--abrechnung)
math.--techn. Anwendungen (Auswert. Ver--reihe)
Systemsoftwarebereich (Übersetzerbau)
auf

-- Betrachten Compiler
Struktur dieser Systeme sehr gut untersucht
Struktur kann aus Def. d. Eingabe u. Ausgabe herleitet werden
(Mech. Herleitung der Architektur i.a. auf Sonder--fälle beschränkt)

-- Bereich, in dem sehr früh versucht wurde
von Handcodierung abzurücken
zu Erzeugung (verschiedene Arten) zu kommen:
E/A--Struktur --> Compiler(Phasen)struktur
deshalb wichtig für Informatik--Ausbildung
(außerdem: kleine Übersetzungsprobleme ergeben sich oft)

-- verschiedene Ansätze:
- Mehrphasencompiler: verschiedene Aufgaben nacheinander,
  Einzelphasen weitgehend generiert
- Einphasencompiler orientiert sich an der kontextfreien Grammatikstruktur:
  alle Übersetzungsaufgaben zusammen

- - Strukturierung Einphasencompiler (rekursiv. Abstieg):
Idee
nichtterminales Symbol **- - >** Prozedur f. sämtl. Über- -

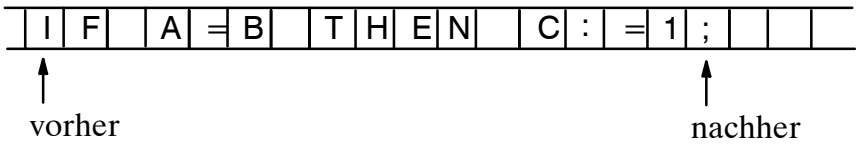
setzungsaufgaben (lexik. Analyse, kontextfreie
Analyse, kontextsensitive Analyse, Adressierung,
Codeerzeugung),
Fehlerbehandlung (Lokalisierung, Erkennung,
Meldung, Beseitigung, Wiederaufsetzen)

rekursiver Zusammenhang nichtterminaler Symbole
= > rek. Prozeduren
absteigend (Top- -down- -Übersetzung)
funktioniert für LL(1)- -Grammatik

```
statement ::= [ ident := expression |
                call ident |
                begin statement {; statement} end |
                if condition then statement |
                while condition do statement ]

procedure statement(...) is
   ...- -- -noetige Deklarationen, s.u.
begin
   if sym = ident then ...  - -- -Uebersetzung Zuweisung
   elsif sym = callsym then ...  - -- -Uebersetzung Pozedurrumpf
   elsif sym = ifsym then ...  - -- -Uebersetzung begingte Anweisung
   elsif sym = beginsym then ...  - -- -Uebersetzung Block
   elsif sym = whilesym then ...  - -- -Uebersetzung while- -Schleife
   end if;
   ...
end statement;
```

Vorbedingung:
Vor Aufruf von statement steht der Eingabezeiger auf dem ersten Zeichen eines Quelltextstücks, das zum nichtterminalen Symbol statement gehört.

Nachbedingung:
Nach Beendigung von statement steht der Eingabezeiger auf dem ersten Zeichen nach diesem Quelltextstück. Das zu statement gehörende Quelltextstück ist übersetzt, es sei denn, es tritt ein Fehler auf.
In diesem Fall ...


I F A = B T H E N C : = 1 ;
vorher
nachher

-- Innenleben der rekursiven Prozeduren ebenfalls mech. herleitbar (vgl. /Wi 84/):
gilt für beliebige Formen von rechten Seiten von Regeln
Erweiterung der Grammatik für einfache Fehler
Fehlerbehandlung
kontextsensitive Analyse
Adressierung
Codeerzeugung

beschränken uns im folgenden auf Architektur--modellierung

-- Compilerhauptteil
jedes nichtterminale Symbol --> funktionaler Modul (Prozedur)
Architektur mechanisch herleitbar aus EBNF
Hauptteil hat die Struktur des Grammatik--Abhängig--

keitsgraphen
Aufspannender Baum --> Enthaltenseinsbezieh.+ lok.

Benutzbarkeit,
restliche Kanten lokale Benutzbarkeit

a)

program ::= block .

block ::= [**const** ident = number {, ident = number} ;]
[**var** ident {, ident} ;]
{ **procedure** ident ; block ; }

statement ::= [ident := expression |
call ident |
begin statement {; statement} **end** |
if condition **then** statement |
while condition **do** statement]

condition ::= **odd** expression |
expression relop expression

expression ::= [addop] term {addop term}

term ::= factor {multop factor}

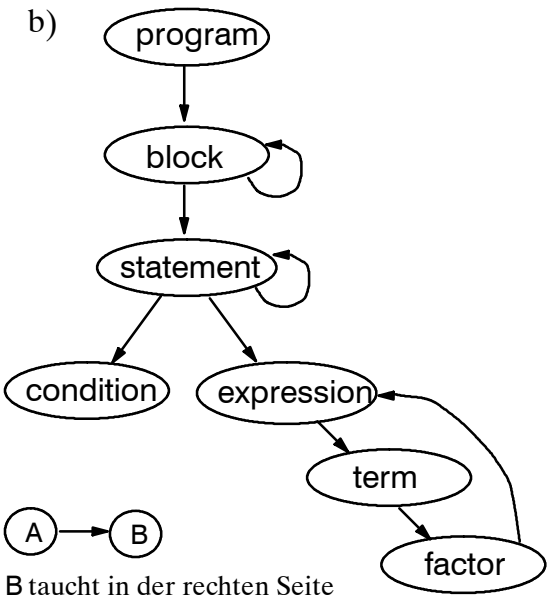
factor ::= ident | number | (expression)

