Hauptseminar "Didaktik der Informatik" im WiSe 2007/08
an der Universität Innsbruck
Leitung: Prof. Dr. Peter Hubwieser

Lernansatz: „Objects First"

Alexander Jäger

MIP Fakultät
Institut für Informatik
Technikerstraße 21A
6020 Innsbruck
a.jaeger@student.uibk.ac.at

Abstract: Wann sollen Objekte im Informatikunterricht eingeführt werden? Dieser Artikel widmet sich dem Ansatz, dass Schüler objektorientiertes Programmieren vor dem iterativen Programmieren lernen sollten und wie das funktionieren kann Außerdem wird der Ansatz „Objects first" kritisch hinterfragt und weiterführende Ansätze diskutiert.

1. Einleitung

Das Lehren von Softwareentwicklung beinhaltet viele Aspekte. Wohl der Wichtigste dabei ist das Programmieren. Deshalb ist es besonders wichtig, den Schülern oder Studenten neben dem Problemlösungsdenken auch das Handwerkszeug für die Programmierung mitzugeben: die Programmierparadigmen. Da es viele dieser Paradigmen gibt, gilt es sich zu entscheiden, mit welchem die Schüler oder Studenten als erstes beglückt werden.

Bis vor einigen Jahren wurde der Unterricht meist mit dem imperativen, strukturierten, prozeduralen Programmieren begonnen. Erst später wurde dann objektorientiertes Programmieren gelehrt. In der Fachdidaktik gibt es jedoch auch seit einigen Jahren Stimmen, welche fordern, dass mit objektorientierter Programmierung begonnen werden soll.

2. Objektorientiert Programmieren

Obwohl die erste objektorientierte Programmiersprache „Simula 67" bereits in den Sechzigerjahren zu Verfügung stand, dauerte es bis zur Entwicklung von C++ (1983), bis die Objektorientierung Fuß fasste. [WikiA]

2.1 Warum objektorientiert?

Inzwischen hat sich das objektorientierte Programmieren als eigenständiges und wichtiges Paradigma etabliert. So beschreiben Pokkunuri, Lieberherr und Decker Vorteile der objektorientierten Programmierung.

> "The activity of updating and maintaining […] changes [in large applications] turns out to be a nightmare. Such applications have forced the different groups to look for alternatives. This search has one result in the form of object-oriented programming methodology […]." [Pok89, 97]

> "Easier software maintenance, less coupling between your methods, better information hiding, narrower interfaces, methods which are easier to reuse, and easier correct.ness proofs using structural induction." [Lie88, 323]

> "OOP was said to support directly many of the software engineering concepts that are among the most difficult to convey in procedural terms: code reuse, encapsulation incremented development and testing, and, of course, program design." [DH94, 51]

Es ist kaum verwunderlich, dass die Objektorientierung in der Softwareentwicklung Einzug gehalten hat. Dies bestätigen auch die in letzter Zeit neu entwickelten Programmiersprachen. Sie sind alle objektorientiert (siehe Abbildung 1).

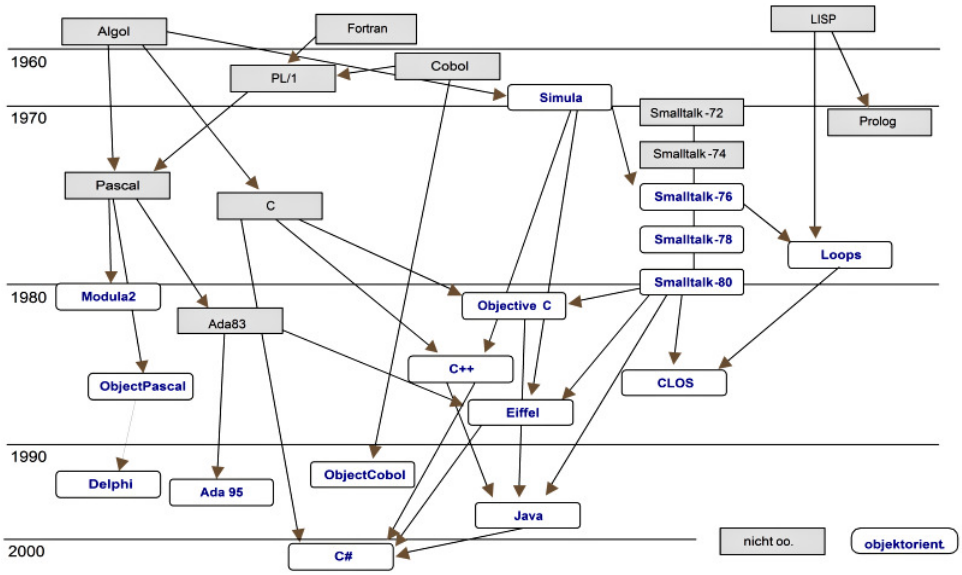

Abbildung 1 Im Laufe der Zeit gewannen objektorientierte Programmiersprachen an Einfluss [WikiA]

