

Welche Programmiersprache unterstützt meine Konzepte für den Informatikunterricht?

Ludger Humbert

22. Juli 2002

begonnen: 14. September 2001

letzte Änderungen: 22. Juli 2002

Zusammenfassung

Einer der Bereiche, über die Informatiklehrerinnen¹, Schulpraktikerinnen, Informatikerinnen aber auch Schülerinnen und Studierende sich intensiv streiten (können und wollen), ist die immerwährende Frage nach „der besten ... richtigen ... lernförderlichsten ...“ Programmiersprache. Ähnlich anderen Glaubensfragen wird nicht über Konzepte reflektiert, sondern i. d. R. beispielhaft über besonders gelungene/misslungene Details von konkreten Implementierungen reflektiert. Benutzungsumgebungen und Entwicklungswerkzeuge scheinen wichtiger als Konzepte.

Da jede Programmiersprache (von Menschen) für Menschen entwickelt wurde und nicht für Computer, wie der zu häufige laienhafte Gebrauch des Wortes „Computersprache(n)“ nahelegt, sei an dieser Stelle daraufhingewiesen, dass jede Programmiersprache es Menschen ermöglichen sollte, ihre Problemlösungen zu kommunizieren. Sobald aber eine Programmiersprache als Konstrukt kommuniziert werden muss (z. B. in Lehr-Lernprozessen) stellt sich die Frage der Vermittlung. Hier wurde für schulbezogene Vermittlungsprozesse(zu) häufig die konkrete Programmiersprache in den Vordergrund gestellt, bis hin zu reinen Sprachkursen. Diese Fehlentwicklung wird zunehmend kritisiert. Aktuell schwingt das Pendel zurück zu den vorgängigen Arbeiten und zur konzeptionellen Ebene.

Die Auseinandersetzung begann bereits *vor* der Einführung der Informatik in der Schule und gipfelte seinerzeit im deutschsprachigen Raum im Arbeitskreis Computer im Unterricht (ACU) in der Auseinandersetzung Pascal versus BASIC. Der ACU gab anschliessend die Empfehlung heraus, eine eingedeutschte Version von Pascal, nämlich Pascal-E zu benutzen². Andere, eher pragmatisch orientierte Ansätze bestanden darin, eine Zwischenebene zur Darstellung von Algorithmen und Datenstrukturen zu wählen, die fast 1:1 in Pascal übersetzt werden kann, aber als „sprachunabhängig“ bezeichnet wurde [Moll und Buhse 1984].

Die Diskussionen bezogen sich auf das imperative Sprachparadigma. Sobald über das Wissensbasierte Paradigma reflektiert wurde [und wird], fällt den meisten Diskutanten nur noch Prolog ein.

In einem wichtigen Bereich des Informatikunterrichts der Sekundarstufe I – in der Differenzierung (Wahlpflichtunterricht) im 9. und 10. Jahrgang – wird an sehr vielen Schulen mit einer funktionalen Programmiersprache gearbeitet. Allerdings fällt dieses offenbar weder den Lehrerinnen noch den Schülerinnen auf, da ausschliesslich eine Teilmenge (die Turtle-Grafik) dieser Sprache – es handelt sich um LOGO – Verwendung findet, bei dem die Stärke des Paradigmas der Sprache nicht thematisiert wird. Es sei die Frage gestattet: Gibt es für diesen Anwendungsfall/Einsatzzweck keine andere Sprache? Gibt es andere funktionale Sprachen? Wer von den Unterrichtenden kennt sie?

Und dann gibt es da immer noch den Kollegen, der unbedingt einen Compiler für HTML anschaffen möchte; ... (Hinweis auf LOG OUT).

Diese Diskussion möchte ich mit diesen Beitrag bereichern. Dabei wird ein Weg gewählt, bei dem die Hoffnung besteht, dass viele Leserinnen die vorgetragenen Thesen für ihren Unterricht fruchtbar werden lassen. Welche Hilfsmittel auch immer im Informatikunterricht eingesetzt werden (und dazu gehören insbesondere Programmiersprachen), letztlich soll durch diese das Lernen unterstützt werden.

Lassen Sie mich Kritik wissen: <mailto:Ludger.Humbert@cs.uni-dortmund.de>

¹Im Text wird - abgesehen von Zitaten - durchgängig das *generische Femininum* verwendet. Männer mögen sich dadurch nicht ausgeschlossen fühlen.

²Dies wurde auf der INFOS 2001 in Paderborn von Klaus BRUNNSTEIN [Brunnstein 2001], der seinerzeit Mitglied im ACU war, mitgeteilt und läßt sich auch den seinerzeit veröffentlichten Dokumenten entnehmen. BRUNNSTEIN gab darüberhinaus an, dass er seinerzeit einen Compiler für diese Version von Pascal entwickelt hat.