relop ::= = | # | < | <= | > | >=

addop ::= + | - -

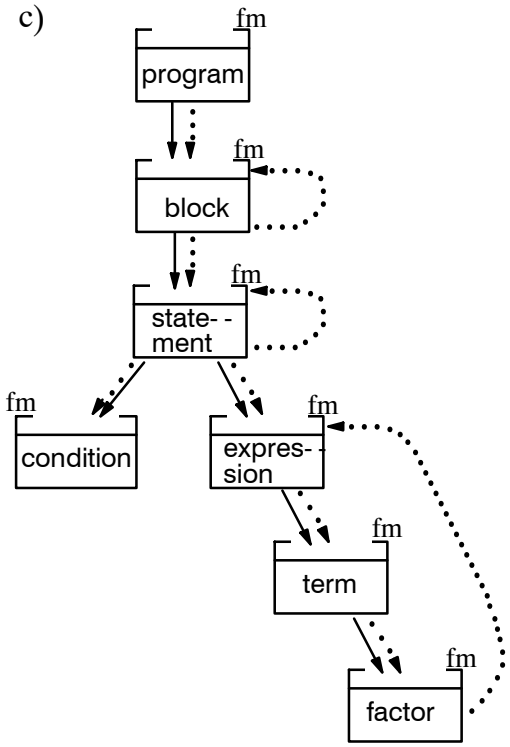
multop ::= * | /

b)



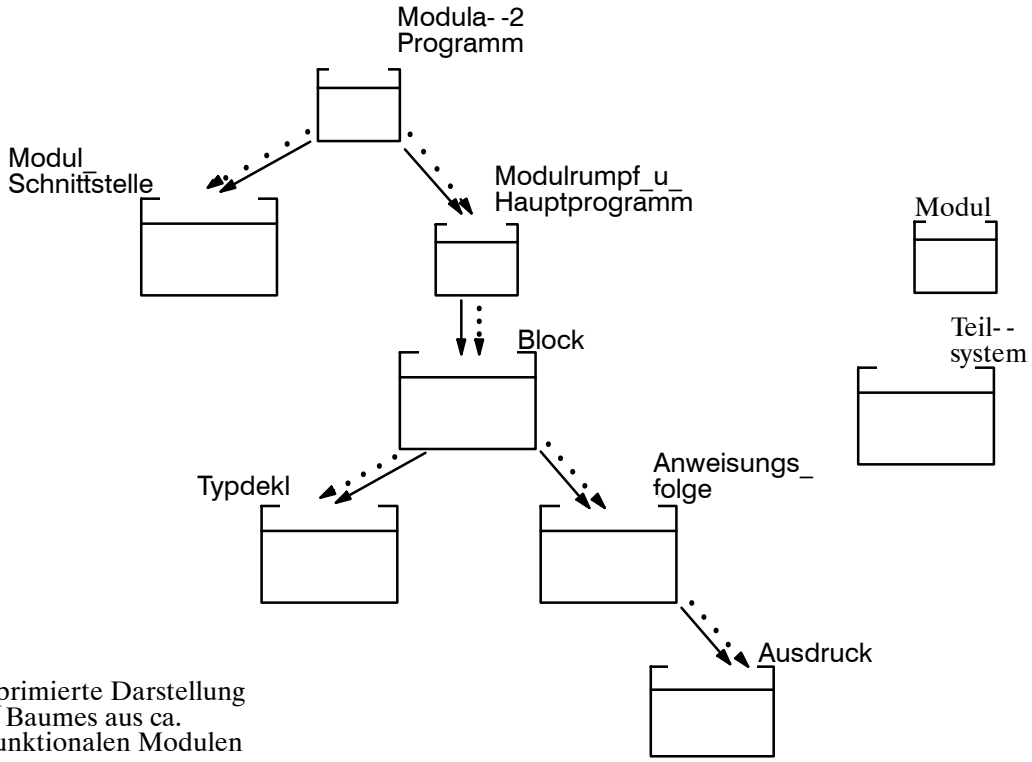
B taucht in der rechten Seite von A auf, oder B wird zur Übersetzung von A gebraucht

c)