2.2 Objektorientierung lehren und lernen

Da objektorientiertes Programmieren in der Wirtschaft an Bedeutung begann, bedurfte es auch entsprechender Kurse, in denen die objektorientierte Inhalte und Konzept gelehrt wurden. Wie Hu es formulierte, besteht ein breiter Konsens darüber, dass auch Programmieranfänger Objektorientierung lernen sollten.

> „In the recent years, the common consensus has been that object-orientation should be taught to beginners given the fact that it has been, and will continue to be, a major programming paradigm used in the software industry." [HU04, 209]

Es gehen jedoch die Meinungen auseinander, wann in einem Anfängerkurs objektorientiertes Programmieren gelehrt werden soll. Während einige Autoren meinen, Kurse sollten gleich mit der Objektorientierung beginnen, finden das andere zu früh. Bevor auf diesen Diskurs näher eingegangen wird (siehe Kapitel 3), sollen noch die Probleme angesprochen werden, die Programmieranfänger mit der Objektorientierung haben.

2.3 Verständnisprobleme von Studenten beim objektorientierten Programmieren

Für Studenten bedeutet die Objektorientierung, dass es eine größere Anzahl an Konzepten gibt. Daraus folgt, dass es auch mehrere Möglichkeiten gibt, diese Konzepte falsch oder unzureichend zu verstehen.

Bereits ein Problem kann das Alter der Studenten bedeuten. So wies bereits Reek darauf hin, dass Jugendliche im Alter von 18 Jahren noch nicht so abstrakt denken können, dass sie die Objektorientierung gänzlich verstehen können:

> "Reek (1995) pointed out that the problem isn't that the students are stupid, but rather that at age eighteen their thinking maturity is still at the concrete level". [HU04, 212]

Auch Hu beschreibt Abstraktionsprobleme der Studenten beim objektorientierten Programmieren. So sei es für sie schwer, kleine Stücke zu programmieren, aber das Konzept (z.B. ein Pattern,…) im Auge zu behalten.

> "Based on author's extensive experience of teaching OO, it has been not the "main" method, the textural details of I/O, or the syntax of OO that causes problems, it is precisely the inability of doing programming-in-the-small and having to think at abstract level that has given students most difficulties." [HU04, 214]

Kate Sanders und Lynda Thomas untersuchten Verständnisschwierigkeiten beim objektorientierten Programmieren von Studenten. Sie kamen zu folgenden Schlüssen:

> „[…] we did not see much evidence of problems with basic mechanics. […] It seems that students were confusing subclass with instances. […] The students did not always use inheritance correctly. […] Finally, students frequently failed to use inheritance when they could have done so. […] Students used interfaces […] that were similar to those they had

> seen in the lectures, the labs, or the text, but did not factor out method signatures into new interfaces of their own. […] The students had more trouble with design patterns." [ST07, 168-169]

Obwohl es so aussieht, dass auf Schüler und Studenten mit der Objektorientierung ein großes Stück Arbeit zukommt, gibt es genügend Verfechter, dass Objekte gleich zu Beginn eines Programmierkurses eingeführt werden sollen. Mehr dazu im nächsten Kapitel.

3. Objekte zuerst

Wann Objekte im Informatikunterricht eingeführt werden sollen, spaltet die Informatik-Didaktik. Bereits der Begriff „Objects first" ist nicht so klar definiert. So findet man in der Literatur folgende Bezeichnungen:

Objects first: Wie Ira Diethelm [Die07, 21] feststellt, ist auch dieser Begriff nicht eindeutig definiert, wird jedoch oft als „Classes first" interpretiert.

