Programmierstile im Anfangsunterricht

Andreas Schwill
Fachbereich Mathematik/Informatik - Universität Paderborn
D-33095 Paderborn - Germany
email: schwill@uni-paderborn.de

Zusammenfassung: Der Informatikunterricht basiert zwar noch weitgehend auf dem imperativen Paradigma via Pascal, es gibt jedoch auch Vorschläge, den funktionalen, den objektorientierten und vor allem den prädikativen Programmierstil in den Unterricht einzubeziehen. Diese Arbeiten lassen jedoch meist offen, welcher Programmierstil für den Anfangsunterricht in Informatik am besten geeignet ist; darüber hinaus berücksichtigen sie nicht die psychologischen Voraussetzungen, die Lernende benötigen, um die entsprechenden Paradigmen zu begreifen. Wir werden uns in dieser Arbeit vor allem aus der Perspektive des Lernenden damit beschäftigen, unterschiedliche Programmierstile hinsichtlich ihrer kognitiven Voraussetzungen zu vergleichen. Schwerpunkt wird der objektorientierte Stil sein, von dem wir zeigen, daß er mit der natürlichen menschlichen Denkweise in Einklang steht und daher besonders geeignet erscheint, Informatik auf einem einführenden Niveau zu vermitteln.

1 Programmierstile im Informatikunterricht - eine Neuauflage des Sprachenstreits?

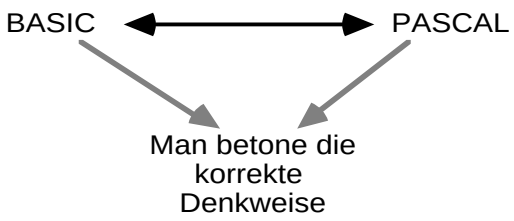
In den 70er Jahren (und teilweise auch heute noch) wurde heftig darüber gestritten, welche Programmiersprache für die Schule am geeignetsten sei. Die beiden Kontrahenten waren damals vor allem BASIC und PASCAL. Diese als Sprachenstreit bekanntgewordene Auseinandersetzung wurde dadurch aufgelöst, daß man die „richtige Denkweise“ in den Mittelpunkt des Unterrichts stellte; die Übertragung einer mit der richtigen Denkweise ermittelten Problemlösung in eine Programmiersprache sei dann lediglich noch eine maschinelle Tätigkeit, die die Vorstellungswelt der Schüler und das systematische Vorgehen beim Programmieren kaum noch negativ beeinflussen könne.

Diese auf den ersten Blick überzeugende Argumentation ignorierte jedoch eine Vielzahl von Überlegungen zum Zusammenhang zwischen Sprache und Welt (etwa bei Wittgenstein) sowie die Sapir-Whorf-These [W73], die die Existenz eines sprachlichen Relativitätsprinzips behauptet, welches anschaulich besagt, daß Menschen, die verschiedene Sprachen benutzen, äußerlich ähnliche Eindrücke unterschiedlich wahrnehmen und bewerten und damit auch zu unterschiedlichen Weltbildern gelangen. So ist die Programmiersprache, auch wenn sie als Gegenstand des Unterrichts zunehmend in den Hintergrund tritt, zentrales Werkzeug, in dem alle informatikrelevanten Sachverhalte formuliert werden, und Medium, mit dem der überwiegende Teil der Informatikinhalte im Unterricht transportiert wird. Sie prägt folglich in erheblichem Maße das informatische Denken und die Ausbildung fundamentaler Ideen.

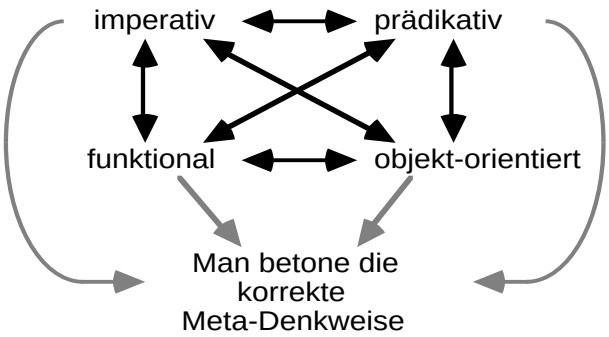
Die aktuelle Situation gleicht dem damaligen Sprachenstreit, nur hat sich das Streitobjekt auf eine Metaebene verlagert: Welches ist die für die Schule geeignete Denkweise (verkörpert durch einen der vier zugehörigen Programmierstile)? Vermutlich kann man den Streit auch hier schlichten, indem man die Verantwortung wie damals in eine Metaebene

(bezogen auf Sprachen ist das dann schon eine Metametaebene) verlagert und damit solange verdrängt, bis man wiederum mehrere Metadenkweisen voneinander isoliert hat und sich die Frage nach der korrekten Metadenkweise erneut stellt (Abb. 1).

