Objektorientierte Modellierung – aber wann und wie?

Zur Bedeutung der OOM im Informatikunterricht

von Ulrich Kortenkamp, Eckart Modrow, Reinhard Oldenburg, Jürgen Poloczek und Magnus Rabel

Objektorientierte Modellierung (OOM) ist ein wichtiges Konzept der Fachinformatik und wird vonseiten der Didaktik der Informatik derzeit für den Unterricht in beiden Sekundarstufen stark empfohlen. So wird in den *Bildungsstandards für die Sekundarstufe I* gefordert, dass alle(!) Schüler und Schülerinnen der Jahrgangsstufen 8 bis 10 „für einfache Sachverhalte objektorientierte Modelle [entwickeln] und diese mit Klassendiagrammen dar[stellen]“ (AKBSI, 2008, S.47). Die Gründe für die Behandlung der OOM in der Schule sind einleuchtend; es stellt sich aber die Frage, wie früh und wie intensiv sie eine Rolle spielen sollte. Etliche der hier zusammengetragenen Argumente sind nicht neu (siehe z.B. Hartmann, 2005), sollen aber noch einmal bewusst gemacht werden, um eine Abwägung zu ermöglichen.

Gründe für die Behandlung der OOM in der Schule

In der didaktischen Literatur findet man verstreut eine Vielzahl von Gründen, die die Behandlung der OOM in der Schule legitimieren. Diese lassen sich in verschiedene Bereiche einteilen, deren Bedeutung im Folgenden teilweise gleich kommentiert wird.

Übergreifende Argumente

- OOM/OOP unterstützt die Forderung nach „mächtigen Konzepten wie Erweiterbarkeit, Anpassbarkeit, Rekonfigurierbarkeit, Vererbbarkeit, Kapselung, evolutionäre Softwareentwicklung sowie Wünsche nach einer stärkeren Anwendungsorientierung durch Betonung der Nutzung des Computers anstelle eines vertieften Verständnisses seiner Funktionsweise“ (Schwill, 1995).

Argumente aus der Fachinformatik

Die Hauptargumente sind:

- OOM hat sich in der Praxis bewährt, sie ist etablierter Bestand des Faches. In diesem Sinne schreibt Spolwig (1999, S.38/40): „Der Siegeszug von OOP ist in den letzten zehn Jahren längst von Forschung, Lehre und Softwareindustrie entschieden.“
- OOM dient der Studienvorbereitung.
- Es ist mit OOM möglich, ein authentisches Bild der Informatik zu vermitteln.

Für allgemeinbildende Schulen sind Argumente aus Studien- und Berufsanforderungen nur bedingt entscheidend. Schüler und Schülerinnen, die nach einem Informatikkurs ein informatikfreies Studium oder einen informatikfreien Beruf anstreben, sind nicht bedauerliche Aussteiger, sondern der Normalfall – und auch in diesem Fall sollte die in die Informatik investierte Zeit gut angelegt sein. Da allgemeinbildende Schulen auch der Berufsorientierung der Schüler dienen, ist ein authentisches Bild der Informatik in der Schule wichtig, da es gegebenenfalls auch eine begründete Entscheidung *gegen* ein Informatikstudium ermöglicht. Hierzu kann OOM (ebenso wie etwa theoretische Informatik) einen Beitrag leisten.

Lerntheoretische Argumente

Zwei wesentliche Argumente sind:

- OOM hilft nicht nur bei der Konstruktion von Software, sondern auch beim Verständnis fertiger Software.

▷ „Objektorientiertes Modellieren [könnte] vernetztes Denken trainieren […], denn die interagierenden Objekte sollen ein benutzbares Softwaresystem ergeben, so dass eine sinnvolle Vernetzung der Objekte hergestellt werden muss“ (Schulte, 2001).

Psychologische Argumente

Im Mittelpunkt stehen kognitionspsychologische Begründungen:

▷ Objektorientierung kommt der menschlichen Wahrnehmung entgegen (vgl. Schwill, 1993, S. 44 f.).
▷ Mit OOP lassen sich nämlich „auf informatischer Ebene Dinge beschreiben […], wie sie uns auch in der täglichen Wahrnehmung und im Umgang mit ihnen erscheinen“ (Spolwig, 1999, S. 40).

Allgemeinbildungsargumente

Hubwieser (^{3}2007, S. 85) spricht der Modellbildung eine „immense Bedeutung für die Allgemeinbildung“ zu; zur Begründung verweist er auf die Literatur. Mögliche Argumente sind:

▷ Fähigkeiten im Strukturieren sind in vielen menschlichen Handlungsfeldern wichtig.
▷ Objektorientierung kann helfen, Informatikunterricht so zu öffnen, dass „sowohl die kognitiven Lernziele (Denkfähigkeiten), die informatischen Lernziele (Fragestellungen und Werkzeuge, Methoden der Softwaretechnik) und der Aspekt der Wechselwirkungen zwischen Informatik und Gesellschaft profitieren können“ (Schulte, 2001).
▷ Die Rolle der Modelle wird in der OOM besonders deutlich.

