

Grundlagen der Programmierung

cps4it

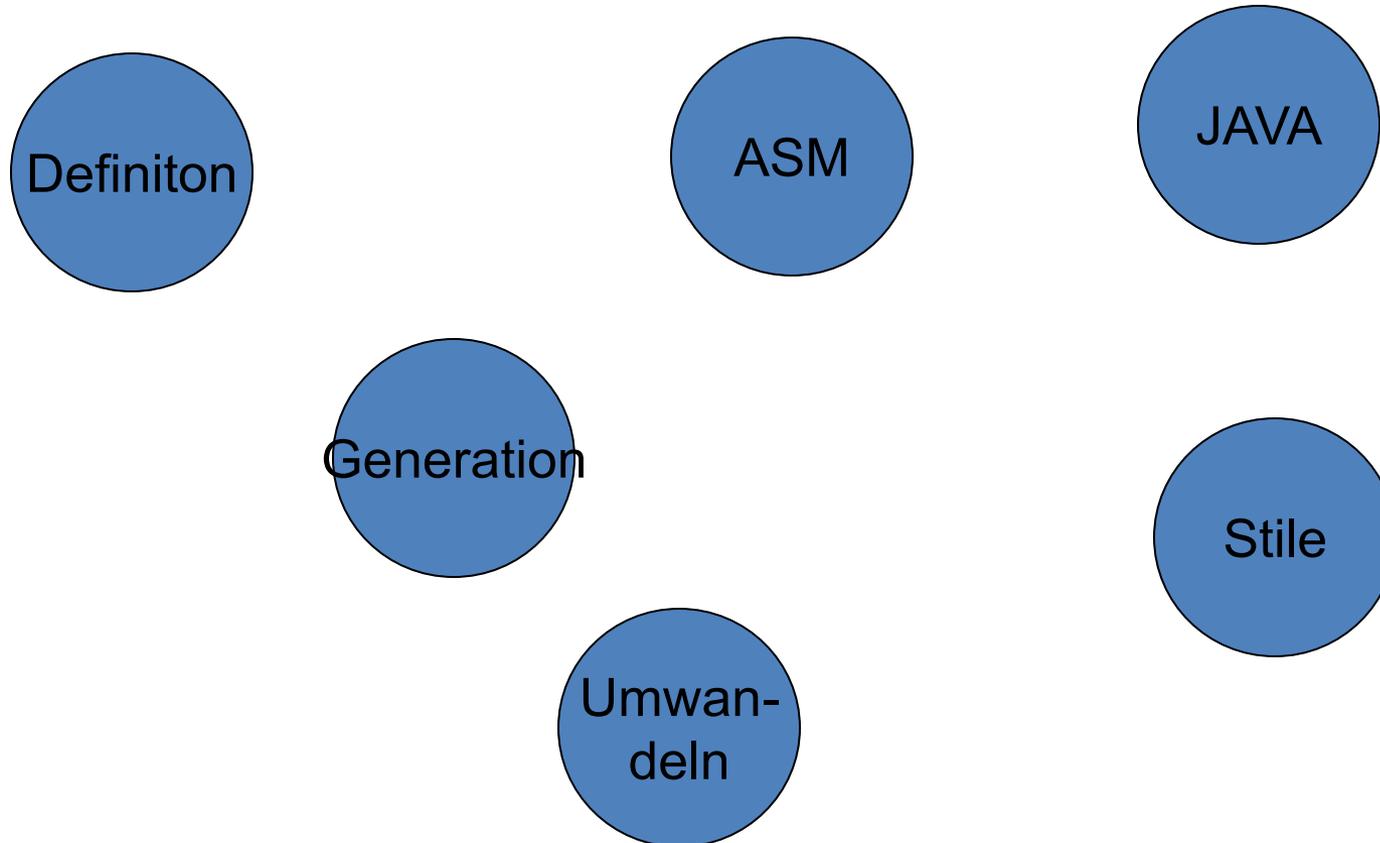
consulting, projektmanagement und seminare für die informationstechnologie

Ralf Seidler, Stromberger Straße 36A, 55411 Bingen

Fon: +49-6721-992611, Fax: +49-6721-992613, Mail: ralf.seidler@cps4it.de

Internet: <http://www.cps4it.de>

-
- 
- A blue arrow pointing to the right, highlighting the first item in the list.
- Programmiersprachen
 - Softwareentwicklung
 - Programmentwicklung
 - strukturierte Programmierung
 - Abbruchbehandlung



Historie

- 1. Generation - Maschinensprache
- 2. Generation - Assembler
- 3. Generation
 - C, COBOL, PASCAL, FORTRAN, PL1, ALGOL, ADA, Basic, SIMULA, JAVA etc.
- 4. Generation
 - Natural (4GL), SQL
 - OO-Sprachen wie JAVA, COBOL, SmallTalk
- 5. Generation - PROLOG (VHLL)

Ausblick – 1 (persönliche Auffassung)

- Maschinensprache
 - abhängig von Hardware
 - trotzdem auch künftig notwendig
 - z.B. Chips
- ASM
 - nicht unabhängig von Hardware
 - schwierige Wartung, da kein Nachwuchs
 - wird wohl auslaufen -> C bzw. Turbo PASCAL
 - wird wohl nicht auslaufen wegen Stärke der IBM ??

- 3. Generation
 - COBOL/PL1 mit hohem Marktanteil; wird daher so schnell nicht wegfallen
 - C übernimmt wohl ASM-Funktionen (nur Teile)
 - Versuch, unabhängig von Hardware zu sein, was nicht komplett gelingt
 - JAVA hat Chancen, aber
 - Eine echte Plattformunabhängigkeit gibt es nicht.
 - Wer versucht, alles mit JAVA zu lösen, wird scheitern.
 - alle anderen sind abhängig von starken Personen

Ausblick – 3 (persönliche Auffassung)

- 4. Generation
 - DB-bezogene Sprachen bleiben
 - OO-Sprachen
 - SmallTalk gibt es (fast) nicht mehr
 - JAVA hat Chancen
 - COBOL als OO-Sprache ohne Zukunft
 - Problem: Overhead
- 5. Generation
 - d.h. KI, die immer noch nicht Markt fähig ist

- imperativ
 - Wertzuweisungen
 - Verzweigungen
 - Basis: Neumann-Rechnerarchitektur
- prozedural
 - Iteration
 - Selektion
 - Daten -> Prozedur -> Daten

- **funktional**
 - einfache Funktionen zu komplexen zusammen bauen
 - Rekursion
 - Variablen präsentieren Wert und sind daran gebunden oder eben ungebunden
- **logisch**
 - es werden nur Fakten und Eigenschaften (Regeln) des Problems formuliert
 - Formulierung über Formalismus
 - System sucht eigenständig die Lösung
 - Variablenkonzept ähnlich wie bei funktionalem Stil

- Objekt-orientiert
 - die Welt ist eine Menge von Objekten
 - Objekte kommunizieren über Nachrichten
 - Objekt ist beschrieben durch Daten und Methoden
 - Daten charakterisieren Attribute des Objekts
 - Methoden regeln Zugriff auf Daten
 - streng hierarchisch orientiert
 - Vererbung

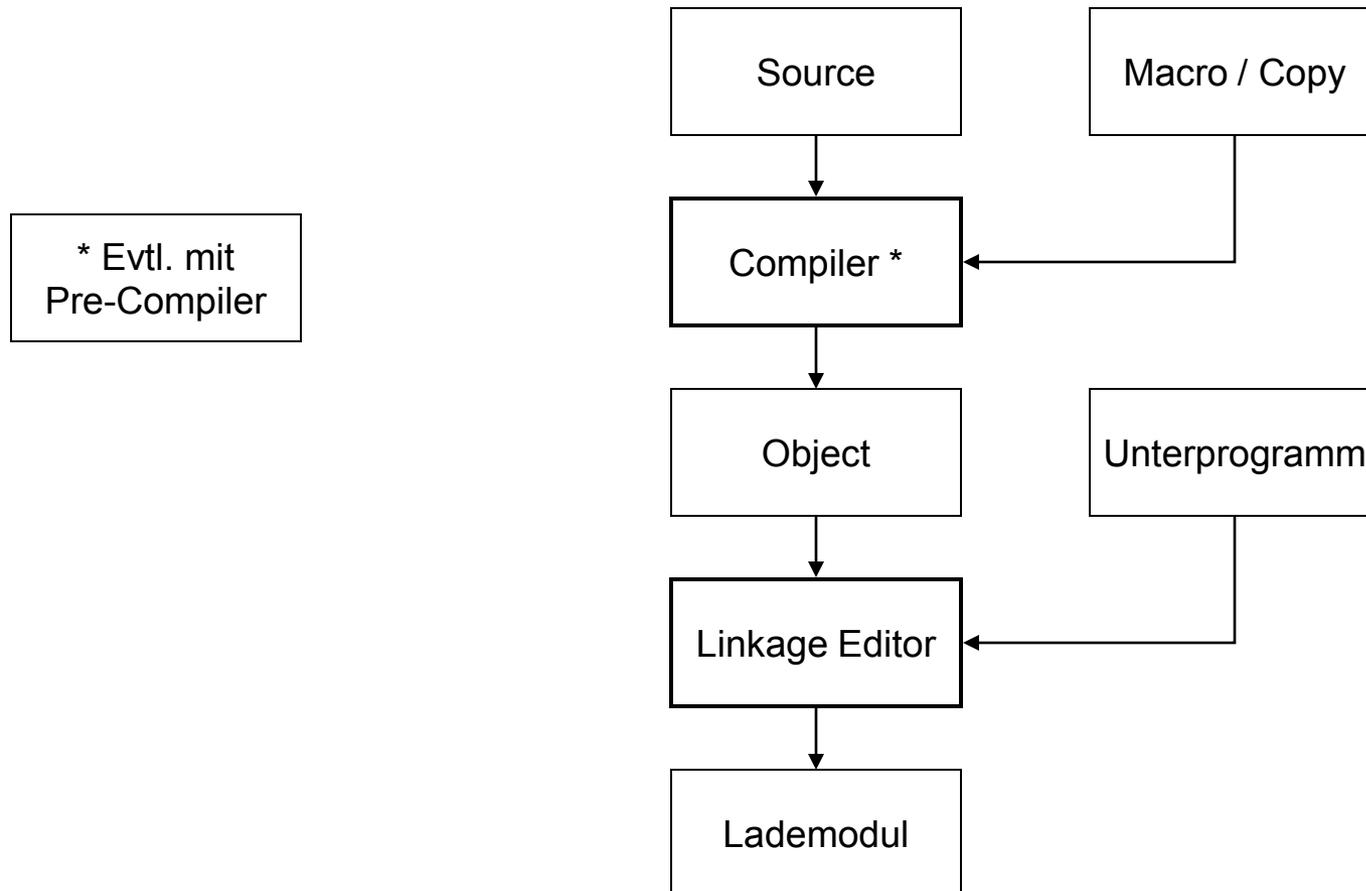
- Parallelisierung
 - Prozesse können parallel abgearbeitet werden
 - es stehen mehrere Prozessoren zur Verfügung
 - Ergebnisse werden zusammen geführt
 - zwei Grenzwerte
 - natürliche Parallelität auf Einprozessorrechnern, die sequentiell abgearbeitet werden
 - sequentielle Prozesse werden für Mehrprozessorrechner parallel abgebildet
- Synchronisierung

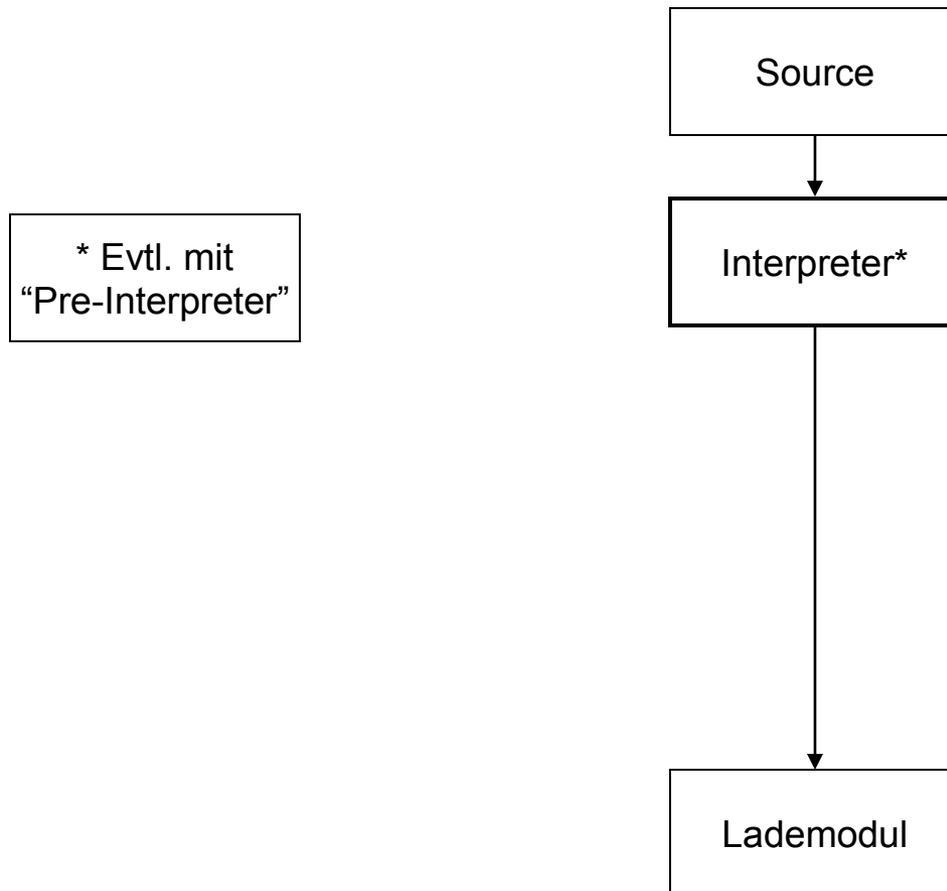
- Es gibt (nahezu) keine reine Sprache
 - OO: Smalltalk, Eiffel, andere mit OO-Erweiterung
 - prozedural: COBOL, PL1, REXX etc.
 - funktional: LISP
 - logisch: PROLOG
 - parallele Systeme: SIMULA, MODULA, ADA

