Programmierstile

Andreas Schwill

Institut für Informatik

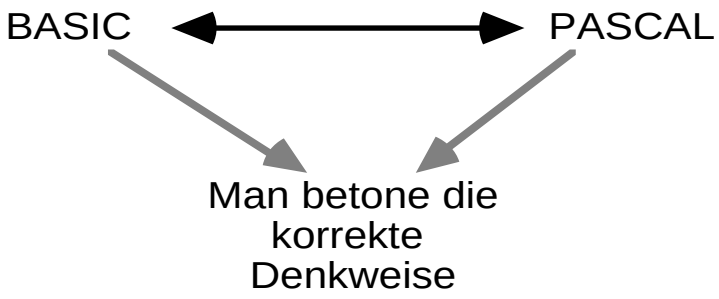
Universität Potsdam

Überblick

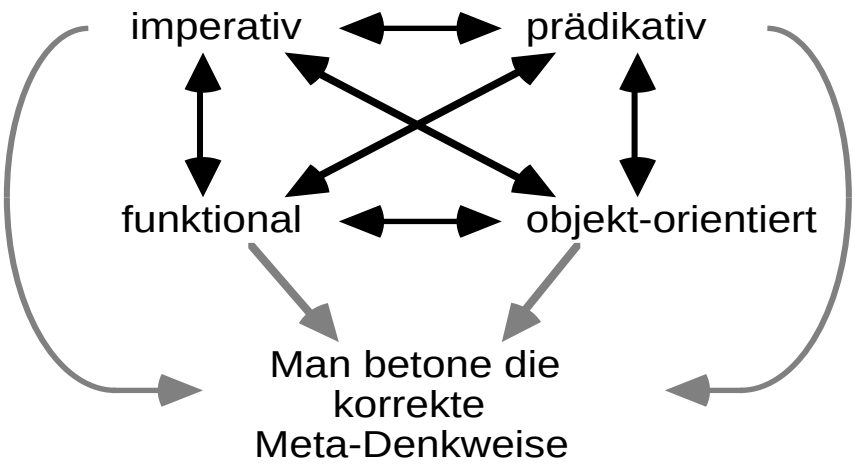
- **Der Sprachenstreit**
- **Klassifikation von Programmierstilen**
  = imperativ
  = funktional
  = prädikativ
- **Programmierstile im Vergleich**
- **Kognitive Aspekte objektorientierter Programmierung**
- **Methodische Hinweise**

1 Der Sprachenstreit

1975: Welche Programmiersprache ist für die Schule am geeignetsten?



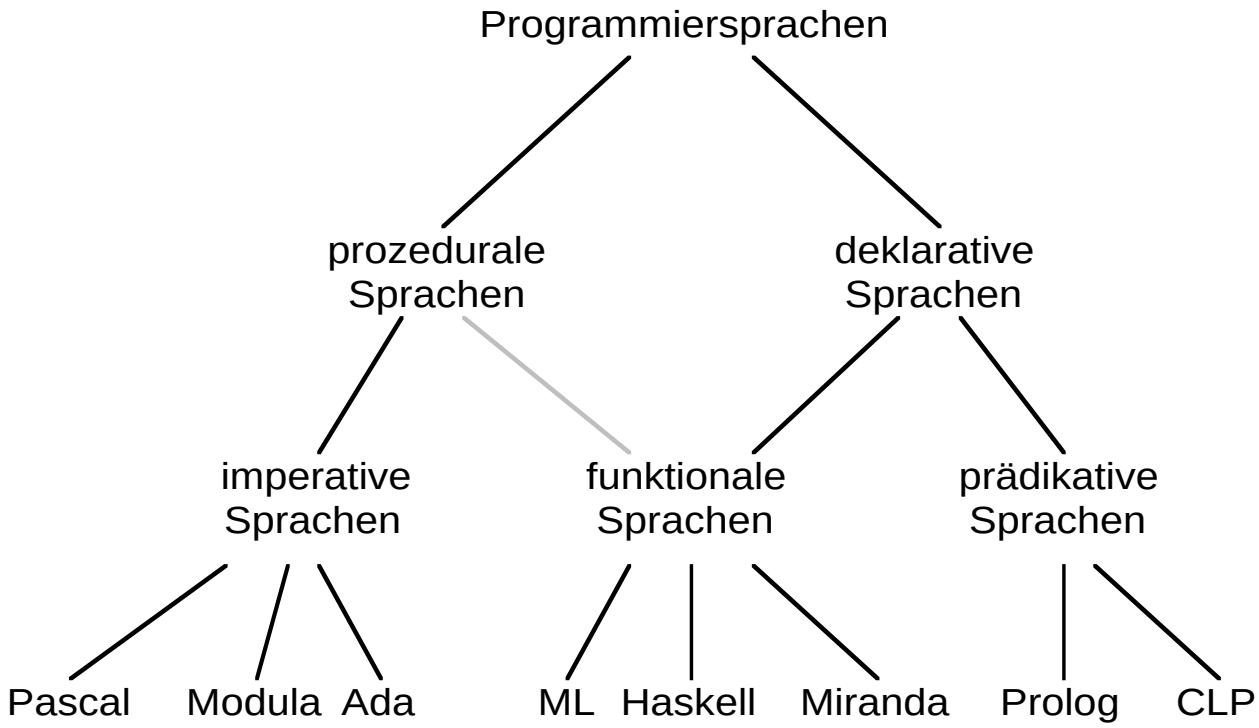
1990: Welche Denkweise ist für die Schule am geeignetsten?