Die Arbeit mit Modellen ist keine Besonderheit der Informatik, aber die Informatik ist besonders reich an verschiedenen Modelltypen und kann damit das Bewusstsein für die Modellverwendung im besonderen Maße stärken. Innerhalb der Informatik stellt die OOM aber nur eine der Modellierungsformen dar.

Probleme bei der Behandlung der OOM

Die vielfältigen Begründungen der OOM in der Schule werden nun im einzelnen einer Kritik unterzogen.

Gegenargumente zur fachwissenschaftlichen Begründung

Zunächst geht es um die tatsächlich vorfindbare Wirklichkeit bei professionellen Software-Projekten:

▷ OOP und OOM sind in der Praxis vor allem deshalb wichtig, weil sie die Arbeit an komplexen Softwaresystemen unterstützen. Die in der Schule realisierten Projekte sind aber in der Regel zu klein, um diesen Vorteil deutlich erfahrbar werden zu lassen.

„Es ist wichtig anzumerken, dass die Vorzüge einer objektorientierten Systementwicklung sich erst bei der Lösung größerer Aufgaben einstellen. Kleine Spielprobleme können nur bestimmte Begriffe veranschaulichen. Um die Möglichkeiten eines Sprachstils zu verinnerlichen, ist die Realisierung größerer Projekte nötig“ (Eirund, 1993, S. 40).

In diesem Sinne argumentiert auch Heubaum (2004) am Beispiel einer graphentheoretischen Aufgabe (Wegsuche in einem Schloss), bei deren Lösung OOM nach Heubaum nur zu einer unnötigen Aufblähung führen würde. Dem haben Diethelm/Geiger/Zündorf (2005) widersprochen. Allerdings erfordert allein die skizzenhafte Beschreibung der von diesen Autoren favorisierten objektorientierten Behandlung sieben Druckseiten. Damit dürfte Heubaums Argument eher bestätigt denn widerlegt sein. Eine Modellierung der o. a. Wegsuche mit Listen in PYTHON kommt im Übrigen – platzgreifend notiert – mit 25 Zeilen (einschließlich Kommentaren) aus, der eigentliche Suchalgorithmus ist acht Zeilen lang und mit Zeilen wie *for raum in nachbarraeume* (hier) auch leicht verständlich.

▷ Es besteht keine Möglichkeit, ein „Softwarepraktikum“ durchzuführen, wie es in vielen Studiengängen verankert ist; die Zentralisierung des Abiturs hat die Sachlage nochmals verschärft.
▷ In der Einschätzung von Fachwissenschaftlern an universitären Informatikinstituten sind die Begriffe der OOM deutlich weniger wichtig als die Begriffe der klassischen Algorithmik, wie Zendler/Spannagel (2008) empirisch nachgewiesen haben.
▷ OOP ist kein fundamentales Konzept, denn es lässt sich auf andere reduzieren. In Sprachen mit *Closures* (Funktionsumgebungen) kann ein vollwertiges Klassensystem „aus dem Stand“ in wenigen Programmzeilen realisiert werden. Das bedeutet, dass Modellbildungsprozesse statt mit den Mitteln der Objektorientierung auch mit funktionalen Mitteln erreicht werden können. Ebenso ist es möglich, prozedurale Sprachen nachträglich mit einem Objekt/Klassen-System auszustatten. Dies ist nicht nur eine theoretische Finesse, sondern wird – wie z. B. die Sprache OBJECTIVE-C zeigt – praktiziert.
▷ Es herrscht keine allgemeine Einigkeit, welche Eigenschaften ein Klassensystem haben sollte (etwa *single-dispatch* oder *multiple-dispatch*). Dergleichen technische Fragen sind aber schon auf der Modellierungsebene wichtig, weil ein Modellierer sein Modell immer im Hinblick auf dessen Realisierbarkeit gestaltet.
▷ Hesse/v. Braun (2001) haben darauf hingewiesen, dass die Sichtweise von OOM-Protagonisten, nach der es lediglich eines Übersetzungsvorgangs bedarf, um von einer Realsituation zu einer Modellierung zu kommen, zu kurz greift, weil sie wesentliche Freiheiten, aber auch Schwierigkeiten beim Modellierungsprozess auslässt.

Genetische Wege zu OOP und OOM

Hinzufügen von Eigenschaften, wenn es nötig und nützlich ist

von Ulrich Kortenkamp

Objektorientierung ist besonders mächtig, wenn für zunächst heterogene Daten (im Verbund, engl.: *record*) eine gemeinsame Eigenschaft identifiziert wird, die aber unterschiedliche Implementierungen benötigt. In Sprachen, deren Klassendefinitionen – wie beispielsweise in RUBY – offen sind, ist es einfach, diesen Schritt von *Typen* zu *Klassen* dann zu machen, wenn er nötig wird. Wir führen es hier vereinfacht am Beispiel der *Kodierungslänge von Daten* vor.
Die *Kodierungslänge* einer natürlichen Zahl n, d.h. die Anzahl der Binärziffern von n, ist durch den abgerundeten Logarithmus zur Basis 2 plus 1 gegeben. In RUBY (hier dargestellt in der interaktiven Umgebung *irb*) kann man die Klasse *Fixnum* öffnen und eine Methode *size* hinzufügen:

```
$ irb
>> class Fixnum
>> def size
>> (Math.log(abs)/Math.log(2)).floor + 1
>> end
>> end
=> nil
```