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?

...

Abb. 1: Evolution des Sprachenstreits

Obgleich die Informatikausbildung in der Schule immer noch vom imperativen Paradigma beherrscht ist, gibt es eine Reihe von Ansätzen, auch den funktionalen, den objekt-orientierten und vor allem den prädikativen Stil in den Unterricht einzuführen. Die meisten Arbeiten argumentieren dabei implizit oder explizit „informatisch“ (z.B. [L93]): Die zuletzt genannten drei Paradigmen würden die Aktivitäten der Informatik (in Zukunft) so stark prägen, daß sie in den Unterricht einbezogen werden müßten, um den Schülern ein angemessenes Bild der aktuellen Entwicklungstendenzen der Informatik zu geben. Diese allgemein akzeptierte Aussage läßt jedoch offen, welcher der Stile für den Anfangsunterricht am geeignetsten erscheint, da sie nicht berücksichtigt, daß die Behandlung der einzelnen

Stile unterschiedliche intellektuelle Voraussetzungen und Fähigkeiten der Schüler erfordert sowie unterschiedliche curriculare Ansätze, die den Fortgang des Unterrichts mehr oder weniger positiv oder negativ beeinflussen.
Wir wollen in diesem Aufsatz obige Begründungen um eine psychologische Argumentation erweitern, die zu einer Favorisierung des objektorientierten Ansatzes für den Einstieg in die Informatik führen wird.

2 Programmierstile im Vergleich

In den folgenden Aussagen beziehen wir uns stellvertretend für die einzelnen Stile meist auf diejenigen Programmersprachen, die in der Schule am weitesten verbreitet sind.

Imperative Programmierung.
Die imperative Programmierung mit PASCAL dominiert zur Zeit noch den Unterricht. Auch wenn hiermit auf lange Sicht gute Erfolge erzielt wurden, stellen sich im Anfangsunterricht erhebliche Schwächen heraus. Obwohl auch als Ausbildungssprache konzipiert, besitzt PASCAL eine vergleichsweise umfangreiche Syntax und viele zu elementare Konzepte, die erst noch zu leistungsfähigen Strukturen zusammengebaut werden müssen, bevor eine angemessene Nutzung auf niedrigem Anfängerniveau möglich ist. Wichtigster Schwachpunkt in diesem Zusammenhang ist das Fehlen des Datentyps Liste; er kann nur (mit Zeigern) simuliert werden, ein Unterrichtsinhalt, der weit über das Anfangsniveau hinausgeht. Ferner behindert der Mangel an Orthogonalität einen sanften Einstieg in die Informatik: Es gibt eine Vielzahl von Vorschriften, Nebenbedingungen und Verboten, die gewisse Kombinationen von Elementarstrukturen und Konstruktoren aus nicht unmittelbar einsichtigen (tatsächlich mit der Implementierung zusammenhängenden) Gründen ausschließen.
Dies bringt Motivationsprobleme mit sich: Die Schüler müssen zunächst mit einer großen Menge von Konzepten vertraut werden („Lernen auf Vorrat“), bevor sie ein vernünftiges Programm mit Anwendungsbezug und ohne Wegwerf-Charakter entwickeln können. Dies hat E. Lehmann [L94] erkannt und einen interessanten Ausweg vorgeschlagen: Er entwirft vorab leistungsfähige Module, die den Schülern schon zu Beginn gewisse nützliche Grundfunktionen zur Verfügung stellen und die praktische Einsatzfähigkeit von PASCAL auf Einstiegsniveau verbessert. Im Ergebnis führt dieser Ansatz in eine objektorientierte Richtung, wie sie auch hier vorgeschlagen wird.
Trotz dieser Mängel gibt es gewichtige pädagogische und informatische Gründe für die Beibehaltung der imperativen Programmierung im Unterricht, so daß auf den imperativen Stil, zumindest im Schulfach Informatik, bis auf weiteres wohl nicht verzichtet werden kann. Denn zum einen besitzt der imperative Stil eine besonderen Bezug zur Lebenswelt und zum Alltagsdenken der Schüler, und er unterstützt, insbesondere in seiner Erweiterung um objektorientierte Konzepte, die elementaren kognitiven Prozesse des Denkens, Erkennens und Problemlösens. Zum anderen werden zur Vermittlung gewisser Informatik-