Inhaltsverzeichnis

| | | |
|----------|---|----------|
| 1 | Strukturierte Darstellung | 2 |
| 1.1 | Aufgabenklassen und Programmiersprachen | 2 |
| 1.2 | Vergleichsuntersuchungen | 3 |
| 1.2.1 | Geschichte der Programmiersprachen | 3 |
| 1.3 | Modellierung | 3 |
| 1.3.1 | Modelle in der Informatik | 3 |
| 1.3.2 | Zur objektorientierten Modellierung | 3 |
| 1.4 | Paradigmen | 4 |
| 1.5 | Mehrere Paradigmen unter ein Sprachendach? | 4 |
| 1.6 | Wie weiter? | 5 |
| 1.7 | War da nicht noch? | 6 |
| A | Programmierersprachen „neben der Spur“ | 8 |
| A.1 | FORTH | 8 |
| A.2 | APL | 8 |
| B | Der besondere Stoff – Software – und wie er beschrieben wird | 9 |

1 Strukturierte Darstellung

Der Streit um die „richtige Sprache“ für den Einsatz im Informatikunterricht macht deutlich:

1. Tatsächlich wird offenbar erheblich mehr Wert (und damit Unterrichtszeit) auf Vermittlung syntaktischer und semantischer Strukturen konkreter Sprachen verwendet, als ihrem Stellenwert im Informatikunterricht zukommen sollte?
2. Die (fehlende) *Eleganz* im konkreten Ausdruck in einer konkreten Sprache ist nicht zuletzt ein ästhetisches Element, über das sich bekanntermassen trefflich streiten läßt.
3. Der Glaube an den Satz von Wittgenstein: „Die Grenzen meiner Sprache sind die Grenzen meiner Welt“ scheint vor allem für die Auseinandersetzung über Programmiersprachen einen entscheidender Begründungszusammenhang zu liefern.

1.1 Aufgabenklassen und Programmiersprachen

Tabelle 1: Auffassungen und ihre Ausprägung in Sprachklassen, nach [?, S. 5]

| Auffassung | Sprachklasse |
|--|---|
| Auswertung von Ausdrücken (einer formalen Sprache) | funktionale und applikative Sprachen ³ |
| Beantwortung von Anfragen (an ein Informationssystem) | relationale und logische Sprachen ⁴ |
| Manipulation von Objekten (der realen Welt) | prozedurale, imperative und objektorientierte Sprachen ⁵ |

„In der Software-Technik lassen sich drei Paradigmen unterscheiden:

- das strukturierte Paradigma,
- das objektorientierte Paradigma,
- das wissensbasierte Paradigma.

Diese verschiedenen Denkmuster werden in der Software-Technik oft als sich gegenseitig ausschließend betrachtet und dementsprechend auch getrennt gelehrt. [...] Jeder Software-Ingenieur muß wissen, welches Paradigma für welches Problem oder Teilproblem am besten geeignet ist. Es geht nicht um »Entweder – Oder« sondern um »Sowohl als auch« [?, S. 40f].

1.2 Vergleichsuntersuchungen

Wer vergleichen will, muss kennen. Das fällt bei dem überbordenden Angebot nicht leicht: in einer bis 1995 gepflegten Liste von Programmiersprachen werden 2350 Sprachen genannt [Kinnersley 1995]. Von den genannten werden allerdings nach Einschätzung der Autoren nur ca. 25 breiter eingesetzt. Bezüglich der Auseinandersetzung um Begründungen für die „richtige“ Sprache – gerade im Anfangsunterricht darf der Artikel [Böszörményi 1998], der auch in einer deutschen Übersetzung (von Andreas SCHWILL) vorgelegt wurde [Böszörményi 2001] nicht unerwähnt bleiben. Genauso scheint es nicht unwichtig zu sein, die Geschichte der Programmiersprachen näher zu untersuchen und den Versuch zu unternehmen, sich einmal unvoreingenommen unbekannte Sprachen anzusehen: Ob GÖDEL oder ERLANG oder HASKELL oder EUKLID, EULER (ja, Herr Wirth hat nicht nur PASCAL [Wirth 1971], MODULA-2 [Wirth 1985] und OBERON [Reiser 1991] entwickelt) aber auch Bloop [Hofstadter 1985], SCHEME [Abelson u. a. 1998], EIFFEL [Rist 1996], SMALLTALK [Allen 1999], RUBY (Quelle ??).

1.2.1 Geschichte der Programmiersprachen

Zur Geschichte der Programmiersprachen finden sich ausnehmend viele Hinweise. Besonders ergiebig finde ich [?] – er berücksichtigt allerdings keine Skriptsprachen.

Einen besonderen Stellenwert nehmen für mich die Proceedings: The Second ACM SIGPLAN Conference on History of Programming Languages April 20 – 23, 1993, Cambridge, United States ein. Sie sind unter <http://www.acm.org/pubs/contents/proceedings/series/hopl/> dokumentiert. Dort läßt sich trefflich nachlesen, was die Entwickler der Programmiersprachen – besonders bezogen auf Konzepte – getrieben hat und wie sie die Konzepte umgesetzt haben. „An introduction to XY“ führt auf verschiedene Programmiersprachen, die jeweils kurz (auf zwei Seiten) vorgestellt werden. Damit sind wir mitten in der Diskussion: Was ist denn nun „Das neue Pascal“? [Liao 1999] – oder sollten wir doch eher dem Autor von PASCAL Glauben schenken [Reiser und Wirth 1994]? Braucht es denn überhaupt eine imperative Sprache? Sowohl funktionale, wie auch wissensbasierte Sprachen haben eine erheblich einfachere Syntax und vor allem: sie sind – bezogen auf ihre jeweiligen Anwendungsfelder – sehr leistungsfähig [Klaeren 1994].

³ Alle Objekte werden als Funktion $f : D \rightarrow E$ aufgefasst, selbst Konstante sind konstante Funktionen. Funktionen können Argument einer weiteren Funktion sein und vice versa. Siehe z. B. [Staudte 1992, Loethe 1997].

⁴ Mit Hilfe des Prädikatenkalküls wird versucht, über Aussageformen Mengen zu beschreiben. Damit kann der Zusammenhang zwischen der mengentheoretischen und der aussagenlogischen Beschreibung von Problemstellungen zum Zweck der Bestimmung von Lösungsmengen geleistet werden.

Siehe z. B. [Baumann 1991, Schubert 1992].