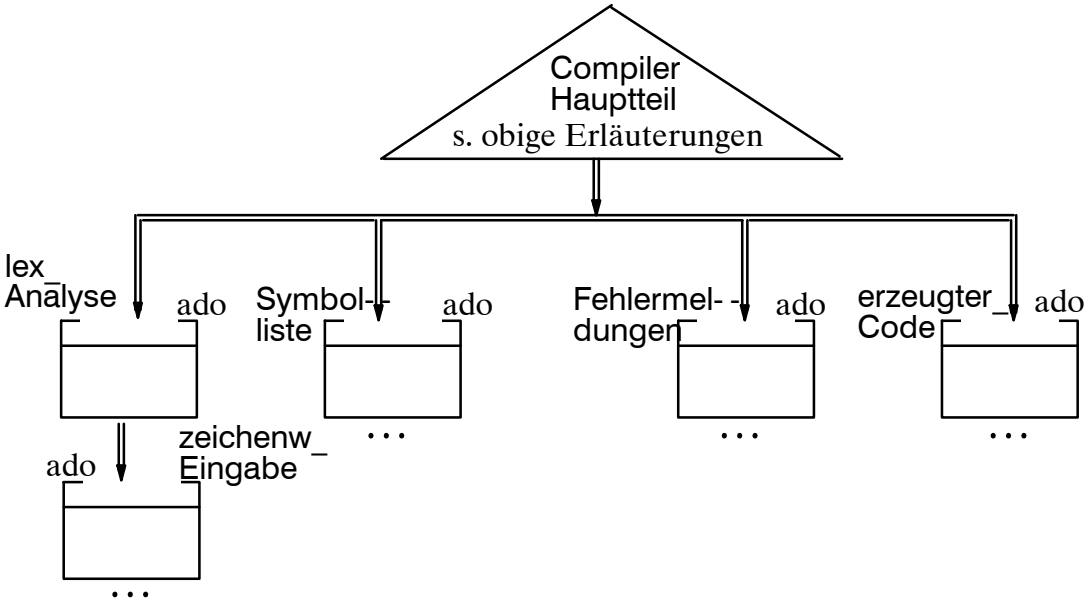
- Für Pascal, Modula-2 großer Baum
  ca. 100 Knoten, 10 Schichten
  für sich kompliziert, aber ein 1-1-Abbild der Grammatik
  dadurch leicht verständlich

- Vergröberung der Architektur durch Teilsysteme
  (hier Vergröberung im nachhinein, sonst umgekehrt)
  Suchen nach "abgeschlossenen" Enthaltenseins-Teil-bäumen
  Sind die komplexen Teile der Sprache



Komprimierte Darstellung
eines Baumes aus ca.
100 funktionalen Modulen

-- allgemein verwendbare Komponenten
Module/Teilsysteme alle zur DA



-- Zusammenfassung
Basisschicht ”invariant”
Compilerhauptteil spezifisch aber leicht herleitbar (Modul/Teilsystemstruktur, Rümpfe der Teilsysteme/Module)

=> Prozeß der Erzeugung wiederverwendbar, nicht Produkt

-- Steigerung der Adaptabilität:

Betrachten Syntaxanalyse allein:

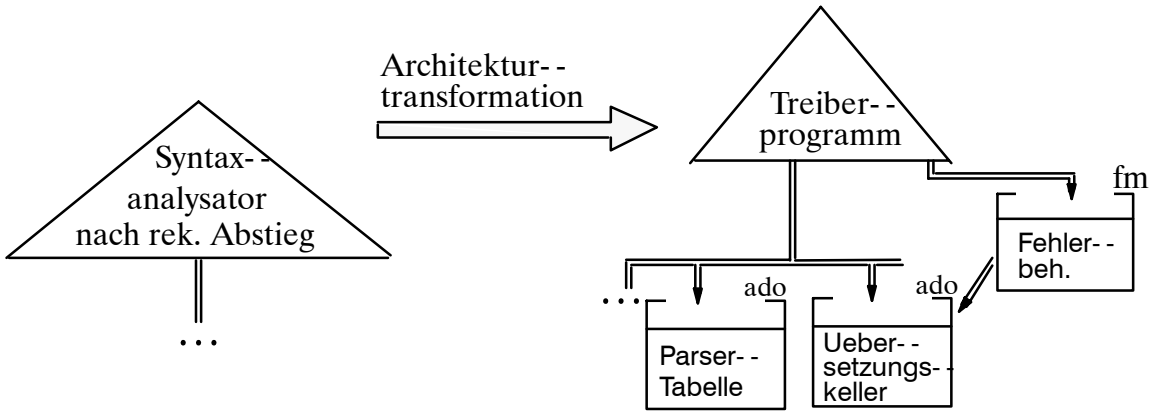
bei rekursivem Abstieg analog zu oben, mit DA--Ab--
leitungsbaum

Herausziehen von Code und in Datenstrukturen stekken:

Tabellenbaustein auswechselbar: LL(1)--Tabelle

Treiberprogrogramm invariant

Übersetzungskeller, Fehlerbehandlung kommen hinzu



7.4 Die Grobarchitektur einer Softwareentwicklungs- -Umgebung

7.5 Zusammenfassung

- Vollständige Architekturen
  - interaktive Systeme: KK--Beispiel, SEU
  - Batch--Systeme: Telefonbeispiel, Compiler

- unterschiedlich detaillierte Modellierung
  - Programmieren im Großen: KK--Bsp., Telefonbsp., Abstiegscomiler
  - Programmieren im Größten: SEU

- Haben Adaptabilität (Wiederverwendbarkeit) durch objektbasierte Architektur nachgewiesen:
  - Realisierunsänderungen
    KK--Beispiel
  - Erweiterungen
  - Adaptabilität des Prozesses: rekursiver Abstiegs--compiler