inhalte, bei denen Bezüge zu konkreten Rechnermodellen bestehen, stets imperative Darstellungen benötigt. Hierzu gehören z.B. die maschinennahe Programmierung in Assembler, die Programmierung von Turing- oder Registermaschinen oder die Effizienzanalyse von Algorithmen, die auf einem verallgemeinerten Registermaschinenmodell (der sog. RAM) basiert, das imperativ programmiert wird.

Prädikative Programmierung.

Neben der mittlerweile auch in Lehrplänen (z.B. von Rheinland-Pfalz und Niedersachsen) festgeschriebenen Verpflichtung, den prädikativen Stil im Informatikunterricht zu behandeln, gibt es auch eine Reihe interessanter Vorschläge, PROLOG in den Anfangsunterricht einzubeziehen [B93,G89,K84,L92,L93]. Motivationsprobleme verbunden mit dem Lernen auf Vorrat sind mit PROLOG nicht zu erwarten, denn die sehr kleine Syntax mit nur wenigen, aber sehr leistungsfähigen Konzepten und ein hohes Maß an Orthogonalität beschränken den Lernprozeß auf das Wesentliche. Folglich können Schüler prinzipiell bereits frühzeitig sehr mächtige Programme schreiben und werden nicht mit Spielbeispielen demotiviert.

Alle bekannten Vorschläge zur Verwendung von PROLOG im Anfangsunterricht basieren jedoch implizit oder explizit auf zwei Thesen:

These 1: PROLOG besitzt eine große Nähe zur natürlichen Sprache, so daß umgangssprachliche Darstellungen von Problemen und deren Lösungen leicht in die PROLOG-Notation übertragen werden können.

These 2: Die Aufgabe, Probleme statt Vorschriften zu ihrer Lösung zu beschreiben, ist für Anfänger leichter zugänglich.

Beide Thesen besitzen zwei Angriffspunkte: Sie beruhen auf nicht-empirischen Einzelerfolgen, und - noch wichtiger - sie widersprechen empirischen Ergebnissen aus der Psychologie.

zu These 1: Die Widerlegung dieser These kann an drei Punkten ansetzen:

- Alltagslogik=Prädikatenlogik.

  Die Nähe, die PROLOG zur natürlichen Sprache zugeschrieben wird, beruht auf einem Gerüst von Folgerungen, das sich grob so darstellen läßt:

  > Logische Operationen sind natürliche Elemente menschlichen Denkens, denn logische Argumentationen finden sich auch in der Lebenswelt. Daher besitzen alle Schüler schon ein gewisses Grundverständnis von Logik, das sich zur Vermittlung von PROLOG im Anfangsunterricht nutzen läßt. [K79, L93]

  Zugleich wird mit dieser Argumentationskette implizit Alltagslogik mit Prädikatenlogik identifiziert, unzulässigerweise, wie man sich an folgenden Beispielen klar macht:

  „Sie war reich, und er heiratete sie“ versus „Er heiratete sie, und sie war reich“.

  „Paula ist eine gute Schülerin, weil sie hart arbeitet“.

  „Wenn es regnet, nehme ich einen Schirm“.

  Konsequenz: „Wenn es nicht regnet, nehme ich keinen Schirm“.

Tatsächlich haben Anfänger häufig große Probleme, umgangssprachliche Aussagen in prädikatenlogische umzusetzen, vor allem dann, wenn sie wie oben temporale oder kausale Elemente enthalten bzw. undifferenziert logische Operationen verwenden [TB87].