⁵ **imperativ:** Im Mittelpunkt steht die an virtuellen Maschinen orientierte Programmiersprache; Ziel ist das Erlernen einer speziellen Programmiersprache - die Problemstellungen werden dem einzuführenden „Befehl“ angepasst. Dieser Ansatz wurde im Laufe der Zeit zunehmend in Richtung auf Strukturen hin verbessert und optimiert und stellt heute an vielen Schulen im Sekundarstufen II-Bereich den ausschließlichen Standard des Informatikunterrichts dar. Siehe z. B. [Balzert 1983].

objektorientiert: Jedes System besteht aus Objekten. Jede Aktivität des Systems ist die Aktivität einzelner Objekte. Gleichartige Objekte werden zu Klassen zusammengefasst. Die Objekte stellen eine Einheit aus *Zustandsvariablen* und *Methoden* dar, d. h. im Objekt wird die imperative Trennung von Daten und Aktion auf (mit) den Daten [bei der Analyse] aufgehoben. Zentral ist die Analyse des Gegenstandes und der anschließende Entwurf, allerdings kommen zentrale imperative Konzepte bei der Erarbeitung von Methoden durchaus zum Einsatz.

Siehe z. B. [Müller 1992, Czischke u. a. 1999].

1.3 Modellierung

1.3.1 Modelle in der Informatik

In letzter Zeit hat ein Promovent der Didaktik der Informatik sich intensiver mit Modellen in der Informatik beschäftigt und einige Überlegungen dazu veröffentlicht: [Thomas 2000, Thomas 2001]. Sicherlich ist in diesem Feld noch maßgebliche Forschungsarbeit zu leisten – dennoch sei hier angemerkt, dass gerade im Bereich der Wissenschaft Informatik eine ungeheure Vielfalt von Modellen gefunden werden kann, ja man kann soweit gehen, dass im Zusammenhang mit der Informatik alle bekannten Modelltypen eine wichtige Rolle spielen. Im Folgenden wird auf die den oben angegebenen Paradigmen entsprechenden Modellierungssichten abgehoben.

1.3.2 Zur objektorientierten Modellierung

Wer hat nicht die Bemerkungen von Klaus FÜLLER im Kontext seines Vortrages anlässlich der INFOS'99 in Ohr [Füller 1999], der fragt, wie er denn der Schülerin im Anfangsunterricht

```
public static void
```

erklären soll, damit sie ihre erste Problemlösung einem Automaten zur Abarbeitung übergeben kann.

[Quibeldey-Cirke 1994] stellt die Entwicklung objektorientierter Ideen im Gesamtzusammenhang dar und macht deutlich, dass hinter den OO-Sprachen das Konzept OO verborgen bleibt, wenn nicht der Schwerpunkt auf der OOM liegt, sondern nur über die Implementierungen diskutiert wird.

In der aktuellen fachdidaktischen Forschung liegt ein ausgewiesener Schwerpunkt im Bereich der objektorientierten Modellierung, wie aktuelle Beiträge [Brinda 2001] und [Schulte 2001] von zwei Promoventen im Feld der Didaktik der Informatik deutlich zeigen.

1.4 Paradigmen

In der Literatur finden sich abstrahierende und konkrete Sprachenvergleiche bis hin zu empirischen Untersuchungen:

- [Floyd 1979, Rede anlässlich der Verleihung des Turing-Awards] – auch andere Reden, die anlässlich der alljährlichen Verleihung des Turing-Award gehalten wurden, beziehen sich ebenfalls explizit auf Überlegungen zur Wahl der Paradigmen und darauf, mit welcher Sprache eine ausbildungsfördernde Umsetzung erfolgen kann (vgl. [?]).
- [Ambler u. a. 1992]
- [Klaeren 1994]
- [Prechelt 2000b, Prechelt 2000a]

Die Ergebnisse sollten von der Didaktik berücksichtigt werden, um „das Rad nicht noch einmal erfinden zu müssen“ – allerdings sind die Anforderungen in der allgemein bildenden Schule nicht ohne weiteres mit den in diesen Studien vorausgesetzten vergleichbar. Warum nicht einmal eine Problemstellung in den verschiedenen Paradigmen bearbeiten? Lassen Sie sich inspirieren – [Brennwalder und Stamm 1994] hilft Ihnen dabei.

1.5 Mehrere Paradigmen unter ein Sprachendach?

Nach meinem Dafürhalten (aber auch nach den Überlegungen von Lehrplankommissionen) ist es sinnvoll, mehrere Paradigmen im Informatikunterricht der Sekundarstufe II zu thematisieren.

Es gibt Prototypen zur Integration mehrerer Paradigmen unter ein (objektorientiertes) Sprachendach – siehe z. B. [Spinellis u. a. 1994] – die aber eher aus Gründen der Forschung, denn der Lehre entwickelt und eingesetzt werden. Für den Ausbildungsunterricht in der Schule sind besondere Anforderungen an eine „Multiparadimen-Sprache“ zu stellen, durch die der Lehr-Lernprozess befördert werden kann:

- Es sollte möglich sein, jedes der Paradigmen „isoliert“ (und damit „sauber“) zu thematisieren – wichtig im Zusammenhang mit der Erarbeitung paradimenspezifischer Fragestellungen.
- Die zur Verwendung kommende Syntax sollte die Integration mehrerer Paradigmen möglichst bruchlos unterstützen, um sie auch praktisch umsetzen zu können (die Implementierung ist nach wie vor eine Hürde, die nicht unterschätzt werden darf).
-

Python gestattet es z. B. sowohl das wissensbasierte Paradigma umzusetzen, aber auch funktionale Wurzeln werden nicht ignoriert.