- Bedeutung der Diskussion "Was kann sich ändern?"
  - Erkennen der DA--Anwendungen
  - Erweiterungen mitbedenken

- Architekturen
  - individuell erstellt
  - für bestimmte Problemklassen herleitbar
    - mechanisch per Hand herleitbar (rekursiver Compiler)
    - mechanisch per Programm (Parser)

- Anzahl der DA--Module: Maß für Adaptabilität

8. ALLGEMEINE HINWEISE: STRATEGIEN ZUR ADAPTABILITÄT UND WIEDERVERWENDBAR-KEIT

- Zielsetzung:
  Hinweise, Regeln, Strategien zur Reduktion der Kosten der Erstellung/ Wartung von Software
  Ansatz, "intelligentere" Architekturen zu entwickeln: Wartbarkeit, Wiederverwendbarkeit

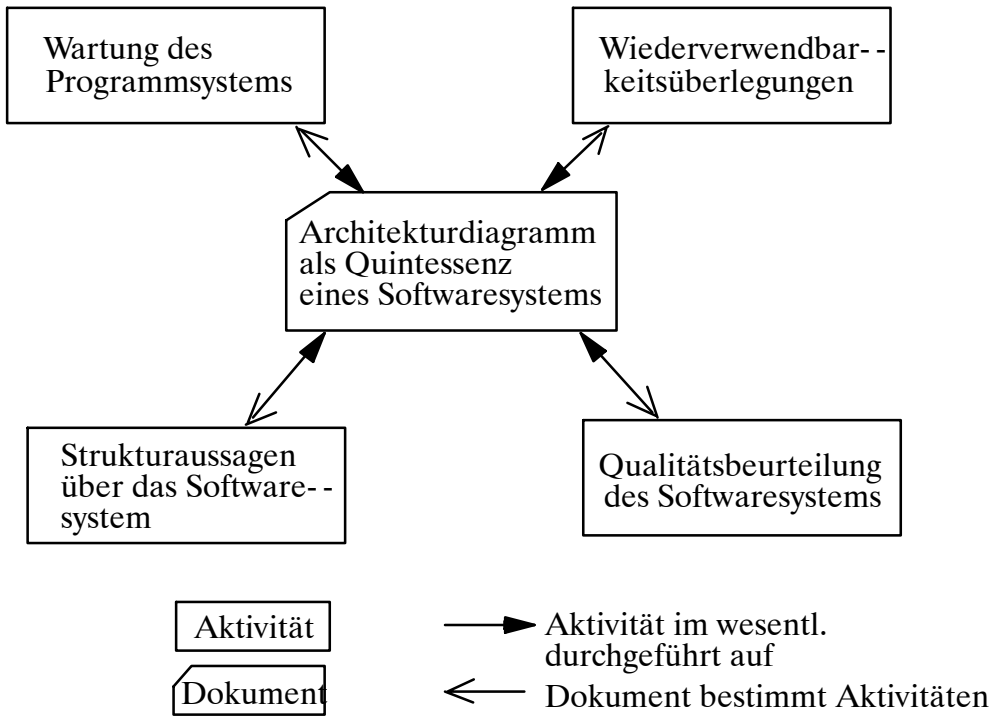
- Fassen bereits gesammelte Erkenntnisse zusammen
  Geben Regeln/ zugehörige Strategien an, die für Wartbarbeit, Wiederverwendbarkeit sorgen

- Ebene der Betrachtung
  nicht von Modulen (Kap. 4)
  Einhängen von Modulen (Kap. 4, 5)
  Teilsysteme (Kap. 5,7)
  Gesamtarchitektur (Kap. 7)
  Meta- -Ebene der Überlegung von Modulen, Teilsystemen,
  Teilarchitekturen, Gesamtarchitektur mit obiger Zielsetzung

- nach Erläuterung der Regeln/ Strategien:
  Was bedeuten sie für W, W

8. 1 Was haben die bisherigen Überlegungen gebracht?

- Motivation:
  Bedeutung: Wartbarkeit, Wiederverwendbarkeit
  Voraussetzung: Struktur von Softwaresystemen best. Anwendungs- und Problemklassen klären, Qualitätsüberlegung wegen W, W ergeben sich erst danach
  Architekturdiagramm enthält die Quintessenz dieser Struktur
- gutes Architekturdiagramm schwierig zu erstellen
  hier eingeführte formale Sprachen sind nur Vehikel, Ideen auszudrücken
  semiformaler Ansatz: Syntax und Semantik der Beschreibungssprache
  formal, Syntax des beschriebenen Softwaresystems
  formal, Semantik erst oberhalb einzelner Module
  Semantik erst oberhalb einzelner Module
- Bedeutung von Architekturdiagrammen: zentrales Dokument



-- für Wartbarkeit: Änderungen (Realisierungsände-- rungen,
Fehlerbeseitigung, Erweiterungen etc.) identifizieren,
Folgewirkungen verfolgen, feststellen, ob Wartbarkeit vorbedacht
(Diskussion: Was ...?)

