Programmiergrundausbildung: Erfahrungen von drei Hochschulen

Stephan Kleuker, Hochschule Osnabrück

s.kleuker@hs-osnabrueck.de

Zusammenfassung

Die Programmierausbildung bleibt ein Grundstein jedweder Informatik-Ausbildung und bildet das Fundament der meisten weiterführenden Informatik-Themen. Über die Art und Weise, wie Programmierung am besten gelehrt werden kann, wird wahrscheinlich seit Beginn dieser Lehre gestritten. Gerade Paradigmen-Wechsel haben diese Diskussion befruchtet. In diesem Artikel werden drei Erfahrungen in der Programmierausbildung in unterschiedlichen Programmiersprachen an verschiedenen Fachhochschulen zusammengefasst und mit einigen Hintergründen verknüpft. Für die Programmiersprache Java werden zusätzlich noch Varianten der didaktischen Herangehensweise diskutiert.

Einleitung

Da fast alle Bachelor-Abschlussarbeiten von Fachhochschulen in Unternehmen stattfinden und sich dabei fast immer ein Programmieranteil in dieser Arbeit befindet, ist fachliche Programmierausbildung ein elementares Ziel von Informatik-Studiengängen. Dabei soll es für angehende Absolventen keine Einschränkungen in der Wahl des Themas durch die am Anfang im Studium gewählte Programmiersprache geben. Es wird davon ausgegangen, dass innerhalb des Studiums die Fähigkeit erlangt wird, sich schnell in die Prinzipien einer vorher nicht benutzten Programmiersprache einzuarbeiten. Die eigentliche Nutzung der Sprache ist die Basis für die durchzuführenden Arbeiten und wird nebenbei erlernt.

In der Grundlagenausbildung soll dabei grundsätzlich eine Einführung in die Programmierung und nicht in eine bestimmte Programmiersprache stattfinden. Dies bedeutet, dass man aus didaktischen Gründen lieber auf Sprach-Features verzichtet, um elementare Konzepte wiederholt intensiver einüben zu können. Studierende sollten spätestens ab dem dritten Semester dann in der Lage sein, sich selbständig in neue nicht intensiv geschulte Programmiersprachen einzuarbeiten. Dies wird dadurch gefördert, dass oft in weiteren Lehrveranstaltungen weitere Programmiersprachen eingeführt werden, die für praktische Übungen unerlässlich sind. Beispiele sind JavaScript für Web-Technologien und Sprachen zur Erstellung von Triggern und Stored Procedures, wie PL/SQL und Transact SQL für Datenbanken.

Im Mittelpunkt der nachfolgenden Analyse steht dabei der Erfolg der Masse der Studierenden, also nicht die Anzahl der Studierenden, die zu sehr guten Ergebnissen kommen. Dieser Ansatz ist gerechtfertigt, da sich alle Hochschulen zwangsweise der Forderung der Politik stellen müssen, dass möglichst viele Studienanfänger in relativ kurzer Zeit zum Studienerfolg geführt werden müssen. Eine Forderung, die teilweise zu politisch diktierten Selbstauflagen von Studiengängen oder ganzen Hochschulen führt, dass 90% der Studierenden zum Studienerfolg gebracht werden. Diese Forderung wird hier als Grundlage genommen, wobei sie natürlich hinterfragt werden muss, da sich gerade in der Informatik immer wieder die Frage stellt, wieviel Prozent der Studienanfänger wirklich realistisch für das Themengebiet Informatik geeignet sind. Natürlich darf die Motivation und Förderung guter Studierender nicht vernachlässigt werden, die im konkreten Fall durch Zusatz- oder Alternativaufgaben und die Aufforderung, Studierenden mit aktuell weniger Kenntnissen zu unterstützen, für Lehrveranstaltungen in der Programmierung umgesetzt werden kann.

Motiviert ist dieser Artikel durch verschiedene Beiträge zu früheren SEUH-Konferenzen, in denen die Möglichkeiten zum Einstieg in die Objektorientierung [SZ07] und die Nutzung von Werkzeugen [Bol09] diskutiert wurden. Generell finden einige Untersuchung zum Lernverhalten von Studierenden statt, die häufiger in Magazinen wie „Computer Science Education“, wie z. B. [MFRW11] mit der Untersuchung der Denkmodelle, was bei Zuweisungen passiert, und [KS12] über Erfahrungen bei ersten Programmieraufgaben, veröffentlicht und auch in GI-Workshops, z. B. [Boe07], behandelt werden. Dieser Erfahrungsbericht will diese Forschungsinteressen unterstützen.

Im folgenden Text werden drei Einführungsveranstaltungen in die Programmierung, die mit unterschiedlichen Programmiersprachen durchgeführt wurden, auf ihren Erfolg im Zusammenhang mit der eingesetzten Programmiersprache analysiert. Da der Erfolg ohne den Aufbau eines komplexeren Messsystems nicht vollständig objektiv messbar ist, basieren Ergebnisse auf persönlichen Erfahrungen und Gesprächen mit beteiligten Studierenden und Lehrenden in ähnlichen Situationen. Es wird deshalb fast konsequent in diesem Artikel darauf verzichtet, aus Durchfallquoten in Prüfungen direkt auf den Erfolg eines didaktischen Ansatzes zu schließen. Im Kapitel zu Java werden weiterhin alternative Vorgehen zur Vermittlung und ihre Unterstützung durch verschiedene Werkzeuge betrachtet.