Zu allen drei Elementen gibt es Implementierungen:

[Keller 2000] <http://sourceforge.net/projects/xoltar-toolkit/> aber auch die Artikelserie von [Mertz 2001b, Mertz 2001d, Mertz 2001a, Mertz 2001c] dokumentieren, wie die funktionalen Elemente von Python mit Sprachmitteln so erweitert werden und wurden, dass es möglich wird, funktionale Programme im Sinne „der reinen Lehre“ zu schreiben.

Eine Anbindung von Python⁶ an gprolog⁷ wurde im Mai 2002 unter <http://bedevere.sourceforge.net/> präsentiert.

In [Lutz 2001, S. 1039f] wird gezeigt und auf der dem Buch beiliegenden CD⁸ an Hand einfacher Beispiele mit Prolog-Syntax dokumentiert, wie ein Expertensystem unter Python implementiert wurde (Quellcode ist auf der CD verfügbar; Quelle im Netz <http://nickel.as.arizona.edu/~barg/pp2e/> – Sun, 3 Jun. 2001).

Was spricht dafür, die Implementierung verschiedener Paradigmen

- in *einer* Programmiersprache vorzunehmen?
- nicht in einer Programmiersprache vorzunehmen?

Vorbemerkung: typischerweise verfügen Sprachen nicht über eine Syntax, die eine positive Entscheidung angemessen unterstützen. Dieses Argument trifft nach m. E. bei den vorgenannten Beispielimplementierungen verschiedener Paradigmen in Python nicht zu.

Als Beispiel mag der Informatik-Bundeswettbewerb stehen, in dem 1992/93 eine dem TSP sehr ähnliche Problemstellung einen Schüler dazu trieb, eine siebenzeilige Prolog-Lösung zu entwickeln. Da er aber nicht glauben konnte, dass dies im Sinne der Problemstellung „richtig“ sei, hat er in Modula-2 noch einen Prolog-Interpreter geschrieben, der diese siebenzeilige Lösung interpretieren konnte (vgl. [Busse 1999, S. 135]).

1.6 Wie weiter?

Datenbanken: Warum werden hier keine Sprachen, wie SQL⁹ genannt?

Generische Datenbankschnittstellen bieten die Möglichkeit, eine Anbindung von Datenbanken in der jeweiligen Programmiersprache transparent und bruchlos, bis hin zur Abarbeitung von SQL-Kommandos, die über definierte Kommandos an die Datenbank weitergeleitet werden, zu nutzen. Siehe z. B. [Lutz 2001, S. 960ff SQL Database Interfaces] (Python); aber auch „alte Bekannte“, wie Turbo Pascal, Modula-2 oder „neue Freunde?“ wie Java verfügen über wohldefinierte Datenbankschnittstellen; i. Ü. in Element, das bereits in frühen Versionen von COBOL erwähnenswert (weil direkt nutzbar) war. Ach so: Delphi bringt eine Datenbankschnittstelle mit.

⁶mittels eines sogenannten Wrappers (engl. – deutsch Hülle)

⁷GNU Prolog – siehe <http://gnu-prolog.inria.fr/>

⁸im Verzeichnis ../EXAMPLES/PP2E/Ai/ExpertSystem/

⁹standard query language – **die** Datenbanksprache, die – soweit ich mich erinnere – von IBM durchgesetzt wurde und ohne die heute (fast) keine Datenbankimplementierung auskommt; es handelt sich um eine Sprache des prädikativen (also wissensbasierten) Paradigmas.

Ein konkretes Beispiel zur Verschränkung verschiedener Paradigmen liefert der Beitrag [Garshol 2001]: durch die Analyse der Problemstellung, *topic-maps* produktiv nutzen zu können, wird eine informale Modellierung vorgenommen, aber daraufhingewiesen:

„The structure could be specified using some object-oriented language (that is, it could be an object model), it could be specified mathematically, or it could be specified using some specialized schema language like HyTime property sets or ISO10303 EXPRESS. [...] we will only use an informal abstract model. [...] tolog is very closely based on Prolog, a near-declarative logic programming language based on first-order predicate logic [ISO13211]. Central to both languages are the concept of a fact database containing statements about the universe of discourse that is, the subject area of the database. If the subject area were Italian Opera, one would expect to find statements like "Tosca (an opera) was composed by Puccini", "Puccini was born in Lucca" and so on in the database. In a topic map, these statements would be expressed as associations, like "Tosca plays the role of opera in a composed-by association where Puccini plays the role of composer", "Puccini plays the role of person in a born-in associations where Lucca plays the role of place", etc.“ [Garshol 2001].

In dem Artikel wird auf dieser Grundlage ausgeführt, wie eine Integration dieses Paradigmas zur Nutzung von *topic-maps* als Erweiterung die serverbasierte (auf der Basis von Java-Server-Pages) Beantwortung von Abfragen vereinfacht. Weiterhin werden Überlegungen ausgeführt, wie mit Hilfe der Abfragesprache die Nutzung von *topic-maps* qualifiziert umgesetzt werden kann.

Feststellung 1: Für alle gängigen Programmiersprachen (und mehr) stehen in einer normalen Linux-Distribution Compiler, Interpreter, ja komplette Entwicklungsumgebungen (oftmals frei) zur Verfügung. Kosten sind damit kein Argument für die Auswahl/Abwahl einer Programmiersprache.