Diese Methode steht dann unmittelbar für alle Zahlobjekte zur Verfügung:

```
>> 5.size
=> 3
>> 8.size
=> 3
>> 9.size
=> 4
```

Noch einfacher kann man die Kodierungslänge von Zeichenketten der Klasse *String* hinzufügen:

```
>> class String
>> def size
>> 8 * length
>> end
>> end
=> nil
>> "Hallo".size
=> 40
```

Warum war das gut? Wir können jetzt ganz einfach die Kodierungslänge auch von Listen bestimmen. Dazu fehlt nur noch die Definition der Methode *size* für die Klasse *Array*, hier in Kurzform:

```
>> class Array; def size; inject(0)
        {|s, e| s += e.size }; end; end
```

Und damit funktioniert problemlos die Bestimmung der Kodierungslänge von Listen, die Zeichenketten, ganze Zahlen größer 0 oder Listen derselben enthalten:

```
>> [2,4,"123123",[123123,"23123"]].size
=> 110
```

Das heißt: Modellieren ist kein Selbstzweck. In der Schule sollen (und können!) auch lauffähige Programme erstellt werden. Die Bildungsstandards äußern sich dazu unmissverständlich: „Beim informatischen Modellieren ist die Implementierung unverzichtbar, um das Ergebnis der Modellbildung erlebbar zu machen“ (AKBSI, 2008, S. 46).

Dies (wie auch die entsprechenden Aussagen in den Lehrplänen) macht es nötig, Grundkonzepte der Programmierung zu thematisieren, damit überhaupt den Objekten Funktionalität zukommen kann. Dabei steht man vor folgenden Schwierigkeiten:

- ▷ Bei einer objektorientierten Betrachtung der Grundkonzepte sowie einfacher Algorithmen, die sehr wohl zum Verständnis beitragen, würde „mit Kanonen auf Spatzen geschossen“; der daraus entstehende programmtechnische „Wasserkopf“ wirkte sich eher kontraproduktiv aus.
- ▷ Eine isolierte Behandlung des Themas bewirkt einen Bruch im Unterricht.

Gegenargumente aus der Lerntheorie

- ▷ Die Behandlung von OOM ohne OOP steht vor einem Problem, das dem der Mengenlehre in der „Neuen Mathematik“ in den Siebzigerjahren des vorigen Jahrhunderts ähnelt: Die Schülerinnen und Schüler lernen neue Wörter zur Beschreibung von Dingen, die sie schon kennen; sie erschließen sich damit aber keine neuen Möglichkeiten, etwas zu tun oder etwas zu verstehen. Natürlich sind „Klassifizieren“ und „Strukturieren“ wesentliche Lernziele, diese müssen aber inhaltlich relevant sein.

Relevant sind beispielsweise bei der Klassifikation der Fauna in der Biologie ihre Fragen zur Klassifikation von Walen oder Schnabeltieren oder in der Mathematik die Klassifikation von Vierecken. Letztere zeigt übrigens, dass die Klassifikation in der Mathematik dynamischer ist als in der OOM: Quadrat ist eine Unterklasse von Rechteck. Wenn bei einem Rechteck aber im Lauf eines mathematischen Argumentationsprozesses klar wird, dass beide Seitenlängen gleich sind, wird es mental sofort zum Quadrat und darf auch als solches verwendet werden. Ein Objekt einer Klasse Rechteck in einer OOP ist normalerweise nicht so dynamisch. „Normalerweise“ heißt hier: in der unterrichtlichen Praxis, unter anderem durch die Auswahl der Modellierungssprache oder -werkzeuge bedingt.

- ▷ Viele Berichte zeigen, dass Schüler Probleme mit der OOM haben. Relativ problemlos ist die Verwendung fertiger Klassen, um Objekte zu erzeugen und dann Methoden aufzurufen (Objekte bilden dann eine Lernumgebung, mit OOM im engeren Sinne hat das aber nichts zu tun); die eigentliche Modellierung bereitet aber Schülern zahlreiche Probleme. Die Gründe dafür sind offensichtlich: Die Lernenden verfügen zu diesem Zeitpunkt meist weder in der Modellierung noch in der Implementierung über ausreichende Kenntnisse und Erfahrungen. Da das Ziel des Modellierungsprozesses sich also noch im Diffusen

versteckt, kann auch kaum ein klarer zielgerichteter Analyseprozess von ihnen erwartet werden. Auch für viele Lehrwerke gilt: Die Modellierung „fällt vom Himmel“ und wird nicht erarbeitet, sondern vorgegeben.