Die Komplexität des Aufbaus eines wirklich aussagekräftigen Messsystems, das auch Vergleiche aushält, wird auch in der Dissertation von Matthew C. Jadud [Jad06] deutlich, in der versucht wird, aus dem Verhalten, wann Studierende was kompilieren, Schlüsse über ihren Lernfortschritt zu ziehen. Die in der genannten Arbeit aufgeworfene Frage, aus welchen Indikatoren, z. B. die Messung von Zeiten, die in einer Entwicklungsumgebung benötigt wurden, die Anzahl von Compiler-Aufrufen oder die Anzahl gescheiterter zur Verfügung stehender Tests, man konkrete Schlüsse ziehen darf, stellt ein offenes spannendes Forschungsgebiet für weitere Untersuchungen dar.

Smalltalk

Smalltalk [Bra08] ist die erste kommerziell eingesetzte Sprache, die konsequent und vollständig auf Objektorientierung gesetzt hat. Alles, also auch einfache Zahlen, sind Objekte, die mit Methoden bearbeitet werden können. Die Geschichte von Smalltalk zeigt deutlich, wie ein hochinnovativer Ansatz historisch einfach völlig zu früh stattgefunden hat. Die Sprache pflegte immer das Image des besonderen und war maßgeblicher Vorreiter in der Entwicklung der Fenster-Technologien und der Maussteuerung. Leider benötigten diese Ansätze in den 1970er und 80er Jahren extrem teure Hardware, so dass Smalltalk zwar in einigen Unternehmen, wie z. B. dem Versicherungsbereich in Deutschland, in der Entwicklung eingesetzt wurde, aber nie einen echten Durchbruch schaffte. Teil dieser Problematik ist die zunächst gewöhnungsbedürftige Syntax, in der z. B. Parameter in Mixfix-Notation in den Methodennamen integriert werden. Die folgenden Befehle erzeugen ein Objekt vom Typ Dictionary, das Paare von Werten verwalten kann, für die dann ein erstes Paar eingetragen wird.

```
d := Dictionary new.
d at: 'Smalltalk' put: 'irgendwie spannend'.
```

Bemerkenswert ist, dass auch diese Schreibweise wesentliche Vorteile hat, da man so genau weiß, wozu die Parameter eingesetzt werden.

Smalltalk wurde und wird in der Programmiereinführung im Studiengang Wirtschaftsinformatik an der privaten Fachhochschule Nordakademie in Elmshorn genutzt. Die Fachhochschule bietet duale Studiengänge an, wobei Unternehmen Studierende aussuchen und deren Studiengebühren bezahlen. Die Studierenden arbeiten zwischen den Vorlesungsblöcken in den Unternehmen. Bemerkenswert ist, dass auch die Unternehmen die Auswahl von Smalltalk als Lernsprache mittragen, was damit zusammenhängt, dass in höheren Semestern konsequent Java eingesetzt wird.

Die Voraussetzungen für die Programmiereinführung unterscheiden sich damit deutlich von denen an öffentlichen Hochschulen. Die Studierenden haben sich bewusst für den dualen Studiengang entschieden, sie wissen, dass ihre Leistungen in der Hochschule auch von ihrem Unternehmen verfolgt werden und oftmals wird die spätere Weiterarbeit im jeweiligen Unternehmen angestrebt.

Fachlich bedeutet dies nach den Beobachtungen des Autors, dass das zumindest anfänglich leistungsschwächere Drittel von Studierenden, das an öffentlichen Hochschulen anfängt, an der Nordakademie kaum vertreten ist. Im oberen Bereich fehlen dafür vereinzelt sehr kreative Köpfe, die ein stark verschultes Ausbildungssystem bewusst umgehen wollen, um experimentelle Freiheiten im Studium zu schaffen.

Die eigentliche Vorlesung wurde mit dem Objects First-Ansatz gehalten, der bereits am Anfang auf die Strukturbegriffe Objekt und Klasse setzt. Generell wurden bei dieser Veranstaltung sehr positive Erfahrungen gemacht, die sich in der auf zwei Semester aufgespaltenen Veranstaltung in großen Lernfortschritten der Studierenden zeigten.

Nachdem anfängliche Schwierigkeiten mit der Syntax und der zunächst kompliziert, dann aber intuitiv nutzbaren Entwicklungsumgebung VisualWorks ausgeräumt waren, fand eine Konzentration auf die eigentliche Programmentwicklung statt.