- closed world-assumption.
  Weitere Schwierigkeiten von Anfängern sind bei der closed-world-assumption zu erwarten, die den natürlichen Schlußweisen im „open-world"-Alltag widerspricht: Auf die Frage

  „Gibt es auf Tahiti Vulkane?"

  wird man im allgemeinen die Antwort „weiß ich nicht" erwarten; PROLOG antwortet stattdessen mit „nein".
- Deklarative versus prozedurale Semantik.
  Natürliche Sprachen können Wissen rein deklarativ formulieren. PROLOG täuscht stattdessen eine deklarative Darstellung nur vor, interpretiert das Wissen aber im Verborgenen prozedural, ohne daß aus der textuellen Darstellung eines PROLOG-Programms irgendwelche Hinweise auf diese Ausführung abgeleitet werden können. Dieser Zwiespalt, der nur durch ein Verständnis der virtuellen PROLOG-Maschine abgebaut werden kann, bereitet Anfängern immense Schwierigkeiten [TB87].

zu These 2: Mit der Maxime „Problemlösen durch Problembeschreiben" ist folgende naive Vorstellung der PROLOG-Programmierung verbunden: Der Computer erhält ein Bündel von zweckfreiem (d.h. problemunabhängigem) Wissen zur Verfügung gestellt und unabhängig davon eine Problemstellung, zu deren Lösung er das Wissen nutzen kann. Diese Vorstellung wird auch in Lehrbüchern häufig propagiert (z.B. [H86, Kap. 1.2]). Tatsächlich ist die Darstellung des Wissens aber in hohem Maße zweckgebunden und problemabhängig.

Beispiel: Das Verhältnis zweier Personen kann man zumindest auf drei Arten beschreiben. Welche Darstellung für welchen Zweck geeignet erscheint, muß vorab entschieden werden:

```
mag(otto,paula).
otto_mag(paula).
es_stimmt_dass(otto,mag,paula).
```

Das folgende Beispiel - ob repräsentativ oder nicht, sei hier nicht entschieden - illustriert abschließend die Schwierigkeiten (die auch der Autor beim Einstieg in PROLOG hatte), selbst Sachverhalte mit eindeutig logischem Hintergrund in PROLOG zu modellieren, und damit die Diskrepanz zwischen natürlichsprachlichen und PROLOG-artigen Problemdarstellungen.