Strictly objects first: Im Gegensatz zu "Objects first", stehen bei "Strictly objects first" laut Diethelm [Die07, 41] wirklich die Objekte im Vordergrund und nicht die Klassen. So stellt David Gries in [Gri08] ein entsprechendes Konzept hierfür vor.

Objects early: In [Lis06, 147] bezeichnet „Objects early", dass die objektorientierte Programmierung anhand einer konkreten Programmiersprache bereits früh in einem Softwareentwicklungskurs eingeführt wird. Dies steht etwas im Gegensatz zu [Die07, 22].

In diesem Kapitel sollen nun Verfechter und Gegner des Ansatzes Objekte zuerst zum Wort kommen.

3.1 Vorurteile gegen „Objects first"

Veränderungen sind schwierig. So auch, wenn ein Kurs nicht zuerst prozedurales Programmieren lehren soll, sondern Objektorientierung. Rick Decker und Stuart Hirshfield wagten bereits 1994 den Schritt. In einem Artikel beschreiben sie, mit welchen Hindernissen und Problemen sie zu kämpfen hatten:

> „[…] we were confronted with a long list of reasons (some real and some, as it turns out imagined) for not changing over to OOP. The fact is, we did it and we're glad." [DH94, 52]

Sie beschreiben zehn Vorurteile, welche Sie gegenüber der Objektorientierung hatten und widerlegen diese [DH94, 52-55]:

„**OOP is to hard for my CS 1 students**! […] many students said that they felt far more comfortable attacking a complex problem from scratch with OOP/C++ than they did with Pascal. […]

You still need algorithms! […] More generally, OOP provides a framework within which conventional algorithm development […] not only can, but must still occur. […]

The "OOP-ish" overhead! […] History and our brief experience teaching OOP has convinced us that such is not the case. Those of us old enough to remember (and we certainly qualify) the early days of the "structured programming" movement which in turn led to the development of Pascal, recall the first texts that used that phrase in their titles. They, too, devoted chapters to graphically illustrating, via structure charts, IPO diagrams, and the like, the salient features of the paradigm. […]

It's too hard for us! […] First, we would have to learn the material ourselves, and become comfortable enough with it to teach it. [Hervorhebung A.J.]"

Sie kommen jedoch zu dem Schluss, dass alle ihre Bedenken weniger mit der Tatsache zu tun hatten, dass Objektorientierung schwierig ist, sondern vielmehr war es ihr Unverständnis gegenüber der Objektorientierung:

"To be sure, we struggled long and hard with each of these, but have since come to recognize them all to reflect one or more of the following: (1) our lack of understanding of the paradigm (2) our fear of the commonly-used OOP languages (most notably C++), (3) our procedural biases derived from years of teaching Pascal or (4) what we now see as a growing body of OOP folklore that reflects a number of myths about OOP." [DH94, 52]

Rick Decker und Stuart Hirshfield empfehlen also, Schülern und Studenten das Programmieren anhand der Objektorientierung zu lernen. Unter anderem argumentieren sie auch damit, dass der sogenannte Paradigmenwechseln von der prozeduralen Programmierung zur Objektorientierung wegfällt. [DH94, 55] Auf diesen Umstieg sei im folgenden Kapitel näher eingegangen.

3.2 Schwierige Konzepte oder schwieriger Paradigmenwechsel

Bereits im vorigen Kapitel wurde in den Zitaten des Öfteren davon gesprochen, dass die Objektorientierung an sich sehr schwer zu verstehen ist. So fragt sich Hu, wie ein Softwareentwicklungskurs leichter werden soll, wenn der Inhalt schwieriger wird:

"What has made this previously considered hard material now become anyhow easier to the beginners in CS1 so that they can learn it from day one? Are students better prepared now than ever before? If not, does OO then replace structured programming (SP, also known as procedure/process-oriented programming) paradigm? If not, does OO then make transition to procedural paradigm easier, or do technologies make the OO learning curve substantially flatter?" [HU04, 210]

Auch Reges findet sogar empirisch belegbare Anhaltspunkte, dass die Objekt-orientierung zu schwer für einen Einsteigerkurs sei:

> "Reges (2006) reports that student evaluation of CS1 improved after reverting from objects-first to objects late in Java." [Lis06, 149]