Smalltalk nutzt den Begriff der Literale, um unveränderliche Objekte (immutable Objects) zu definieren, so kann ein String-Objekt in den meisten Smalltalk-Varianten nicht verändert werden, etwaige Methoden liefern ein neues String-Objekt als Ergebnis. Diese Unterscheidung ist aus Sicht von Anfängern nicht sehr intuitiv und führt zu Problemen. Kritisch aus Anfängersicht ist auch, dass für die Ausgabe eine Klassenmethode genutzt werden muss.

Insgesamt war die Vorlesung erfolgreich, da durch eine schnell angebotene Nachprüfungsmöglichkeit alle Studierenden letztendlich die Veranstaltung bestanden. Weiterhin war der Einstieg in die Objektorientierung erfolgreich, so dass der Übergang zu Java keine besondere Herausforderung darstellte.

Zusammenfassend kann man den Ansatz, Smalltalk als Anfängersprache zu nutzen, als klaren Erfolg ansehen, der allerdings maßgeblich durch die sehr guten Randbedingungen unterstützt wird. Die Möglichkeit, mit Smalltalk gerade an anderen Fachhochschulen anzufangen, schätzt der Autor als sehr gering an, da neben der mangelnden Akzeptanz unter den Kollegen gerade auch Studierende frühzeitig praxisrelevante Lehrstoffe vermittelt bekommen wollen. Dies ist besonders zu beachten, da viele Studierende ihr Studium nebenbei finanzieren müssen und der frühe Einstieg in einfache Programmierjobs in Unternehmen für Studierende und Unternehmen sehr vielversprechend sein kann.

Die Programmiersprache Ruby [Fla08] hat viele der zentralen Ideen übernommen und eignet sich sehr gut zur schnellen Prototyp-Entwicklung für Web-Systeme mit Ruby on Rails [RTH11]. Es wäre deshalb eine Überlegung wert, ob eine Programmiereinführung mit Ruby sinnvoll für Erstsemester sein könnte.

C

Die Programmiersprache C ist in aktuellen Untersuchungen weiterhin eine der zentralen Sprachen bei der Software-Entwicklung in Unternehmen. Da die Sprache sehr hardware-nah ist und gerade der Markt der embedded Systeme kontinuierlich vom Auto bis zum Kühlschrank wächst, wird die Sprache weiterhin mittelfristig ihre Bedeutung behalten.

C wurde 2006 in der Programmierausbildung für Erstsemester in einer Veranstaltung mit fünf Leistungspunkten an der Fachhochschule Wiesbaden (jetzt Hochschule RheinMain) genutzt. Die Rahmenbedingungen waren eine zu dem Zeitpunkt leicht sinkende Anzahl von Erstsemestern, wobei es in dieser Veranstaltung eine besonders hohe Anzahl von Wiederholern gab. C wird als gute Sprache zum Lernen der Programmierung von Befürwortern angesehen, da sie praxisnah ist und mit sehr wenigen Schlüsselwörtern auskommt, was zur Annahme führt, dass die Sprache leicht zu erlernen sei.

In der Durchführung zeigte sich deutlich, dass die notwendige Zeigerprogrammierung allein schon beim Umgang mit Strings, also char-Arrays mit besonderem Terminierungssymbol, und anderen Arrays ohne Möglichkeit, deren Länge implizit zu bestimmen, ein großes Problem darstellt. Ein Verheddern in der Zeigerwelt bei einfach und doppelt verketteten Listen ist für Anfänger oft unausweichlich. Gegenüber Java hat C allerdings den Vorteil, dass man alle Variablen auch per Referenz (`int* wert`) übergeben kann. Da so leider auch Arrays übergeben werden können, geht der intuitive Vorteil schnell verloren.

Weiterhin kann in C auch auf eine Kopiersemantik gesetzt werden, wenn z. B. Arrays in ein `struct` eingebettet sind. Dieser Ansatz wird zwar zunächst als intuitiver von Studierenden angesehen, führt aber schnell zu Problemen, wenn eine Verknüpfung mit der Referenzsemantik notwendig wird. Generell sollte deshalb auf die Kopiersemantik in der Grundausbildung in C verzichtet werden.

Gerade in C ist die Verlockung sehr groß, für Anfänger überflüssige, aus Sicht eines begeisterten C-Nutzers aber sehr interessante Sprachkonzepte wie Funktionszeiger und Makros durchzunehmen, was verpflichtend durchgeführt, für viele Studierende im ersten Semester zu großen Verständnisproblemen führt.

Eine verbreitete besondere Lernumgebung für C existiert nicht. Mit der SDL-Library [@SDL] besteht zwar die Möglichkeit, sehr schnell Spiele zu entwickeln, aber die Lernkurve ist beim Einstieg recht hoch.

Die hier dokumentierte Lehrveranstaltung wurde von vielen verschiedenen Lehrenden hintereinander gehalten, wobei die plumpe Aussage „Beim ersten Mal hält ein Dozent eine Lehrveranstaltung hauptsächlich für sich selbst“ einen Wahrheitswert enthält, da didaktische Erfahrungen aufgebaut werden müssen. Gerade die Beobachtung, dass der Übergang vom Lesen eines Programms zur Erstellung eines Programms selbst mit einfachen if-Anweisungen, ein enormer Schritt sein kann oder dass die didaktische Hürde zwischen einer einfachen Schleife und geschachtelten Schleifen sehr groß ist, muss oft in Selbsterfahrungen erworben werden.