Feststellung 2: Was der Bauer nicht kennt, ... oder „Die Macht der Gewöhnung.“

So durfte der Autor in einer laufenden Projektgruppe in der Universität Dortmund erfahren, dass ein Studierender im Hauptstudium Informatik meinte: „Nee, ich bleibe lieber bei Java, mit Python habe ich keine Erfahrung und das kommt dann für das Projekt nicht in Frage.“

Ähnliche Aussagen erhielt der Autor, als er Aufgaben zu verschiedenen Paradigmen für Übungen zur Didaktik der Informatik im Grundstudium stellte: „Wir haben aber doch überhaupt keine Vorlesungen zum wissensbasierten oder zum funktionalen Paradigma gehabt, wir hatten doch nur Java, wie sollen wir uns denn dann zu den anderen Paradigmen äußern können?“

1.7 War da nicht noch?

Im Kontext der Diskussion um die Folgen der ersten Softwarekrise [Naur und Randell 1969] verdienen Entwicklungen um die Aufkündigung des Phasenmodells (Wasserfall) im Kontext evolutionärer Systementwicklung eine besondere Beachtung (vgl. [Floyd 1994, S. 35]). Interessant ist, dass im Kontext der Beschreibung der Elemente der Design-Schicht bzgl. der Produktionssicht die Verschränkung von Softwareentwicklerinnen und Anwenderinnen durch gemeinsames Lernen gefordert wird. „Maßgeblich für das Gelingen sind Exploration, Experimente, Kommunikation und Reflexion zwischen den Beteiligten“ [Floyd 1994, S. 33]. „Die technische Unterstützung von Lernprozessen bei der Softwareentwicklung erfolgt durch exploratives, experimentelles und evolutionäres Prototyping ...“ [Floyd 1994, S. 35].

Für diese Prozesse sind Voraussetzungen zu schaffen, die zur allgemeinen Bildung zu zählen sind, d. h. in der Schule für alle Schülerinnen Bestandteil des Pflichtunterrichts sein müssen. Um eine aktive Rolle in der partizipativen Softwareentwicklung wahrnehmen zu können, bedarf es der Kenntnis grundlegender Möglichkeiten der Gestaltung von Informatiksystemen. Diese bestehen u. a. darin, prinzipielle Konzepte, ihre Möglichkeiten, aber auch ihre Grenzen zu kennen. Dann kann der oben beschriebene Prozess erfolgreich umgesetzt werden.

Allerdings sollte nicht verkannt werden, dass eine grosse Zahl von Benutzerinnen mit vorgefertigten Informatiksystemen konfrontiert wird und zu arbeiten gezwungen ist, die zum Standardbestandteil ausgelieferter Systeme gehören: sogenannte Standardbetriebssysteme, -anwendungssoftware [und -pakete]. Hier liegt eine andere Situation vor: die Nutzerinnen sollten in der Lage sein, die hinter den Systemen stehende Modellierung zu erkennen und sie kritisch zu würdigen. Diese Forderung ist bereits im Pflichtunterricht in der Sekundarstufe I einlösbar. Die Umsetzung konnte der Autor im Jahre 2001 bei Besuchen in bayrischen Schulen (Jahrgangsstufe 6) beobachten. Einige dieser Unterrichtsbesuche wurden in [Frey u. a. 2001] und [Hubwieser u. a. 2001] dokumentiert. Somit steht zu erwarten, dass es möglich ist, Modellkritik und Modellierung unterrichtlich im Pflichtbereich unserer allgemein bildenden Schulen zu verankern.