▷ Bei vielen Beziehungen ist nicht klar, welche Mittel der OOM angemessen sind. Dies stellt die Schüler vor Schwierigkeiten, die über die intrinsische Problemkomplexität hinausgehen.

Ein Beispiel hierzu: Die ganzen Zahlen $\mathbb{Z}$ sind eine Teilmenge der rationalen Zahlen $\mathbb{Q}$. Dieser Sachverhalt könnte durch zwei eigenständige Klassen realisiert werden, wobei die Klasse *Q* ein Prädikat „*inZ*“ besitzt. Man könnte aber auch $\mathbb{Z}$ als Unterklasse von $\mathbb{Q}$ (weil Spezialisierung) oder $\mathbb{Q}$ als Unterklasse von $\mathbb{Z}$ (weil der Körper mehr Methoden als der Ring hat) modellieren. Sich zwischen diesen Möglichkeiten zu entscheiden, stellt eine erhebliche, für Schülerinnen und Schüler kaum passierbare, kognitive Hürde dar.

Grundsätzlich entspricht die Überzeugung, man könne die richtige Klassenstruktur einfach von einer realen Situation „ablesen“, einer Abbildtheorie, wie sie in der Philosophie wohl zuletzt von Wittgenstein in seinem *Tractatus* vertreten wurde, und die heute auf breiter Front von Kognitivisten bis zu Konstruktivisten abgelehnt wird. Es erscheint deshalb sinnvoll, das Problem der Entwicklung einer Klassenhierarchie recht spät anzugehen und dabei verschiedene Modellierungskonzepte und Modellierungen gegeneinander abzuwägen.

Die Abbildtheorie führt insbesondere zur problematischen Tendenz der Einengung auf das vermeintlich einzig Richtige. Für die Mathematikdidaktik hat Hans Freudenthal seinerzeit zu bedenken gegeben: „Daß es mehrere Winkelbegriffe gibt, ist schon früher zur Sprache gekommen. Manche Didaktiker wollen uns davon überzeugen, daß nur einer der richtige sei. Ordnungsliebe ist lobenswert, aber sie sollte nicht so weit gehen, daß man wichtige Begriffe verbietet, weil sie nicht ins System passen“ (Freudenthal, 1973. S. 438). Ähnlich liegen die Dinge in der Informatikdidaktik.

Gegenargumente aus der Psychologie

▷ Während mentale Modelle von Objekten leicht gebildet werden, mangelt es vielen Schülerinnen und Schülern an – im Sinne des Konstruktivismus – „viablen“ mentalen Modellen zu Klassensystemen. Eine Analyse von Lehrbüchern zeigt, dass es wenig explizite Anregung von Modellvorstellungen zu Klassen gibt (bis auf die Stempel- und Fabrikmetapher).

▷ Die Motivation, etwas zu beschreiben, ist in aller Regel geringer als die, etwas zu tun. (Dies bedeutet natürlich nicht, dass im Unterricht nichts beschrieben werden sollte; aber das Beschreiben sollte nicht über das Tun dominieren.)

Allgemeinbildungsargumente

▷ Strukturieren und Planen sind ohne Zweifel wesentliche allgemeinbildende Kompetenzen. Da es aber eine Wechselbeziehung zwischen modelliertem Gegenstand und Modellierungswerkzeugen gibt, müssen auch diese in einem allgemeinbildenden Unterricht in den Blick genommen werden. Im Idealfall geschieht dies, indem Strukturierungstechniken situationsadäquat von den Schülerinnen und Schülern entwickelt werden, wie dies den Ideen des genetischen Unterrichts entspricht.

Mentale Stolperfallen

In dem Bemühen, sinnvolle Anwendungen für OOM im Bereich der Schule zu finden, wurden viele – darunter viele gute – Vorschläge gemacht. Man findet aber auch immer wieder Vorschläge, die Stolperfallen beinhalten.

▷ *Über-Modellierung:* Die Feinheit von Modellen kann nahezu beliebig gesteigert werden; dies kann zu einer extrem hohen Anzahl an Klassen führen.

Ein Beispiel: Das Gesellschaftsspiel *Fang den Hut* könnte man so modellieren, dass es eine Klasse *Brett* gibt. Diese hat vier Startplätze und das eigentliche Spielfeld. Das Spielfeld hat viele Felder. *Felder* ist die Oberklasse für normale Felder und Ruhefelder und so weiter. Der Effekt dieser Vorgehensweise ist, dass man aus dem Modellieren nicht mehr herauskommt, d. h. das naheliegende Ziel, ein Programm zum Spielen zu schreiben, läuft Gefahr, aus dem Auge verloren zu werden. Auch in der Praxis der Softwareentwicklung hat die Ausdifferenzierung der Klassen ihre Grenzen (vgl. z. B. Kortenkamp, 1999).

▷ *Computerartefakt:* Spolwig (2004) hat deutlich darauf hingewiesen, dass es merkwürdig ist, wenn eine Ampel so modelliert wird, dass sie einen Mal- oder Zeichenstift benötigt, um sich rot zu färben.