Als Fazit eignet sich C wohl dann als Anfängersprache, wenn viele sprachliche Besonderheiten, wie die Kopiersemantik, Zeiger-Arithmetik und Makros, nicht behandelt werden. Ob solch ein konsequenter Ansatz schon umgesetzt wurde, ist leider unbekannt. Weiterhin zeigen Erfahrungen in höheren Semestern deutlich, dass zumindest bei aktuell nicht sehr leistungsstarken Studierenden die Ideen der Objektorientierung durch eine spätere Einführung nicht verinnerlicht werden.

Die ursprünglich mit C, C++ und Java dreisemestrige Programmierausbildung mit je fünf Leistungspunkten wurde in der Reakkreditierung des Bachelors nach sehr kontroversen Diskussionen durch eine zweisemestrige Programmierausbil-

dung mit jeweils 10 Leistungspunkten mit Schwerpunkt in Java, ergänzt um eine kompakte Einführung in C, ersetzt [HSRM10].

Java

Java verbreitet sich seit seiner Einführung Mitte der 1990er Jahre kontinuierlich in Unternehmen und in der Hochschulausbildung. In diversen Statistiken ist Java die am meisten benutzte Programmiersprache, wenn es um Programme im Business-Bereich, wie ERP geht.

Java ist aus Sicht der Objektorientierung eine hybride Sprache, deren Großteil aus der konsequenten Nutzung von Klassen besteht, die allerdings auf einfachen Basistypen aus C, wie `int` und `char` basieren. Diese Problematik spiegelt sich in verschiedenen didaktischen Ansätzen wider. Der klassische Ansatz besteht darin, zunächst den imperativen Teil der Programmiersprache zu lehren, die auf `if` und `while` fokussieren, und danach zur Objektorientierung überzugehen. Dieser Ansatz wurde lange Zeit auch in imperativen Sprachen, wie Basic, Pascal und C verfolgt, wo der zweite Teil sich auf die Entwicklung komplexerer Datenstrukturen meist zusammen mit ihrer Verlinkung in Listen fokussiert. Didaktisch steht man dann am Anfang vor dem Problem, dass man bei `public static void main(String[] s)` mit „public“, „static“ und „String[]“ drei Sprachanteile vorstellen muss, deren genauer Sinn erst viel später in der Lehrveranstaltung vermittelt werden kann. Gerade dieses „magische Verhalten“ führt aber bei Programmieranfängern oft zur verständlichen Idee, dass diese Magie auch für andere Sprachanteile gilt. Ein Beispiel ist der Konstruktor

```
public Punkt(int x, int y){
}
```

bei dem vermutet wird, dass eine Zuweisung zur Objektvariablen gleichen Namens (`this.x=x`) nicht programmiert werden muss.

Der klassische didaktische Ansatz ist anfänglich durchaus erfolgreich, führt aber beim Übergang zu Klassen oft zu Problemen, da diese dann analog zum `struct` in C als reine Daten-Container ohne eigene Funktionalität angesehen werden. Weiterhin setzt sich bei durchschnittlich begabten Studierenden bei frei gestaltbaren Programmieraufgaben in höheren Semestern eher der Ansatz durch, alles möglichst mit Klassenmethoden lösen zu wollen.

Einen ersten Bruch mit der imperativen Herangehensweise gibt es spätestens, wenn Strings oder Arrays genutzt werden sollen, da es sich hier um Objekte handelt, auf denen Methoden genutzt werden können. Arrays, deren Umsetzung in Java aus didaktischer Sicht sicherlich diskutabel ist, werden dann zum Problem, wenn sie an Funktionen automatisch per Referenz übergeben werden.

Unabhängig vom didaktischen Ansatz bietet Java als hybride Sprache eine weitere Falle für Anfänger mit dem Übergang vom elementaren Datentypen zu seiner kapselnden Klasse, z. B. von `int` zur Klasse `java.lang.Integer`, die z. B. bei der Nutzung von Listen benötigt wird. Zwar findet im Wesentlichen die Umwandlung durch Autoboxing automatisch statt, wird aber durch null-Werte und die Festlegung, dass die Integer-Werte von -128 bis 127 identisch, genauer Singletons, sind, die anderen nicht, bemerkenswert kompliziert. Das folgende Programm zeigt diese Probleme.

```
public static void main(String[] s){
  ArrayList<Integer> l = new ArrayList<>();
  l.add(10);
  l.add(1000);
  Integer vgl1 = 10;
  Integer vgl2 = 1000;
  for(Integer val:l){
    if (val == vgl1){
      System.out.println("val ist 10");
    }
    if (val == vgl2){
      System.out.println("val ist 1000");
    }
  }
  for(int val:l){
    if (val == vgl1){
      System.out.println("val ist 10");
    }
    if (val == vgl2){
      System.out.println("val ist 1000");
    }
  }
  l.add(null);
  Integer valn = l.get(2);
  System.out.println("valn = "+valn);
  int vali = l.get(2);
  System.out.println("vali = "+vali);
}
```