Arbeitsschritte beim Implementieren

- Editieren
- Übersetzen
- Binden
- Laden
- Ausführen

Arbeitsschritte beim Implementieren – Schaubild mit Compiler





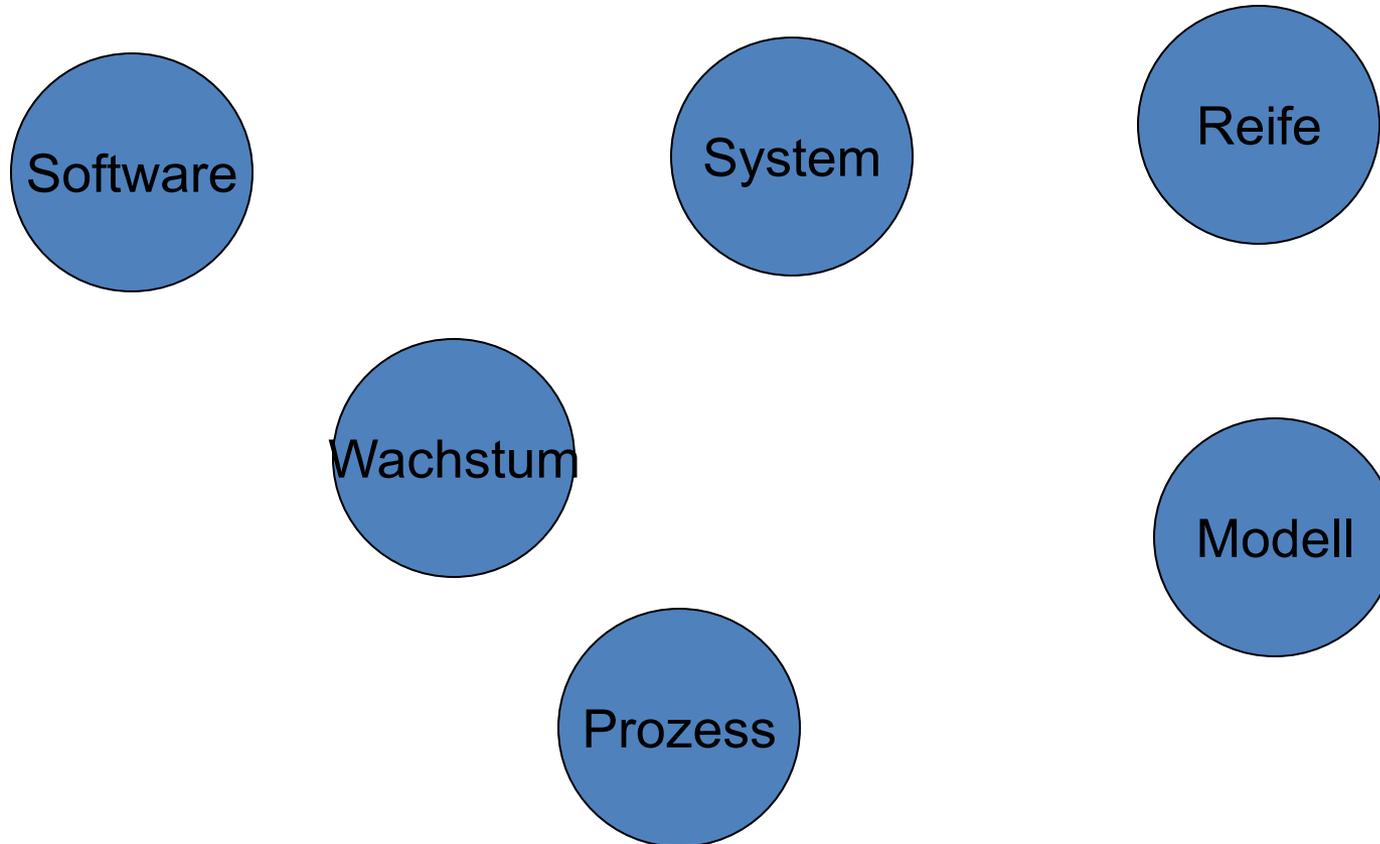
- **Compreter**
 - Zwischencode (P-Code oder Byte-Code)
 - Zwischencode soll unabhängig von Hardware sein
 - Zwischencode wird mit Interpreter ausgeführt
- **aktuell**
 - Compile auf beliebigem Rechner mit erzeugen Byte-Code
 - Bilden ausführbares Modul auf Zielmaschine

Aufgabe des Binders und der Laufzeitumgebung

- Binder
 - Zusammenfügen aller benötigten Ressourcen wie
 - Referenzen von Dateien
 - Unterprogramme bzw. Referenzen dazu
 - erzeugen von Maschinencode
- Laufzeitumgebung
 - Laden des Programms
 - Zuordnen aller externen Referenzen mit absoluten Adressen
 - Kontrolle bei der Ausführung



-
- Programmiersprachen
 - • Softwareentwicklung
 - Programmentwicklung
 - strukturierte Programmierung
 - Abbruchbehandlung



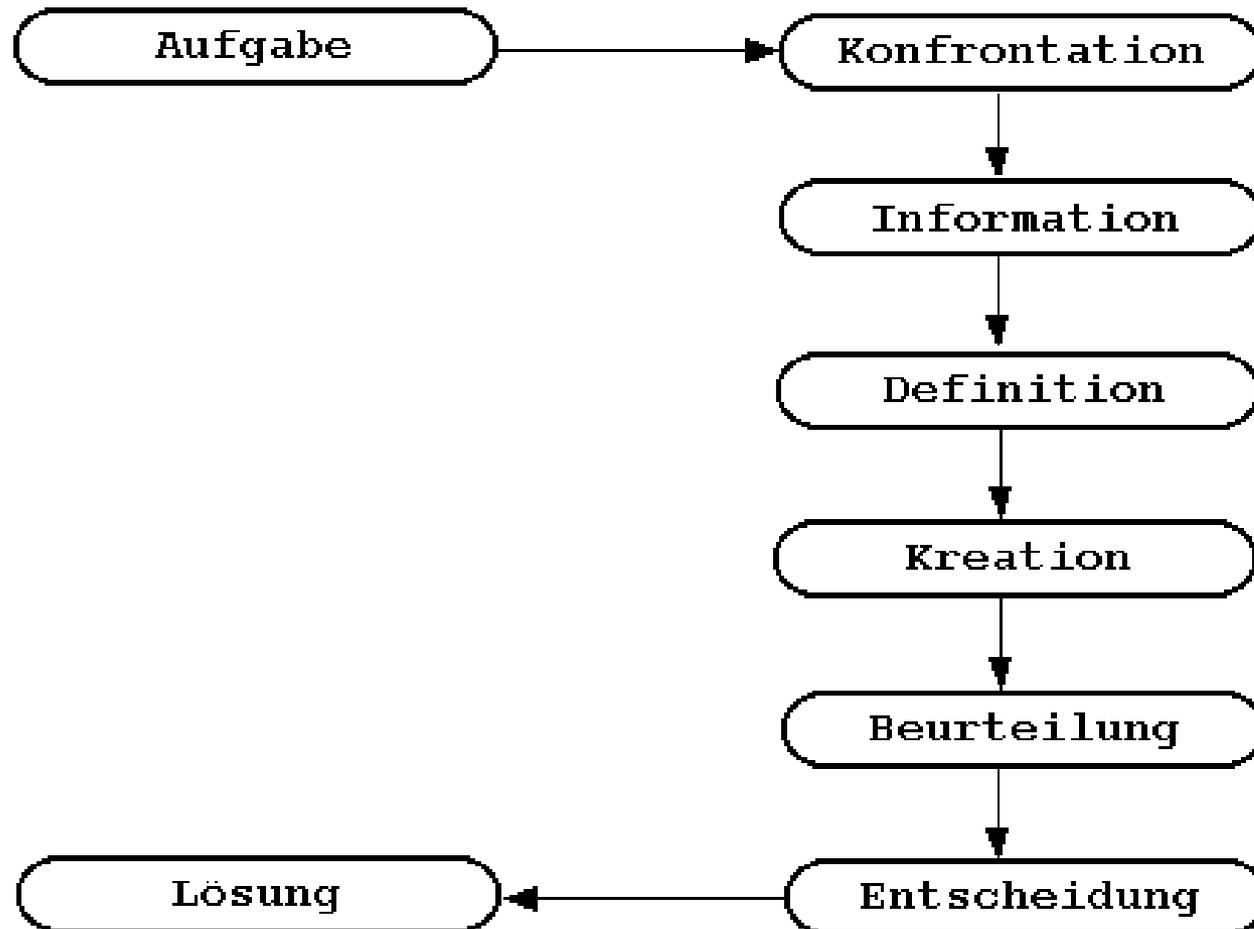
Definition von Software

- Software ist ein Produkt. Es ist eine Kombination von Programmen, Dokumentationen und Daten. Sie unterliegt dem allgemeinen Zyklus von Anwendungssystemen.

Lebenszyklus – 1

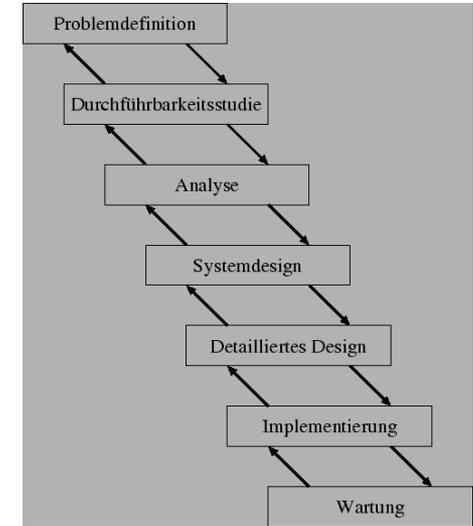
- Systementwicklung
 - Entwicklung der Software
- Systemeinführung
 - Vorbereitungen zur Nutzung der Software
- Wachstum
 - Verbreitung der Nutzung der Software
- Reife
 - Umsetzen von Verbesserungen an der Software, Beseitigung von Fehlern

- Rückgang
 - Schrittweiser Übergang zu einem neuen Software-Produkt oder zu einer neuen Version des Produkts
 - Ablösung des Produkts mit Löschen aller nicht mehr benötigten Ressourcen / Objekten



Wasserfallmodell – Phasen

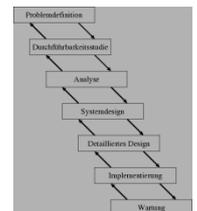
- Planung / Grobentwurf
- Analyse / Fachentwurf
- Design / technischer Entwurf
- Programmierung mit Modultest
- Integration und Systemtest
- Auslieferung, Einsatz und Wartung



Wasserfallmodell – Schwachstellen

- tatsächlicher Prozess nicht nacheinander
- Testen und damit Einbinden der Fachbereiche sehr spät
- Prototyping nicht möglich

- Beginn der Entwicklung von verschiedenen Modellen, die den Entwicklungsprozess variabler berücksichtigen

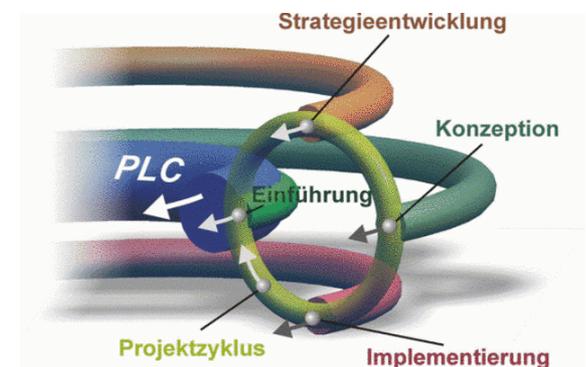


- Planungssicherheit (vergleichbare Produkte vergleichbar erstellen)
- Effizienz (nichts vergessen, keine Nachbereinigungen)
- Kostenreduzierung (Fehler schnell erkennen)
- Qualität (von Anfang an kontrolliert)

- Wasserfallmodell (1970)
- Iteratives / evolutionäres Prototyping (1980)
- V-Modell (1986)
- Spiralmodell (1988)
- agile Softwareentwicklung (1990)
- extreme Programmierung (1999)
- V-Modell XT (2005)
- agiles V-Modell (aktuell)

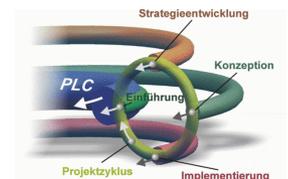
Iteratives Modell – Eigenschaften

- Das Softwareprodukt wird allmählich und stufenweise entwickelt.
- Die Steuerung basiert aufgrund gesammelter Erfahrungen bei der Anwendung.
- Die Wartung der Software wird als Erstellung einer neuen Version verstanden.