2005: Welche Meta-Denkweise (??) ist die geeignetste?

...

2 Klassifikation von Programmierstilen



objektorientierte Programmierung ???

=> schief zu o.g. Stilen, mit allen drei Stilen kombinierbar

objektorientierte Programmierung <-> strukturierte Programmierung

Softwareentwurfsmethoden

evolutionäre Software-Entwicklung (rapid prototyping).

Programmieren → (Re-)Konfigurieren

Objektorientierte Programmierung: leistungsfähige Typsysteme
Datenabstraktion, -kapselung,
Vererbung,
Polymorphie

Objekt = Daten + Operationen (*Methoden*) + Vererbung

Klasse = Zusammenfassung von Objekten mit gemeinsamen Merkmalen

Kommunikation durch **Nachrichten** (Daten oder Anweisungen, bestimmte Methoden auszuführen)

objektorientierte Programmiersprachen (im Kern imperativ):
Smalltalk-80 (als Reinform),
Oberon, Simula (als Urahn), C++ oder Eiffel.

Prozedurale Programmierung

Zweck: exakte Festlegung, auf welchem Weg die Lösung zu einem Problem maschinell zu berechnen ist,
unmittelbar ersichtlich: gegebene Größen, gesuchte Größen, Lösungsweg

Deklarative Programmierung

zweckfrei: exakte Beschreibung der allgemeinen Eigenschaften gewisser Objekte sowie ihrer Beziehungen untereinander (*Wissen*), keine Problemstellung,
unklar: bekannte und gesuchte Größen, Lösungsweg

Zweck: Ergänzung um eine Problembeschreibung mit Angabe der bekannten und gesuchten Größen.
Aufgabe des Computers: Nutzung des Wissens zur Lösung des Problems.

Bisher keine deklarativen Programmiersprachen im engeren Sinne.

Beispiel: deklaratives „Programm“ („**zweckfrei**“):

Die Kaufkraft eines Geldbetrages sinkt nach Ablauf eines Jahres um die Preissteigerungsrate.

Mögliche Zwecke:

- Berechnung der Kaufkraft K am Schluß eines Jahres zu einem gegebenen Geldbetrag G zu Beginn eines Jahres und zu gegebener Preissteigerungsrate P,
- Ermittlung von P nach Ablauf eines Jahres zu G und K,
- Berechnung einer Tabelle von Paaren (G,P) zu K, so daß G vermindert um P der Kaufkraft K entspricht,
- Ausgabe aller Tripel (G,P,K) mit der spezifizierten Eigenschaft.

Weitere Beispiele für deklarative „Programme“:

(1) Katzen trinken Milch.
(2) Wenn A und B dieselbe Mutter haben, dann sind A und B Geschwister.
(3) Das Quadrat einer geraden/ungeraden Zahl ist gerade/ungerade.

Beispiel: prozedurales Programm (zweckhaltig):

Die Kaufkraft eines Geldbetrages nach Ablauf eines Jahres erhält man, indem man den Geldbetrag um die Preissteigerungsrate ver mindert.

Präzisierung eines Problems und des zugehörigen maschinell nachvollziehbaren Lösungswegs,
Festlegung, welche Größen gegeben und welche gesucht sind.

Weitere Beispiele für prozedurale „Programme“:

(1) Zum Öffnen Lasche anheben, zusammendrücken und farbige Ecke abreißen.
(2) Falls Sie weitere Informationen wünschen, brauchen Sie nur den ausgefüllten Coupon zurückzusenden.
(3) Zur Installation der Software müssen Sie mindestens die Dateien X und Y auf Ihre Festplatte kopieren. Alles weitere entnehmen Sie der Datei „Liesmich“.

Die Programmierstile im Detail.

Unterscheidungsmerkmale:

- **Kalkül** oder **mathematisches Modell**.
  Programmiersprache = Ergänzung des Modells um syntaktische Konstrukte (*syntactic sugar*)

- der Begriff des **Programms**.

- **Start** eines Programms.

