Ira Diethelm(Hrsg.): INFOS 2017 – Informatische Bildung zum Verstehen und Gestalten,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 1

Modularisierung im Informatikunterricht aus lernpsychologischer Perspektive

Johannes Fischer, Arno Pasternak[1]

Abstract: Objektorientiertes Modellieren und Programmieren ist eine weitverbreitete Technik, um informatische Prinzipien wie Abstraktion, Automation und Modularisierung umzusetzen. Letztgenannte Begriffe gehören zweifelsohne zu den wichtigsten Konzepten des *Informatischen Denkens* nach Jeannette Wing [Wi06], und ihre Vermittlung ist aus einem allgemeinbildenden Informatikunterricht nicht wegzudenken. Unklar ist nur der zu beschreitende Weg, um diese Kompetenzen zu erlangen. Kölling und Rosenberg [KR01] geben hierfür einige Ratschläge, die sich insbesondere durch die Verwendung *großer* Projekte von Anfang an auszeichnen. In diesem Beitrag wird dafür plädiert, statt einer konkreten Technik wie der Objektorientierung die Konzepte Abstraktion, Automation und insbesondere Modularisierung in den Fokus zu nehmen und diese altersgerecht mit einer geeigneten Sprache zu unterrichten.

Keywords: Informatikunterricht, OOM, OOP, Module, Lernpsychologie, Cognitve Load Theory

1 Informatische Bildung an Schulen

Heymann hat 1996 überzeugend dargelegt, was unter *Allgemeinbildung* heute verstanden werden soll [He96]. Entsprechend ist der Allgemeinbildungscharakter der Informatik unter Fachdidaktikern unbestritten [Wi03, SS11, Pa13]. Diese Bedeutung wird zusätzlich durch die Darstellung von *Jeannette Wing* deutlich, die 2006 in einem kurzen Beitrag die Besonderheit des *informatischen Denkens* dargestellt hat [Wi06].

Es versteht sich dabei von selbst, dass in der Informatik im Schulunterricht als Allgemeinbildung in erster Linie grundlegende Strukturen und Konzepte anstatt die Nutzung bestimmter aktueller Werkzeuge in Form konkreter Informatiksysteme wie beispielsweise eine Textverarbeitung oder eine konkrete Datenbankimplementierung vermittelt werden sollen. Ebenso gilt, dass auch bei der Verwendung informatikinterner Werkzeuge der Schwerpunkt auf die Vermittlung von Konzepten und Prinzipien und nicht auf beispielsweise die Anwendung einer oder mehrerer Programmiersprachen und mit denen in diesen Sprachen realisierbaren Ideen gelegt werden muss.

1.1 Konzepte der Informatik

Welches sind die grundlegenden Konzepte der Informatik? Ausgehend von ihren ersten Feststellungen schreibt Wing 2008 [Wi08]: *„Abstraction and automation“* und etwas später in anderer Formulierung:

[1] TU Dortmund, Fakultät für Informatik, Otto-Hahn-Str. 14, 44227 Dortmund,
{Johannes.Fischer,Arno.Pasternak}@cs.tu-dortmund.de

„Computing[3] is the automation of our abstractions“. Diese kurze und knappe Beschreibung verlangt natürlich nach einer genaueren Begriffsanalyse. Erst die Kombination aus *abstrakter* Betrachtung und Modellbildung der realen Welt und deren Umsetzung in eine durch Maschinen *ausführbare* Implementierung macht den besonderen Charakter der Informatik aus und hat durch die dafür notwendigen geistigen Vorstellungen und Bemühungen den intellektuellen Horizont der Menschen wesentlich erweitert und durch die dadurch erzeugten Produkte die Welt real verändert.

Der Kerngedanke der Aussagen von Wing lässt sich als Konzept folgendermaßen beschreiben: Durch *Abstraktion* wird als Abbild der realen Welt ein Modell erschaffen. Dies stellt aber erst eine *statische* Beschreibung der (Modell-)Welt dar. Erst durch die Beschreibung von *abstrakten Handlungen* wird ermöglicht, dass bei Vorhandensein einer entsprechenden Implementierung ein Ergebnis erzielt werden kann, das (hoffentlich) bei der Lösung eines Problems hilft. Durch das Ausführen dieser Implementierung wird aus dem statischen ein *dynamisches* Modell.