Im Gegenzug stellt Ira Diethelm in [Die07] ein fundiertes Konzept vor, wie „Strictly objects first" am Beginn eines Anfängerkurses unterrichtet werden kann. Sie stützt sich dabei auf die Erkenntnis von Joseph Bergin [Ber00], dass der Umstieg von prozeduralen Denkmustern auf objektorientierte sehr schwierig sein kann. Diese Sichtweise bestätigen auch Hu, Stroustrup und Decker:

> "It was said (Stroustrup, 1994) that it takes an average programmer 6 to 18 months to switch his/her mindset from a procedural to an object-oriented view of the world, which has been often cited to justify teaching objects-first." [Hu04, 213]

> "The results of the midterm exam are extremely encouraging, in that they support the idea that objects should be introduced early and emphasized after their introduction. The results also show that previous experience with procedural programming does not positively affect performance in object-oriented programming" [Dec03, 245]

Schließlich werden obige Aussagen durch die Zusammenfassung einer Diskussion im März 2004 etwas relativiert:

> "We identify the cognitive claims made in the email discussion and find there is not a consensus in the research literature as to whether the objects first approach or the imperative approach is harder to learn." [Lis06, 146]

> "There is a fairly strong consensus that programming is hard both to teach and to learn, but the case that objects-early is harder (or easier) than objects-late has not yet been made conclusively." [Lis06, 150]

3.3 Vorteile des Lernansatzes „Objects first"

Aus den „Pedagogical Patterns" von Joseph Bergin [Ber07] ergibt sich, dass wichtige Lerninhalte möglichst früh gelernt und öfters wiederholt werden sollen. Bei Kommentaren bezüglich des Ansatzes „Objects first" wird meist hervorgehoben, dass besonders die Problemlösungskompetenz der Studenten gesteigert wird:

> "The advantages of this approach to teaching novices are both numerous and tangible. First, introducing the object-oriented paradigm from the beginning allows us to exploit it as a design medium. Second doing so puts the procedural paradigm (along with the ideas of top-down design and stepwise refinement) into a meaningful and useful problem-solving context. 'Third it eliminates (at least for the student!) the dreaded "paradigm shift" from procedural programming to object-oriented programming. Finally, and most importantly, it helps students to develop their problem solving skills in conjunction with their programming ones." [DH94, 55]

Außerdem berichten Decker und Hirshfield, dass es Studenten leichter fiel, komplexere Problemstellungen mittels Objektorientierung anzugehen. Dies mag auch deshalb sein, da wie oben dargestellt, die Objektorientierung einer abstrakteren und strukturierteren Herangehensweise bedarf.

> „[…] Decker and Hirschfield[!] (1994) reported that students felt more comfortable attacking a complex problem from scratch with object-oriented programming than with Pascal." [Lis06, 149]

Grieß unterscheidet in [Gri08, 33] Aspekte von Programmiersprachen in einen strukturellen/organisatorischen Aspekt und einen algorithmischen/prozeduralen. Er findet es wichtig diese Unterscheidung zu diskutieren und klarzustellen, dass zuerst der strukturelle Aspekt behandelt wird. Er kommt zum Schluss:

> "We believe that the principles discussed in this paper lead to the conclusion that OO first is the only way to go. But don't go this way without paying sufficient attention to: (1) the programmingskill aspect, (2) a model of objects and classes to which students can relate, (3) notation and terminology, (4) the order in which topics are introduced, and (5) an IDE that does not require an application." [Gri08, 35]

3.4 Probleme mit „Objects first"

Besonders, wenn mathematische Algorithmen abgebildet werden sollen, sehen viele Autoren Probleme bei einer objektorientierten Implementierung. Sie verweisen darauf, dass Studenten zuerst einmal das algorithmische Denken lernen sollten:

> "It is almost impossible to separate programming issues completely from mathematical issues, and in these days of falling math standards the level of math skills (or lack thereof) that an instructor may assume is a large factor in course design." [BB03, 111]

> "When a real-world problem is not naturally OO-represented, programmers often have to twist the structure in order to map those nonmathematical objects to algebraic-type-based classes, thus much of what can be observed as OO Programming is just a complex form of "encapsulation", which is not an OO approach (Ledgard, 2001)." [HU04, 212]

Außerdem bestünde eine natürliche Abfolge des Lernprozesses der Programmierparadigmen, die dadurch begründet wird, dass die Objektorientierung eine Erweiterung des prozeduralen Programmierens sei.