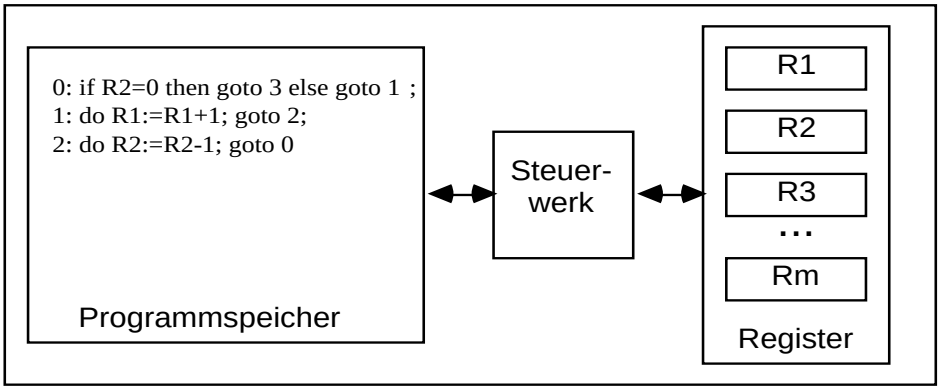
- der Begriff des **Befehls**, besser des elementaren Bausteins zur Beschreibung von Programmen und der **Konstruktoren**, um die elementaren Bausteine zu komplexeren zusammenzusetzen.

- das **Variablenkonzept**.

- das **Beispiel**: Mischen/Verschmelzen von Zahlenfolgen:
  Gegeben sind zwei aufsteigend sortierte Folgen von ganzen Zahlen. Gesucht ist ein Algorithmus, der die beiden Folgen zu einer aufsteigend sortierten Folge mischt.

Imperative Programmierung.

Modell: Registermaschine.



Programm: Zusammenstellung von Befehlen oder Anweisungen.

Start eines Programms: Ausführung des ersten Befehls.

Anweisung: Aufforderung an den Computer, eine Handlung auszuführen,

elementare Anweisungen: Zuweisung,

Konstruktoren: Konkatenation (Sequenz von Anweisungen),
Alternative (bedingte Anweisung: if, case),
Iteration (Schleife: while, repeat).

Variablenkonzept: Behälter mit Bezeichner,

Operationen auf Variablen: **Lesen** und **Schreiben**.

Historische Entwicklung:

1950: Maschinen- und Assemblersprachen.
1954: FORTRAN
1958: ALGOL60
1959: COBOL
1962: BASIC
1970: PASCAL
1974: C
1977: MODULA
1979: ADA

Beispiele:

1) Mischen in einer fiktiven imperativen Programmiersprache:

```
program misch:
var f,g,h: sequence of integer;
while f ≠ [ ] and g ≠ [ ] do
    if first(f) < first(g) then
        h ← h • [first(f)]
        f ← rest(f)
    else
        h ← h • [first(g)]
        g ← rest(g)
    end
end
h ← h • f • g.
```

Start des Programms: run misch.

2) Mischen in PASCAL:

```
program misch(f,g,h);
var f,g,h: file of integer;
begin
    reset(f); reset(g); rewrite(h);
    while not (eof(f) or eof(g)) do
    begin
        if f↑ < g↑ then
        begin
            h↑:=f↑; put(h); get(f)
        end else
        begin
            h↑:=g↑; put(h); get(g)
        end
    end;
    while not eof(f) do
    begin
        h↑:=f↑; put(h); get(f)
    end;
    while not eof(g) do
    begin
        h↑:=g↑; put(h); get(g)
    end
end.
```

Funktionale Programmierung.

Kalkül: **λ-Kalkül**: mathematischer Formalismus zur Beschreibung von Definition und Anwendung von Funktionen, die durch Rechenvorschriften gegeben sind (A. Church, 1941).

Programm: Menge von Funktionsdefinitionen (Funktionalgleichungen), die nach bestimmten Grundprinzipien aufgebaut sind.

Start eines Programms: Aufruf einer Funktion mit Parametern.

Funktion: Funktionen im mathematischen Sinne, also Abbildungen f: A→B von einer Menge A in eine Menge B.
Definition durch **Rechenvorschriften** statt mengentheoretisch.
elementare Funktionen: Addition,
Subtraktion,
Operationen auf linearen Listen,
Alternative if B then F(...) else G(...) usw.
Konstruktoren: Komposition/Einsetzung,
Anwendung einer Funktion mit Parameterersetzung,
Abstraktion (Parametrisierung eines Ausdrucks).

Variablenkonzept: Bezeichner, der an ein Objekt gebunden ist,
keine Zuordnung von Variablen zu Speicherzellen (=> keine Speicherung von Zwischenwerten, keine Seiteneffekte), **referentielle Transparenz**.
Wichtigste Operation: Substitution.
Übliche Substitutionsregeln: Call-by-value-Strategie,
Call-by-name-Strategie,
Call-by-need-Strategie (lazy evaluation).

Historische Entwicklung:

1930: A. Church: Entwicklung des λ-Kalküls.

1958: J. McCarthy: Programmiersprache LISP auf der Basis des λ-Kalküls. Weiterentwicklungen von LISP in der Folgezeit (z.B. SCHEME).

1965: P. Landin: Sprache ISWIM (Vorläufer von ML). In der Folgezeit: Entwicklung vieler zentraler Ideen im Zusammenhang mit der Anwendung, Notation und Implementierung funktionaler Sprachen.