Beispiel: 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 beide lügen. 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).
```

Wer hätte erwartet, daß trotz der versteckten Implikationen in den Aussagen von A, B und C in der Modellierung keine Implikation mittels :- auftaucht?

Folgerung: Da Alltagslogik und Prädikatenlogik sowie natürlichsprachliche und PROLOG-Modellierung nur wenig gemeinsam haben, darf zunächst nicht davon ausgegangen werden, daß Schüler, zumal auf einführendem Niveau und ohne jedes Vorwissen, gut in Logik sind bzw. die Modellierungsprobleme intuitiv lösen können. Folglich ist für ein vertieftes Verständnis von PROLOG zumindest eine vorhergehende Einführung in die Prädikatenlogik erforderlich, die bisher von der Mathematik nicht erbracht wird, einen möglichen Vorteil von PROLOG beim Einstieg von Anfängern in die Informatik aber wieder zunichte macht, wenn die Informatik diese Einführung leisten muß.

Funktionale Programmierung.

Mit modernen funktionalen Programmiersprachen wie ML, HOPE, MIRANDA liegen bisher kaum Erfahrungen im Unterricht vor. Ein Plädoyer, diese Sprachen im Unterricht stärker zu erproben, habe ich bereits in [S93] formuliert. Lediglich für LISP/SCHEME (z.B. [S94]) und natürlich LOGO wurden einige Unterrichtsvorschläge ausgearbeitet.

Ob die mit LOGO gewonnenen positiven Erfahrungen auf die modernen funktionalen Sprachen übertragen werden können, kann hier nicht geklärt werden. Gesichert ist aus psychologischen Untersuchungen jedenfalls, daß Anfänger i.a. erhebliche Schwierigkeiten haben, Rekursionen zu formulieren und nachzuvollziehen [GV88,HR89,K82,W89]. Möglicherweise wirken funktionale Sprachen durch ihre mathematische Notation auch abschreckend.

Vorteilhaft im Vergleich zu prädikativen Sprachen ist jedoch zu bewerten, daß funktionale Programme zwar deklarativ formuliert werden, dabei aber immer noch erkennen lassen, wie sie ausgeführt werden.

Objektorientierte Programmierung.

Wir favorisieren für die Einführung in die Informatik den objektorientierten Ansatz vor allem aus drei Gründen: Erstens erfüllt dieses Paradigma unbestrittenermaßen informatisch orientierte Forderungen nach einem zeitgemäßen Unterricht mit mächtigen Konzepten, wie Erweiterbarkeit, Anpaßbarkeit, Rekonfiguration, Vererbbarkeit, Kapselung, evolutionäre Softwareentwicklung sowie Wünsche nach einer stärkeren Anwendungsorientierung

durch Betonung der Nutzung des Computers anstelle von einem vertieften Verständnis seiner Funktionsweise. Zweitens ist dieser Ansatz im Sinne des didaktischen Prinzips der Fortsetzbarkeit, ein zentrales Merkmal eines nach dem Spiralprinzip organisierten Curriculums, auf höherem Niveau beliebig ausbaufähig. Eine Umstellung von einer speziellen Anfangssprache auf eine „echte“ Programmiersprache mit den damit verbundenen Reibungsverlusten, wie sie beim Einstieg mit Roboter NIKI oder LOGO notwendig werden, ist nicht erforderlich. Drittens - und dies erscheint aus pädagogischer Sicht der wichtigste Pluspunkt - ordnet sich der objektorientierte Stil in besonderer Weise harmonisch den elementaren kognitiven Prozessen unter, die beim Denken, Erkennen und Problemlösen im menschlichen Gehirn ablaufen. Diesen Aspekt werden wir im weiteren Verlauf ausführlicher erläutern.

3 Kognitive Aspekte objektorientierter Programmierung