Die Ausgabe liefert

```
val ist 10
val ist 10
val ist 1000
valn = null
Exception in thread "main" java.lang.NullPointerException
```

ein Ergebnis, dass mit reiner Logik ohne zusätzliches Wissen kaum verständlich ist. Es ist möglich, dieses Problem im Wesentlichen zu vermeiden, indem konsequent nur die dazugehörigen Klassen wie `Integer` und `Double` für Variablen genutzt werden. Ein Ansatz, der vom Autor erfolgreich genutzt wurde. Leider kann man nicht vollständig in der Welt der Objekte und Klassen arbeiten, da die Java-

Klassenbibliothek viel mit int-Werten arbeitet. Kritisch ist weiterhin, dass man sehr häufig früh in Programmen viele null-Überprüfungen einbauen muss und es faktisch kein begleitendes Lehrbuch gibt, das diesen Ansatz auch verfolgt.

Java behandelt Zahlen- und String-Objekte wie für Smalltalk bereits andiskutiert als immutable Objects, was für Anfänger nicht unmittelbar verständlich ist.

Objects First

Trotz dieser Mängel wird Java oft und recht erfolgreich in der Programmiergrundausbildung eingesetzt, was auch auf die zweite didaktische Herangehensweise „Objects first" [BK09] zurück zu führen ist, bei der von Anfang an auf die Entwicklung von Objekten und Klassen gesetzt wird.

Die Grundidee besteht darin, sich möglichst früh mit den zentralen Strukturen der Objektorientierung auseinander zu setzen. Der Objektbegriff selbst ist intuitiv und Studierende finden ihn oft in ihrer realen Welt wieder, sei es selbst als Student-Objekt in der Software der Hochschulverwaltung oder als Kundin, die ein Konto-Objekt nutzt. Generell bietet sich hier die Hochschule zur Modellierung an, da Studienanfänger neu in diesem System sind, in das sie sich für drei oder mehr Jahre einarbeiten sollen. So kann z. B. der Modul-Begriff und sein Zusammenhang zur konkreten Vorlesung genauer analysiert werden.

Setzt man den Objekt-Begriff an den Anfang, spielen Begriffe wie Objekt, Objektvariable (auch Exemplarvariable, Instanzvariable oder Attribut genannt), konkrete Werte der Variablen, Objekte, die andere Objekte als Eigenschaft haben und Sammlungen von Objekten zunächst die zentrale Rolle. In Java stößt man dann schnell auf den Typ-Begriff, da jede Variable einen Typen haben muss, was entweder ein elementarer Typ aus der C-Welt oder wieder eine Klasse sein kann. Im nächsten Schritt werden Methoden und Parameter betrachtet, wobei hier der leider inkonsistente Ansatz von Kopien bei elementaren Typen und Referenzen bei Objekten zum ersten Bruch führt. Die eigentlichen Ablaufstrukturen werden dann später in den Methoden eingeführt. Man erhält so einen recht langen Anteil an Vorlesungen, die sich ausschließlich mit Objekten, Methoden und sequentiellen Programmabläufen beschäftigen, wozu man bemerkenswert viele Beispiele konstruieren kann. Dies ist auch ein gutes Beispiel, wie man Vorlesungen verlangsamen kann, da eine aus C bekannte Anweisung

```
x = x+1;
```

für einen Informatiker intuitiv erscheint, für jemand, der noch nicht programmiert hat, aber nach einer nicht erfüllbaren Formel aussieht. Ähnliches gilt für die Frage, warum man nicht `x+1 = x;` schreiben darf, obwohl wenig später `x==y+1` und `y+1==x` semantisch äquivalent sind.

Dieser konsequente Weg des Starts ausschließlich mit Objekten wurde an der Hochschule Osnabrück in zwei einführenden Lehrveranstaltungen des Bachelor-Studiengangs „Informatik - Medieninformatik" genutzt. Der Ansatz wurde im Rahmen einer Reakkreditierung in sehr kontroversen Diskussionen 2010 ausgewählt, in dem nach einer Einführung in Java, in einer Zweitsemesterveranstaltung C und C++ gelehrt werden. Ursprünglich wurde nach einer Einführung in C im zweiten Semester C++ (mit all seinen sprachlichen Besonderheiten) auch in Veranstaltungen zu jeweils zehn Leistungspunkten gelehrt. Die Java-Einführung war ein kleiner Block im Rahmen einer Usability-Veranstaltung.

Die Rahmenbedingungen der ersten Umsetzung des neuen Ansatzes waren relativ schlecht, da es durch einen doppelten Abiturjahrgang und Wiederholer, die diese neue Veranstaltung auch hören mussten, zu einer sehr hohen Teilnehmerzahl mit anfänglich schlechter räumlicher Ausstattung kam.