> "Students should only be asked to learn one paradigm at a time, and there is clearly a natural order for teaching PP and OOP." [BB03, 111]

> "The basis of the argument is that considered as a paradigm, Object Oriented Programming comes in addition to the Procedural Programming paradigm and not as a replacement for it." [BB03, 111]

4. Umsetzungsmöglichkeiten

Um Objekte den Studenten und Schülern beizubringen gibt es in der Zwischenzeit eine beachtliche Anzahl von Werkzeugen. Diese stellen eine vereinfachte Entwicklungsumgebung zur Verfügung. Die meisten bieten einen spielerischen Zugang zur Programmierung.

> „To help with „teaching objects first", educators […] have developed software for visualizating objects and teaching material using visual objects […]. Do they work? The answer depends on who you ask." [HU08, 20]

Welche Werkzeuge besser geeignet sind Objektorientierung zu lernen muss jeder selber entscheiden. Dies hängt letztendlich auch vom jeweiligen Lehrer ab. Die nun folgenden Werkzeuge stellen lediglich einen Ausschnitt dar. Sie sind allerdings je, die weiter verbreitet sind und im Unterricht schon erprobt.

4.1 „Objects first" mit Karel the Robot

Becker vorschlägt die Möglichkeit vor, mit Karel the Robot, Schüler und Studenten in die objektorientierte Welt einzuführen:

> "We have found a third way to be most satisfying for both teachers and students: using interesting predefined classes to introduce the fundamentals of object-oriented programming (object instantiation, method calls, inheritance) followed quickly by the traditional fundamentals of iteration and selection, also taught using the same predefined classes." [Bec01, 50]

Becker beschreibt hierzu den Ablauf des Kurses wie folgt [Bec01, 52]:

- Woche 1: „Instantiating and Using Objects"
- Woche 2: „Extending Existing Classes"
- Woche 3: „Selection and Iteration"
- Woche 4: „Methods with Parameter"
- Woche 5: „Instance Variables"

Nach Becker [Bec01, 53] liegen die Vorteile klar auf der Hand: Die Studenten lernen objekt-orientierte Grundlagen von Beginn an. Bewegen sich die Roboter falsch, wissen die Studenten, dass bei ihrem Programm etwas nicht stimmt. Roboter bedeuten Spaß.

Auch Buck et al. [Buc01] verwenden ein Tool, welches auf Karel the Robot aufbaut und führen damit Studenten in die Objektorientierung ein.

4.2 „Objects first“ mit BlueJ

Eine weitere Möglichkeit bietet BlueJ [BlueJ]. Mithilfe der Beispiele, people und people2, die mit der Standardinstallation mitgeliefert werden, kann sehr gut der Polymorphismus gezeigt werden. Die Schüler oder Studenten haben zwei Klassen (Staff und Student), welche von der Klasse Person abgeleitet sind. Personen können bei einer einem Objekt der Klasse Database hinzugefügt werden.

Kölling, Miterfinder von BlueJ, beschreibt in Kapitel zwei von [Köl03, 1-3], dass viele Entwicklungsumgebungen einem guten Softwareentwicklungsunterricht nicht entsprechen. Deshalb hat er mit anderen BlueJ entwickelt. Es stellt eine Entwicklungsumgebung zur Verfügung, in der die Schüler und Studenten zunächst keinen Code schreiben müssen und trotzdem mit Klassen und Objekten hantieren können.

> “BlueJ offers a unique mechanism of direct parameterised method calls. This mechanism allows teachers to delay the introduction of other interface technologies such as text based interfaces, GUIs or applets” [Köl03, 3]

4.3 „Objects first“ mit Alice

Ein recht ausgereiftes Konzept bietet Alice [alice]. Dies ist eine Entwicklungsumgebung, wo mittels Drag ‚n’ Drop virtuelle Tiere, Personen und Fabelwesen programmiert werden können. Diese befinden sich in einer 3D-Welt (siehe Abbildung 2). Die Schüler und Studenten können so sofort die Handlungen und Bewegungen sehen.

Abbildung 2 Startszene in einer Welt in Alice [Coo03, 192]

Cooper et al [Coo03] erforschten die Erfolgsrate unter Studenten, welche mit und ohne Alice programmieren lernten. Sie stellten fest, dass jene Studenten, welche Alice benutzten:

> „The results show that the 11 students who took the Alice-based course did better in CS1 than the total group, and significantly better than the 10 students who were of a similar background." [Coo03, 2:5]

Torben Lorenzen und Abdul Sattar beschreiben in [LS08] einen von ihnen entwickelten Kursablauf vor, der Studenten objektorientiertes Programmieren mit Alice beibringt. In nur sechs Stunden erklären sie ihren Studenten die Grundbegriffe der Objektorientierung.