▷ *Pseudolösungen:* Ein Verständnis der OOM kann bei der Arbeit mit Standardanwendungen helfen. Die rekonstruierende Modellierung, die unterstellt, die Buchstaben eines Textes seien Objekte der Klasse *Zeichen* mit Attributen wie *Schriftart*, *Zeichengröße*, *Stilattribut* usw., ist aber realitätsfern, denn eine solche Implementation würde unangemessen viel Speicherplatz verbrauchen und wäre in ihrer Komplexität fehleranfällig. Das Beispiel zeigt, dass es möglich ist, auch Software, die ohne OOM/OOP konstruiert wurde, durch OO-Remodellierung zu deuten. Es ist aber eine offene didaktische Frage, nach welchen Kriterien beurteilt werden kann, wann dies sinnvoll ist.

Die bisherige Zusammenstellung von Argumenten macht ein Dilemma deutlich: Einerseits gibt es wichtige Gründe für die OOM in der Schule (auch wenn einige davon relativiert wurden), andererseits gibt es eine Reihe von Schwierigkeiten, die teilweise der Situation der Schule, teilweise dem Ansatz als solchem geschuldet sind. Diese Zusammenfassung der Problemlage ist nicht neu, aber wichtig als Grundlage für die Überlegungen zu einem konstruktiven Umgang mit den Problemen.

Schüler und OOM

Der Informatikunterricht muss zwischen seinen Zielen und den Interessen der Schülerinnen und Schüler vermitteln. Man hat sich zu fragen, ob OOM dies erleichtert oder erschwert. Bei einer Befragung von Schülern und Studierenden (Rabel/Oldenburg, 2009) ergab sich, dass Schüler die Konzepte der OOM von denen der klassischen Algorithmik getrennt sehen und anders bewerten. In der genannten Befragung haben Schüler außerdem eine Vielzahl von Projekten genannt, die sie gerne durchführen würden, u.a. 3-D-Grafik, Spiele-Programmierung u.v.m. Nur bei einem kleinen Teil davon stellt aber OOM eine wesentliche Hilfe dar. Wenn weniger Zeit auf OOM verwendet würde, könnte man also mehr von dem realisieren, was Schülerinnen und Schüler wünschen. Das ist kein zwingendes Argument gegen OOM; es ist aber festzuhalten, dass es einen Konkurrenzkampf um Unterrichtszeit gibt, in dem Dinge wie UML und Computergrafik als Konkurrenten auftreten.

Natürlich sollte die Orientierung an Schülerinteressen nicht soweit gehen, dass man nur noch „Ballerspiele“ baut; die Interessen können aber so gelenkt werden, dass die Lernenden sich gerne mit attraktiven und „sozial akzeptablen“ Problemen beschäftigen (Robotik, Grafik, Simulationen etc.). Ein Schulfach muss seine Lernenden „erreichen“, wenn es nachhaltige Lernerfolge erzielen will; dies gilt umso mehr, als es sich weitgehend um ein Wahlfach handelt. Die Schule muss sich der Grenzen der Erziehbarkeit bewusst sein und daher den Unterricht so gestalten, dass die Interessen der Lernenden mit denen des Fachs in Einklang gebracht werden, und nicht in einen Gegensatz (in der allgemeinen Didaktik übrigens eine Selbstverständlichkeit).

Genetischer Unterricht zur OOM – eine Alternative?

Bisher wurden viele Kritikpunkte zur OOM zusammengetragen. Gleichwohl bleiben die Argumente dafür bestehen, und für ein Hineinwachsen in die Kultur der Informatik ist OOM unerlässlich. Es ist eine wesentliche Aufgabe der Didaktik der Informatik, dieses Dilemma zu entschärfen.

Eine Möglichkeit dazu könnte ein *genetischer Weg* in die Objektorientierung sein. Bei einem solchen Zugang beginnt man früh mit der Benutzung von Objekten fertiger Klassen; die OOM, insbesondere das Erstellen eigener Klassen und ihrer Beziehungen zueinander, werden aber zunächst zurückgestellt. Stattdessen zielt der Unterricht darauf ab, die zentralen klassischen Themen der Algorithmik gründlich und sinnstiftend zu behandeln. Dabei ist nachrangig, ob das mit textbasierter Programmierung oder mit einem grafischen System wie *SCRATCH* oder *Etoys* erfolgt. Der Unterricht konzentriert sich darauf, dass die Schülerinnen und Schüler schöne und für sie attraktive Dinge produzieren. Geeignete Themen sind Spiele, Bildverarbeitung, Grafik, Simulationen (z.B. Schwarm-Simulationen in der Biologie), Audio- und Sprachverarbeitung, Arbeit mit Webcams usw.

Vor allem in der Sekundarstufe I wollen Kinder aus unterschiedlichen Bausteinen etwas bauen. Ob das LE-

Genetische Wege zu OOP und OOM

Von Listen zu Objekten

von Reinhard Oldenburg

Modellieren ist eine offene Tätigkeit, d.h. die Ergebnisse können ganz unterschiedlich aussehen, und es stellt sich dann die Frage, welches Modell besser, welches schlechter ist.
Beispiel: Die Daten eines Kassenzettels sollen gespeichert und verarbeitet werden.