Iteratives Modell – Vorteile / Nachteile

- Gut, wenn der Auftraggeber der Softwareentwicklung seine Anforderungen nicht vollständig überblickt.
- Gut, da die Entwicklung sich auf lauffähige Teillösungen konzentriert, die über verschiedene Versionen nacheinander freigegeben werden.
- Gefahr, dass zu einem späteren Entwicklungszeitpunkt die Architektur geändert und angepasst werden muss.



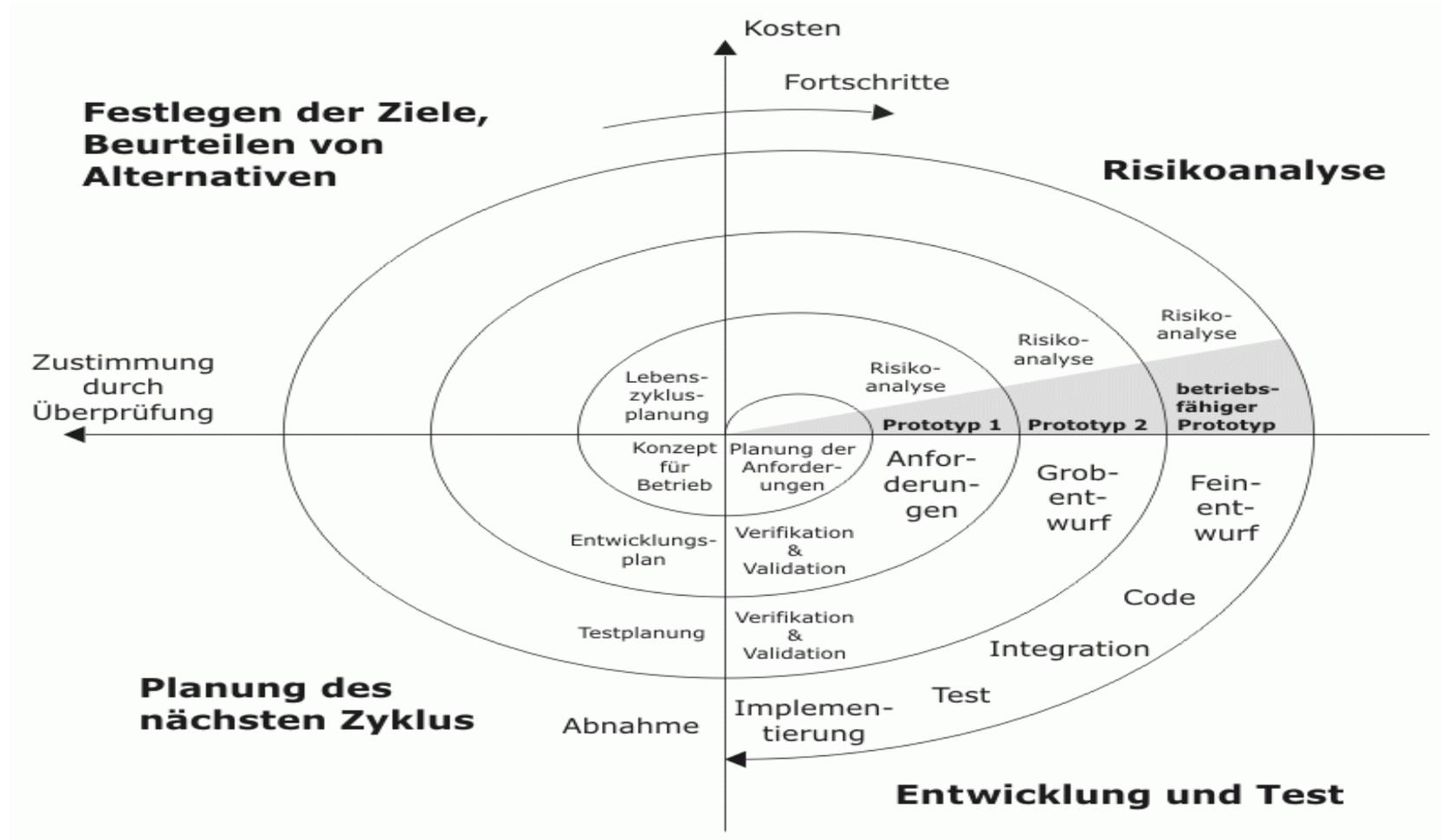
V-Modell

- System-Anforderungsanalyse
- System-Entwurf
- SW-/HW-Anforderungsanalyse
- SW-Grobentwurf
- SW-Feinentwurf
- SW-Implementierung
- SW-Integration
- System-Integration
- Überleitung in die Nutzung



- Iterativer Prozess, jeder Zyklus enthält:
 - Festlegung von Zielen, Alternativen und Rahmenbedingungen
 - Evaluierung der Alternativen und das Erkennen und Reduzieren von Risiken
 - Realisierung und Überprüfung des Zwischenprodukts
 - Planung der Projektfortsetzung.
- Die Phasen des Wasserfallmodells werden mehrfach spiralförmig durchlaufen.

Spiralmodell – Grafik



- Eigenschaften
 - reine Entwurfsphase auf ein Mindestmaß reduzieren
 - im Entwicklungsprozess so früh wie möglich zu ausführbarer Software gelangen
 - Abstimmung jederzeit mit Kunden
- Ziele
 - flexible Handhabung von Kundenwünschen
 - hohe Kundenzufriedenheit
 - einfach (KISS)
 - gemeinsamer Code-Besitz
 - vorhandene Ressourcen mehrfach verwenden

- Beispiele
 - Adaptive Software Development (ASD)
 - Dynamic System Development Method (DSDM)
 - Extreme Programming (XP)
 - Feature Driven Development (FDD)
 - Pragmatic Programming
 - Scrum
 - Software-Expedition / Universal Application
 - Usability Driven Development (UDD)
 - Testgetriebene Entwicklung
- <http://www.agilemanifesto.org/principles.html>

- Beispiel eines Teams bei DaimlerChrysler
- kleine bis mittelgroße Teams
- „übertriebener“ Einsatz anerkannter Techniken
- 14 Grundprinzipien



- interessant: es gibt keine (Uni-) Web-Adresse mehr mit Vorlesungsfolien / kaum Artikel

Gesetze von Murphy

1. Die Dinge sind komplexer als sie scheinen!
2. Die Dinge brauchen länger als erwartet!
3. Die Dinge kosten mehr als vorgesehen!
4. Wenn etwas schief gehen kann,
so geschieht es!

Anmerkung: Murphy war ein Optimist !

Softwarequalität – 1

- Software muss zuverlässig die Aufgaben lösen, zu deren Erledigung sie entwickelt wurde. Kann die Software (unter bestimmten Umständen) eine Teilaufgabe nicht (zuverlässig) lösen, so sollte sie dies dem Nutzer - unmissverständlich - mitteilen
- Sie muss so einfach wie möglich zu bedienen sein, d.h. sie muss benutzerfreundlich sein.
 - Achtung: das Auslösen kritischer Operationen darf nicht zu einfach sein!

Softwarequalität – 2

- Software muss so einfach wie möglich wartbar sein; gute Wartbarkeit ist Voraussetzung für Flexibilität !
- Software sollte so effektiv wie notwendig arbeiten
 - Achtung: zu uneffektiv arbeitende Software ist benutzerunfreundlich!

- Allgemeiner Nutzen
 - Brauchbarkeit
 - Portabilität
 - Zuverlässigkeit
 - Effizienz
 - Benutzerfreundlichkeit
 - Wartbarkeit
 - Testbarkeit
 - Verständlichkeit
 - Änderbarkeit

- Geräteunabhängigkeit
- Autarkie
- Genauigkeit
- Vollständigkeit
- Robustheit
- Integrität
- Konsistenz
- Zählbarkeit
- Geräte-Effizienz
- Zugänglichkeit
- Assimilationsfähigkeit
- Selbsterklärung
- Strukturierung
- Kompaktheit
- Lesbarkeit
- Erweiterbarkeit

Grundsätze für die Softwareentwicklung – 1

- Trauen Sie Ihren Nutzern zu, dass sie in der Lage sind, jede sich bietende Fehlermöglichkeit zu nutzen.
(Murphy'sche Regel: Wenn etwas schief gehen kann, so geschieht es!)
- Gehen Sie als Software-Entwickler vom schlimmsten Fall aus:
 - Sie müssen ihr eigenes Produkt nutzen.
 - Sie müssen ihr eigenes Produkt warten.
- Die Produktivität des Nutzers ist zu messen an der Anzahl der Eingabehandlungen, die er tätigen muss, bis er das Problem gelöst hat.

- Routinierte Nutzer bewerten die Benutzeroberfläche eines Programms anders als neue oder gelegentliche Nutzer.
Der routinierte Nutzer sieht mehr den Aufwand bei der täglichen Bedienung des Programms.
Der neue bzw. gelegentliche Nutzer sieht – zunächst – mehr den Aufwand, um die Bedienung des Programms zu lernen.
(ease of use - ease of learning)

- Beide Aussagen haben eine gewisse Berechtigung:
 - Nutzer wissen, was sie wollen!
 - Nutzern muss gesagt werden, was sie wollen!
- Ein Programm, welches der Nutzer nicht nutzt, ist wertlos!
- Der Nutzer weiß, was er will, er kann es nur nicht exakt und nicht vollständig ausdrücken.
- Ein Nutzer kann nicht Dinge wollen, die er nicht kennt!
- 10 % der Wünsche == 90 % des Aufwands.

Grundsätze für die Softwareentwicklung – 4

- Ein Programm sollte so arbeiten, wie der Nutzer es erwartet. Er sollte durch Reaktionen des Programms nicht überrascht werden!
(no surprises)
- Der Software-Entwickler sollte nicht Probleme lösen, die es nicht gibt!
Entscheidend sind nicht die Probleme, die der Entwickler sieht, sondern die, die der Nutzer hat!
Selten ist der Chef der wichtigste Nutzer!
(Allerdings weiß er dies nicht immer!)

- Folgende Fragen sind abzuwägen:
 - Was kostet es, wenn das Programm einen Fehler zulässt?
 - Was kostet es, ein Programm zu entwickeln, welches keine Fehler zulässt?
 - Was kostet es, ein Programm zu nutzen, welches Fehler zulässt?
 - Was kostet es, wenn ein Programm nicht optimal mit Zeit und Speicherplatz umgeht?
 - Was kostet es, wenn niemand – einschließlich des Entwicklers – in der Lage ist, ein "optimales" Programm (rechtzeitig) zu verändern?

- Was ein Programmierer nicht in natürlicher Sprache ausdrücken kann, das kann er auch nicht in Programmcode ausdrücken.
- Wenn der Algorithmus zur Lösung eines Problems zu kompliziert wird, dann suche einen neuen. (Mut zum Neuanfang)
- Wenn ein Problem zu umfangreich ist, dann zerlege es! (Teile und herrsche!)