Durch die Programmierung mittels Drag ‚n' Drop schreiben jedoch beide, dass es Probleme bei der Umstellung auf eine „gewöhnliche" objektorientierte Programmiersprache gab: Die Studenten lernen keine Syntaxfehler. Sie lernten jedoch schnell mit diesen Fehlern umzugehen.

Alice stellt somit ein gutes Hilfsmittel dar, Klassen und Objekte nach einer Einführung mittels BlueJ, die objektorientierte Programmierung zu vertiefen.

5. Ausblick

In der Fachdidaktik gibt es nicht nur eine Diskussion ob zuerst prozedural oder gleich zu Beginn objektorientiert programmiert werden soll. Einige Autoren bereichern die Diskussion mit weiterführenden Ansätzen. Zwei davon sollen hier angeführt werden.

5.1 Entwurfsmuster zuerst

Rudolf Pecinovský und sein Team gehen noch einen Schritt weiter und vertreten einen Ansatz, wo als erstes Entwurfsmuster gelehrt werden:

> "The first mentioned rule of OOP is to program against the interface and not against the implementation. […]The second key rule is to use design patterns wherever it is feasible. We have also felt the need to introduce this concept as early as possible in the lecture course. Currently, we teach the design patterns to our students already at the end of the first lecture; even before their first experience with the code! […]These experiences have demonstrated that if we modify the Objects first approach into the Design Patterns First we will improve the results in teaching of OOP paradigm." [Pec06, 188]

In ihrem Artikel beschreiben Sie, wie die ersten fünf Einheiten ihres Kurses für Softwareentwicklung aussehen. Bereits nach diesen fünf Einheiten kennen die Studenten mehrere Entwurfsmuster und lernten mit Schnittstellen (Interfaces) umzugehen. Da sie den Kurs auch an Jugendlichen zwischen 12 und 16 Jahren testeten, konnten sie zeigen, dass junge Jugendliche bereits mit diesen Inhalten umgehen können. Ihre Folgerung:

> "We have verified in practice that using this methodology we can teach even young people from an age as low as 12 years. These very young students perceive the matter very quickly; mostly with fewer conceptual problems than experienced professional

> programmers, because they don't need to incorporate the new information into the present knowledge and habits […]." [Pec06, 191]

5.2 Datentypen

Chenglie Hu fordert eine Besinnung auf die Wurzeln. Hu plädiert dafür, dass die Datentypen in den Mittelpunkt gerückt werden:

> „Yet the study of data types is fundamentally important to learning programming with a typed programming language." [Hu08, 19]

Somit könnten nach Hu's Interpretation die Diskussion, wann Objekte eingeführt werden sollen, ad acta gelegt werden:

> "A data type characterizes how a set of entities is internally represented and algorithmically manipulated. […] A class (as in an object-oriented programming language like Java) defines a data type. […] Having learned primitive data types and standalone methods when covering ADTs, novices typically experience a drastic change in the level of abstraction (an ADT is essentially a mathematical algebra) they were unprepared for while learning primitive data types." [Hu08, 19]

6. Zusammenfassung

In den Artikeln über „Objects first" zeigt sich für mich eine klare historische Entwicklung. Konnten in den achziger und neunziger Jahren noch wenige Autoren mit dem Ansatz etwas anfangen, dass Objekte in Programmierkursen zuerst behandelt werden sollen, so hat sich das im neuen Jahrtausend weitgehend verändert. Einer der Gründe hierfür wird wohl gewesen sein, dass auch die Professoren mit der Objektorientierung noch nicht so vertraut waren, wie sie es heute sind.

Eine weitere Entwicklung stellen jene Werkzeuge dar, mit denen Studenten und Schülern programmieren anhand von grafischen Oberflächen lernen (siehe Kapitel 4). Diese vereinfachen die Handhabung mit Syntax,… so weit, dass sich der Lehrende auf die wichtigen Konzepte der Objektorientierung konzentrieren kann.