Dieses dynamische Modell wird auf einer (konkreten) Maschine ausgeführt. Diese Maschine „versteht“ leider keine natürlichen Sprachen. Die künstlichen Sprachen der Maschinen sind eine Anpassung an technische Gegebenheiten und stellen eine eigene intellektuelle Struktur dar. Diese Programmiersprachen müssen erlernt und durch Übungen gefestigt werden. Es ist eine intellektuell hoch zu wertende Leistung, ein Modell in einer konkreten Programmiersprache zu „kodieren“ und dieses Programm zu testen und zu korrigieren. Bei der Umsetzung eines Modelles in eine derartige Sprache sind gerade sogenannte *Novizen* erheblich intellektuell gefordert. Entsprechend ist der Aufwand nicht zu unterschätzen, der im Unterricht beim Erlernen einer Programmiersprache nötig ist. Dieser hohe Aufwand darf aber nicht dazu führen, auf die Automatisierung durch Programmierung im Unterricht zu verzichten.

Die beiden Aspekte des *informatischen Denkens — Abstraktion und Automation —* müssen in einem Schulfach Informatik (mit didaktischer Reduktion) gleichermaßen abgebildet werden, wenn es seinem Charakter gerecht werden will. Betrachten wir also die beiden Aspekte etwas genauer.

2 Abstraktion

Um ein Problem der realen Welt mit einer Maschine — dem Computer — zu bearbeiten, muss das Problem in seine Bestandteile zerlegt und strukturiert analysiert werden. Daraus folgt im Endeffekt eine Modularisierung und eine strukturierte Programmierung.

Diese *Modellierung* zu einer reduzierten abstrakten Welt erhalten wir durch Entwicklung von *Modulen*. Im *Informatik-Duden* werden derartige Module beschrieben als:

- „Er[4] ist logisch oder funktional in sich abgeschlossen.

[3] Mit *Computing* ist in der englischen Fachsprache *Informatik* gemeint.
[4] gemeint ist hier: das Modul als *der* Baustein

- Wie er arbeitet oder implementiert ist, braucht außen nicht bekannt zu sein [...].
- Er besitzt klar definierte Schnittstellen nach außen.

[...] Ein System ist *modular* aufgebaut, wenn es aus abgrenzbaren Einheiten zusammengesetzt ist und wenn diese Einheiten einzeln ausgetauscht, verändert oder hinzugefügt werden können, ohne dass andere Teile des Systems hierdurch beeinflusst werden oder das System arbeitsunfähig wird." [CS03, S.414]

Abstraktion der Daten Diese Definition macht deutlich, wie zentral die Idee der Modularisierung in der Informatik ist. Beschreibt ein derartiges Modul Daten aus der Realität, so sprechen wir von einer *abstrakten Datenstruktur* oder einem *abstrakten Datentyp* [Po84, S.85ff,S.154ff][5] Da die Beschreibung von Daten eine zentrale Aufgabe im Rahmen der Modellierung ist, ist das Erlernen der Entwicklung von derartigen Modulen ein wichtiger Teil im Rahmen von Lehrplänen für das Schulfach Informatik.

Abstraktion der Abläufe Eine systematische Abstraktion der Abläufe führt zur *strukturierten Programmierung*, die im Informatikduden beschrieben wird als: „Programmiermethode, bei der das vorgegebene Problem in Teilprobleme und die Beziehungen zwischen diesen Teilproblemen(Schnittstellen) zerlegt wird. [...]" [CS03, S.643] Je nach Programmierparadigma werden bei der imperativen Programmierung *Kontrollstrukturen* oder bei der funktionalen Programmierung *rekursive Funktionen und Prozeduren* verwendet.

Abstraktion im Unterricht Beide Formen der Abstraktion müssen im Informatikunterricht gleichermaßen für eine Problemlösung vermittelt werden. Eine ausschließliche Abstraktion der Daten führt zu einer Beschreibung der statischen Welt, vergleichbar mit dem Kunstunterrricht, eine ausschließliche Abstraktion der Abläufe führt de fakto zu einer Variante des Mathematikunterrichtes. *Schubert* und *Schwill* machen daher bei ihrer beispielhaften Vorstellung des objektorientierten Vorgehens deutlich: „Es soll deutlich werden, dass die traditionellen, grundlegenden Konzepte der strukturierten Programmierung unverzichtbarer Lerngegenstand sind, auch wenn objektorientierte Lösungen von den Schülern konstruiert werden. Folgende Teilziele werden angestrebt: Die Schüler können das Konzept der Modularisierung anwenden. ..." [SS11, S.157] Sie berufen sich dabei auf *Böszörményi*, der richtigerweise feststellt, dass „das Konzept der Modularisierung viel grundlegender [ist] als das Konzept der Objektorientierung" [Bö01, S.15]. Folgt man *Wirth*, handelt es sich dabei noch nicht einmal um ein neues, eigenständiges Konzept, sondern nur um eine neue Technik: „Noch ein Wort zur Objektorientierung: [...] Man lernt neue Programmiertechniken – aber auf dem Boden des bisher Mitgebrachten." [Wi91, S.60,61]