1978: J. Backus: Sprache FP

1978: R. Milner: Sprache ML mit leistungsfähigem Typkonzept und Definition von abstrakten Datentypen.

Beispiele:

1) Mischen in einer fiktiven funktionalen Programmiersprache:

```
function misch f: sequence of integer g: sequence of integer →
                                    sequence of integer ≡
if f=[ ] then g else
     if g=[ ] then f else
          if first(f) < first(g) then
               first(f)•misch(rest(f),g)
          else first(g)•misch(f,rest(g)).
```

Anwendung der Funktion durch Aufruf, z.B.: misch [3,7,19,54] [2,3,5,16,74].

2) Mischen in ML:

```
fun misch [ ] g  = g: int list  |
     misch f [ ] = f   |
     misch (x::r) (x'::r') = if x<x' then [x]@(misch r (x'::r'))
                                    else [x]@(misch (x::r) r').
```

Prädikative Programmierung.

Kalkül: Prädikatenlogik.

Beispiel: typische Formel des Prädikatenkalküls:

$$(\exists f)\ (f(a)=b \wedge (\copyright x)\ (p(x) \Rightarrow f(x)=g(x,f(h(x)))))).$$

Programm: Menge von wahren Aussagen (**Fakten**) und von logischen **Schlußregeln**.

Wissen = Fakten + Regeln.

Start eines Programms: Eingabe einer Behauptung.

Computer versucht, Behauptung mithilfe der Regeln und Fakten zu beweisen.

Faktum: wahre Aussage über Eigenschaften von oder Beziehungen zwischen Objekten.

Zwischen Objekten $a_1,...,a_n$ besteht die Beziehung e: $e(a_1,...,a_n)$.

Beispiel: „Z ist die gemischte Version der Folgen X und Y“: misch(X,Y,Z).

Faktum: $(\copyright Y)\ (misch([\ ],Y,Y))$.

Schlußregel: Allgemeine Form:

$$Prämisse \Rightarrow Konklusion.$$

Beispiel: sinnvolle Regel für die Formulierung des Mischens zweier Folgen

$$(\copyright A,B,C,D,E,F,G)\ ((A \leq C \wedge misch([A,B],[C,D],[E,F,G])) \Rightarrow misch([B],[C,D],[A,E,F,G]))$$

Variablenkonzept: Variable = Unbestimmte.

Operation: Unifikation = „Gleichmachen“ zweier Ausdrücke durch konsistente Ersetzung von Variablen durch Werte.

Beispiel: Gegeben Faktum: (©Y) (misch([ ],Y,Y)).

Frage: Gilt misch([ ],[1,3],[1,3]) ?

Antwort: Ja, Unifikation der Variablen Y durch die Folge [1,3].

Historische Entwicklung:

1965: J.A. Robinson: Resolutionsprinzip = Verfahren für die Ableitung von Aussagen aus Fakten und Regeln

1972: R.A. Kowalski: Idee, Logikkalküle als Programmiersprachen zu verwenden.

1973: A. Colmerauer: Sprache PROLOG.

Beispiele:

1) Mischen in einer fiktiven prädikativen Programmiersprache:

(©Y) (misch([ ],Y,Y)).
(©Y) (misch(Y,[ ],Y)).
(©A,B,X,Y,Z) (A≤B ∧ misch(X,[B|Y],Z) ⇒
misch([A|X],[B|Y],[A|Z])).
(©A,B,X,Y,Z) (B<A ∧ misch([A|X],Y,Z) ⇒
misch([A|X],[B|Y],[B|Z])).

2) Mischen in PROLOG:

```
misch([ ],Y,Y).
misch(Y,[ ],Y).
misch([A|X],[B|Y],[A|Z]) :– A≤B, misch(X,[B|Y],Z).
misch([A|X],[B|Y],[B|Z]) :– B<A, misch([A|X],Y,Z).
```

3 Programmierstile im Vergleich

Imperative Programmierung mit Pascal

Erfahrungen:

- dominiert den Informatikunterricht
- gute Lernerfolge auf lange Sicht
- besitzt Bezug zur Lebenswelt der Schüler
- ist in der Informatik im Zusammenhang mit Rechnermodellen und Effizienzanalysen unverzichtbar

Mängel:

* umfangreiche Syntax
* (zu) viele (zu) elementare Konzepte
* Mangel an Orthogonalität
* Motivationsprobleme zu Beginn
* Programme mit Wegwerf-Charakter
* Lernen „auf Vorrat“

Prädikative Programmierung mit PROLOG

Erfahrungen:

- viele interessante Vorschläge, PROLOG in den Anfangsunterricht einzubeziehen
- kein Motivationsproblem: Schüler können sehr früh mächtige Programme schreiben
- hohes Maß an Orthogonalität

Mängel: Alle Vorschläge setzen voraus,

* daß PROLOG eine große Nähe zur natürlichen Sprache besitzt,
* daß Anfänger leichter Probleme beschreiben als Vorschriften zu ihrer Lösung angeben können,

im Widerspruch zu empirischen Ergebnissen aus der Psychologie.