Statistiken zeigen, dass dieser erste Ansatz trotz schlechter Rahmenbedingungen nicht schlechter abgeschlossen hat, als der vorherige Weg. Weiterhin zeigte die Analyse eine Herausforderung, dass alle an einer Lehrveranstaltung Beteiligten das Konzept der Vorlesung verstehen und aktiv fördern müssen. Dies ist gerade wichtig, da es dauert, bis die Welt aus Objekten, Klassen und Methoden in den Köpfen verankert wird, was den zweiten Teil einer Lehrveranstaltung mit den Ablaufsteuerungen dann aber deutlich vereinfacht, da sich hier die objektorientierten Strukturen festigen. Da im konkreten Fall nicht genügend Zeit zur Schulung der Beteiligten vorlag und diese auch ihre eigenen Vorstellungen vom Programmieren vermitteln wollten, konnte der Weg in den zugehörigen Praktika nicht immer konsequent beschritten werden.

Die Wiederholung der Veranstaltung hatte mit deutlich geringeren Studierendenzahlen bessere Voraussetzungen, wobei sich dann klar die Vorteile des Ansatzes mit einer geringeren Durchfallquote, die von 32,2% auf 20,9% sank, wobei die Quote bei den Erstsemestern von 16,9% auf 12% zurückging, zeigten.

Im Folgeverlauf konnten einige Studierende bereits im zweiten Semester erfolgreich unter Anleitung eigenständige Programmier- und Analyseaufgaben in Forschungsprojekten als studentische Hilfskräfte übernehmen.

Zusammenfassend kann der Ansatz als sehr erfolgsversprechend angesehen werden, wird aber

nicht konsequent weiter betrachtet, da die verantwortlichen Dozenten sich nicht auf diesen Weg einigen konnten.

Die folgenden Unterabschnitte analysieren einige weitere Faktoren, die für den Erfolg einer Programmierveranstaltung relevant sein können, was z. B. die Wahl der Entwicklungsumgebung betrifft.

Eclipse

Eclipse [@Ecl] ist eine mächtige Plattform, die mit vielen Arten von Erweiterungen individuell an Projekte angepasst werden kann und deshalb in Unternehmen in den meisten Java-Projekten gesetzt ist. Neben Java ist die Entwicklungsumgebung auch für weitere Programmiersprachen wie C und C++ effizient nutzbar. Für absolute Programmieranfänger ist aber oft die große Vielzahl an angebotenen Arbeitsschritten sehr verwirrend. Die Flexibilität, die Eclipse bei Entwicklern sonst sehr beliebt ist, macht die Umgebung für Anfänger schwer einschätzbar. Es gibt z. B. mindestens drei Wege, eine Klasse anzulegen, es gibt viele Sichten, die konfiguriert werden können und die Syntaxprüfung weist bereits beim Tippen auf potenzielle Fehler hin. Weiterhin werden für Java viele vereinfachende Schritte, wie die Generierung von get- und set-Methoden sowie `hashCode` und `equals` angeboten. Gerade bei diesen Methoden ist es sinnvoll, dass Anfänger sie für sich durch die eigene Erstellung entdecken, um gerade das sehr gute, aber nicht einfache Konzept der Basisklasse `Object` zu verstehen.

Man kann zwar die Nutzung von Eclipse auf wenige Klicks reduzieren, die für erste Programme wirklich benötigt werden, benutzt allerdings ein Nachbar bei der Programmentwicklung ihm schon bekannte Features, wächst schnell ein innerer Druck, dies auch beherrschen zu wollen.

Ein Einstieg in Eclipse ist zumindest optional sicherlich im Laufe der ersten Programmiererfahrungen sinnvoll und kann zum Selbststudium in der vorlesungsfreien Zeit vorgeschlagen werden.

Eclipse unterstützt nicht vollständig den Objects-First-Ansatz, da zumindest eine `main`-Methode in einer Klasse existieren muss.

BlueJ

BlueJ [@Blu] ist gegenüber Eclipse eine wesentlich einfachere Entwicklungsumgebung mit wesentlich weniger Möglichkeiten. Ein einfaches Googeln nach BlueJ mit der Ergänzung „filetype:pdf“ zeigt, dass BlueJ an vielen Schulen aber auch Hochschulen in der Programmierausbildung genutzt wird.

BlueJ ermöglicht es, einfach Objekte gegebener Klassen zu erzeugen und mit diesen Objekten über deren Methoden zu interagieren. Dazu werden die verfügbaren Klassen in einem Klassendiagramm angezeigt, von denen dann über einen Rechtsklick Objekte erzeugt und Klassenmethoden aufgerufen werden können. Die Objekte liegen unten in einer Objektleiste, so dass über einen Rechtsklick deren Methoden ausführbar sind. Durch einen Doppelklick auf ein Objekt werden alle Objektvariablen mit Namen und Werten angezeigt. Änderungen über Methodenaufrufe werden auch in die angezeigten Objektvariablen übernommen. Hat eine Objektvariable eine Klasse als Typ, kann auch weiter in diese Objekte hineingeschaut werden. BlueJ bietet einen einfachen Debugger, der wie üblich über Break Points die schrittweise Ausführung und Überprüfung von Programmen ermöglicht.