Bei Erwachsenen ebenso wie bei kleinen Kindern kann man das typisch menschliche Verhalten beobachten, alle Dinge zunächst danach zu beurteilen, was man mit ihnen machen kann. So ist z.B. ein Schraubenzieher ein Werkzeug, mit dem man in erster Linie Schrauben lösen und festziehen kann; als Vertreter einer Klasse mit allgemeineren Eigenschaften wie „länglich“, „spitz“ kann man ihn zur Not aber auch als Brechstange, Meißel, Bohrer oder Stichwaffe verwenden. Umgekehrt unterscheidet man diverse Teilklassen mit spezielleren Merkmalen: Schraubenzieher für Schlitzschrauben, für Kreuzschlitzschrauben, mit Einrichtung zur Spannungsprüfung usw. Gerade Kleinkinder besitzen in hohem Maße die Fähigkeit zwischen den Ebenen dieser Hierarchie hin und her zu wechseln und dabei Eigenschaften und Operationen von Objekten zu entdecken, die sie für einen völlig anderen als ihren vorbestimmten Zweck geeignet erscheinen lassen. An diesem Beispiel konkretisiert sich bereits die klassische objektorientierte Denkweise.

Ausgehend von einer Reihe von Untersuchungen zur Problemlösefähigkeit und den verfügbaren Problemlösemethoden von Menschen in der ersten Hälfte dieses Jahrhunderts [D66,S13,W57] gibt es eine Vielzahl von Untersuchungen bei Kindern und Erwachsenen [P52,BGA56,A77,D90] über die Wahrnehmung von Objekten und die Repräsentation von Wissen sowie darüber, wie dieses Wissen menschliche Entscheidungen und Handlungen leitet. Alle diese Untersuchungen belegen, daß Menschen Objekte überwiegend anhand der Handlungen identifizieren, die mit ihnen möglich sind, als anhand äußerlicher Eigenschaften wie Farbe oder Form. Wichtige Bausteine dieser psychologischen Theorie sind die sog. *Schemata* [A80] oder *Kategorien* [GHS79], das sind große komplexe Einheiten, die wesentliche Teile menschlichen Wissens und Verhaltens organisieren. Aus Sicht der objektorientierten Programmierung handelt es sich bei den Schemata oder Kategorien um *Klassen*. Diese verblüffende Analogie beider Begriffe zeigt Tab. 1.

objektorientierte Sicht	**Beispiel**	**psychologische Sicht**
Klasse	Hund	Kategorie/Schema
Definition durch Attribute:		Definition durch Attribute:
- Variablen	hat vier Beine	- Wahrnehmungsattribute
- Methoden	kann bellen	- Funktionalattribute
- Vererbung		-- relationale Attribute
-- einfache	Ein Hund ist keine Katze	-- Kategorien können sich gegenseitig ausschließen
-- mehrfache	Hunde und Katzen sind Haustiere	-- Kategorien können sich überlappen

Tab. 1: Analogie zwischen Klassen und Kategorien/Schemata

Beispiel: Duncker's Kerzenproblem und seine objektorientierte Interpretation.
Das folgende Experiment [D66] verdeutlicht diese Überlegungen: Mehrere Versuchspersonen, die sich jeweils allein in einem Versuchsraum befanden, erhielten die Aufgabe, an einer Wand in Augenhöhe nebeneinander drei Kerzen zu befestigen und anzuzünden. Hierfür standen den Probanden eine Reihe von willkürlich auf dem Tisch verteilten Gegenständen zur Verfügung. Unter den meist nutzlosen Dingen befanden sich auch einige für die Lösung brauchbare: Heftzwecken, Streichhölzer und drei kleine in Farbe und Größe etwas unterschiedliche Pappschachteln von der Form einer Streichholzschachtel.
Lösung der Aufgabe: Mit je einer Heftzwecke werden zunächst die Pappschachteln an der Wand befestigt; sie dienen den Kerzen als Standflächen. Anschließend werden die Kerzen angezündet und mit etwas Wachs auf den Schachteln festgeklebt.
Die Versuchspersonen mußten diese Aufgabe in zwei leicht unterschiedlichen Ausgangssituationen lösen: Bei den Personen der ersten Gruppe waren die drei Pappschachteln mit Versuchsmaterialien gefüllt, die erste mit den Kerzen, die zweite mit Heftzwecken und die dritte mit Streichhölzern. Bei der zweiten Gruppe waren die Schachteln leer; Kerzen, Heftzwecken und Streichhölzer lagen hier auf dem Tisch verstreut.