„**Nähe zur natürlichen Sprache**“: Alltagslogik=Prädikatenlogik.

1. **Problem**: Umsetzung umgangssprachlicher Aussagen mit temporalen, kausalen oder undifferenzierten logischen Operationen in prädikaten-logische Aussagen [Taylor/duBoulay87]:

   Sie war reich, und er heiratete sie <=> Er heiratete sie, und sie war reich.

   Paula ist eine gute Schülerin, weil sie hart arbeitet.

   Wenn es regnet, nehme ich einen Schirm

   => Wenn es nicht regnet, nehme ich keinen Schirm.

2. **Problem**: closed world assumption versus open world-Alltag:

   Mensch: weiß nicht

   Gibt es auf Tahiti Vulkane?

   PROLOG: nein

3. **Problem**: Deklarative <=> prozedurale Semantik.
   Anfänger benötigen Verständnis für virtuelle PROLOG-Maschine [Taylor/duBoulay87].

„**Problemlösen durch Problembeschreiben**“:

Idee: zweckfreies Wissen

Computer Lösung

Problem

Tatsächlich: Eingegebenes Wissen ist in hohem Maße zweckorientiert.

***Beispiele**:*

1) Modellierungsentscheidungen:

```
mag(otto,paula)
```

oder `otto_mag(paula)`

oder `es_stimmt_dass(otto,mag,paula)`

2) A, B und C stehen vor Gericht. A sagt aus, daß B lügt. B sagt aus, daß C lügt. C sagt aus, daß A und B lügen beide. Wer lügt, wer sagt die Wahrheit? Die PROLOG-Modellierung:

```
ist_Lügner(wahr,lügt).
ist_Lügner(lügt,wahr).
beide_lügen(wahr,lügt,lügt).
beide_lügen(lügt,wahr,lügt).
beide_lügen(lügt,lügt,wahr).
beide_lügen(lügt,wahr,wahr).
?- ist_Lügner(A,B),ist_Lügner(B,C),beide_lügen(C,A,B).
```

Beachte: Keinerlei logische Schlüsse !

Konsequenzen:

* Schüler sind nicht gut in Logik

  => Einführung in Prädikatenlogik nötig

* Schüler können Modellierungprobleme nicht intuitiv lösen

  => Vertieftes Verständnis von PROLOG und der virtuellen PROLOG-Maschine nötig

* Vorteile von PROLOG im Anfangsunterricht sehr zweifelhaft

Funktionale Programmierung

Erfahrungen:

- kaum Vorschläge und Ergebnisse mit modernen Sprachen
- positive Erfahrungen mit LOGO evtl. nicht übertragbar
- mathematische Notation für Anfänger evtl. zu abschreckend
- Psychologie: Rekursion schwierig für Anfänger

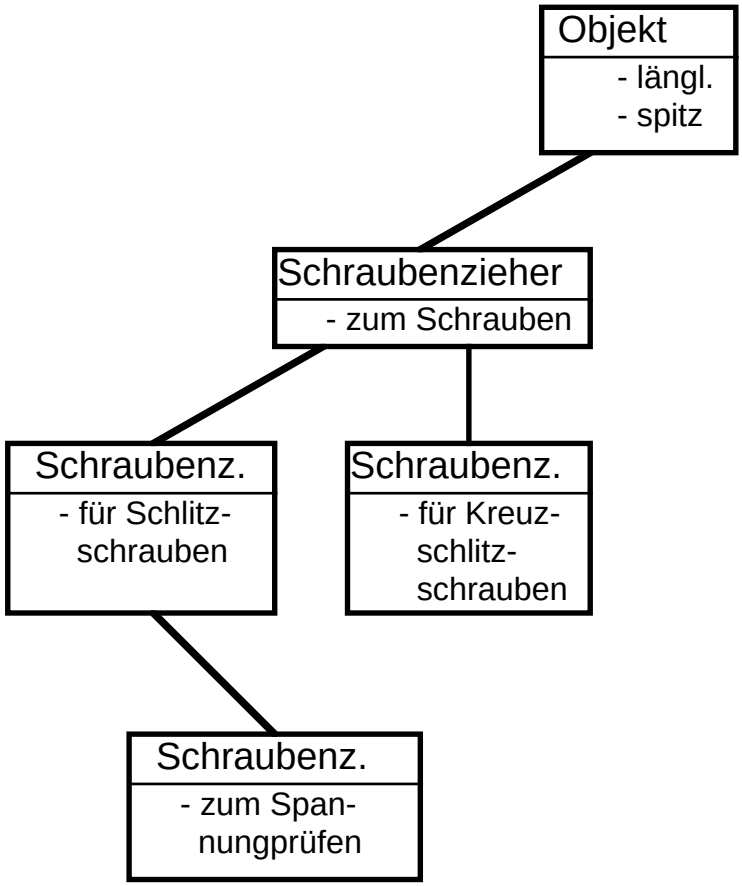
Objektorientierte Programmierung

Erfahrungen:

- erfüllt Forderungen nach einem zeitgemäßen Unterricht mit modernen Konzepten, wie Erweiterbarkeit, Vererbbarkeit, Kapselung etc.
- erfüllt Wünsche nach einer stärkeren Anwendungsorientierung durch Nutzung des Computers anstelle von seinem vertieften Ver ständnis
- erfüllt das Prinzip der Fortsetzbarkeit eines nach dem Spiralprinzip organisierten Unterrichts;
  Umstellung von speziellen Anfängersystemen wie LOGO oder Roboter NIKI auf eine „echte“ Sprache entfällt
- paßt zu den psychologischen Prozessen, die im menschlichen Ge hirn beim Denken, Erkennen und Problemlösen ablaufen,
- paßt damit insbesondere zu den kognitiven Voraussetzungen von Anfängern.

4 Kognitive Aspekte objektorientierter Programmierung

Objekte aus kognitiver und informatischer Sicht



Ergebnisse der Kognitionspsychologie
[z.B. Bruner et al. 1956, Piaget 1952, Anderson 1980]:

Menschen identifizieren Objekte
mehr durch die Aktionen, die mit ihnen möglich sind,
als durch ihr Aussehen wie Farbe oder Form.

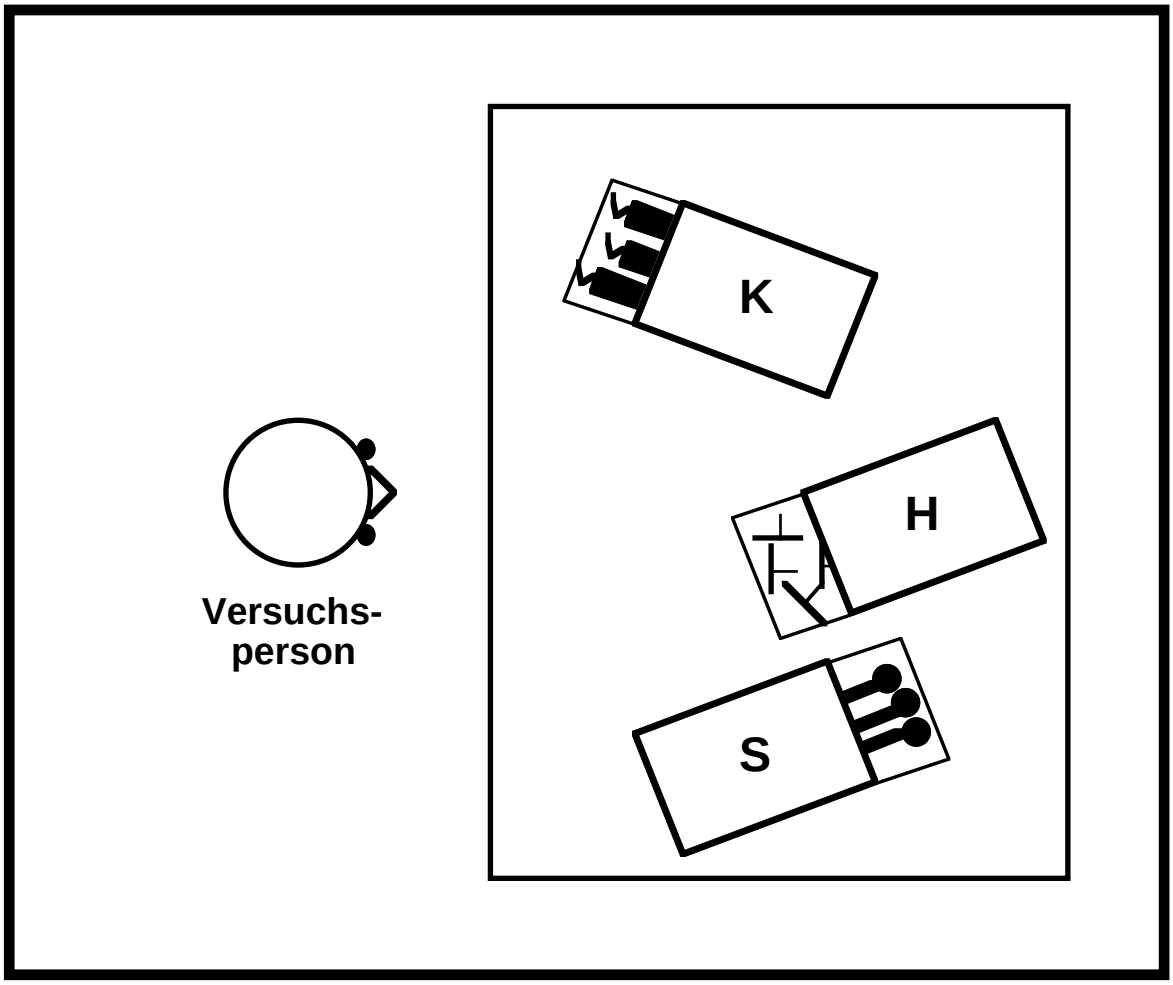
Standardmodell der Kognitionspsychologie: Kategorie/Schema