Objektorientierte Technologie im Unterricht In der Ausbildung werden heute häufig objektorientierte Sprachen verwendet. Dass modulare Modellierung und Programmierung Objektorientierung nicht voraussetzt, wird damit den Lernenden nicht immer deutlich. Wir haben beobachtet, dass selbst Informatik-Studierende nur die objektorientierte Darstellung kennen und daher glauben, dass modulares Programmieren mit Objektorientierung gleichzusetzen ist. So verwundert es nicht, dass dieser Eindruck auch in einem Schulbuch vertreten wird:

[5] Eine *abstrakte Datenstruktur* definiert nur ein Exemplar, ein *abstrakter Datentyp* definiert eine Menge von gleichen abstrakten Datenstrukturen. [Po84, S.100]

„Vor allem in Zeiten des Internets und der Verbreitung von Software in allen Lebensbereichen bietet die OOP große Vorteile:

- Wiederverwendbarkeit von schon programmierten Elementen
- Aufteilung überschaubarer Einzelteile
- Erweiterung durch Schnittstellen“ [KT10, S.15]

Diese unvollständige fachliche Beschränkung der Modularisierung auf Objektorientierung verführt leicht zu fragwürdigen didaktischen Entscheidungen [6].

3 Automation

Aus der abstrakten Beschreibung von Daten und Abläufen soll ein Programm in einer konkreten Programmiersprache erstellt werden, das auf einer Maschine ablauffähig ist. Die Erstellung eines solchen Programmes ist aber mitnichten einfach oder trivial. Die gewählte Programmiersprache stellt entsprechend der Syntax bestimmte grundsätzliche Datentypen und Ablaufstrukturen als Bausteine zur Verfügung. Um vom abstrakten Modell zum konkreten Programm zu gelangen, muss ein *Graben* zwischen den abstrakten Ideen und den konkreten Strukturen der jeweiligen Programmiersprache überwunden werden.

Der Modellierungsgraben Dieser Graben wird am Beispiel des Kernlehrplanes für die gymnasiale Oberstufe in NRW [MS13, S.21] deutlich, wenn die (verkürzte) Aufzählung um die Darstellung dieses Grabens erweitert wird:

„Die Schülerinnen und Schüler
• ermitteln bei der Analyse einfacher Problemstellungen Objekte, ihre Eigenschaften, ihre Operationen und ihre Beziehungen (M), • modellieren Klassen mit ihren Attributen, ihren Methoden und Assoziationsbeziehungen (M),
GRABEN
• ordnen Attributen, Parametern und Rückgaben von Methoden einfache Datentypen, Objekttypen oder lineare Datensammlungen zu (M)“

Beispielhaft ist es für Schüler überhaupt nicht selbstverständlich, das Sortieren einer Menge von Karten mit einem Feld oder einer Liste zu modellieren. Die Darstellung in den meisten Schulbüchern suggeriert allerdings diese scheinbare Selbstverständlichkeit, die von vielen Lehrern auch ohne große didaktische Reflexion den Schülern übergestülpt wird. Beeinflusst der Lehrer die Schülerinnen und Schüler nicht, so modellieren diese ohne Kenntnis der Datenstruktur Feld oder Liste wahrscheinlich einzelne Karten.

Ebenso wie in der obigen Darstellung für die Daten angegeben, ist auch ein entsprechender Graben für die algorithmische Abstraktion zur Programmiersprache vorhanden. Jeder Lehrer weiß, wie lange es dauert, bis die Schülerinnen und Schüler beispielsweise die Iteration über ein Feld als geeignetes programmiersprachliches Konstrukt einer Problemstellung erkennen und auch anwenden können.

[6] In der Ausgabe von 2014 ist die obige Darstellung so nicht mehr enthalten [KL14, S.20]

Ein Graben kann überwunden werden, wenn man von der einen Seite auf die andere Seite springen kann. Dieses muss aber erlernt werden. Dabei ist ein Sprung über einen Graben von der einen Seite zumeist genauso kompliziert wie von der anderen Seite. Das bedeutet hier: Wenn es mir als Schüler gelingt, die Wirklichkeit zu modellieren, muss ich zusätzlich lernen, wie diese Modelle in der gewählten Programmiersprache abgebildet werden können. Kann ich mich zuerst in dieser Programmiersprache für kleine Problemstellungen adäquat bewegen, muss ich zusätzlich lernen, die Wirklichkeit in diese mir bekannten Strukturen „herunterzubrechen."