Als weitere Besonderheit bietet BlueJ eine Konsole, das Code Pad, mit dem Java als Skriptsprache genutzt werden kann. Anweisungen werden hier direkt eingetippt und ausgeführt. Weiterhin sind Ausdrücke eingebbar, deren Ergebnisse sofort ausgegeben werden. Handelt es sich hierbei um Objektreferenzen, können diese in die Objektleiste gezogen und weiter analysiert werden.

Der Editor von BlueJ ist recht einfach gehalten, unterstützt aber die sinnvolle automatische Code-Formatierung und die Generierung von Java-Doc. Alle Änderungen werden unmittelbar abgespeichert. Der Compiler gibt nur jeweils die erste Fehlermeldung aus, um nicht mit teilweise unverständlichen Folgefehlern zu verwirren.

Zusammen mit BlueJ werden bereits einige Beispiele geliefert, die den didaktischen Ansatz unterstützen, dass Anfänger zunächst spielerisch lernen, Objekte zu erzeugen und deren Methoden auszuführen. Ein bekanntes Beispiel „shapes“ erlaubt es, verschiedene graphische Elemente auf einer Canvas-Oberfläche zu platzieren und diese zu verschieben und umzufärben. Dieses prinzipiell sehr schöne Beispiel hat allerdings später den Nachteil, dass an zentraler Stelle eine Klassenmethode genutzt wird, ein Konzept, dass man erst spät in einer Einführung in die objektorientierte Programmierung vorstellen sollte, da sonst die zunächst zu erlernende Grenze zwischen einer nicht ausführbaren Klasse und Objekten, mit denen Programmcode ausführbar ist, wieder verschwimmt.

Generell ist die Idee, möglichst einfach graphische Ausgaben oder sogar Interaktionen mit der Oberfläche zu erlauben, sehr sinnvoll, da man als Lernender schnell zu anschaulichen Programmen kommt, was oft auch den Spiel- und den für das Lernen dringend benötigten Probier-Trieb anspricht. Für das gezeigte Beispiel und auch einige Anfangsprogramme in [BK09] gilt leider, dass gegen die konsequente Objektorientierung verstoßen und schnell z. B. zu `System.out.println()` gegriffen wird.

Um auch bei Ein- und Ausgaben konsequent auf Objekte zu setzen, ist es sinnvoll, diese Funktionalität in einer Klasse mit Objektmethoden zu kapseln. Dies löst aber das Problem nicht vollständig, da es typischerweise trotz mehrfacher Konstruktornutzung jeweils nur ein Objekt dieser Klasse gibt.

Greenfoot

Die aus der graphischen Ausgabe gezogene Visualisierung ist auch eine Motivation für eine Alternative zu BlueJ namens Greenfoot [@Gre], ebenfalls einer Entwicklungsumgebung für Anfänger, die allerdings auf BlueJ basiert. Typischerweise werden mit Greenfoot sogenannte Objektwelten definiert, deren Objekte auf einem zweidimensionalen Feld als Objekte platziert werden können. Wie in BlueJ sind dann direkt alle Methoden des Objekts und das Objekt selbst zur Analyse zugreifbar und Methoden können ausgeführt werden.

Das typische Einführungsbeispiel sind Wombat-Objekte, die über das Feld mit Hilfe der Methoden gesteuert werden können und z. B. Blätter suchen und fressen. Generell vermittelt Greenfoot so sehr schön den Objektbegriff, man kann sogar einfache Vererbungen verfolgen, allerdings wird im Beispiel-Code wieder sehr schnell auf Klassenvariablen und Klassenmethoden zugegriffen. Greenfoot nutzt im Wesentlichen dabei eine Objektwelt, um zentral die Nutzung von `if` und `while` zu trainieren. Man muss dabei klar zwischen der reinen Nutzung der Objektwelt, bei der die Objekte nur über Methoden gesteuert werden können, und der Erstellung neuer Welten, Szenarien genannt, die von erfahreneren Entwicklern angelegt werden sollten, trennen. Von Programmieranfängern wird nur die Erweiterung existierender Szenarien erwartet.

Greenfoot eignet sich auch, um einfache interaktive Spiele zu erstellen, wobei hier von der konsequenten Objektorientierung abgewichen wird.

```
if (Greenfoot.mouseClicked(null))
```

Die vorherige Fallunterscheidung dient z. B. dazu, festzustellen, ob die Maus zum Klicken genutzt wurde. Um dann das geklickte Objekt zu finden, muss ein Cast-Operator und ein Klassen-Objekt genutzt werden, wie das folgende Code-Fragment zeigt.

```
Balloon balloon = (Balloon) getOneObjectAtOffset(x , y
                                        , Balloon.class);
```

Verwandt mit Greenfoot sind einige andere Ansätze, wie das Hamster-Modell [Bol08], Kara [@Kara], Karol [@Karo] oder Scratch [@Scr], mit denen eigene Umgebungen, typischerweise in Java, geschaffen werden, um Schüler an die ersten Schritte des Programmierens, also die systematische Hintereinanderausführung von Befehlen, heranzuführen. Dieser Ansatz mag zu schnellen „Erfolgen“ bei den Lernenden führen, was die Motivation hochhalten kann, ob der Übergang dadurch in reale Programmiersprachen einfacher oder durch ein dann fundiertes zu einfaches mentales Entwicklungsmodell erschwert wird, ist bisher in Studien nicht ernsthaft untersucht worden.