Erstaunlicherweise wurde die Aufgabe von der zweiten Gruppe signifikant häufiger und schneller gelöst als von der ersten. Duncker erklärt dieses Ergebnis wie folgt: Die erste Gruppe nimmt die Schachteln als Behälter für Kerzen, Heftzwecken und Streichhölzer wahr. Diese Funktion „Behälter" ist anschließend so eng mit den Schachteln verknüpft, daß die Probanden ihr Denken kaum noch davon lösen können und unfähig sind, die Schachteln zu einem völlig anderen Zweck zu nutzen, nämlich als Standfläche für die Kerzen. Duncker nennt dieses Phänomen *funktionale Gebundenheit*. Die zweite Gruppe nimmt die Schachteln ohne die Bindung an eine spezielle Funktion wahr. Die Versuchspersonen können sie daher völlig frei auch zu scheinbar ungewöhnlichen Zwecken (als Standfläche) einsetzen.

Was bedeuten diese Beobachtungen aus Sicht der Programmierung? Die Versuchspersonen denken offenbar objektorientiert: Gegenstände werden zuerst unter dem Gesichtspunkt in Klassen eingeteilt, welche Operationen mit ihnen möglich sind. Eine Schachtel gehört damit für die erste Gruppe zur Klasse der Objekte, auf denen Operationen wie „öffnen“ und „schließen“ erlaubt sind und die einen Zustand wie „leer“ oder „gefüllt“ besitzen. Diese Operationen bestimmen fortan das Denken. Dabei übersieht die Gruppe im Gegensatz zur zweiten, daß Schachteln auch als Objekte einer Oberklasse mit allgemeineren Eigenschaften aufgefaßt werden können, etwa als Objekte der Klasse quaderförmiger, flacher Gegenstände mit Operationen wie „als Unterlage verwenden“, „stapeln“ usw. Abb. 2 zeigt eine graphische Darstellung der Klassenhierarchie der Schachteln so, wie sie von den Versuchspersonen möglicherweise wahrgenommen wurde. Funktionale Gebundenheit ist aus objektorientierter Sicht also die mangelnde Fähigkeit, zwischen unterschiedlichen Verfeinerungen einer Klassenhierarchie gedanklich hin und her zu wechseln.

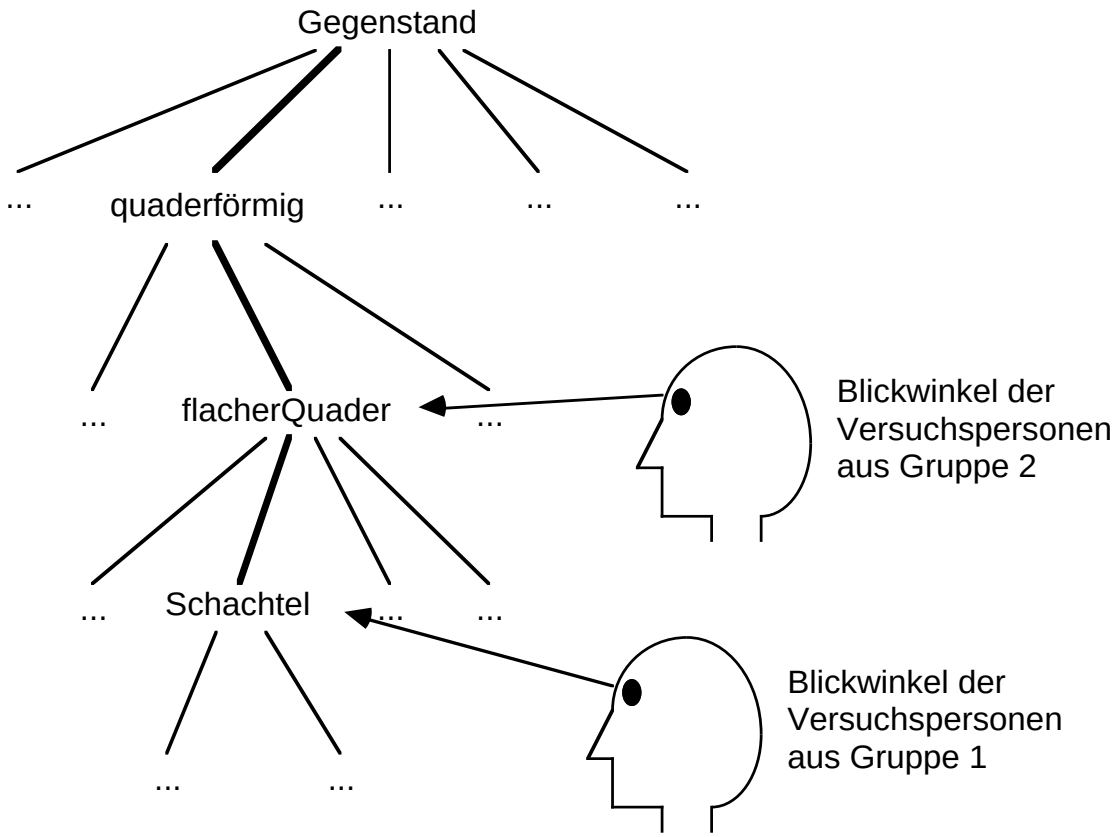


Abb. 2: Gedankliche Wahrnehmung der Klassenhierarchie durch die Probanden

In diesem Versuch von Duncker wurden das Denken und die Problemlösefähigkeit durch die objektorientierte Vorgehensweise mehr oder weniger behindert. Dies spricht nur auf

den ersten Blick gegen die objektorientierte Denkweise. Vielmehr scheint diese Denkweise, also das Kombinieren der Eigenschaften und Operationen, die lokal mit den Objekten verbunden sind, im Mittel schnell zu akzeptablen Lösungen zu führen. Sonst hätte sich die Denkweise im Laufe der menschlichen Evolution vermutlich auch nicht durchgesetzt. Denkfehler wie in obigem Versuch gilt es, durch Problemlösetraining zu vermeiden.

4 Schlußbemerkungen und methodische Hinweise

In diesem Aufsatz haben wir unterschiedliche Programmierstile auf ihre Tauglichkeit für den einführenden Unterricht in Informatik untersucht.
Der häufig favorisierte prädikative Stil in Form von PROLOG scheint sich hierfür überraschenderweise weniger gut zu eignen, weil er hohe Anforderungen an das logische Denken und die Fähigkeit stellt, reale Sachverhalte durch prädikatenlogische Formeln zu modellieren, die von Anfängern nicht zu erwarten sind. Zumindest besteht hier eine Diskrepanz zwischen kognitionspsychologischen Erkenntnissen und Resultaten aus einzelnen Unterrichtsversuchen, die noch aufzuklären ist.
Wir haben versucht zu begründen, daß von den bisher erprobten Programmierstilen die objektorientierte Programmierung der geeignetste Ansatz ist, um Informatik im Anfangsunterricht zu vermitteln, weil sie fundamentale kognitive Prozesse widerspiegelt und damit der natürlichen Denkweise am nächsten kommt und .
Auch diese Überlegungen bedürfen noch einer Absicherung durch Unterrichtsversuche. Ferner ist ein geeignetes curriculares Konzept zur Vermittlung von Informatik auf der Basis objektorientierter Programmierung zu erarbeiten.