Unterrichtliches Vorgehen Notwendig sind also Kenntnisse der Datenrepräsentation und Ablaufsteuerung. Beide Aspekte müssen erlernt und geübt werden. Welcher davon im unterrichtlichen Verlauf sinnvollerweise wann im Vordergrund steht, ist nicht offensichtlich. Diese Einschätzung wird durch die Untersuchung von *Ehlert* gestützt, der in seiner Studie verglichen hat, ob eine objektorientierte Sichtweise gebenüber der prozedural-imperativen Vorgehensweise im Anfangsunterricht Vorteile hat: „Es kann das Vorgehen gewählt werden, welches der Lehrer bzw. Dozent für das richtige hält." [Eh12, S.218]

4 Erfahrungen aus der Schulpraxis

In den letzten Jahren hat sich in der Schule die Sprache *Java* gegenüber beispielsweise *(Objekt-)Pascal* durchgesetzt. Java *verlangt* das Arbeiten mit Klassen. Die Technik der Objektorientierung und damit auch die Sprache Java ist zweifelsohne für die Implementierung großer Programmsysteme sehr gut geeignet. Daraus kann abgeleitet werden, auch in der Schule mit diesen Ideen und Techniken der Praxis zu beginnen. Entsprechend formulieren Schubert und Schwill richtigerweise: „Die Einführung in die Informatik kann mit objektorientiertem Modellieren (OOM) und Programmieren (OOP) [...] beginnen. Zwingend ist das nicht." [SS11, S.157] Eine derartige Entscheidung hat allerdings auch Nachteile. Klassen repräsentieren *Datenkapseln*. „Programmsysteme, die mit Datenkapseln arbeiten, enthalten deshalb fast immer mehr Prozeduren als konventionelle Programmsysteme." [Po84, S.99] Gerade bei kleineren Programmen, die im Anfangsunterricht vermehrt auftreten, stellen diese oft einen *Overhead* dar.

OO-Guidelines Diese Grundproblematik objektorientierter Systeme und zusätzlich speziell für Anfänger schwer durchschaubare syntaktische Strukturen der Sprache *Java* verleiten Kölling und Rosenberg zu Vorschlägen, wie ein Anfangsunterricht mit der Sprache Java erfolgversprechend sein könnte [KR01]. Der relative Overhead sei kleiner, je größer das System ist. Also folgern sie, daß der Unterricht mit einem großen System beginnen sollte.

Einige dieser von Kölling und Rosenberg empfohlenen *guidelines* beziehen sich auf die Ideen der Objektorientierung (z.B. *„use 'large' projects", „objects first"*), andere auf die Problematik der konkret verwendeten Sprache Java (z.B. *„don't start with 'main'", „be careful with the user interface"*). Es werden damit zwei völlig unterschiedliche didaktische Fragestellungen in diesen Vorschlägen vermischt.

Umsetzung im Unterricht Wir haben uns gefragt, ob die sprachspezifischen Aspekte von Java tatsächlich das Erlernen der Objektorientierung erschweren und haben in einem kleinen Unterrichtsprojekt von ca. 10 Stunden eine erste dahingehende Analyse versucht. In einem 12. Jahrgang einer Gesamtschule wurden zwei etwa gleich große Kurse eingerichtet (je ca. 15 Schüler). In einem Kurs wurde mit Java unterrichtet, in dem anderen Kurs mit der Scriptsprache *Groovy*[7], die die sprachspezifischen Probleme von Java teilweise umgehen lässt. Wie weiter unten beschrieben wird, erwiesen sich hier die Sprachunterschiede aber nicht als problematisch.

In beiden Kursen wurde ein Schülerverwaltungsprogramm in der jeweiligen Sprache mit ca. 500 Zeilen verwendet. Entsprechend den Vorstellungen von Kölling und Rosenberg waren in diesem Programm mehrere Klassen enthalten, zumindest teilweise wurden mehrere Instanzen dieser Klassen verwendet. Die Schüler haben diesen Programmtext gelesen, getestet, kleine Veränderungen vorgenommen, neue Instanzen erzeugt und auch einzelne Operationen zielgerichtet verändert.