Modell 1: Artikel und Preise werden abwechselnd notiert, am Ende steht die Summe:

```
meinkassenzettel1 =
  ["Brot", 1.98, "H-Milch 1,5%", 0.55, 2.53]
```

Modell 2: Artikel und Preis bilden Unterlisten:

```
meinkassenzettel2 =
  [["Brot", 1.98], ["H-Milch 1,5%", 0.55]]
```

Nachteil der bisherigen Formen ist, dass der Kauf von drei Milchpackungen zu drei Nennungen führt.
Besser ist vielleicht *Modell 3*:

```
meinkassenzettel3 =
  [["Brot", 1.98, 1], ["H-Milch 1,5%", 0.55, 3]]
```

Welche Vor-, welche Nachteile hat das in Bezug auf Konsistenz und Aufwand? Welche Operationen sind mit Kassenzetteln sinnvoll?
Eine Gesamtpreisfunktion für die verschiedenen Typen ist leicht zu schreiben; es entstehen aber Probleme, wenn man (in einer Buchhaltung) Kassenzettel aller drei Typen bearbeiten will.
Eine Lösung ist, als 0-tes Element bei allen Listen eine Zeichenkette für den Typ zu speichern:

```
meinkassenzettel4 =
  ["ZettelTyp4", ["Brot", 1.98, 1],
                 ["H-Milch 1,5%", 0.55, 3]]

def summe(zettel): # Für Zettel vom Typ 4
  if not(zettel[0] == "ZettelTyp4"):
  raise Exception("summe benötigt
                   Typ 4 Zettel")
     ... # Summe berechnen
```

Das sind fast schon Klassen. Was man jetzt noch braucht:

- ▷ Eine Funktion, die solche Listen erzeugt (Konstruktor).
- ▷ Die Funktionen zur Bearbeitung werden auch in der Liste gespeichert (Methoden).

Genetische Wege zu OOP und OOM

Simulationen als Einstieg in die Objektorientierung

von Magnus Rabel

Die Veröffentlichung der Programmiersprache SIMULA (1967) wird im Allgemeinen als Ursprung der objektorientierten Programmierung gesehen. Wie der Name schon andeutet, wurde die Sprache für die Simulation physikalischer Prozesse entwickelt; sie ist der Vorgänger von SMALLTALK. Eine ähnliche (genetische) Vorgehensweise böte sich für den Informatikunterricht an.

Der Einsatz einer Sprache für die agentenbasierte Simulation biologischer oder sozialer Phänomene wie NETLOGO (http://ccl.northwestern.edu/netlogo/) macht es in der Sekundarstufe I möglich, spannende Naturphänomene wie *Schwarmverhalten* auf sehr einfache Weise zu programmieren. In vergangenen Unterrichtsgängen wurde jedoch vonseiten der Schülerinnen und Schüler hin und wieder die Kritik laut, dass sie gerne eine „erwachsenere" und vielseitigere Programmiersprache verwenden würden.

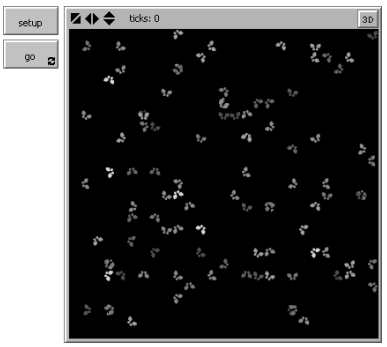

Bild 1: Schwarmsimulation mit NETLOGO.

Da liegt ein erneutes Aufgreifen des Themas Schwarmsimulation nahe, um objektorientiertes Denken zu schulen. Die einzelnen Agenten werden als Objekte verstanden, die mit den anderen Agenten interagieren können. Der Unterschied hinsichtlich Modellierung zwischen NETLOGO und einer objektorientierten Sprache liegt darin, dass die Agenten in NETLOGO bereits mit allen Funktionalitäten, die sie zur Verfügung stellen, entworfen sind und man die komplexeren Schwarmabläufe quasi außerhalb der Objekte als Anweisungen formulieren muss, während man die Agenten in einer OO-Sprache mithilfe eines Bauplans – einer Klasse – entwerfen kann. So werden die Agenten zu eigenständigen „Wesen", die wie in der Realität keinem externen „Imperator" gehorchen. Durch diese Modellierung wird es möglich, die selbst entwickelten Agenten beispielsweise zum kursinternen Vergleich oder Wettbewerb an andere Gruppen oder an die Lehrkraft weiterzugeben.