Grundsätze für die Softwareentwicklung – 7

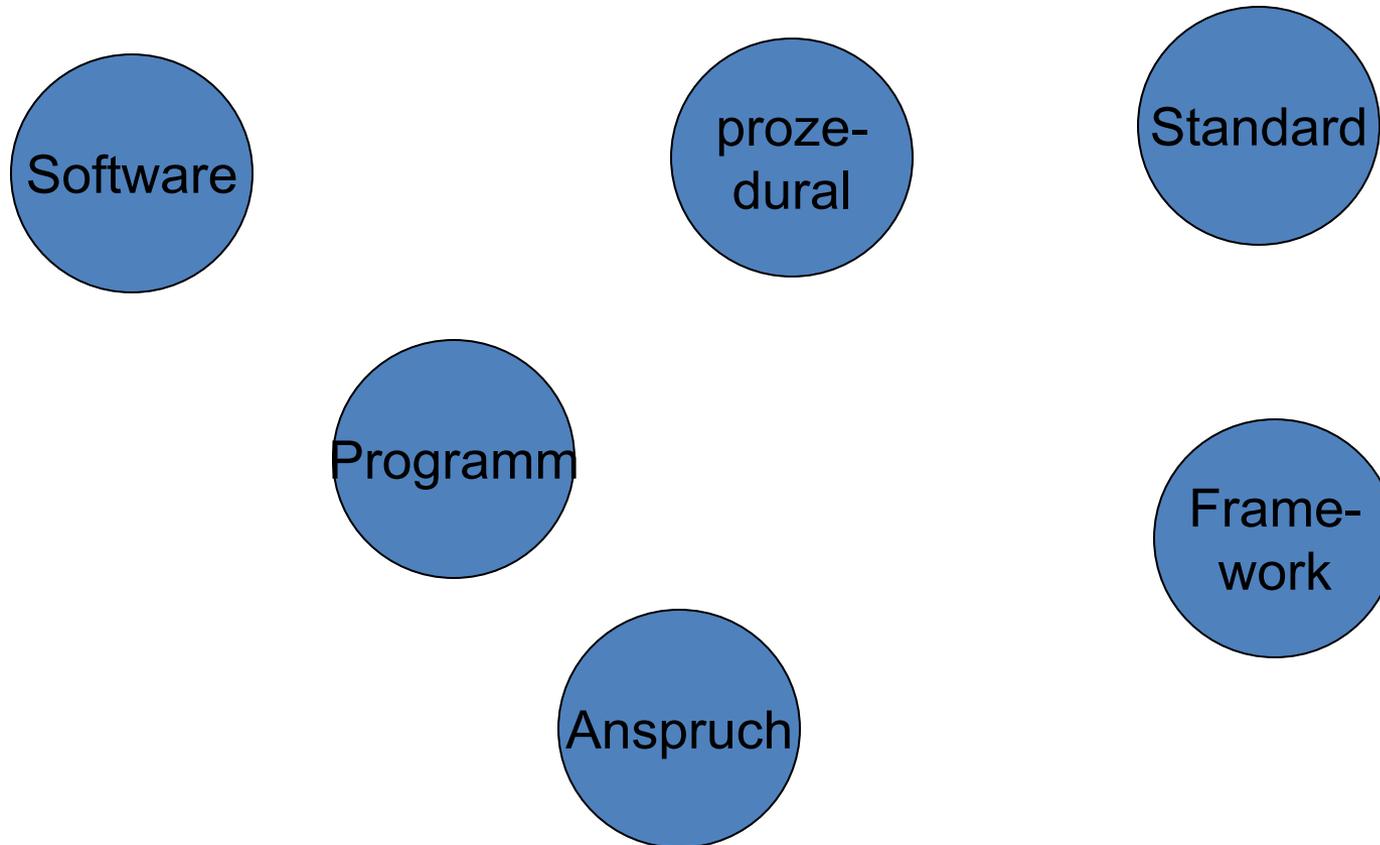
- Ein Problem ist nur dann gut zerlegt, wenn die Teilprobleme wenig voneinander abhängig sind.
- Es ist billiger, einen missratenen Entwicklungsschritt zu wiederholen, als ein missratenes Entwicklungsprodukt jahrelang mühevoll zu warten – oder es nie einzusetzen.

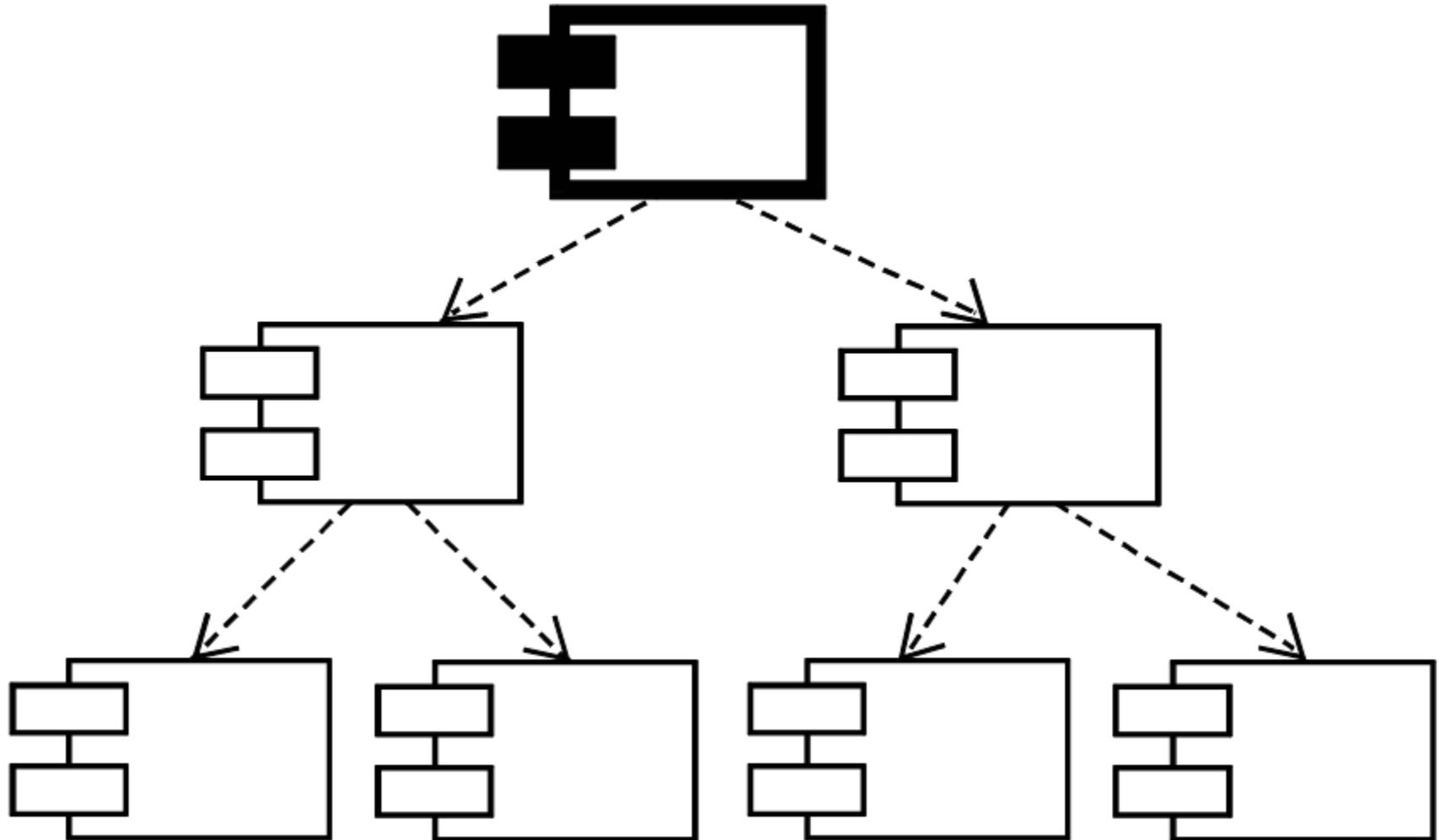
Grundsätze für die Softwareentwicklung – Anmerkungen

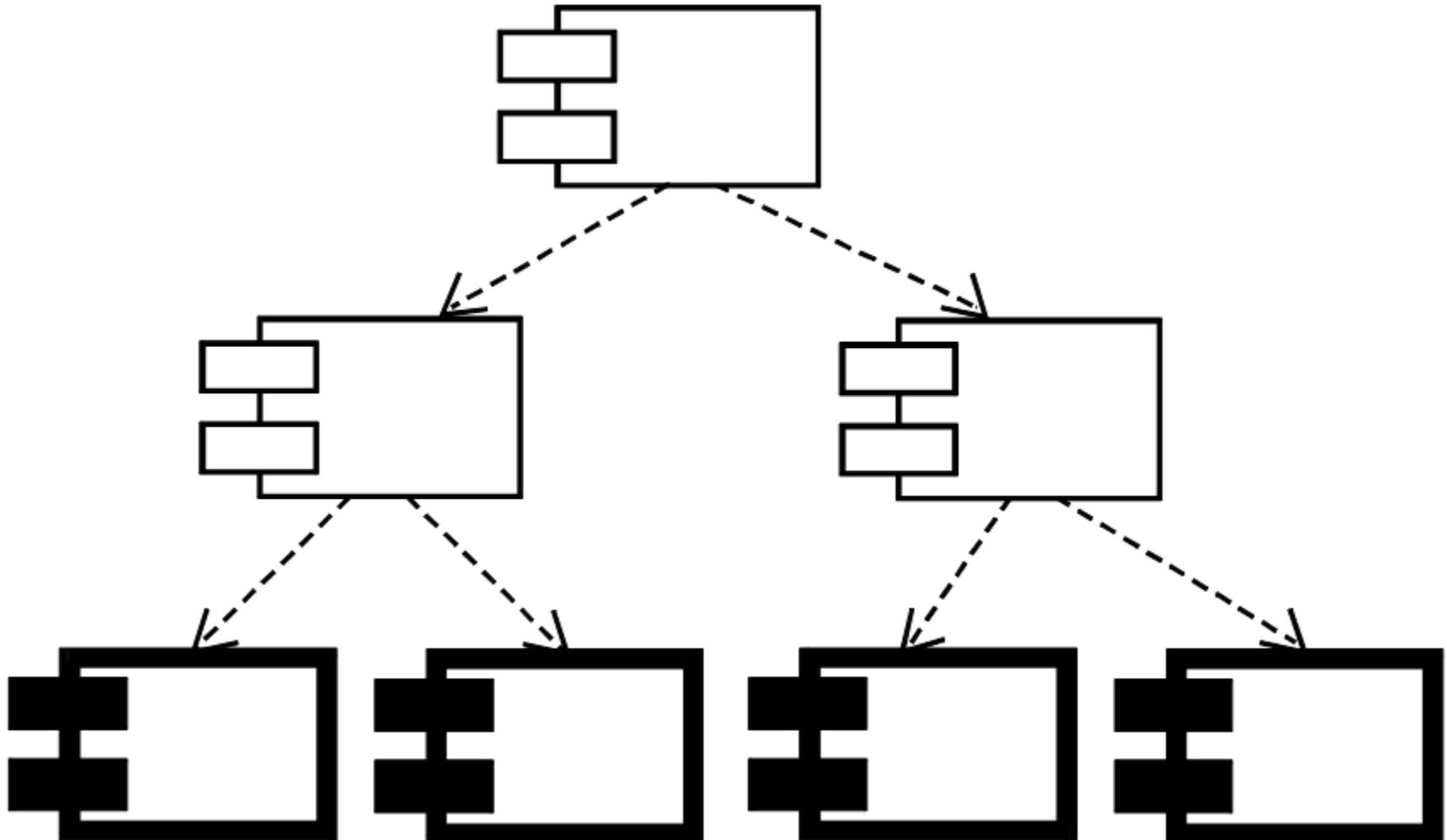
- Auch wenn die Grundsätze teilweise lustig wirken, sollte doch der ernste Hintergrund beachtet werden!
- Es wird misslingen, alle Grundsätze gleich wichtig zu nehmen!