Erwartungen Erwartet hatten wir, dass sich die Schülerinnen und Schüler in der Groovy-Gruppe aufgrund der einfacheren Syntax leichter tun würden. Dies spielte allerdings in dieser Phase kaum eine Rolle, da das Programm als Ganzes vorlag und nur an einzelnen Stellen, die sich in Java und Groovy textuell kaum unterschieden, verändert wurde. Allerdings zeigte sich bei der Analyse eine andere, viel bedeutsamere Problematik.

Das Ziel des Ansatzes von Kölling und Rosenberg ist es, dass die Lernenden durch das explorative Erarbeiten eines nicht zu kleinen Programmes die Vorteile des objektorientierten Vorgehens selbst erkennen und die Begrifflichkeiten anhand des Beispieles deutlich werden. Um dies zu überprüfen, fragten wir zu Beginn des Schuljahres und nach dieser Unterrichtsphase schriftlich die Schüler, was sie sich unter Begriffen wie beispielsweise *Klasse* und *objektorientierter Programmierung* vorstellen. Es muss dabei angemerkt werden, dass die Begriffe im Lauf des Unterrichts bei der Analyse selbstverständlich erläutert, erarbeitet und auch verschriftlicht wurden. Daher gingen wir davon aus, dass bei diesen rein reproduktiven Fragen zumindest die erlernten Definitionenen angegeben werden würden.

Erste Ergebnissse Das Ergebnis hat uns völlig überrascht. Nicht eine einzige Antwort in beiden Gruppen war zielführend. Eine Anwort wie *„Eine Klasse wird mit Attributen gefüllt“* kann dabei noch als Ansatz einer korrekten Idee angesehen werden, aber Antworten wie *„Das ist das Programmieren mit Hilfe von Datenbanken“* oder *„Unter objektorientierter Programmierung kann man mit speziellen Objekten programmieren“* waren häufiger. Entsprechend der von uns gestellten Ansprüche an den Lernzuwachs der Schülerinnen und Schüler war das Ergebnis vernichtend. Die Ideen des Konzepts der Modularisierung – hier mit der Technik der Objektorientierung realisiert – waren offensichtlich nicht erlernt und verstanden worden, sodass selbst eine einfache reproduktive Wiedergabe der Begriffe von den Schülern nicht geleistet werden konnte.

Da keine Vergleichsdaten aller Schüler wie in dieser Befragung von früheren Kursen mit Java bzw. Groovy mit einem anderen didaktischen Vorgehen vorliegen, ist es nicht ausge-

[7] `http://www.groovy-lang.org`, letzter Zugriff: 6.1.2017

schlossen, dass auch in der Vergangenheit keine besseren Ergebnisse erzielt worden sind. Dagegen spricht allerdings folgende Beobachtung: In der folgenden Klausur, an der allerdings nur sechs der insgesamt 28 Schülerinnen und Schüler teilgenommen haben, wurden wie in vergleichbaren Klausuren der vergangenen Jahre diese Begriffe erneut erfragt. Diese Aufgabe ist normalerweise als reproduktive Aufgabe ein „Punktelieferant“ für die Klausurteilnehmer. Bis auf eine Lösung waren die Antworten wieder nicht ausreichend.

Die Kurse in den früheren Jahren sind didaktisch mit deutlich kleineren Beispielen an die Thematik herangeführt worden. Die Antworten der Schüler, die vor einem Jahr an der Klausur teilgenommen haben, waren deutlich besser.

Da wir immerhin noch auf 8 der damals geschriebenen 22 Klausuren Zugriff hatten, konnten wir die Antworten an diesen Klausuren verifizieren. Zusätzlich liegen ähnliche Daten aus einem Kurs des 11.(!) Jahrganges aus dem Jahre 2010 vor. Dieser Kurs hatte allerdings ein etwas anderes Curriculum (und wurde in (Objekt-)Pascal unterrichtet). Die Schülerantworten stellten den Lehrer im Gegensatz zu den Antworten der beiden Kurse im aktuellen Jahr im Wesentlichen zufrieden.

Spekulativer Vergleich dieser Ergebnisse Es handelt sich bei diesen Zahlen nicht um eine systematische Untersuchung. Die Ergebnisse und der Vergleich mit früheren Kursen zeigen trotzdem, dass die Schülerinnen und Schüler beim Vorgehen nach Kölling und Rosenberg die Modularisierung und deren konkrete Umsetzung mit der Objektorientierung nicht verstanden haben. Selbst die unter Groovy niedrigere Syntax-Hürde konnte dies nicht verhindern. Die Schülerinnen und Schüler aus den Kursen vor einem Jahr und dem Jahre 2010 konnten die Begriffe der OOP im Wesentlichen wiedergeben.