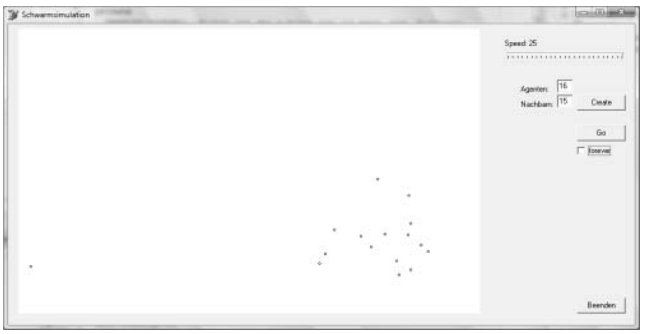

Bild 2: Eine ähnliche Situation in DELPHI.

Diese Ideen warten noch auf eine konkrete unterrichtliche Realisierung. Eine mögliche programmtechnische Umsetzung in NETLOGO und DELPHI gibt es schon; man findet sie unter

http://www.math.uni-frankfurt.de/~rabel/schwarm.zip

GO-Teile, die Objekte einer grafischen Benutzungsoberfläche oder Karas Welten sind – die Hauptsache ist Tun und Ausprobieren. Das ist auch richtig, weil die dabei gemachten Erfahrungen dann später systematisiert werden können. Die „erfahrenen" Lernenden sind dann offen für Modellierungstechniken, weil sie „erfahren"(!) haben, dass „Versuch und Irrtum" allein ab einer gewissen Komplexitätsstufe nicht mehr ausreicht. Aus der Arbeit an Projekten dieser Art kann so die Einsicht entstehen, dass es sinnvoll ist, die Dinge besser zu strukturieren, zu planen, zu modellieren. Wenn das gelingt, kann die OOM für Schüler eine Antwort auf eine Frage darstellen, die sie tatsächlich haben.

Ein anderer, aber möglicherweise damit auch kombinierbarer Weg besteht in der Stärkung der Modellvorstellungen. Es spricht viel dafür, dass eine tragfähige Vorstellung von der Arbeitsweise eines Computers auch die OO-Modellierung unterstützt: Bei der Erzeugung eines Objekts wird ein Stück Speicher belegt, das so strukturiert ist, wie es der „Klassenbauplan" angibt. In diesem Zusammenhang kann eine fundierte Vorstellung aufgebaut werden, was eine Referenz ist; dies ist dann hilfreich bei den Überlegungen, wie man Klassen gestaltet. (Diese Sichtweise lässt sich gut auf einem OH-Projektor und Objekten, die als Kopie der Klassenbeschreibung auf Folie produziert werden, darstellen.)

Wir haben verschiedene Vorschläge für mögliche (und möglichst) genetische Wege zu OOP und OOM erarbeitet und skizzieren diese in den Kästen „Hinzufügen von Eigenschaften, wenn es nötig und nützlich ist" (S.43), „Von Listen zu Objekten" (vorige Seite) und „Simulationen als Einstieg in die Objektorientierung" (oben).

Fazit

Mit diesem Beitrag soll keineswegs die objektorientierte Modellierung aus der Schule verbannt werden. Es scheint aber sinnvoll, ihre Bedeutung etwas zu relativieren, wodurch sie eher (wieder) Ziel denn Ausgangspunkt

der Beschäftigung mit Informatik wird. Die geringe Zahl von Schülerinnen und Schülern, die Informatik in der Qualifikationsphase des Gymnasiums belegt, ist möglicherweise nicht nur den Zwängen des Zentralabiturs geschuldet, sondern auch eine Reaktion darauf, dass der Unterricht (immer noch) nicht an den Bedürfnissen der Schülerinnen und Schüler orientiert ist.

Prof. Dr. Ulrich Kortenkamp
CERMAT
Pädagogische Hochschule Karlsruhe
Postfach 11 10 62
76060 Karlsruhe

E-Mail: kortenkamp@ph-karlsruhe.de

Prof. Dr. Eckart Modrow
Institut für Informatik
Universität Göttingen
Goldschmidtstraße 7
37077 Göttingen

E-Mail: emodrow@informatik.uni-goettingen.de

Prof. Dr. Reinhard Oldenburg
Dr. Jürgen Poloczek
Magnus Rabel
Institut für Didaktik der Mathematik und Informatik
Goethe-Universität
Senckenberganlage 9
60325 Frankfurt

E-Mail: {oldenbur, rabel}@math.uni-frankfurt.de
E-Mail: juergen.poloczek@auge.de

Literatur und Internetquellen

AKBSI – Arbeitskreis „Bildungsstandards" der Gesellschaft für Informatik (Hrsg.): Grundsätze und Standards für die Informatik in der Schule – Bildungsstandards Informatik für die Sekundarstufe I. Empfehlungen der Gesellschaft für Informatik e. V. vom 24. Januar 2008. In: LOG IN, 28. Jg. (2008), Heft 150/151, Beilage.
http://www.gi-ev.de/fileadmin/redaktion/empfehlungen/Bildungsstandards_2008.pdf

Brinda, T.: Ein didaktisches System zum objektorientierten Modellieren. In: LOG IN, 24. Jg. (2004), H. 128/129, S. 103–107.

Broy, M.; Siedersleben, J.: Objektorientierte Programmierung und Softwareentwicklung – Eine kritische Einschätzung. In: Informatik Spektrum, 25. Jg. (2002), H. 1, S. 3–11.