-- für Strukturaussagen: funktionale, objektbasierte, ob-- jektorientierte Modellierung, Teilsysteme, generische Teile, welche Entwurfsentscheidungen

-- für Qualitätsaussagen:
Korrektheit: zur Überprüfung geg. Anforderungsdef. Diagramm
zur Überprüfung der Impl.: Textnotation
Robustheit, Auswahlsicherheit: Vorhandensein be-- stimmter Module/ Teilsysteme
Bedienerfreundlichkeit: ”
Adaptabilität/Portabilität: obj.bas.Module/Schichten
Lesbarkeit: Verständlichkeit zus. mit Design Rationale

Effizienz (des Erstellungs- -/ Wartungsprozesses)

- - für Wiederverwendbarkeit
  einzelne Basismodule/ - -teilsysteme
  gesamte Architektur
  Teile einer Gesamtarchitektur
  Vorgehensweise (vgl. Compilerabschnitt)

8.2 Einige Strategien zur Erstellung wartungs- - freundlicher Architekturen

- - Regeln für Wartbarkeit und Wiederverwendbarkeit
  Vermeiden, täglich das Rad neu zu erfinden
  Regeln keine konkr. Handlungsanleitungen, induzieren Strategien
  Beispiele:
    allgemeine
    spezielle innerhalb SEU

- - erste Strategie: Erkennen/Herausz. von Basisbausteinen
  - Module/ Teilsysteme
    über allg. Benutzbarkeit/ Generalisierung in Architek- - tur eingehängt
    verschiedene Grade von Allgemeinheit

  - anderen Personen zur Verfügung gestellt
    innerh. Teilprojekt, Projekt, Abt., Firma/ Institution, Benutzerkreis, Klasse v. Anwendungen, Sprachimpl.

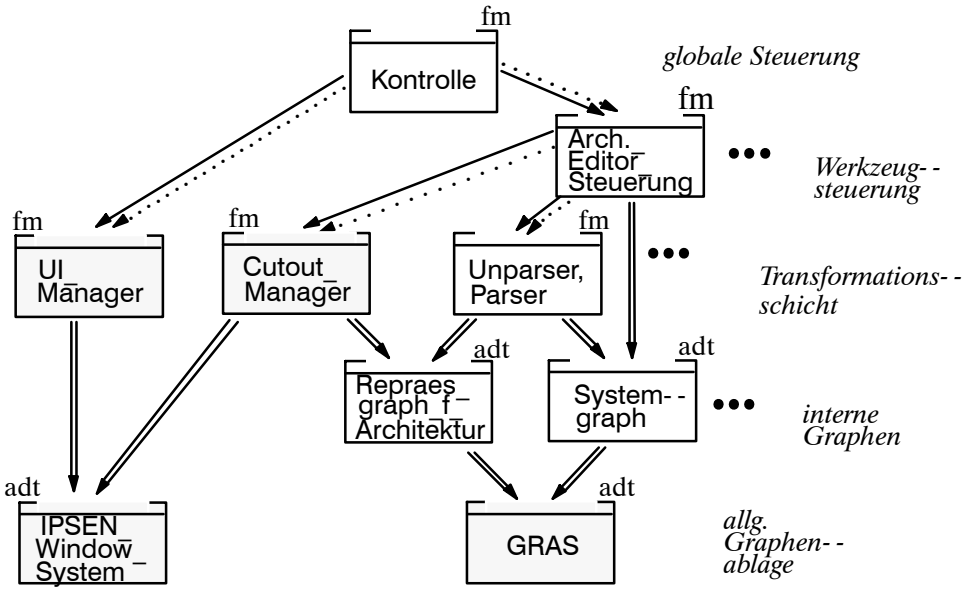
  - Wiederverwendungsbeitrag trivial
    Wartbarkeitsbeitrag:
      Wartung der Basiskomponenten von Spezialisten
      der Rest wird kleiner

  - Werkzeugunterstützung fehlt bisher

  - allgemeine Beispiele
      f: mathematische Bibliothek

DA: Einträge, Kollektionen, E/A- -Geräte, Bedien- -oberfläche

- SEU IPSEN


fm
Kontrolle
globale Steuerung
fm
Arch.
Editor
Steuerung
Werkzeug-
steuerung
fm
UI
Manager
fm
Cutout
Manager
fm
Unparser,
Parser
Transformations-
schicht
adt
adt
Repraes
graph f
Architektur
System-
graph
interne
Graphen
adt
adt
IPSEN
Window
System
GRAS
allg.
Graphen-
ablage

- - zweite Strategie:
  Softwaresysteme als Bausteinkasten

  - hier nicht Basisbausteine
    spezielle Module/ Teilsysteme eines Softwaresystems