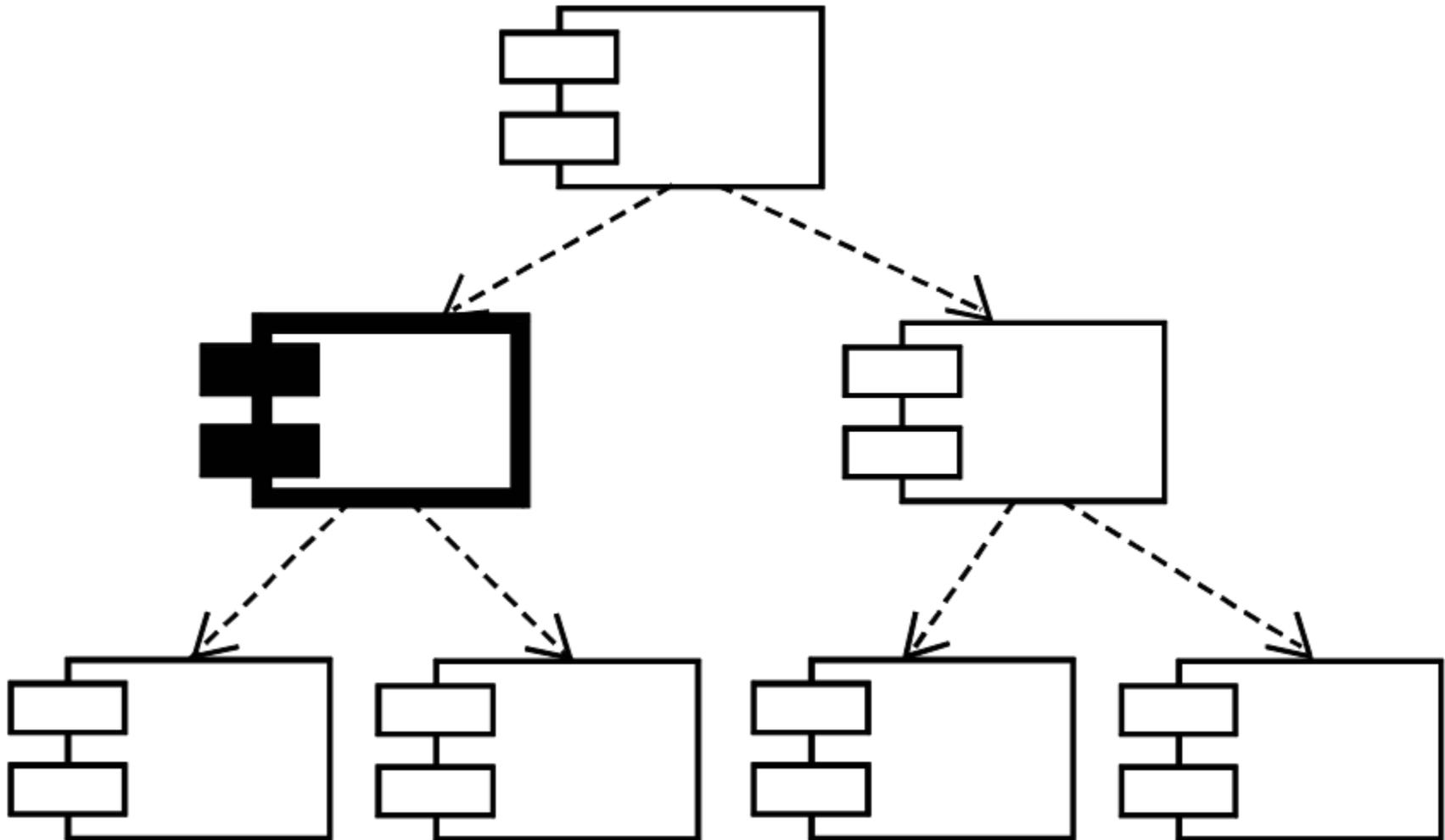


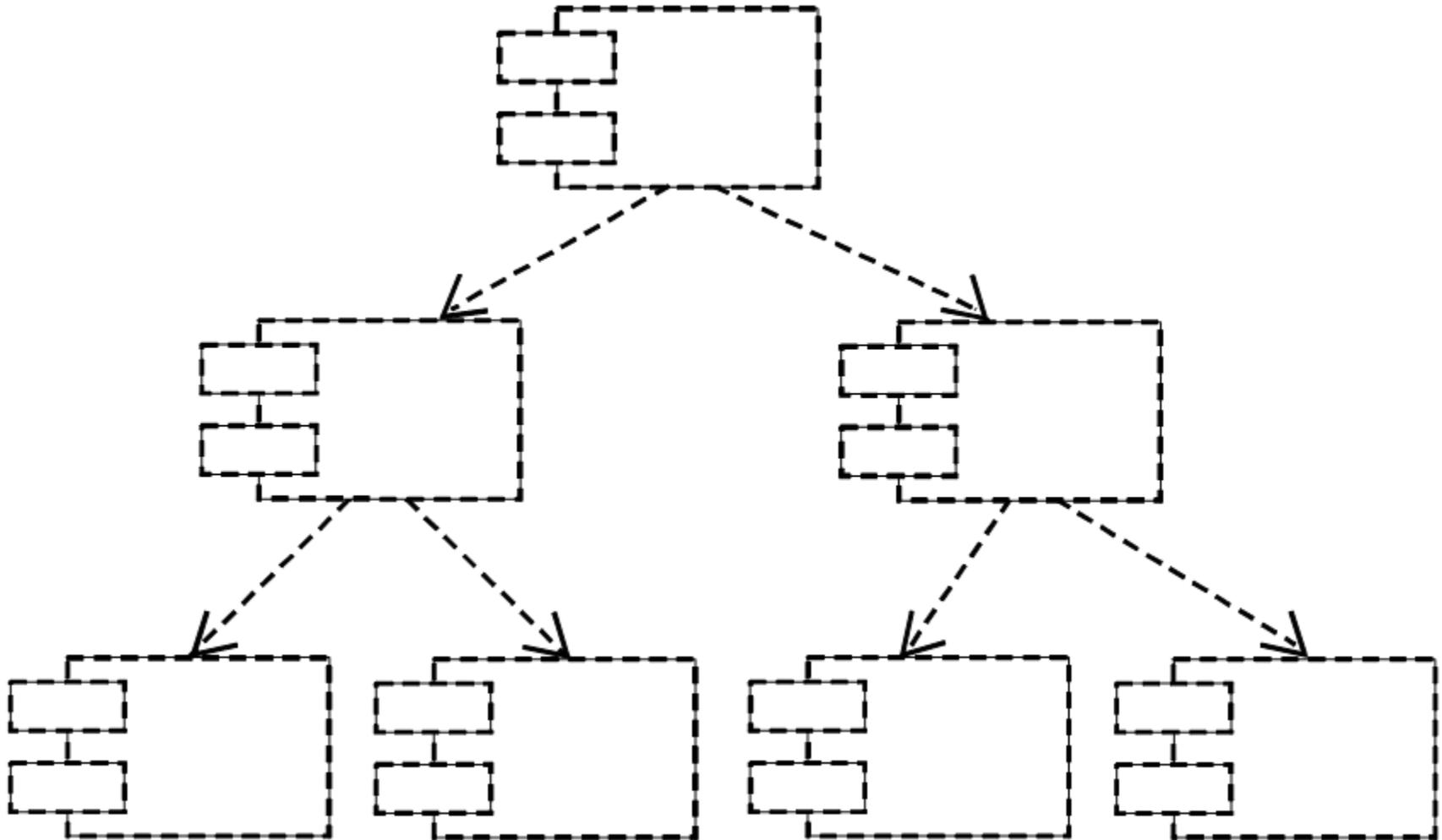
-
- Programmiersprachen
 - Softwareentwicklung
 - • Programmentwicklung
 - strukturierte Programmierung
 - Abbruchbehandlung











- deklarativ
 - Beschreibung des Problems
 - Lösung übernimmt der Inferenzmechanismus
- prozedural
 - Algorithmen in sequentieller Reihenfolge
 - kaum beeinflussbar durch außen
- Ereignis gesteuert
 - Beschreibung, wie auf Ereignisse reagiert wird
 - Reihenfolge beliebig

Steuerung des Programmflusses – deklarativ

- Gebunden an Programmiersprache
- keine weite Verbreitung
- hoher Investitionsaufwand zu Beginn

- PROC (event,struct_event);
- INIT;
- REPEAT;
 - GetEvent(event);
 - HandleEvent(event);
 - UNTIL quit(event);
- END-REPEAT;
- END;

Steuerung des Programmflusses – prozedural

- PROC;
- INIT;
- action-1;
- action-2;
- ...
- action-n;
- END;

- Rahmen für Programmfluss
- Inhalt wird hinzugefügt
- Vorteile
 - Konzentration auf Lösung
 - allgemeine “Probleme” löst schon der Rahmen
 - Bausteine leichter zu schreiben
 - erprobte Teillösung liegt vor

- Nachteile
 - Effizienz des Programms geringer
- Herausforderungen
 - Existiert ein geeigneter Rahmen?
 - Aufwändige Entwicklung des Rahmens
 - Inhalt und Vorgehensweise des Rahmens muss bekannt sein
- lohnenswert daher erst ab bestimmter Problemgröße -> aber: hat Kolleg/in/e etwas?

- ANSI
 - American National Standards Institute
 - auf Programmiersprachen
 - teilweise auf Umgebung
- ISO
 - International Standards Organization
 - Funktionen innerhalb Programmiersprachen
 - Programmierkonstrukte
- teilweise (freundschaftlich) konkurrierend

Programm – Versuch von Definitionen

- Reihenfolge von Reden, Darbietungen
 - Konferenzprogramm, Konzertprogramm
- Plan, Vorhaben
 - Arbeitsplan
- Grundsätze, Zielstellungen
 - Parteiprogramm
- Aneinanderreihung von Algorithmen
 - Computerprogramm

Programm – Gemeinsamkeiten

- Anspruch - es soll etwas erreicht werden
 - Wird das Programm dem Anspruch gerecht?
- Ziel - es gibt eine Vision, ein Ergebnis
 - Wie stehen die Ziele zum Anspruch des Programms?
- Änderungen und Störungen beeinflussen ein Programm
 - Welche Vorkehrungen sind getroffen, damit das Programm keinen Schaden anrichtet?

Programm – Unterschiede

- Ein Mensch geht im Allgemeinen sehr flexibel mit Programmen um. Er ist in der Lage, spontan zu reagieren.
- Der Computer nimmt das Programm sehr ernst. Er macht genau und ausschließlich das, was im Programm steht.

- Fremdwörterbuch, 1977
 - Realisierung eines Algorithmus in der Sprache eines elektronischen Rechenautomaten. Eindeutige und geordnete Zusammenstellung von Befehlen und Daten zur Lösung einer Aufgabe durch elektronische Datenverarbeitungsanlagen.
- Schülerduden, 1986
 - Formulierung eines Algorithmus und der dazugehörigen Datenbereiche in einer Programmiersprache
- Webster's 1995
 - a sequence of coded instructions for a computer



Auswahl der Programmiersprache . . . – 1

- ... der/die Entwickler beherrschen
- ... dem Entwickler am besten gefällt
- ... dem Entwicklerteam am besten gefällt
- ... dem Chef am besten gefällt
- ... dem Kunden am besten gefällt
- ... von vielen anderen in vergleichbaren Fällen verwendet wird
- ... sehr schnellen Programmcode erwarten lässt
- ... sehr Speicher sparenden Programmcode erwarten lässt

Auswahl der Programmiersprache . . . – 2

- ... sehr gut lesbaren Quelltext erwarten lässt
- ... es dem Programmierer schwer macht, unbemerkt Programmierfehler in ein Programm einzubauen
- ... auf allen wichtigen Rechnerplattformen zur Verfügung steht ...
- ... und in Zukunft auf allen wichtigen Rechnerplattformen zur Verfügung stehen wird
- ... für die es zukünftig noch Entwickler geben wird

- Namen - Länge
- Konstanten und Typen
 - Strukturen, Typprüfung, symbolische Variablen
- Wertzuweisungen
 - eindeutig ohne Nebenwirkungen
- Ablauf und Stil
 - Konstrukte klar, Wesen klar und deutlich, nicht zusammen gestoppelt
- Compiler
 - verfügbar, effizienter Code, benutzerfreundlich
- **Portabilität wenn erforderlich**



weitere Kriterien für die Auswahl der Programmiersprache ;-)

- ADA Ein als amerikanischer Straßenkreuzer getarnter Schützenpanzerwagen.
- Assembler Ein Go-Cart ohne Sicherheitsgurt und Überrollbügel. Gewinnt jedes Rennen, wenn es nicht vorher im Graben landet.
- BASIC Eine Ente - weder modern noch besonders schnell, aber für jeden erschwinglich. Und manch einer, der sich daran gewöhnt hat, will gar nichts anderes mehr haben.
- C Ein offener Geländewagen. Kommt durch jeden Matsch und Schlamm, der Fahrer sieht hinterher auch entsprechend aus.
- COBOL Ein dunkelroter Benz mit getöntem Panzerglas und kostbaren Intarsienarbeiten im Fond. Kein Mensch fährt diesen Wagen selbst; man läßt fahren.
- FORTRAN Ein Schlitten aus den fünfziger Jahren mit riesigen Heckflossen. Erntet bei der technischen Überprüfung stets mißtrauische Blicke, überholt aber noch manch neueres Gefährt.
- LISP Ein Prototyp mit Telepathie-Steuerung. Kann außer von seinem Erfinder von niemanden bedient werden.
- PASCAL Entwurf eines amerikanischen Straßenkreuzers, der nur durch Versehen in die Serienproduktion gelangte.
- MODULA-2 Wie Pascal, aber mit dreifachen Sicherheitsgurten, seitlichen Stoßstangen und separatem Gaspedal für jeden der fünf Gänge.
- PL/1 Ein handgefertigter Eigenbau, mit Teilen von FORTRAN, COBOL, PASCAL und ADA. Entsprechend sieht er aus.
- PROLOG Enthält statt eines Lenkrades eine Automatik, die alle Straßen solange absucht, bis das gewünschte Fahrziel erreicht ist.

weitere Kriterien für die Auswahl der Programmiersprache ;-)

- Fluchen ist die einzige Sprache, die alle Programmierer wirklich beherrschen.
- Man muß ein Idiot sein, um das Programm eines anderen Idioten verstehen zu können.
- Die einzige gute Programmiersprache ist die, die man nicht benutzen darf.
- Es hat wenig Sinn, eine Programmiersprache verstehen zu wollen.