Literatur

- [Abelson u. a. 1998] ABELSON, Harold ; SUSSMAN, Gerald J. ; SUSSMAN, Julie (Coautorin): *Struktur und Interpretation von Computerprogrammen*. 3. überarb. Aufl. Berlin : Springer, 1998 (Springer-Lehrbuch). – Übersetzung von: *Structure and Interpretation of Computer Programs*, Cambridge, 1996 (1st Ed. 1985) übersetzt von Susanne Daniels–Herold
- [Allen 1999] ALLEN, Russell. *Squeak, an open, highly-portable Smalltalk-80 implementation*. <http://squeak.org/index.html>. 1999
- [Ambler u. a. 1992] AMBLER, Allen L. ; BURNETT, Margaret M. ; ZIMMERMANN, Betsy A.: Operational Versus Definitional: A Perspective on Programming Paradigms. In: *IEEE Computer* 25 (1992), September, Nr. 9, S. 28–42
- [Backus u. a. 1963] BACKUS, J.W. ; BAUER, F.L. ; J.GREEN ; KATZ, C. ; MCCARTHY, J. ; NAUR, P. ; PERLIS, A.J. ; RUTISHAUSER, H. ; SAMUELSON, K. ; VAUQUOIS, B. ; WEGSTEIN, J.H. ; WIJNGAARDEN, A. van ; WOODGER, M. ; NAUR, Peter (Hrsg.): *Revised Report on the Algorithmic Language Algol 60*. diverse Zeitschriften, 1963. – CACM, Vol. 6, pp 1–17; The Computer Journal, Vol. 9, p. 349; Numerische Mathematik, Vol. 4, p. 420. <http://www.masswerk.at/algol60/report.htm>
- [Balzert 1983] BALZERT, Helmut: *Informatik: 1. Vom Problem zum Programm – Hauptband*. 2. Aufl. München : Hueber-Holzmann Verlag, 1983. – 1. Aufl. 1976
- [Baumann 1991] BAUMANN, Rüdiger: *Prolog. Einführungskurs*. Stuttgart : Klett-Schulbuchverlag, 1991
- [Bosler 1992] BOSLER, Ulrich u. a. (Hrsg.): *Schulcomputerjahrbuch '93/94*. Hannover, Stuttgart : Metzler Schulbuch und B.G. Teubner, 1992
- [Böszörményi 1998] BÖSZÖRMÉNYI, László: Why Java is not my favorite first-course language. In: *Software – Concepts & Tools* 19 (1998), S. 141–145. – <http://www.ifi.uni-klu.ac.at/ITEC/Publications/showabs?1998-0033-Boes>
- [Böszörményi 2001] BÖSZÖRMÉNYI, László: JAVA für Anfänger? Warum JAVA nicht meine Liebessprache für einen Anfängerkurs ist. In: *LOG IN* 21 (2001), Nr. 1, S. 14–19. – übersetzt von Andreas Schwill
- [Brennwalder und Stamm 1994] BRENNWALDER, Daniel ; STAMM, Christoph: *Gruppenunterricht zum Thema: Paradigmen von Programmiersprachen*. <http://educeth.ethz.ch/informatik/puzzles/paradigmen/>. September 1994. – Eidgenössische Technische Hochschule Zürich – Institut für Verhaltenswissenschaft / Departement für Informatik PDF-Dokument vom 6. November 1997
- [Brinda 2001] BRINDA, Torsten: Einfluss fachwissenschaftlicher Erkenntnisse zum objektorientierten Modellieren auf die Gestaltung von Konzepten in der Didaktik der Informatik. In: **[Keil-Slawik und Magenheim 2001]**,
- [Brunnstein 2001] BRUNNSTEIN, Klaus: Mit IT-Risiken umgehen lernen: Über Probleme der Beherrschbarkeit komplexer Informationssysteme. In: **[Keil-Slawik und Magenheim 2001]**, S. 9–12
- [Busse 1999] BUSSE, Johannes: *Attribution von Verantwortung durch Metapheranalyse*. Wilhelm Schickard Institut für Informatik Sand 13, D-72 076 Tübingen, Universität Tübingen, Diss., Juni 1999. – <http://www-pu.informatik.uni-tuebingen.de/users/busse/diss/>
- [Czischke u. a. 1999] CZISCHKE, Jürgen ; DICK, Georg ; HILDEBRECHT, Horst ; HUMBERT, Ludger ; UEDING, Werner ; WALLOS, Klaus ; LANDESINSTITUT FÜR SCHULE UND WEITERBILDUNG (Hrsg.): *Von Stiften und Mäusen*. 1. Aufl. Bönen : DruckVerlag Kettler GmbH, 1999
- [Floyd 1994] FLOYD, Christiane: Software-Engineering - und dann? In: *Informatik Spektrum* 17 (1994), Februar, Nr. 1, S. 29–37
- [Floyd 1979] FLOYD, Robert W.: The Paradigms of Programming. In: *Communications of the ACM* 22 (1979), August, Nr. 8, S. 455–460. – Turing Award Lecture
- [Frey u. a. 2001] FREY, Elke ; HUBWIESER, Peter ; HUMBERT, Ludger ; SCHUBERT, Sigrid ; VOSS, Siglinde: Erste Ergebnisse aus dem Informatik-Anfangsunterricht in den bayerischen Schulversuchen. In: *LOG IN* 21 (2001), Nr. 1, S. 25–37. – http://ddi.cs.uni-dortmund.de/ddi_bib/forschung/pub/Informatik-Anfangsunterricht.pdf
- [Füller 1999] FÜLLER, Klaus: Objektorientiertes Programmieren in der Schulpraxis. In: SCHWILL, Andreas (Hrsg.): *Informatik und Schule – Fachspezifische und fachübergreifende didaktische Konzepte*. Berlin : Springer, September 1999 (Informatik aktuell), S. 190–201
- [Garshol 2001] GARSHOL, Lars M.: tolog – A topic map query language. In: *XML Europe*, 2001. – <http://www.gca.org/papers/xml europe2001/papers/pdf/s31-3.pdf> – geprüft: 30. May 2002
- [Hofstadter 1985] HOFSTADTER, Douglas R.: *Gödel, Escher, Bach: ein endloses geflochtenes Band*. 5. Aufl. Stuttgart : Klett-Cotta, 1985. – Original: Gödel, Escher, Bach: an Eternal Golden Braid, 1979 im Verlag Basic Books, New York
- [Hubwieser u. a. 2001] HUBWIESER, Peter ; HUMBERT, Ludger ; SCHUBERT, Sigrid: Evaluation von Informatikunterricht. In: **[Keil-Slawik und Magenheim 2001]**, S. 213–215
- [Keil-Slawik und Magenheim 2001] KEIL-SLAWIK, Reinhard (Hrsg.) ; MAGENHEIM, Johannes (Hrsg.): *Informatik und Schule - Informatikunterricht und Medienbildung INFOS 2001 – 9. GI-Fachtagung 17.–20. September 2001, Paderborn*. Bonn : Gesellschaft für Informatik, Köllen Druck + Verlag GmbH, September 2001 (GI-Edition – Lecture Notes in Informatics – Proceedings P-8)