Soweit ich das nun anhand der Recherche beurteilen kann, gibt es kaum mehr Gründe, nicht mit Objekten einen Programmierkurs zu beginnen. „Objects First" ist eines von drei wichtigen Konzepten [Coo03, 191] für die Einführung in die Software-Entwicklung. Deshalb ist es auch legitim, dieses Konzept zu wählen. Wichtig ist meines Erachtens nur, dass zu Beginn mit Hilfe von verschiedenen Werkzeugen, wie Karel the Robot, BlueJ, Alice,… die Konzepte den Studenten oder Schülern nähergebracht werden, bevor sie selbständig Code schreiben.

Literaturverzeichnis

[Alice] Alice: http://www.alice.org, Abruf 6.1.2009

[BB03] Burton, P. J.; Bruhn, R. E.: Teaching Programming in the OOP Era. In ACM SIGCSE Bulletin, Volume 35, Issue 2, June, 2003.

[Bec01] Becker, B. W.: Teaching CS1 with Karel the Robot in Java. From Technical Symposium on Computer Science Education, Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, February, 2001, Charlotte, North Carolina, United States.

[Ber00] Bergin, J.: Why Procedural is the Wrong First Paradigm if OOP is the Goal http://csis.pace.edu/~bergin/papers/Whynotproceduralfirst.html, Abruf: 3.1.2009

[Ber07] Bergin, J,: Fourteen Pedagogical Patterns, http://csis.pace.edu/~bergin/PedPat1.3.html, Abruf: 3.1.2009

[BlueJ] BlueJ: http://www.bluej.org, Abruf 6.1.2009

[Buc01] Buck, D. et al.: JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum. In SIGCSE 2001 2/01 Charlotte, NC, USA

[Coo03] Cooper, S. et al.: Teaching Objects-first In Introductory Computer Science. In ITiCSE'03, February 19-23, Reno, Nevada, USA.

[Dec03] Decker, A.: A tale of two paradigms. In Journal of Computing Sciences in Colleges Volumne 19, Issuse: Eastern Conference, 2003

[Die07] Diethelm I.: "Strictly models and objects first" - Unterrichtskonzept und -methodik für objektorientierte Modellierung im Informatikunterricht, 2007, Universtät Kassel

[DH94] Decker, R.; Hirshfield, S.: The Top 10 Reasons Why Object-Oriented Programming Can't Be Taught in CS 1. From Technical Symposium on Computer Science Education, Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, March, 1994, Phoenix, Arizona, United States.

[Gri08] Gries, D.: A Principled Approach to Teaching OO First. In SIGCSE'08, March 12-15, 2008, Portland, Oregon, USA.

[Hu04] Hu, C.: Rethinking of Teaching Objects-First. In Education and Information Technologies, Volume 9, Issue 3, September, 2004.

[Hu08] Hu, C.: Just Say 'A Class Defines a Data Type'. In Communications of the ACM, Volume 51, Issue 3, March 2008.

[Köl03] Kölling, M., et al.: The BlueJ system and its pedagogy. In the Journal of Computer Science Education, Special Issue on Learning and Teaching Object Technology, Volume 13, No 4, Dec 2003.

[Lie88] Lieberherr, K. et al.: Object-oriented programming: an objective sense of style. From the Conference on Object Oriented Programming Systems Languages and Applications, San Diego, California, United States, 1988

[Lis06] Lister, R. et al.: Research perspectives on the objects-early debate. In ITiCSE'06, June 26-28, 2006, Bologna, Italy.

[LS08] Lorenzen T.; Sattar A.: Objects First Using Alice to Introduce Object Constructs in CS1. In SIGCSE'08, June, 2008.

[Pec06] Pecinovský R., et al.: Let's Modify the Objects-First Approach into Design-Patterns-First. In ITiCSE'06, June 26-28, 2006, Bologna, Italy.

[Pok89] Pokkunuri, B. P.: Object Oriented Programming. In ACM SIGPLAN Notices, Volume 24 , Issue 11, November, 1989.

[ST07] Sanders, K.; Thomas L.: Checklists for Grading Object-Oriented CS1 Programs: concepts and misconceptions. In ITiCSE'07, June 23-27, 2007, Dundee, Scotland, United Kingdom.

[WikiA] Wikipedia: Geschichte der Programmierung,
http://de.wikipedia.org/wiki/Geschichte_der_Programmiersprachen, Abruf: 2.1.2009