Crutzen, C.K.M.; Hein, H.W.: Objektorientiertes Denken als didaktische Basis der Informatik. In: S. Schubert (Hrsg.): Innovative Konzepte für die Ausbildung. 6. GI-Fachtagung „Informatik und Schule – INFOS '95", Chemnitz 1995. Berlin; Heidelberg u. a.: Springer, 1995, S. 149–158.

Diethelm, I.; Geiger, L.; Zündorf, A.: Rettet Prinzessin Ada – Am leichtesten objektorientiert. In: St. Friedrich (Hrsg.): Unterrichtskonzepte für informatische Bildung. INFOS 2005 – 11. GI-Fachtagung Informatik und Schule, 28.–30. September 2005 in Dresden. Reihe „GI-Edition LNI – Lecture Notes in Informatics", Band P-60. Bonn: Köllen Verlag, 2005, S. 161–172.

Eirund, H.: Objektorientierte Programmierung – Motivation und Einführung. In: LOG IN, 13. Jg. (1994), H. 4, S. 40–48.

Freudenthal, H.: Mathematik als pädagogische Aufgabe. Band 2. Stuttgart: Klett, 1973.

Graham, P.: Hackers & Painters – Big Ideas from the Computer Age. Sebastopol (USA, CA): O'Reilly, 2004.

Hartmann, W.: Informatik – EIN/AUS-Bildung. In: St. Friedrich (Hrsg.): Unterrichtskonzepte für informatische Bildung. INFOS 2005 – 11. GI-Fachtagung Informatik und Schule, 28.–30. September 2005 in Dresden. Reihe „GI-Edition LNI – Lecture Notes in Informatics", Band P-60. Bonn: Köllen Verlag, 2005, S. 43–56.

Hesse, W.; v. Braun, H.: Wo kommen die Objekte her? Ontologisch-erkenntnistheoretische Zugänge zum Objektbegriff. In: K. Bauknecht u. a. (Hrsg.): Informatik 2001 – Tagungsband der GI/OCG-Jahrestagung, Bd. II. Wien: Österr. Computer-Gesellschaft, 2001, S. 776–781.

Heubaum, A.: Möglichkeiten und Grenzen maschineller Intelligenz – Unterrichtsvorschläge in JAVA. Teil 1: Suchbaum und Rückziehungsverfahren. In: LOG IN, 24. Jg. (2004), H. 128/129, S. 62–79.

Hubwieser, P.: Didaktik der Informatik. Berlin u. a.: Springer, ^{3}2007.

Kortenkamp, U.: Foundations of Dynamic Geometry. Zürich: ETH Zürich, 1999 (Dissertation).
http://kortenkamps.net/papers/1999/diss.pdf

Rabel, M.; Oldenburg, R.: Konzepte, Modelle und Projekte im Informatikunterricht – Bewertungen und Erwartungen von Schülern und Studenten. In: B. Koerber (Hrsg.): Zukunft braucht Herkunft – 25 Jahre „INFOS – Informatik und Schule". INFOS 2009 – 13. GI-Fachtagung Informatik und Schule, 21.–24. September 2009 in Berlin. Reihe „GI-Edition LNI – Lecture Notes in Informatics", Band P-156. Bonn: Köllen Verlag, 2009, S. 146–156.

Schubert, S.; Schwill, A.: Didaktik der Informatik. Heidelberg: Spektrum, 2004.

Schulte, C.: Vom Modellieren zum Gestalten – Objektorientierung als Impuls für einen neuen Informatikunterricht. In: id – informatica didactica, 2. Jg. (2001), Nr. 3.
http://ddi.cs.uni-potsdam.de/InformaticaDidactica/Schulte2001.htm

Schwill, A.: Objektorientierte Programmierung – Eine Rechtfertigung aus kognitionspsychologischer Sicht. In: LOG IN, 13. Jg. (1993), H. 4, S. 44–45.

Schwill, A.: Programmierstile im Anfangsunterricht. In: S. Schubert (Hrsg.): Innovative Konzepte für die Ausbildung. 6. GI-Fachtagung „Informatik und Schule – INFOS '95", Chemnitz 1995. Berlin; Heidelberg u. a.: Springer, 1995, S. 178–187.

Spolwig, S.: „Hello World" in OOP – Zum Beitrag von Harro von Lavergne in LOG IN 18. Jg. (1998), Heft 5. In: LOG IN, 19. Jg. (1999), H. 5, S. 38–42.

Spolwig, S: Kritisches zu „Stiften und Mäusen" – Was ist objektorientierte Modellierung? In: LOG IN, 24. Jg. (2004), H. 130, S. 35–39.

Zendler, A.; Spannagel, C.: Empirical Foundation of Central Concepts for Computer Science Education. In: Journal of Educational Resources in Computing, Volume 8 (2008), Issue 2, Article No. 6.
http://portal.acm.org/citation.cfm?id=1362787.1362790

Alle Internetquellen wurden zuletzt am 4. Januar 2010 geprüft.