- [Keller 2000] KELLER, Bryn. *Support for a functional style of Python programming*. <http://sourceforge.net/projects/xoltar-toolkit/> – geprüft: 30. May 2002. November 2000
- [Kinnersley 1995] KINNERSLEY, Bill: *The Language List*. <ftp://wuarchive.wustl.edu/doc/misc/lang-list.txt>. January 1995. – Version 2.4
Started Mar 7, 1991 by Tom Rombouts
- [Klaeren 1994] KLAEREN, Herbert: Probleme des Software-Engineering, Die Programmiersprache - Werkzeug des Softwareentwicklers. In: *Informatik Spektrum* 17 (1994), Februar, Nr. 1, S. 21–28
- [Liao 1999] LIAO, Luby: *Python as the Pascal of 2000's*. ORA Open Source Convention at Monterey. August 1999. – <http://www.acusd.edu/~liao/python.pdf> <http://conferences.oreilly.com/cd/python/presentations/lliao/python.pdf>
- [Loethe 1997] LOETHE, Herbert M.: Computer integration in the mathematics study of pre-service teacher education – experiences with Project CIMS. In: PASSEY, Don (Hrsg.); SAMWAYS, Brain (Hrsg.): *Information Technology – Supporting change through teacher education*. London : Chapman & Hall, 1997, S. 238–244
- [Lutz 2001] LUTZ, Mark: *Programming Python. Solutions for Python Programmers*. 2nd. Ed. Sebastopol : O'Reilly, March 2001. – Chapter 15: Advanced Internet Topics <http://www.oreilly.com/catalog/python2/chapter/ch15.html>
- [Mertz 2001a] MERTZ, David: Charming Python: Even More Functional Programming in Python. In: *IBM developerWorks* (2001), June. – Series: Charming Python 19 http://gnosis.cx/publish/tech_index.html <http://www-106.ibm.com/developerworks/library/l-prog3.html>
- [Mertz 2001b] MERTZ, David: Charming Python: Functional programming in Python. In: *IBM developerWorks* (2001), March. – Series: Charming Python 13 http://gnosis.cx/publish/tech_index.html <http://www-106.ibm.com/developerworks/library/l-prog.html>
- [Mertz 2001c] MERTZ, David: Charming Python: Iterators and simple generators. In: *IBM developerWorks* (2001), September. – Series: Charming Python http://gnosis.cx/publish/tech_index.html <http://www-106.ibm.com/developerworks/library/l-pycon.html>
- [Mertz 2001d] MERTZ, David: Charming Python: More Functional Programming in Python. In: *IBM developerWorks* (2001), April. – Series: Charming Python 16 http://gnosis.cx/publish/tech_index.html <http://www-106.ibm.com/developerworks/library/l-prog2.html>
- [Moll und Buhse 1984] MOLL, Gerhard ; BUHSE, Reinhard: *Metzler Informatik Lehrerband*. 1. Aufl. Stuttgart : J.B. Metzler, 1984
- [Müller 1992] MÜLLER, Klaus: Objektorientierte Programmierung. In: [Bosler 1992], S. 180–189
- [Naur und Randell 1969] NAUR, Peter (Hrsg.) ; RANDELL, Brian (Hrsg.): *Software Engineering. Report on a conference sponsored by the NATO Science Committee*. Brussels 39, Belgium : NATO, Scientific Affairs Division, January 1969 . – Garmisch, Gernay, 7th to 11th October 1968
- [Padawitz 1995] PADAWITZ, Peter: *Programmierung II - Sommersemester*. Dortmund : Universität Dortmund, 1995. – Vorlesungsskript - FB Informatik, LS 5, Universität Dortmund
- [Padawitz 1998] PADAWITZ, Peter: *Einführung ins funktionale Programmieren – Vorlesungsskriptum*. Dortmund : Universität Dortmund, 1998. – LV Grundlagen und Methoden funktionaler Programmierung <http://ls5-www.cs.uni-dortmund.de/padawitz.html>
- [Prechelt 2000a] PRECHELT, Lutz: An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl. In: *IEEE Computer* 33 (2000), October, Nr. 10, S. 23–29. – http://wwwipd.ira.uka.de/~prechelt/Biblio/jccpprt_computer2000.pdf
- [Prechelt 2000b] PRECHELT, Lutz: An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl for a search/string-processing program / Fakultät für Informatik, Universität Karlsruhe. Karlsruhe, March 2000b (2000-5). – Forschungsbericht. <http://wwwipd.ira.uka.de/~prechelt/Biblio/jccpprtTR.pdf> 34 Seiten
- [Quibeldey-Cirkel 1994] QUIBELDEY-CIRKEL, Klaus: *Das Objekt-Paradigma in der Informatik*. Stuttgart : Teubner-Verlag, 1994
- [Reiser 1991] REISER, Martin: *The Oberon System: user guide and programmer's manual*. New York : ACM Press, 1991
- [Reiser und Wirth 1994] REISER, Martin ; WIRTH, Niklaus: *Programmieren in Oberon: Das neue Pascal*. New York : ACM Press, 1994
- [Rist 1996] RIST, Robert S.: Teaching Eiffel as a first language. In: *Journal of Object-Oriented Programming* (1996), März/April, S. 30–41
- [Schubert 1992] SCHUBERT, Sigrid: Logische Programmierung. In: [Bosler 1992], S. 171–179
- [Schulte 2001] SCHULTE, Carsten: Vom Modellieren zum Gestalten – Objektorientierung als Impuls für einen neuen Informatikunterricht? In: *informatica didactica* (2001), Juli, Nr. 3. – <http://didaktik.cs.uni-potsdam.de/InformaticaDidactica/Issue3> Ausgewählte Beiträge der Tagung „IAB2000 - Informatik und Ausbildung“
- [Spinellis u. a. 1994] SPINELLIS, Diomidis ; DROSSOPOULOU, Sophia ; EISENBACH, Susan: Language and Architecture Paradigms as Object Classes: A Unified Approach Towards Multiparadigm Programming. In: GUTKNECHT, Jürg (Hrsg.): *Programming Languages and System Architectures* Bd. 782. Berlin, Heidelberg : Springer-Verlag, March 1994, S. 191–207
- [Staudte 1992] STAUDTE, Rainer: Funktionale Programmierung. In: [Bosler 1992], S. 160–171