Aufgrund der schwierigen Vergleichbarkeit der Materialien und der unterschiedlichen Kursverläufe dürfen diese Ergebnisse nicht überbewertet werden. Deutlich wird aber, dass dringender Bedarf nach empirischen Untersuchungen besteht.

5 Lernpsychologische Sicht

Die aus fachlicher Sicht überzeugende Position von Kölling und Rosenberg hatte uns angeregt, in diesem Schuljahr nach dieser Konzeption vorzugehen. Das Ergebnis war ernüchternd. Es erscheint uns naheliegend, dass hier lernpsychologische Gründe für das Scheitern verantwortlich sind.

In den letzten ca. zwanzig Jahren hat es erhebliche Fortschritte in der Lernpsychologie gegeben. Mit diesen ist es möglich, die Funktionsweise des Lernens im menschlichen Gehirn zu verstehen. Eine wesentliche Theorie stellt die *Cognitive Load Theorie (CLT)* [PRS03] dar, die in inzwischen vielfältigen Untersuchungen bestätigt wurde. Nach dieser Theorie lässt sich die Arbeitsweise des Gehirns als das Zusammenspiel zwischen einem *Arbeitsgedächtnis* und einem *Langzeitgedächtnis* vorstellen. Das nur gering belastbare Arbeitsgedächtnis nutzt dabei das Langzeitgedächtnis als Wissens- und Kompetenzspeicher. Es lassen sich aber nur wenige neue Eindrücke mit diesem Speicher gleichzeitig verarbeiten. Daher ist es sinnvoll, in einen Lernprozess zu einem bestimmten Zeitpunkt nur we-

nige neue Elemente einzugliedern. Die empirischen Untersuchungen von *Hattie* [HBZ13] und die neurowissenschaftlichen Begründungen von ihm und *Yates* bestätigen dies auf eindrucksvolle Weise [Ha15, S.107ff]. Für die Informatik bedeutet dies, dass gerade im Anfangsunterricht das *bottom-up*-Prinzip oft dem *top-down*-Prinzip vorzuziehen ist.

Mit diesen neurowissenschaftlichen Überlegungen können die Theorien von *Piaget* und *van der Hiele* aus der Mitte des letzten Jahrhunderts begründet werden [PI93, HHG78]. Entsprechend dieser Erkenntnisse hat die Gruppe der *Neo-Piagisten* wie beispielsweise *Lister* [Li16] diese Theorien weiterentwickelt. Für den Informatikunterricht speziell folgert *Lister* [Li16, S.9ff] ein intensives Arbeiten an und mit dem Quellcode eines Programmes. Entsprechendes wurde auch einige Jahre zuvor von *Shaffer, Doub* und *Tuovinen* aus der CLT-Theorie gefolgert [Sh03]. Eigene Untersuchungen aus dem Anfangsunterricht mit einer Programmiersprache bestätigen dies [Pa16].

Nach der CLT-Theorie ist das aus Sicht der Objektorientierung sinnvolle Vorgehen mit dem Erarbeiten eines größeren Softwareprojektes als Einführung in die Modellierung und Programmierung eine Überlast des lernenden Gehirns und muss daher im Allgemeinen scheitern. Es ist daher nach einem anderen Vorgehen zu suchen.

6 Eine mögliche Alternative

Datenabstraktion bleibt das Ziel. Die objektorientierte Programmierung ist eine Möglichkeit, die offensichtlich im Unterricht der allgemeinbildenden Schulen nicht auf jedem Weg — wie gezeigt — zum Ziel führt. Auch wenn die OOM und die OOP als programmiertechnisches Vorgehen angestebt wird, ist es sinnvoll, den Weg dorthin anders zu gestalten. Das lässt sich nur mit Programmiersprachen realisieren, die verschiedene Umsetzungen der Datenabstraktion ermöglichen. Gut ist es, wenn die verschiedenen Techniken in der Darstellung sich nicht groß unterscheiden, sodass ein Wechsel der Darstellung aus didaktischer Sicht keine unnötigen syntaktischen Schwierigkeiten bereitet. Hierzu sind allerdings noch empirische Untersuchungen nötig.

Module und Klassen in einer geeigneten Sprache Im Folgenden wird ein Beispiel bezüglich der Datenabstraktion in der Programmiersprache `Tcl` [Fl12] vorgestellt. Es geht hier nicht um diese konkrete Sprache, sondern um das Prinzip. In `Tcl` sind alle Formen der Datenabstraktion möglich. Die Nutzung der entsprechenden Techniken weist in der Darstellung kaum Unterschiede auf.