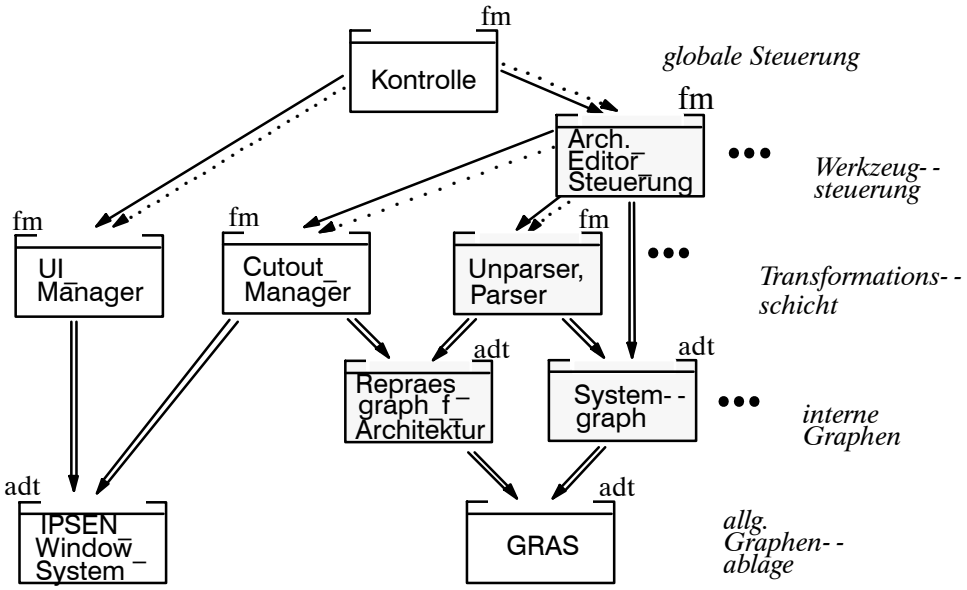
  - Sinn: verschiedene Varianten des Gesamtsystems
    vereinfachte (Einführungssystem), spez. (für best. E/A- -Geräte),
    erweiterte Systeme
    alles durch Bausteinkasten

  - in Modulkonzept hierfür
    Teilsysteme, Addition von Funktionalität
    Objektorientierung
    Schichten von Softwaresystemen

  - Beitrag zur Wartbarkeit:
    Wegnehmen/ Hinzufügen von Bausteinen
    Beitrag zur Wiederverwendung:
    Bausteine sind f. die wahlw. Hinzunahme konzipiert

  - allgemeine Beispiele kaum angebbar
    Anwendung Mehrphasencompiler: Optimierung, Postoptimierung;
    Vorderteil, Hinterteil kann ausgewechselt werden

- Anwendung SEU IPSEN



-- dritte Strategie: veränderliche Teile in ”Daten”

- Daten leichter auswechselbar als feste ”Verdrahtung” in Code

widerspricht nicht DA- -Auss.: alle Teile in Modulform:
Programm vs. Daten stets als Module identifizierbar
anwendbar auf Basisbausteine sowie auf Bausteine des
Bausteinkastens

- allgemeine Beispiele
  Fehlermeldungen, Warnungen, Systemmeldungen
  Layout- -Eigenschaften
  E/A- -Gerät
  endliche Automaten, Kellerautomaten

- SEU
  Meldungen (s.o)
  zulässige Kommandos zu logischen Inkrementen
  Parser/ Unparser
  Layoutfestlegung
  über die ganze Grobarchitektur verschmiert

- Umwandlung in datengetriebenes Progr. verschieden:
  Meldungen: enthalten jetzt Verweis auf Tabelle
  Syntaxanalyse: Architekturtransformation

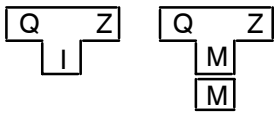
- Nutzen für Wartbarkeit:
  leicht änderbare Stellen in DA- -Modulen
  Real. ist austauschbar: Wechsel einer Datenstruktur

Wiederverwendung:
Interpreterbaustein

- - vierte Strategie: Bootstrapping

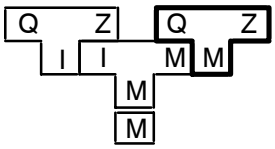
  - aus Compilerbau
    Verwendung für
    (1) Sprach- - und Compilererweiterungen
    (2) Portieren Compiler auf andere Maschine
    (3) Compiler verbessern

  - T- -Diagramm Notation:
    Q Quell- -, Z Ziel- -, I Implementierungssprache
    Ausführbarkeit eins Q_MZ- -Compilers:
    Interpreter für M in Hard- - oder Software



T- -Diagramme und
Ausführung

  - Zusammenfügen von T- -Diagrammen
    I_MM, M- -Assembler
    vorh., prod. Q_MZ- -Compiler
    Q_IZ- -Compiler
    Regeln des Zusammenfügens

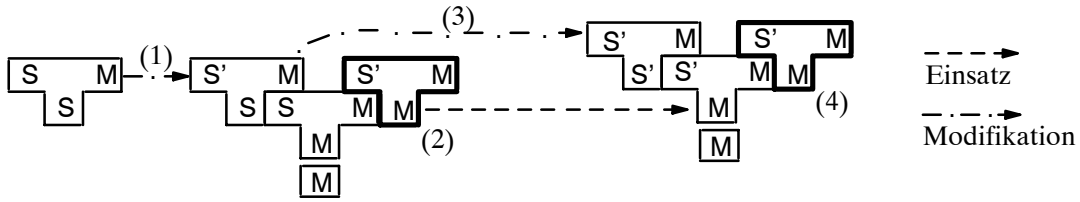


Compilieren eines
Compilers

- Ausgangspunkt S- -Compiler in S geschrieben

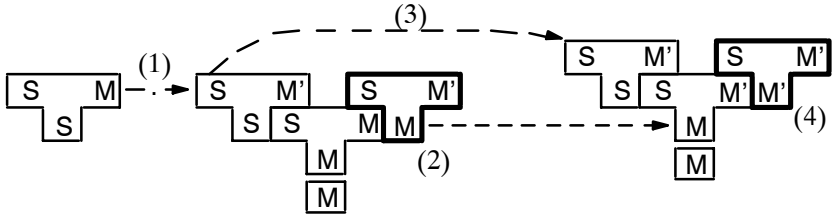
- erste Anwendung: Erweiterung S zu S' u. zurückspielen