[Thomas 2000] THOMAS, Marco: Modellbildung im Schulfach Informatik. In: HISCHE, Horst (Hrsg.): *Modellbildung, Computer und Mathematikunterricht*. Hildesheim : Verlag Franzbecker, 2000 (Bericht über die Arbeitstagung vom 1. bis 4. Oktober 1998 in Wolfenbüttel 16). – <http://didaktik.cs.uni-potsdam.de/HyFISCH/Modellbildung/ModellbildungMuI.pdf>, S. 39–46

[Thomas 2001] THOMAS, Marco: Die Vielfalt der Modelle in der Informatik. In: [Keil-Slawik und Magenheimer 2001], S. 173–186

[Wirth 1971] WIRTH, Niklaus: The Programming Language Pascal. In: *Acta Informatica* 1 (1971), Nr. 1, S. 35–63

[Wirth 1985] WIRTH, Niklaus: *Programmieren in Modula-2*. 1. dt. Aufl. Berlin : Springer, 1985

A Programmiersprachen „neben der Spur“

A.1 FORTH

humbert@haspe:~/MATERIAL/PDF/HOPL >

gforth GForth 0.4.0, Copyright (C) 1998 Free Software Foundation, Inc.

GForth comes with ABSOLUTELY NO WARRANTY; for details type 'license' Type 'bye' to exit

3 4 + 5 * ok . 35 ok

: SUM-OF-SQUARES (a b - c) DUP * SWAP DUP * + ; ok

: SQUARED (n - n**2) DUP * ; ok

: SUM-OF-SQUARES (a b - c) SQUARED SWAP SQUARED + ; redefined SUM-OF-SQUARES ok

3 SQUARED . 9 ok

3 4 SUM-OF-SQUARES . 25 ok

A.2 APL

...

B Der besondere Stoff – Software – und wie er beschrieben wird

Verfolgen wir die Geschichte der Informatik, so fällt uns als Erstes auf, dass bezüglich der wissenschaftstheoretischen Einordnung nach Vergleichen mit biologischen Systemen, wie sie von von NEUMANN und WIENER deutlich hervorgehoben wurden / teilweise unter Mißachtung der den Informatiksystemen zu grunde liegenden deterministischen Struktur über die Zeit geradezu eine Ablehnung der Auseinandersetzung mit der Hardware stattgefunden hat – obwohl kein Informatiksystem ohne Hardware existieren kann. Auch wenn Information als eine von Energie und Materie unabhängige neue Kategorie bereits von WIENER beschrieben wurde: „Information ist Information, nicht Energie nicht Materie“ so ist festzustellen, dass geschichtlich betrachtet, die ersten Softwareexperten allesamt Frauen waren: ADA, die ENIAC GIRLS, Grace HOPPER. Andererseits muss der Tatsache Rechnung getragen werden, dass die Entwicklung von Programmiersprachen erfolgte, noch bevor sich irgendein Mensch Gedanken über Compiler gemacht hat: Sowohl die Notation von TURING, wie auch ZUSES Plankalkül aber nicht zuletzt die Bemerkungen von ADA über das Programmieren machen deutlich, dass Programmiersprachen in aller erster Linie nicht dazu entwickelt wurden, konkrete Hardware damit zu bedienen, sondern Kommunikation über die in Programmen gegossene Ideen zu ermöglichen.

Es erstaunt, dass unter völliger Ignoranz dieser geschichtlichen Tatsachen, Programmiersprachen wie FORTRAN, COBOL u. a. , „gestrickt“ wurden.

Die Entwicklung von Algol 60 erfolgt auf einer soliden Fachbasis (mit einer klar definierten Syntax in BNF) und wird von den Autoren explizit zur Kommunikation von und über Algorithmen gestaltet:

„... three different levels of language are recognized, namely a Reference Language, a Publication Language, and several Hardware Representations [...]. [The] Publication Language ... is used for stating and communicating process“ [Backus u. a. 1963, Introduction].

Didaktische Gesichtspunkte spielten bei der Entwicklung keine Rolle, wenn auch das Kriterium der Orthogonalität eine gute Voraussetzung für die Erlernbarkeit darstellt.

„One of the greatest impacts ALGOL 60 had was a result of its description as found in [Backus u. a. 1963]. A major contribution of this report was the introduction of BNF notation for defining the syntax of the language. Overall, ALGOL is considered to be perhaps the most orthogonal programming language, meaning it has a relatively small number of basic constructs and a set of rules for combining those constructs. Every construct has a type associated with it and there are no restrictions on those types. In addition, most constructs produce values. Several of ALGOL's other characteristics are listed below:

- Dynamic Arrays - one for which the subscript range is specified by variables so that the size of the array is set at the time storage is allocated.
- Reserved Words - the symbols used for keywords are not allowed to be used as identifiers by the programmer.
- User defined data types - allow the user to design data abstractions that fit particular problems very closely.

Areas of Application ALGOL was used mostly by research computer scientists in the United States and in Europe. Its use in commercial applications was hindered by the absence of standard input/output facilities in its description and the lack of interest in the language by large computer vendors. ALGOL 60 did however become the standard for the publication of algorithms and had a profound effect on future language development.“

<http://www.engin.umd.umich.edu/CIS/course.des/cis400/algol/algol.html>

Dennoch werden auch in der Folgezeit immer wieder Sprachen entwickelt, die eine solide informatische Basis vermissen lassen.

Die Featuritis mancher Vorschläge (im Kontext von ALGOL-68) führte zu qualifizierten Alleingängen, wie sie von Niklaus Wirth mehrfach erfolgreich (aber leider mit abnehmender Akzeptanz durch Lehrende) realisiert wurden: Pascal, Modula-2, Oberon.