Analogie: Klasse <-> Kategorie/Schema

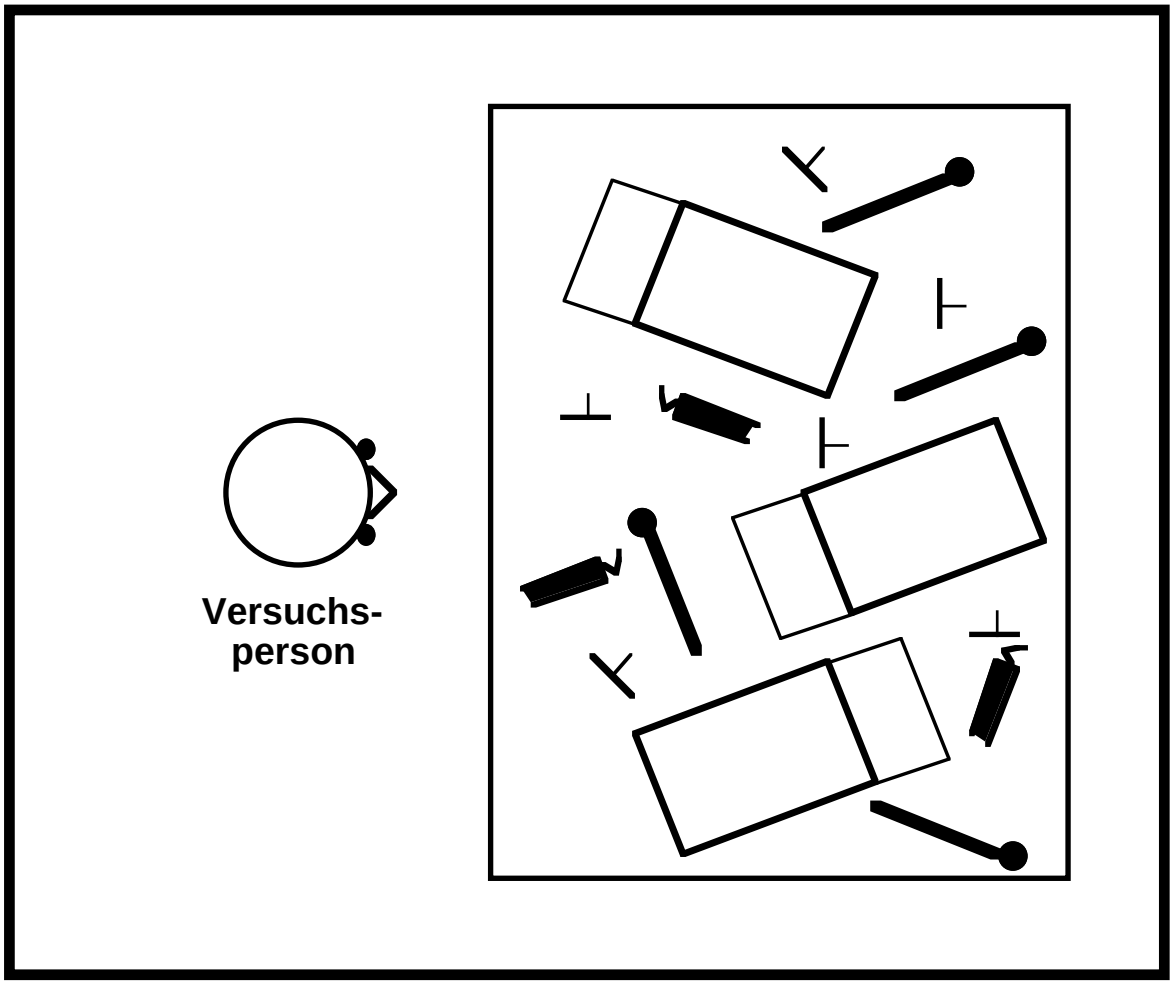
objektorientierte Sicht	Beispiel	psychologische Sicht
Klasse	Hund	Kategorie/Schema = große komplexe Ein heit, die große Teile menschl. Wissens und Verhaltens organisiert
Definition durch **Attribute**:		Definition durch **Attribute**:
- Variablen attribute	hat 4 Beine	- Wahrnehmungs-
- Methoden	kann bellen	- Funktionsattribute
- Vererbung		- relationale Attribute
— einfach	Hund ≠ Katze	* Kategorien können sich gegenseitig ausschließen
— mehrfach	Hunde und Katzen sind Haustiere	* Kategorien können sich überlappen

Beispiel: Das Kerzenproblem von K. Duncker

Situation 1



Situation 2



Beobachtung:

- **Versuchspersonen 1 nehmen die Schachteln als Behälter wahr; Die Funktion „Behälter“ beschränkt die nachfolgenden Denkabläufe (funktionale Gebundenheit)**