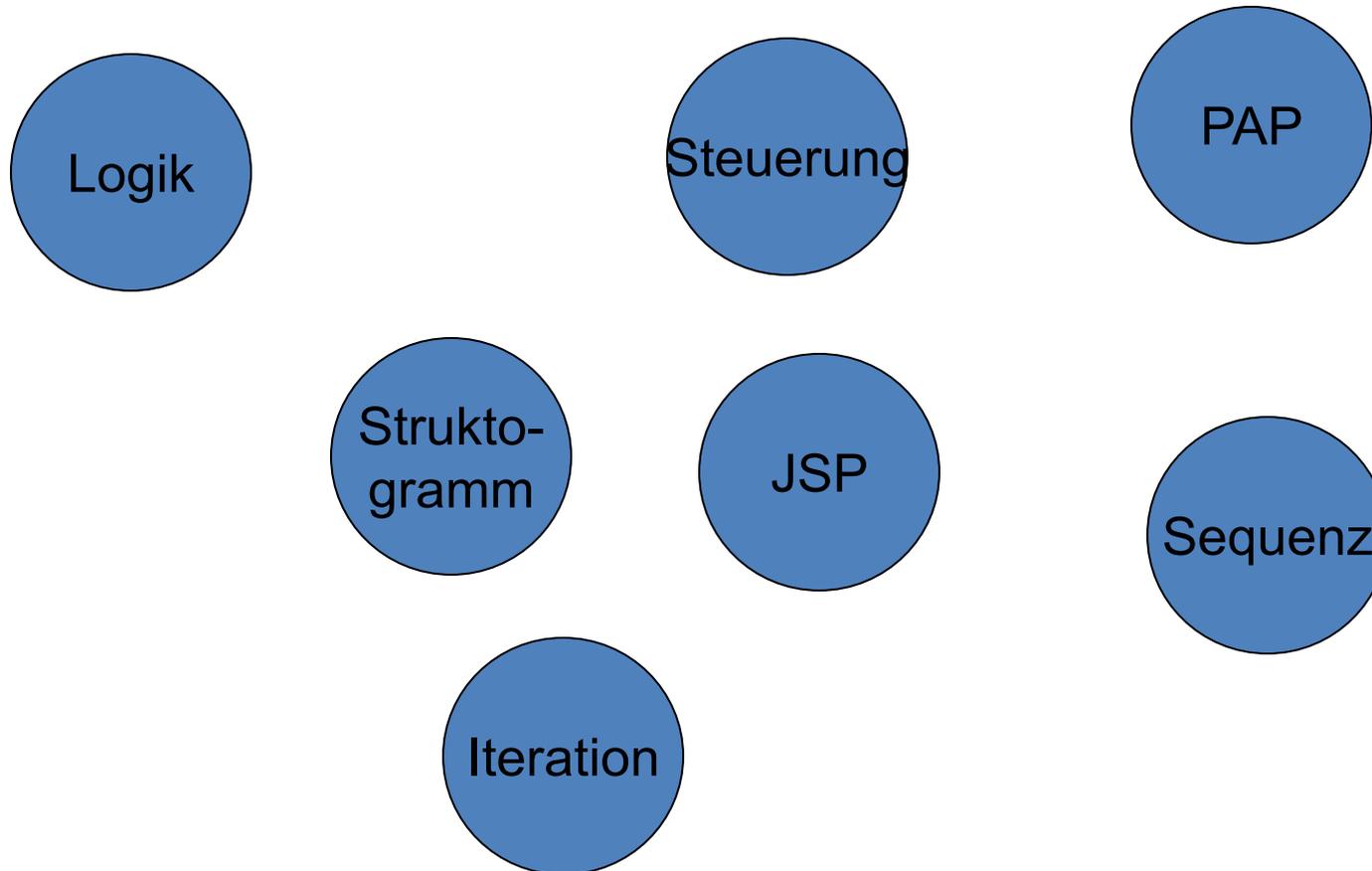
wichtige Aussagen zu Programmen ;-)

- Jedes fertige Programm, das läuft, ist veraltet.
- Wenn ein Programm korrekt läuft, muss es geändert werden.
- Wenn ein Programm nutzlos ist, muss es dokumentiert werden.
- Erst wenn ein Programm mindestens 6 Monate läuft, wird der schlimme Fehler entdeckt werden.
- Der Fehler befindet sich immer in der Routine, die niemals getestet wurde.
- Es gibt immer einen Fehler mehr als gedacht.

und ... ;-)

- Ein Computerprogramm tut, was du schreibst, nicht was Du willst.
- Es ist unmöglich, ein Programm fehlerfrei zu programmieren; Dummköpfe sind erfinderisch.
- Bedenke: keine Sprache an sich ist gut oder schlecht, sondern der, der sie benutzt!
 - Wenn man einen Teelöffel Wein in ein Fass Jauche gießt, ist das Resultat Jauche.
 - Wenn man einen Teelöffel Jauche in ein Fass Wein gießt, ist das Resultat ebenfalls Jauche.

-
- Programmiersprachen
 - Softwareentwicklung
 - Programmentwicklung
 - • strukturierte Programmierung
 - Abbruchbehandlung



Grundkonzepte

- Bildung von logischen Programmeinheiten
- hierarchische Programmorganisation
- Definition einer zentralen Programmsteuerung
- Beschränkung der Ablaufsteuerung
- Beschränkung der Datenverfügbarkeit

- Sequenz
- Verzweigung
 - (un)vollständige Alternative
 - Mehrfachverzweigung
 - Fallauswahl
- Iteration
 - (Nicht)Abweisschleife
 - verallgemeinerter Zyklus

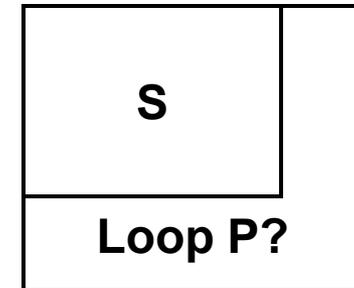
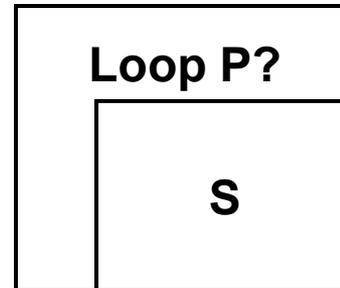
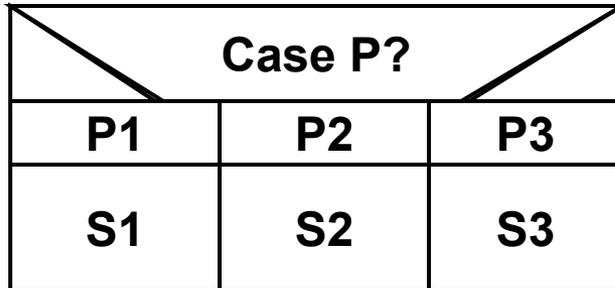
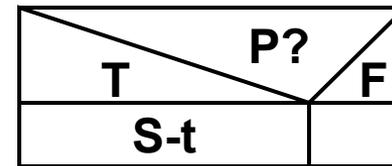
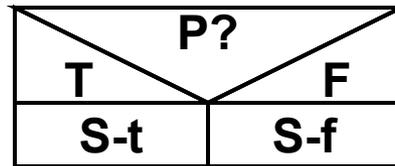
Darstellungsmittel

- Struktogramm
- JSP-Diagramm
- Programmablaufplan
- Pseudocode
- “höhere Programmiersprache”

Vorgehensweise

- Fluss zurückführen auf Grundelemente
- von oben nach unten (konsequent)
- komplexe Aktionen schrittweise verfeinern
- Grundelement kann weitere Grundelemente einschließen
- Unterprogramme werden separat behandelt
- Komplexität vermindern
 - der Mensch kann nicht mehr als 5 bis 9 verschiedene Informationen gleichzeitig erfassen

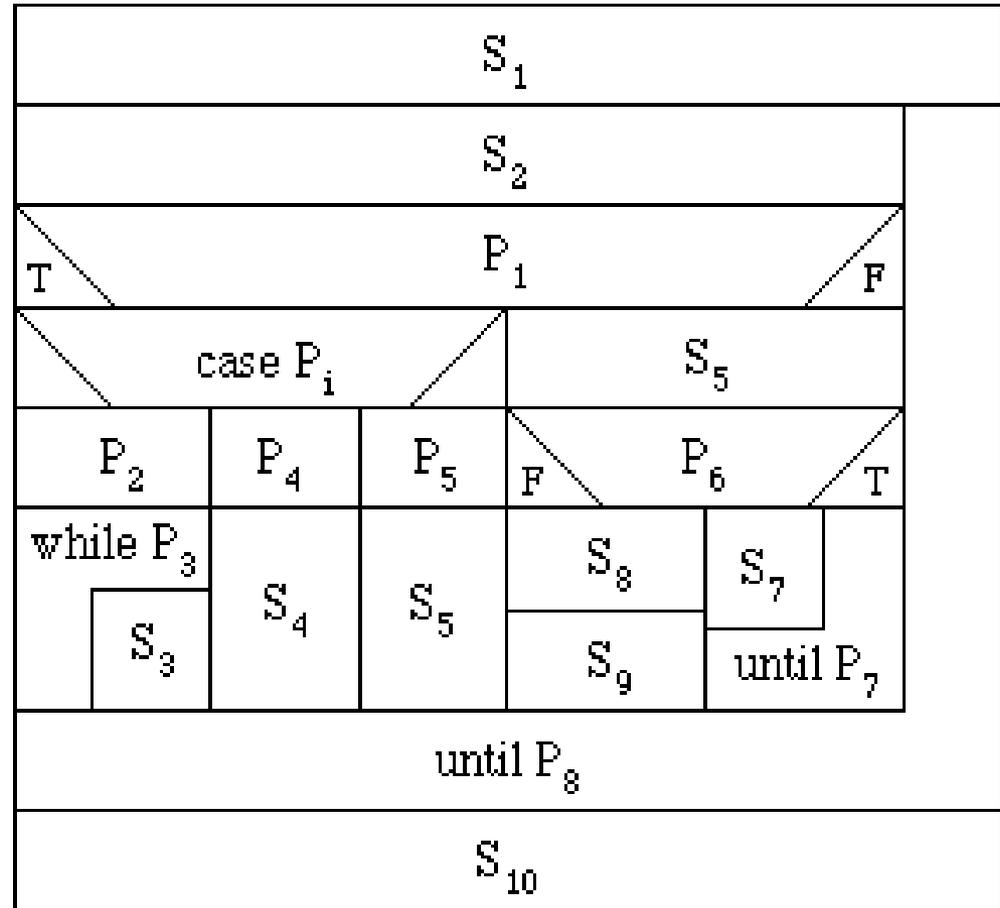
Struktogramm – Grundelemente



Struktogramm – Beispiel

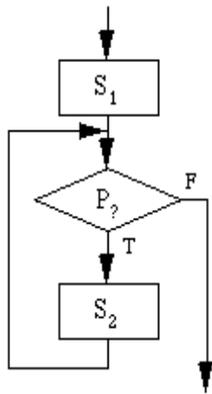
```

S1;
repeat
  S2
  if P1 then
    case (P1 i = 2,4,5)
    of P2:
      while P3 do S3 end while
    of P4: S4
    of P5: S5
    end case
  else S6;
  if P6 then
    repeat S7 until P7
  else S8; S9
  end if
end if
until P8;
S10.
    
```

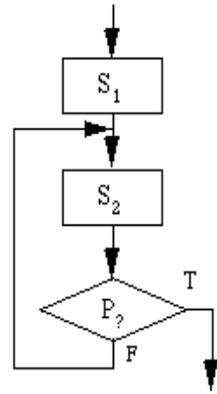


- Maximal 1 Seite pro Struktogramm
- Modularisieren
- es gibt unterstützende Software
 - Weiterentwicklung auf Basis Struktogramm möglich
 - EasyCase (Siemens)
 - AllFusion:Gen (CA)
 - etc.

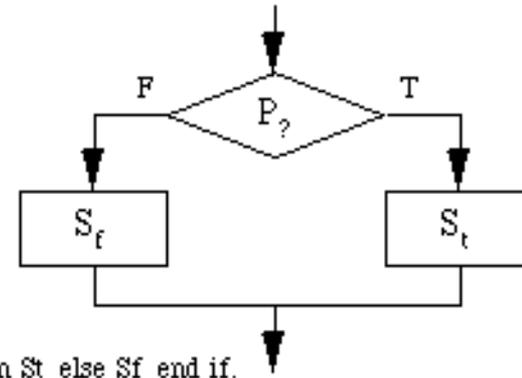
Programmablaufplan



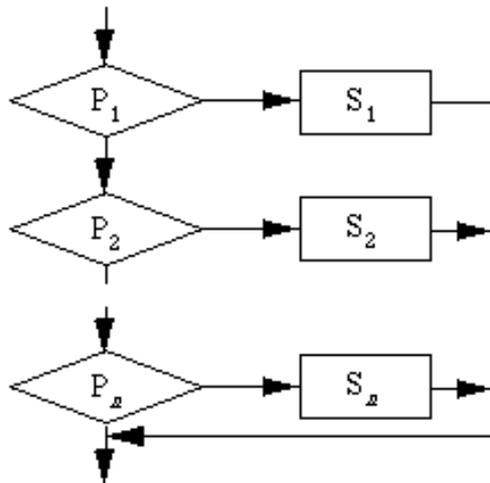
while P do S2 end while.



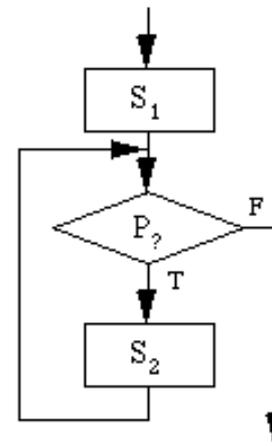
repeat S2 until P end repeat.



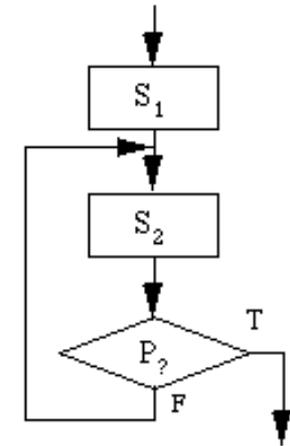
if P then S_t else S_f end if.



case P?
of P_1 : S_1 ;
of P_2 : S_2 ;
...
of P_n : S_n ;
end case.



while P do S2 end while.