In jedem Fall muß davor gewarnt werden, die Ausbildung nun einfach mit einer objektorientierten Programmiersprache zu beginnen, denn die prognostizierten Lernerfolge hängen nicht allein von der Sprache sondern vor allem auch von der Programmierumgebung ab, die die objektorientierte Denkweise sichtbar machen muß. Das heißt, sie muß eine künstliche Welt generieren, in der möglichst viele, klar visualisierte Objekte zur Verfügung stehen, ferner muß sie eine Benutzungsschnittstelle bieten, auf der die Schüler interaktiv und spielerisch explorativ Objekte auf ihre Funktionen analysieren und sie mit zunehmendem Niveau manipulieren, kombinieren, rekonfigurieren, evolutionär erweitern und schließlich neu entwickeln können.
Beispiele für Systeme, die diesen Anforderungen nahe kommen, bei denen also Sprache und Entwicklungsumgebung die objektorientierte Denkweise widerspiegeln, sind auf einem einführenden Niveau u.a. das System HyperCard mit seiner festen Klassenhierarchie, vereinfachtem Nachrichtenaustausch und der Sprache HyperTalk oder auf einem fortgeschrittenen Niveau das Smalltalk-80-System mit der Sprache Smalltalk-80.

Literatur

[A80] Anderson, J.R.: Cognitive psychology and its implications. Freeman 1980

[A77] Anglin, J.M.: Word, object, and conceptual development. New York 1977

[B93] Baumann, R.: Prädikative Denk- und Programmiermethoden im Informatikunterricht. LOGIN 13,3 (1993) 55-57 und 13,4 (1993) 56-60 und 13,5 (1993) 52-58

[BGA56] Bruner, J.S.; Goodnow, J.J.; Austin, G.A.: A study of thinking. Wiley 1956

[D66] Duncker, K.: Zur Psychologie des produktiven Denkens. Springer 1966

[D90] Dux, G.: Die Logik der Weltbilder. Suhrkamp 1990

[G89] Gasper, F.: Nichtprozedurale Sprachen im Informatikunterricht der Oberstufe. In: Informatik und Schule (F. Stetter, W. Brauer, eds.) 1989

[GHS79] Glass, A.L.; Holyoak, K.J.; Santa, J.L.: Cognition. Addison-Wesley 1979

[GV88] Göbel, R.; Vorberg, D.: Rekursionsschemata als Problemlösepläne. Tech. Report (1988) FB Psychologie, Uni Marburg

[H86] Hanus, M.: Problemlösen mit Prolog. Teubner 1986

[HR89] Haussmann, K.; Reiss, M.: Strategien bei Problemen mit rekursiver Lösung. J. für Mathematik-Didaktik 10 (1989) 39-61

[K79] Kowalski, R.: Logic for problem solving. North-Holland 1979

[K84] Kowalski, R.: Logic as a computer language for children. In: New horizons in educational computing (M. Yazdani, ed.) 1984

[K82] Kruse, R.L.: On teaching recursion. SIGCSE 14,1 (1982) 92-96

[L94] Lehmann, E.: Programmieren in Turbo-PASCAL mit Bausteinen. Dümmler 1994

[L92] Lehmann, G.: Ziele im Informatikunterricht. Beispiele für den Einsatz und Stellenwert von PROLOG. LOGIN 12,1 (1992) 26-30

[L93] Lehmann, G.: Sprache in der informatischen Bildung aus didaktisch-methodischer Sicht. Mathematik in der Schule 31 (1993) 628-636

[P52] Piaget, J.: The origins of intelligence in children. Intern. University Press 1952

[S93] Schwill, A.: Funktionale Programmierung mit CAML. LOGIN 13,4 (1993) 20-30

[S94] Seiffert, M.: Verschlüsselungsmethoden. LOGIN 14,2 (1994) 25-31 und 14,3 (1994) 33-40

[S13] Selz, O.: Über die Gesetze des geordneten Denkverlaufs: Eine experimentelle Untersuchung. Spemann 1913-1922

[TB87] Taylor, J., duBoulay, B.: Studying novice programmers: Why they may find learning PROLOG hard. in: Computers, Cognition and Development (J. Rutkowska, C. Crook, eds.) (1987) 99-114

[W57] Wertheimer, M.: Produktives Denken. Kramer Verlag 1957/1964

[W73] Whorf, B.: Linguistics as an exact science. In: Language, thought, and reality (B.L. Whorf, J.B. Carroll, ed.) Cambridge 1973

[W89] Wiedenbeck, S.: Learning iteration and recursion from examples. Intern. J. Man-Machine Stud. 30 (1989) 1-22