Als Beispiel diene das Einfügen in *binäre Bäume*. Ohne Nutzung von Modulen bzw. mit importierten Modulen lautet der Aufruf: `fuege_ein mein_baum $info` . Bei der Nutzung von Klassen ändert sich beim Aufruf `mein_baum fuege_ein $info` nur die Reihenfolge in der Anweisung. Bei Verwendung einer abstrakten Datenstruktur statt eines abstrakten Datentypes vereinfacht sich die Anweisung sogar noch: `fuege_ein $info` .

Die fast gleichartige Darstellung bei der Verwendung von Modulen oder Klassen für abstrakte Typen macht deutlich, dass das Denken in diesen Strukturen sich programmiersprachlich ohne große Brüche in der jeweils gewünschten Form umsetzen lässt. Das

ermöglicht die Umsetzung einer abstrakten Struktur in die technische Form einer Programmiersprache, die zu dem unterrichtlichen Zeitpunkt entsprechend dem Stand der Schüler angemessen ist und kann eine Überforderung an technischem und begrifflichem Overhead im Anfangsunterricht verhindern helfen. Auf diese Weise können die Schülerinnen und Schüler sanft ohne intellektuelle Überforderung von der Strukturierung der Abläufe mit Prozeduren über die Gestaltung von Modulen bis zur objektorientierter Programmierung geführt werden und damit die Gedanken und Ideen der Modularisierung auf vielfältige Weise erfahren und erlernen. Das erfordert allerdings die Auswahl einer Programmiersprache für den Unterricht, die verschiedene Darstellungsmöglichkeiten gleichberechtigt ermöglicht.

Mit einem derartigen Vorgehen wird auch einem weiteren wichtigen Prinzip gefolgt, das nicht nur, aber auch in der Informatik Gültigkeit besitzt, das *Kiss-Prinzip* [Ri95]: *Keep it small and simple*. Eine Hoffnung aus der Anwendung dieses Prinzip in der Modellierung und vor allem in der Umsetzung in eine konkrete Programmiersprache ist, die Hürden für die Schüler so niedrig wie möglich zu setzen, sodass diese auch außerhalb des Unterrichts und im späteren Leben für sich selbst nicht ausschließen, ein konkretes Problem durch ein informatisches Vorgehen selber programmieren und lösen zu wollen.

7 Zusammenfassung

Ein wesentliches „Denkwerkzeug“ der Informatik ist die *Modularisierung*, die mit verschiedenen Techniken durchgeführt werden kann. Eine davon ist die *objektorientierte Modellierung und Programmierung*. Diese hat allerdings oft einen erheblichen Overhead zur Folge.

Bei einem Vergleich von Schülerleistungen verschiedener Jahrgänge zeigte sich, dass vor ein Vorgehen nach den „guidelines“ von Kölling und Rosenberg vielen Schülerinnen und Schülern große Schwierigkeiten macht. Es ist daher zu überlegen, ob es zweckmäßig ist, den Schulunterricht nicht von Beginn an mit objektorientierten Sprachen zu beginnen. Im Rahmen der didaktischen Reduktion könnte ein Vorgehen, dass den Aufbau der Elemente der Modularisierung stufenweise ermöglicht, geeigneter sein. Dieser Weg realisiert die Ideen, wie sie von *Kortenkamp u.a.* unter dem Begriff *genetischer Unterricht* [Ko09] beschrieben sind. Ein solches Vorgehen verlangt eine didaktisch begründete Auswahl der Programmiersprache, mit der diese Ideen sukzessive im Unterricht vermittelt werden.

Literaturverzeichnis

[Bö01] Böszörményi, László: Java für Anfänger? LOG IN, (1/2001):14–19, 2001.

[CS03] Claus, Volker; Schwill, Andreas: Duden Informatik. Dudenverlag, Mannheim, 2003.

[Eh12] Ehlert, Albrecht: Empirische Studie: Unterschiede im Lernerfolg und Unterschiede im subjektiven Erleben des Unterrichts von Schülerinnen und Schülern im Informatik-Anfangsunterricht (11. Klasse Berufliches Gymnasium) in Abhängigkeit von der zeitlichen Reihenfolge der Themen (OOP-First und OOP-Later) . Dissertation, Freie Universität Berlin, Berlin, 2012.

[Fl12] Flynt, Clif: Tcl/Tk: A Developer's Guide. The Morgan Kaufmann Series in Software Engineering and Programming. Amsterdam, 2012.