- **Versuchspersonen 2 nehmen die Schachteln ohne Bindung an irgendeine Funkton wahr.**

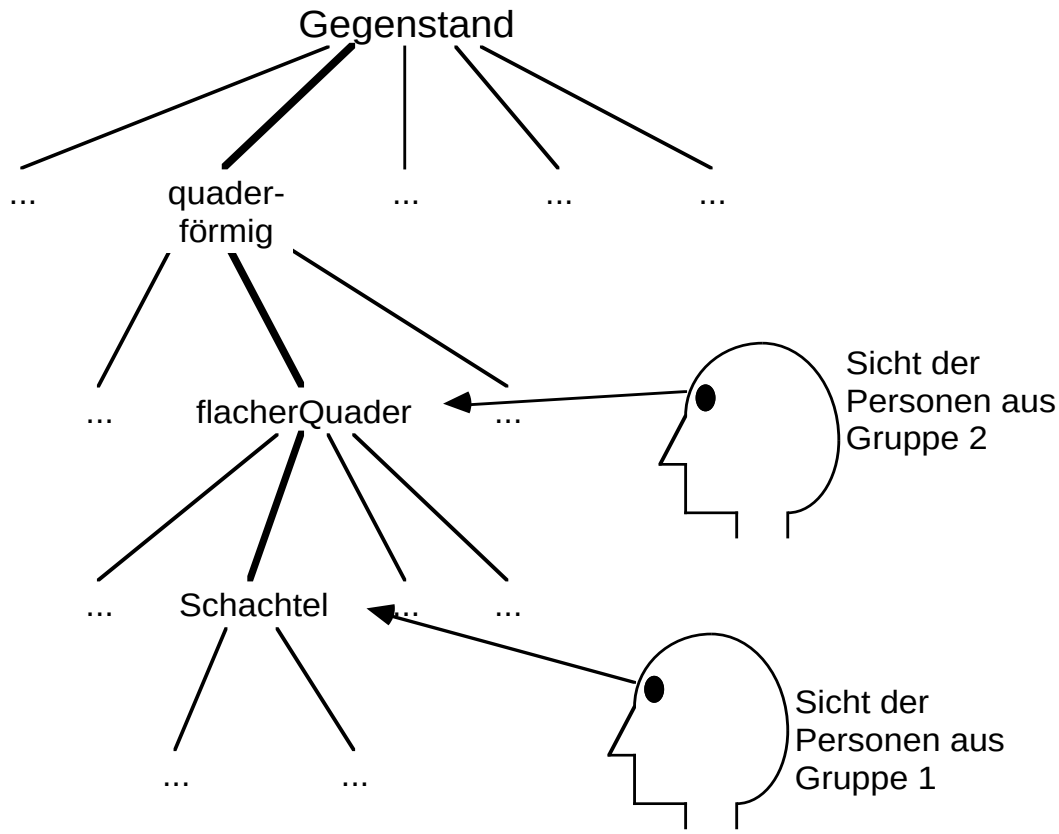
Eine objektorientierte Interpretation (OBERON-artig)

type **Gegenstand** = record
    hat eine räuml. Ausdehnung;
    kann man greifen;
    hat Farbe und Form;
    ...
end;

type **quaderförmig** = record (**Gegenstand**)
    hat Länge, Breite, Höhe;
    ...
end;

type **flacherQuader** = record (**quaderförmig**)
    Höhe≤5cm;
    kann man stapeln;
    kann man als Unterlage verwenden;
    ...
end;

type **Schachtel** = record (**flacherQuader**)
    kann man öffnen;
    kann man schließen;
    ist leer oder gefüllt mit ...;
    ...
end.


Gegenstand
...
quader-
förmig
...
...
...
...
flacherQuader
...
Sicht der
Personen aus
Gruppe 2
...
Schachtel
...
...
...
...
Sicht der
Personen aus
Gruppe 1

5 Methodische Hinweise für den Anfangsunterricht

Objektorientierte Programmierung

- ist aus Sicht der Informatik ein moderner und leistungsfähiger Ansatz,
- spiegelt fundamentale kognitive Prozesse im menschlichen Gehirn wider,
- ist folglich besonders geeignet für den einführenden Informatik-unterricht,

vorausgesetzt, daß

- man eine Programmierumgebung wählt, die die objektorientierte Vorgehensweise sichtbar macht, d.h.
  = eine künstliche Welt generiert,
  = viele, klar visualisierte Standardobjekte bietet,
  = eine Benutzungsschnittstelle besitzt, mit der man spielerisch explorativ Objekte manipulieren, analysieren, kombinieren, rekonfigurieren, evolutionär erweitern, neu entwickeln kann.

Systeme, die dieser Philosophie nahekommen, sind z.B.:

HyperCard für Anfängerniveau
(mit fester Klassenhierarchie, vereinfachtem Nachrichtenaustausch, HyperTalk)

Smalltalk-80 für Fortgeschrittene.