  auf die Realisierung von S
  (1) $S_S M$ zu $S'_S M$- -Compiler
  (2) liefert $S'_M M$- -Compiler
  (3) Modifikation $S'_S M$ zu $S'_{S'} M$- -Compiler



Bootstrapping für Spracherweiterung

- zweite Anwendung: Portierung auf M'- -Maschine



Bootstrapping für die Portierung

- Initialschritt $S_M M$- -Compiler
  auch dieser herstellbar (Übung)

- weitere Anwendung Compileroptimierung (Übung)

- Bootstrapping
  u. Wiederverwendbark.: Compiler--Struktur u. Code

  Wartung: obige Probleme sind die Wartungspro--bleme von Compilern

- Verallgemeinerung Definition:
  Bootstrapping ist die Entwicklung von Konzepten, Methoden, Sprachen, Regeln, die für ihre eigene Realisierung eingesetzt werden können

- SEU--Beispiel
  PiG--Konzepte: Architektur IPSEN und somit auch der Architekturwerkzeuge benutzen Modulkonzept
  IPSEN für weitere IPSEN--Werkzeuge (zukünftig)
  PROGRESS--Umgebung für Spezifikationen
  Rapid--Prototypes von Werkzeugen mit PROGRESS--Interpreter

-- nächste 2 Strategien noch einmal Ebene höher
bisherige Strategien genügen diesen neuen

- - fünfte Strategie: Softwaresysteme ”erzeugen”
  - Abkehr von Überlegungen zur ”Handcodierung” von Überlegungen für ein System zu Überlegung f. ”Systemklasse”
    Erzeugen auf die spezifischen Teile gerichtet

  - Bausteinkastenstrategie und Bootstrapping sind Spezialfälle:
    Bausteinkasten: Zusammenbau durch Auswählen geeigneter Komponenten
    Bootstrapping: Erzeugen heißt Modifikationen von Compilergrundversion

  - 1. Art: Methodische Handfertigung (Abstiegscompiler, Jackson):
    Aus Anforderungsdefinition zur Architektur

  - 2. Art: Generierungsprogramm (Compiler- - Compiler- -Ansatz):
    Programm oder Tabelle

  - 3. Art: direkte Interpretation einer Spezifikation in der Regel für Komponenten eines Systems in gegebenem Rahmen:
    z.B. PROGRESS- -Spezifikation
    spätere Überführung in effizientes Programm durch 1. oder 2.

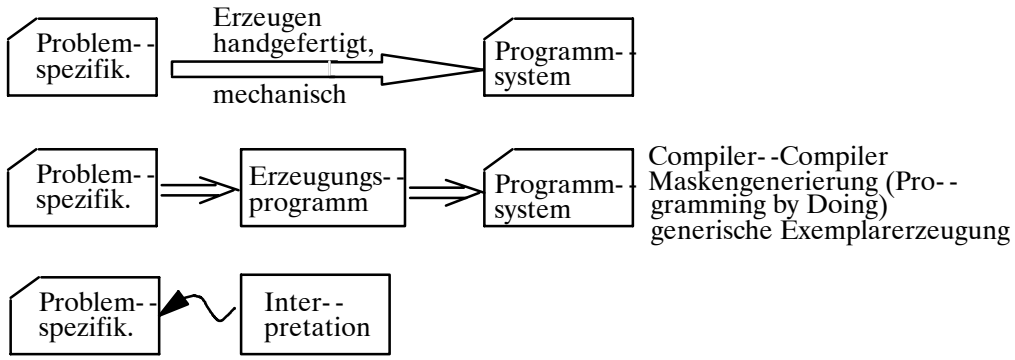
  - für Komponenten eines Systems generischer Mechanismus
    Programming by doing (durch Mechanik umgesetzt in Programm)

- PiG- -Sprachkonstrukte und Verwendung
  Generizität
  direkte Abb. einer Problemspez. (Abstiegscompiler)
  erlaubt Strukturierung des Generator- -Programms

als auch dessen Ergebnis
  Tabellen als DA- -Bausteine in Architektur
  Zusammenfügen gen. Teile (z.B. Mehrphasencomp.)
  durch allgemeinen Baustein
  direkte Ausführung: formale Spez. ist DA- -Modul,
  Interpreter kann strukturiert werden
  Einfügung d. formalen Spezifikation (+ Interpreter)
  in Architekturrahmen
  Programming by doing: Baustein, der interaktive
  Eingabe umsetzt

=>
sowohl generierende Programme, als auch deren
  Ergebnis, als auch deren Einbettung kann
  strukturiert werden