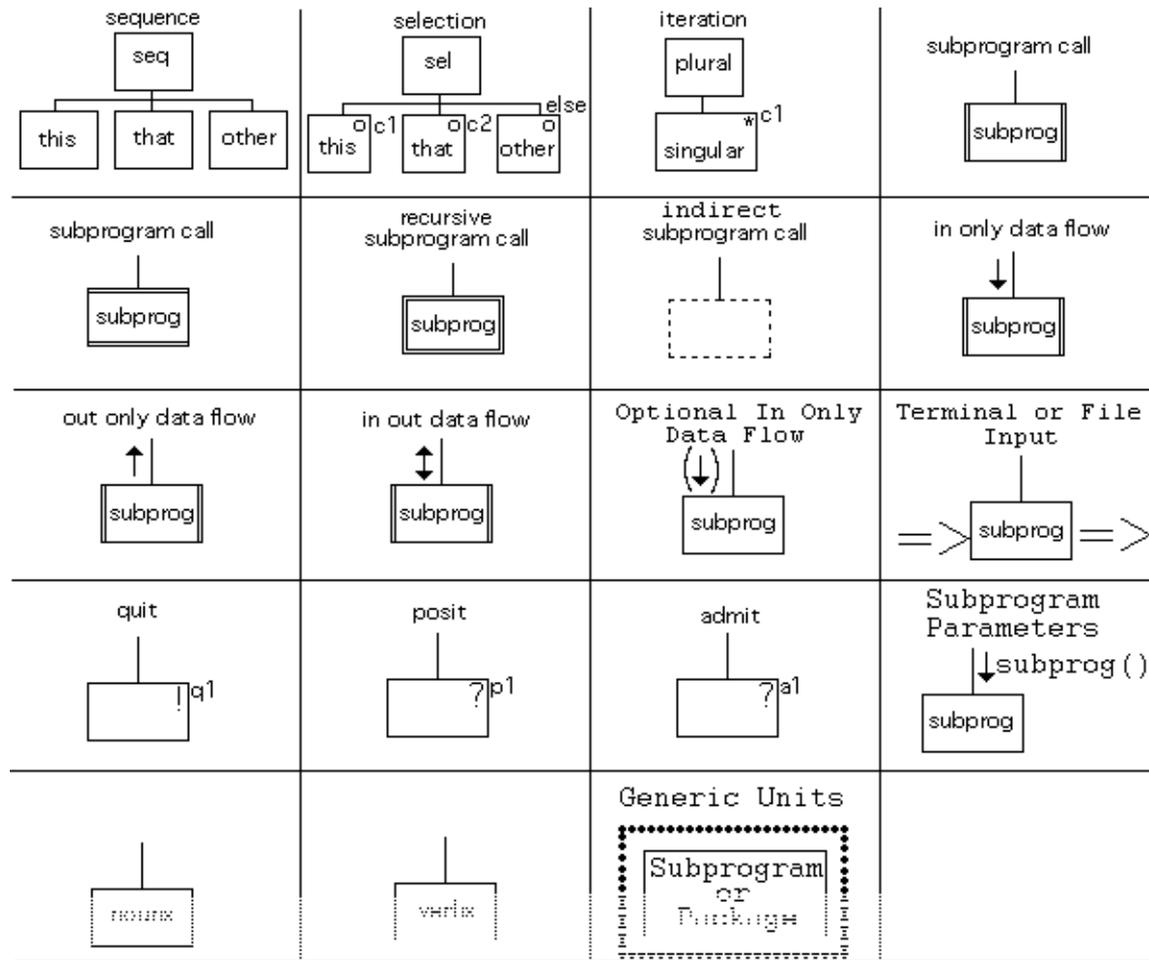


repeat S2 until P end repeat.

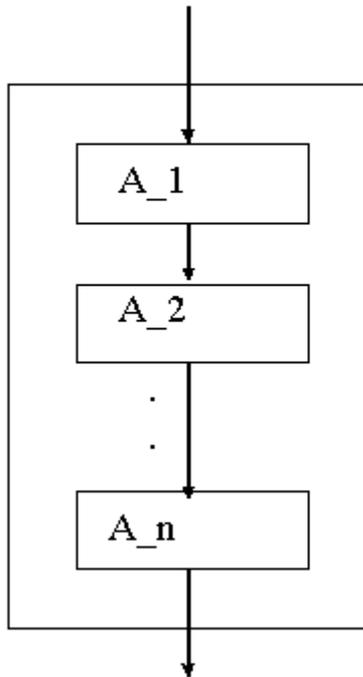


strukturierte Programmierung

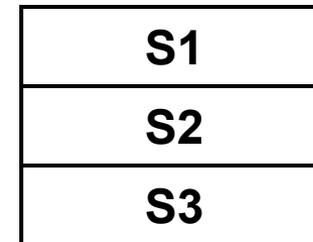
JSP – Jackson structured programming



Anweisungssequenz



```
begin  
  A_1;  
  A_2;  
  ....  
  A_n;  
end;
```



Sequenz

- ist eine Folge von Anweisungen
 - Wertzuweisungen
 - Aufrufe von Unterprogrammen, Prozeduren, Funktionen
 - Sprungbefehle
 - Alternativen
 - Zyklen
- bei Fehler kann die Sequenz unterbrochen werden

S1
S2
S3

Sequenz – Beispiele

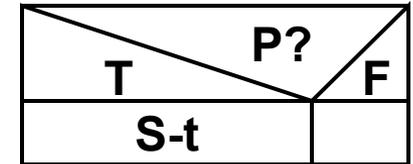
```
MOVE A TO B  
CALL HUGO  
GO TO OTTO  
IF / ELSE  
PERFORM  
CONTINUE  
END  
GOBACK
```

```
B = A;  
CALL HUGO;  
GO TO OTTO;  
IF / ELSE  
DO  
;  
END;  
RETURN;
```

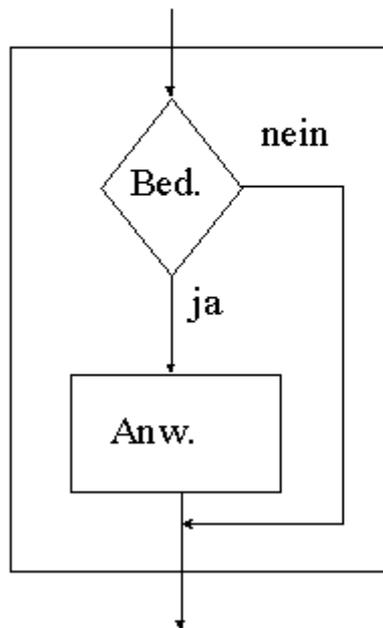
```
b = a;  
call hugo;  
goto otto;  
if / else  
loop while  
say  
end  
return
```

S1
S2
S3





Einseitige Auswahl

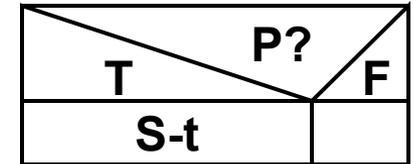


**if Bedingung
then Anw ;**

- S-t ist eine Folge von beliebigen Anweisungen
- S-t wird genau dann ausgeführt, wenn P erfüllt ist

strukturierte Programmierung

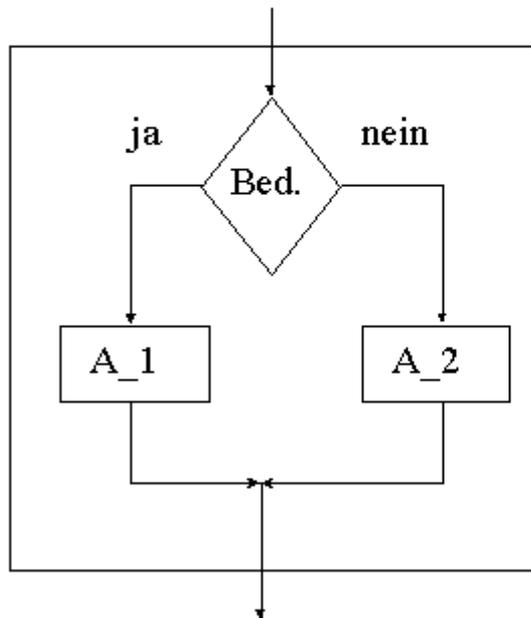
unvollständige Verzweigung – Beispiel



```
if environment = 'TSO' then do;  
    say 'Umgebung = ' sysvar(sysenv);  
end;
```



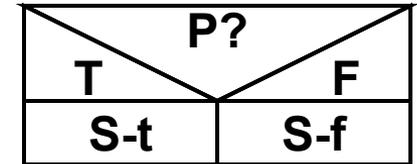
Zweiseitige Auswahl



```
if Bedingung  
  then A_1  
  else A_2;
```

P?	
T	F
S-t	S-f

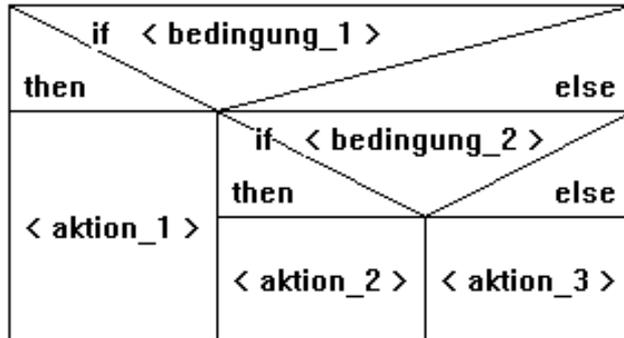
- S-t und S-f sind Folgen von beliebigen Anweisungen
- S-t wird genau dann ausgeführt, wenn P erfüllt ist
- S-f wird genau dann ausgeführt, wenn P nicht erfüllt ist



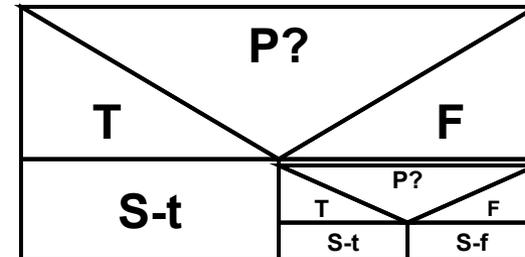
```
if environment = 'TSO'  
then do;  
    say 'Umgebung = ' sysvar(sysenv);  
end;  
else do;  
    say 'wo bin ich denn?'  
end;
```



Mehrfachverzweigung



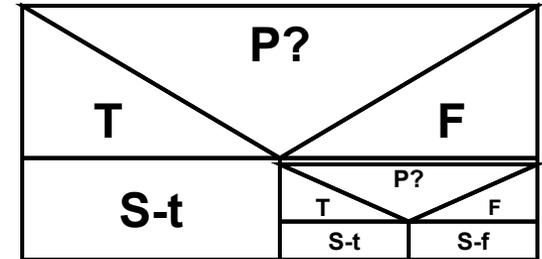
Kein separates PAP-Bild



- Die alternativ auszuführenden Aktionen hängen von verschiedenen Bedingungen ab, die in einer vorgegebenen Reihenfolge ausgewertet werden, d.h. die Bedingungen beruhen auf der Auswertung unterschiedlicher Ausdrücke.

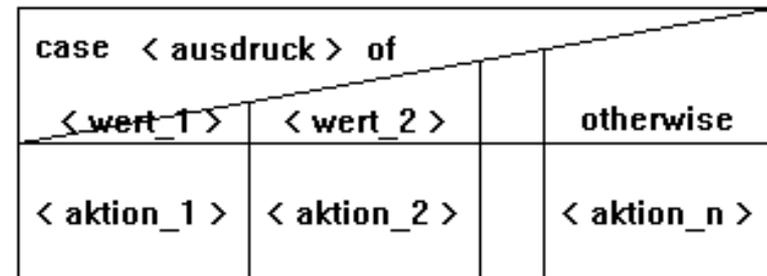
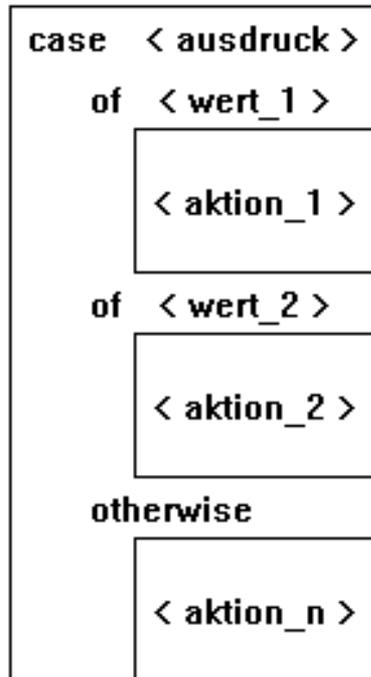
Mehrfachverzweigung – Beispiel

```
if enviroment = 'TSO' then do;  
    say 'Umgebung = ' sysvar(sysenv);  
end;  
else do;  
    if environment = 'CICS' then do;  
        say 'Umgebung = CICS';  
    end;  
    else do;  
        say 'wo bin ich denn?'  
    end;  
end;
```

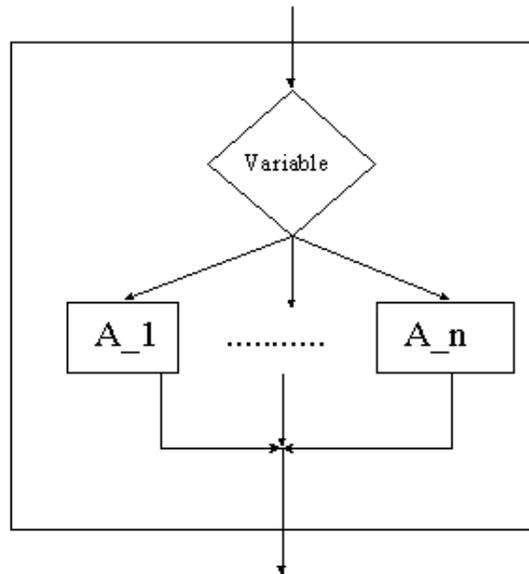


Fallauswahl

- Die alternativ auszuführenden Aktionen hängen von der Auswertung eines einzigen Ausdrucks ab. Kriterium ist der Wert, den dieser Ausdruck liefert.
- Die Fallauswahl ist eine spezielle Form der Mehrfachverzweigung. Die für sie in vielen Programmiersprachen vorgesehene spezielle Notationsform ist jedoch einfacher und übersichtlicher: Alle Bedingungen der Mehrfachverzweigung lassen sich auf die Auswertung nur eines Ausdrucks zurückführen.