[Ha15] Hattie, John; Yates, Gregory C.R.; Beywl, Wolfgang; Zierer, Klaus: Lernen sichtbar machen aus psychologischer Perspektive. Schneider Verlag GmbH, Baltmannsweiler, 2015.

[HBZ13] Hattie, John; Beywl, Wolfgang; Zierer, Klaus: Lernen sichtbar machen. Schneider Verlag GmbH, Baltmannsweiler, 2013.

[He96] Heymann, Hans Werner: Allgemeinbildung und Mathematik. Studien zur Schulpädagogik und Didaktik ; 13. Beltz, Weinheim [u.a.], 1996.

[HHG78] Hiele, Pierre Marie van; Hiele-Geldorf, Dina van: Die Bedeutung der Denkebenen im Unterrichtssystem nach der deduktiven Methode. Wege der Forschung. Wissenschaftliche Buchgesellschaft, Darmstadt, 1978.

[KL14] Kempe, Thomas; Löhr, Annika: Informatik 1. Schöningh, Paderborn, 2014.

[Ko09] Kortenkamp, Ulrich; Modrow, Eckart; Oldenburg, Reinhard; Poloczek, Jürgen; Rabel, Magnus: Objektorientierte Modellierung - aber wann und wie? LOG IN, (160/161):41–47, 2009.

[KR01] Kölling, Michael; Rosenberg, John: Guidelines for Teaching Object Orientation with Java. In: Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education. ITiCSE '01, ACM, New York, NY, USA, S. 33–36, 2001.

[KT10] Kempe, Thomas; Tepaße, David: Informatik 1 * Softwareentwicklung mit Greenfoot und BlueJ. Schöningh, Paderborn, 2010.

[Li16] Lister, Raymond: Toward a Developmental Epistemology of Computer Programming. In: Proceedings of the 11th Workshop in Primary and Secondary Computing Education. WiPSCE 2016, ACM, New York, NY, USA, S. 5–16, 2016.

[MS13] MSW NRW: Kernlehrplan für die Sekundarstufe II, Gymnasium/Gesamtschule in Nordrhein-Westfalen, Informatik. Düsseldorf, 2013.

[Pa13] Pasternak, Arno: Fach- und bildungswissenschaftliche Grundlagen für den Informatikunterricht in der Sekundarstufe I. Dissertation, Westfälische Wilhelms-Universität, Münster, 2013.

[Pa16] Pasternak, Arno: Contextualized Teaching in the Lower Secondary EducationLong-term Evaluation of a CS Course from Grade 6 to 10. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education. SIGCSE '16, ACM, New York, NY, USA, S. 657–662, 2016.

[PI93] Piaget, Jean; Inhelder, Bärbel: Die Psychologie des Kindes. dtv-Taschenbücher dialog und praxis. Dt. Taschenbuch-Verlag, Stuttgart, 1993.

[Po84] Pomberger, Gustav: Softwaretechnik und Modula 2. Hanser Fachbuchverlag, München, 1984.

[PRS03] Paas, Fred; Renkl, Alexander; Sweller, John: Cognitive Load Theory and Instructional Design: Recent Developments. Educational Psychologist, 38(1):1–4, 2003.

[Ri95] Rich, Ben R.: Clarence Leonard (Kelly) Johnson, A Biographical Memoir. National Academies Press, Washington DC, USA, 1995.

[Sh03] Shaffer, Dale; Doube, Wendy; Tuovinen, Juhani; Sturt, Charles; Wagga, Wagga; Wales, New South: In (Petre, M.; Budgen, D., Hrsg.): Proceedings of the Joint Conference EASE & PPIG 2003. 2003.

[SS11] Schubert, Sigrid; Schwill, Andreas: Didaktik der Informatik. Spektrum Akademischer Verlag, Heidelberg, 2. Auflage, 2011.

[Wi91] Wirth, Niklaus: Die Dinge beim Namen nennen. In: Erstes European Software Festival. Chip – Vogel-Verlag, Würzburg, S. 60–64, 1991.

[Wi03] Witten, Helmut: Allgemeinbildender Informatikunterricht? Ein neuer Blick auf H. W. Heymanns Aufgaben allgemeinbildender Schulen. INFOS 2003, GI, Bonn, S. 59–75, 2003.

[Wi06] Wing, Jeannette M.: Computational Thinking. Commun. of the ACM, 49(3):33–35, 2006.

[Wi08] Wing, Jeannette M.: Computational thinking and thinking about computing. Philosophical Transactions of The Royal Society, (366):3717–3725, 2008.