- Zusammenfassung



- Betrag zu Wartung/ Wiederverwendung
  Generizität: Wartung durch Erzeugung, Wiederver- - wendung der generischen Schablone
  mech. Handfertigung: Wartung durch Neuerzeugung
  mit vertretbarem Aufwand, Wiederverwendung: Vorgehensweise und Basisbausteine
  Compiler- -Compiler- -Ansatz: durch Neuerzeugung,
  Wiederverwendung, Compiler- -Compiler
  Programming by doing: analog
  dir. Ausführung: Wartung: Veränderung der Spez.
  Wiederverwendung: Interpreter

-- sechste Strategie: Ähnlichkeiten herausfinden

- zur Reduktion des Erstellungsaufwands
- alle bisherigen Strategien sind Spezialfälle
  (a) Basisbausteine herausziehen:
  (b) Bausteinkasten:
  (c) Tabellensteuerung: Interpreter
  (d) Bootstrapping: Ähnlichkeit eines Problems mit Verallgemeinerung
  (e) Erzeugung:
  Handfertigung:
  Compiler- -Compiler:
  direkte Ausführung:
  generischer Mechanismus:

- Objektorientierung als Sprachkonstrukt hierfür sowie andere Konzepte für (a) - - (e)

- allgemeine Beispiele:
  Architektur für Anwendungs- - oder Problemklasse

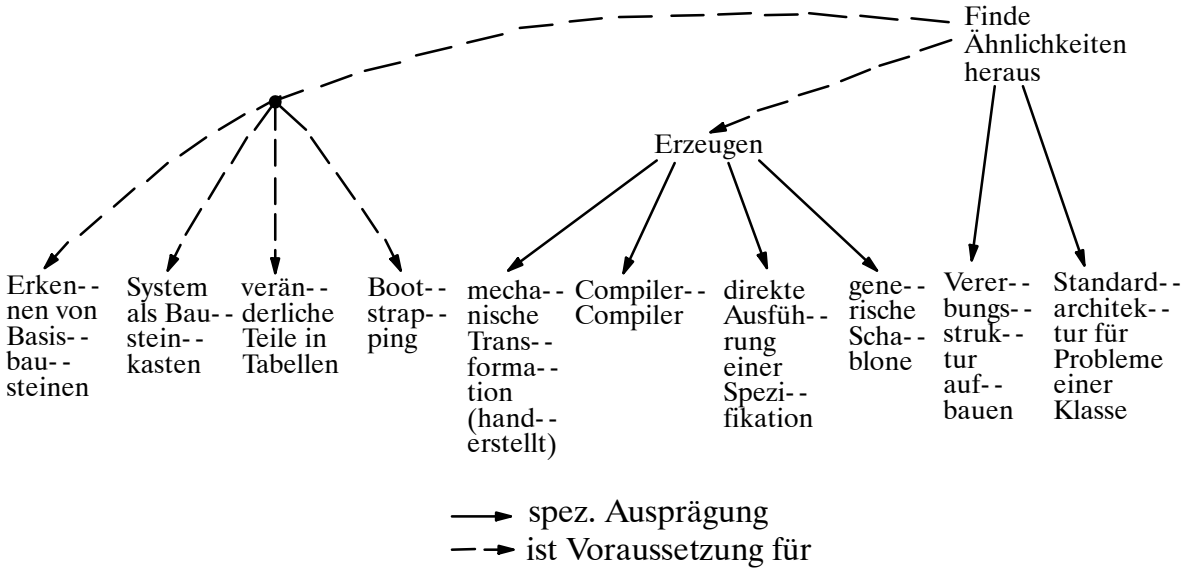
- Standardarchitektur mit obigen Strategien
  Basisbausteine werden Standardbausteine

- Sprachmittel
  zur Objektorientierung
  ansonsten bedarf es des Nachweises, daß Architektur Standardarcharakter hat

- Strategie fördert
  Wartung: Vor Anerkennung als Standardarchitektur muß dieser Nachweis erbracht sein
  Wiederverwendung: Rahmen u. darin enthaltene Standardbaust.

- Anwendung auf IPSEN (führen nur Argumente auf, die nicht bereits erwähnt wurden):
  stets gleiche Werkzeuge (Editoren, Analysatoren, ...)
  Werkzeuge haben stets gleiche ex. Charakteristika
  einheitliches Transformationsschema
  Graph- -Grammatik- -Engineering
  Architektur der Werkzeuge ähnlich und ableitbar
  Rückführung (Instrumentierung auf Editor- -operation,Parser- -Codeerzeugung etc.)

-- Zusammenfassung



-- Nachtrag zur 1. Strategie Basisbausteine:

Interpreter für Tabellen
Interpreter für die direkte Ausführung
generische Schablone (zum Prozeß gehörig)
Generierungsprogramm (”)

8.3 Zusammenfassung

- Architektur ist Zentrum für alle Überlegungen zur Gewinnung einer guten Programmsystemstruktur

- Regeln/ Strategien zur Erstellung ”intelligenterer” Software

- Generalthema war Wartbarkeit/ Wiederverwendbarkeit als die drängendsten Probleme der Softwareproduktion