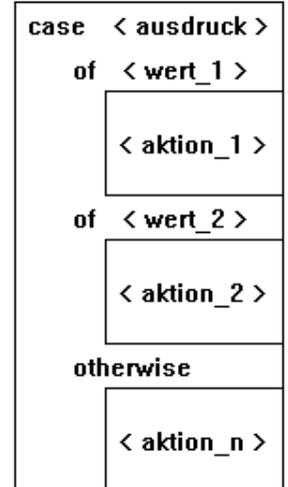
Mehrfachauswahl



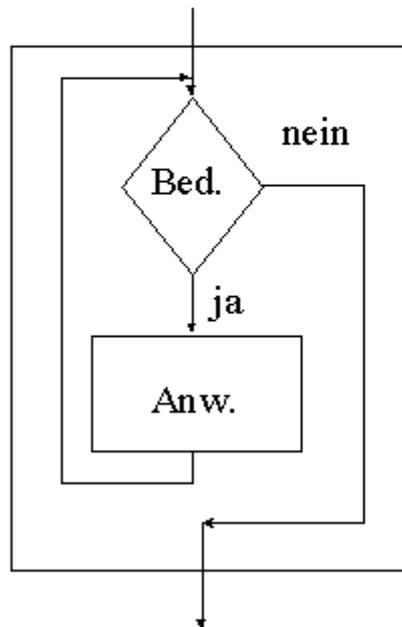
CASE Ausdruck of
wert_1: A_1;
wert_2: A_2;
.
.
.
wert_n: A_n;
END; { case }

Fallauswahl – Beispiel

```
select (environment);  
  when ('TSO')  
    say 'Ich bin im TSO';  
  when ('CICS')  
    say 'Ich bin im CICS';  
  otherwise  
    say 'Ach du Schrott!!';  
end-select;
```



Abweisende Schleife

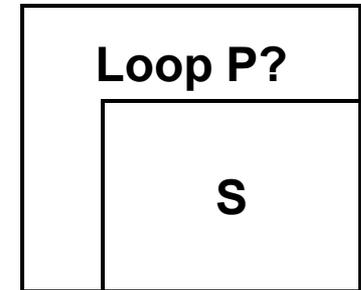


while Bedingung do Anw;

der Schleifenkörper kann 0-mal ausgeführt werden, wenn die Bedingung schon zu Beginn *falsch* ist.

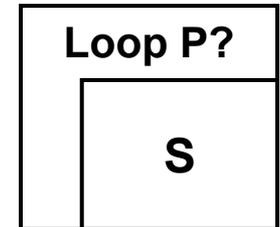
Abweisschleife

- Die Aktion wird solange wiederholt, wie die Bedingung erfüllt ist.
- Die Bedingung wird vor der Aktion geprüft, d.h. die Aktion wird möglicherweise nie ausgeführt.

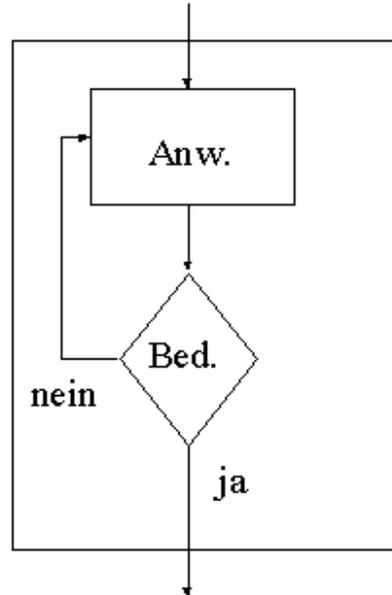


Abweisschleife – Beispiel

```
do while (buli1="bayern")  
  say "Auf! schalke, mach mal!"  
  call bundesligaspiele  
  if punkte(schalke) > punkte(bayern)  
    buli1 = "schalke"  
  end-if  
end-do
```



Nicht abweisende Schleife

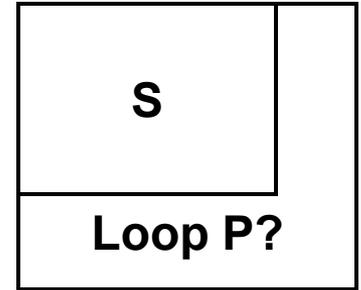


```
repeat  
  A_1;  
  A_2;  
  .  
  .  
  A_n;  
until Bedingung;
```

der Schleifenkörper wird mindestens 1-mal ausgeführt, auch wenn die Bedingung schon zu Beginn *falsch* ist.

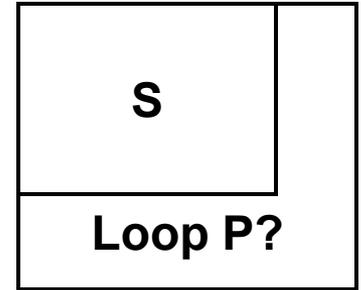
Nichtabweisschleife

- Die Aktion wird solange wiederholt, bis die Bedingung erfüllt ist.
- Sie wird immer mindestens einmal ausgeführt.



Nichtabweisschleife – Beispiel

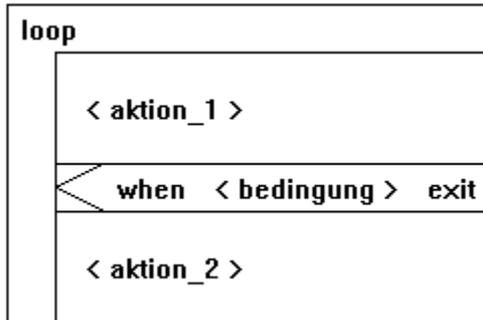
```
do until (buli1="schalke")  
  say "Auf! schalke, mach mal!"  
  call bundesligaspiele  
  if punkte(schalke) > punkte(bayern)  
    buli1 = "schalke"  
  end-if  
end-do
```



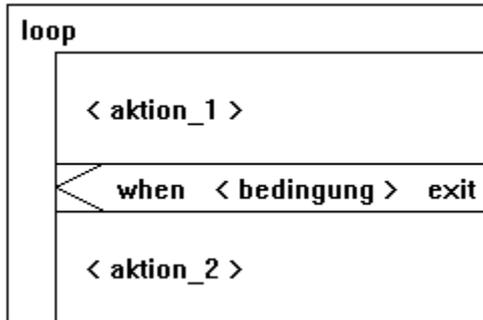
- **Achtung!**
In den Programmiersprachen sehr uneinheitliche Sprachregelung.
Genau hinsehen, was der Loop-Befehl bewirkt.



verallgemeinerter Zyklus



- Innerhalb des Aktionsteils befindet sich eine Abbruchbedingung, d.h. der Zyklus wird verlassen, wenn diese Bedingung erfüllt ist.
- Die Aktionen 1 und 2 werden wiederholt, bis die Abbruchbedingung erfüllt ist. Aktion 1 wird dabei einmal mehr ausgeführt als die Aktion 2.
- Fast keine Sprache kennt diesen Konstrukt.



do for ever;

say "Auf! schalke, mach mal!"

call bundesligaspiele

if punkte(schalke) > punkte(bayern)

leave

end-if

end-do



Zählschleife



- Es gibt eine Laufvariable, die von einem festen Anfangswert in einer Schrittweite bis zu einem Endewert verändert wird.
- Pro Wert der Laufvariable wird aktion ausgeführt.

Zählschleife – Beispiel



```
do i = 1 to CPUS.0
```

```
  say 'CPU' i ' has CPU info ' CPUS.i
```

```
end
```



Rekursion

- besondere Art einer Schleife
- kein eigenes Struktogramm definiert
- Beispiel: Fakultät
 - $n! = n * (n-1)!$ für $n > 1$
 - $n! = 1$ für $n = 1$
- wichtig sind
 - Rekursionsvorschrift (was ist zu tun)
 - Rekursionsverankerung (wie lange ist es zu tun)

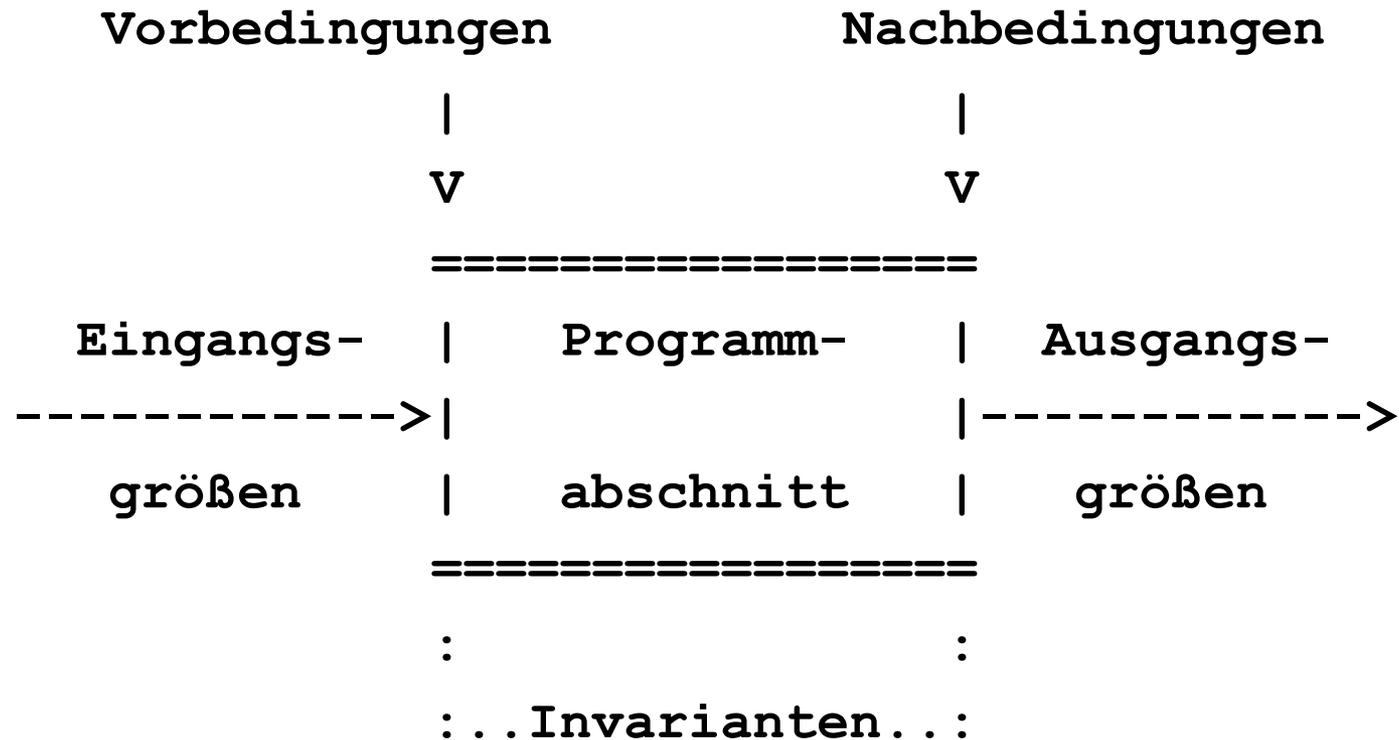


Sprungbefehl

- Ein Sprungbefehl widerspricht den Regeln der strukturierten Programmierung.
- Typen
 - direkte Sprünge zu (fast) beliebigem Code
 - kontrollierte Sprünge in Form von Verlassen einer Sequenz



Zusicherungen



-
- Programmiersprachen
 - Softwareentwicklung
 - Programmentwicklung
 - strukturierte Programmierung
 - • Abbruchbehandlung

ABEND

Abbruch

Speicher

Datei

Strukto-
gramm

Anwei-
sung

Situationen

- Hardware-Probleme
- Software-Probleme
- Kommunikationsprobleme
- Autorisierungsprobleme
- Verfahrensprobleme
- Datenprobleme

- Auslöser im Normalfall
 - Technik
 - technisches Modul
 - selten Konstrukt in der Programmiersprache
- Behandlung
 - durch Technik
 - durch technisches Modul
 - selten Konstrukt in der Programmiersprache
 - keinen erneuten Fehler provozieren!

- das Schließen von Dateien
- das Aufheben von Zugriffssperren
- die Freigabe von dynamisch angefordertem Speicherplatz
- also: gesamtes technisches Umfeld sauber zurücksetzen für
 - Neustart
 - Restart



-
- Programmiersprachen
 - Softwareentwicklung
 - Programmentwicklung
 - strukturierte Programmierung
 - Abbruchbehandlung