Interaktionsbrett

Die Motivation von Lernenden mit graphischen Möglichkeiten hochzuhalten, ist in vielen Ansätzen vertreten. Aus diesem Grund wurde für die hier betrachteten Java-Veranstaltungen in Osnabrück motiviert von der kritischen Analyse von Greenfoot eine eigene Klasse Interaktionsbrett zur Visualisierung erstellt [@Int].

Die zentrale Idee ist es, konsequent objektorientiert eine Klasse nutzbar zu machen, mit der die einfachen graphischen Elemente Punkt, Linie, Kreis und Rechteck gezeichnet und mit der Maus bewegt werden können. Weiterhin wird die Tastatursteuerung unterstützt. Zum Einstieg werden auch hier einfache Befehlsketten genutzt, die ein Bild auf der Oberfläche zeichnen. Danach werden eigene Klassen für graphische Objekte, wie Dreiecke, angelegt, die dann selbst Methoden enthalten, um sich auf ein Interaktionsbrett zu zeichnen.

Das folgende Programm zeigt die Möglichkeit, einen Kreis mit Hilfe der Pfeil-Tasten nach links und rechts zu schieben. Etwas „negative Magie“ wird auch im Interaktionsbrett benötigt, da ein Objekt, das über Tastatureingaben informiert werden möchte, eine Methode der Form `public void tasteGedrueckt(String s)` realisieren muss. Das Interaktionsbrett nutzt Reflektion, um dann diese Methode aufzurufen.

```
public class Steuerung {
  private Interaktionsbrett ib = new Interaktionsbrett();
  private int pos = 0;

  public Steuerung() {
    ib.willTasteninfo(this);
  }

  public void tasteGedrueckt(String s) {
    if (s.equals("Rechts")) {
      pos = pos + 5;
    } else if (s.equals("Links")) {
      pos = pos - 5;
    }
    ib.abwischen();
    ib.neuerKreis(pos, 5, 20);
  }

  public static void main(String[] s){
    new Steuerung();
  }
}
```

Mit Hilfe des Interaktionsbretts können auch leicht etwas kompliziertere Spiele programmiert werden. Abb. 1 zeigt ein Beispiel einer Abschlussaufgabe, in der mehrere Monster, die eine Mauer erklimmen, mit Steinen von einem auf der Burgmauer laufenden Wächter abgewehrt werden müssen.

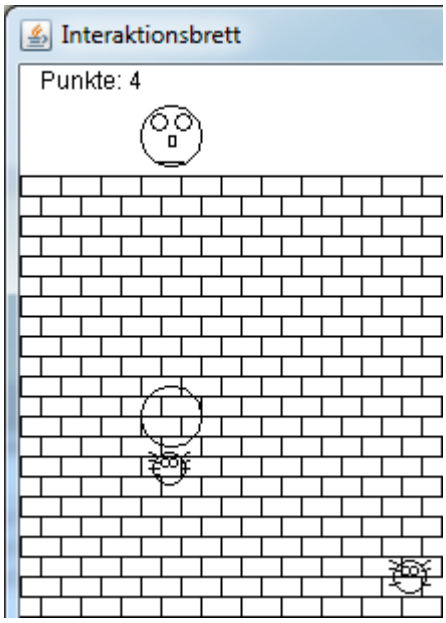

Abb. 1: Interaktives Spiel mit Objekten

Jedes der graphischen Elemente kann mit der Maus bewegt werden, wozu auch etwas Magie notwendig ist. Wieder muss festgelegt werden, welches Objekt informiert werden soll, wenn eine Mausaktion stattfindet. Weiterhin ist es sinnvoll, dass mehrere Objekte über einen Namen als String unterschieden werden können, wenn das gleiche Objekt informiert werden soll. Für einen verschiebbaren Kreis mit x- und y-Koordinate sowie einem Radius, sieht dann eine Methode zum Anmelden beim Interaktionsbrett wie folgt aus.

```
ib.neuerKreis(this,"Ball",ib.zufall(0,300),42,10);
```

Hier wird das Objekt, das diesen Befehl ausführt, auch über die Mausaktion, die in angeklickt, verschoben und losgelassen unterschieden werden, informiert. Dazu muss zumindest eine der drei zu den Aktionen gehörenden Maus-Methoden realisiert werden, die als Parameter den Namen des Objekts und seine neuen Koordinaten übermittelt bekommt.

```
public Boolean mitMausAngeklickt(String name
                          , Integer x, Integer y){
  return !versenkt;
}
```

Der Boolesche Ergebniswert legt fest, ob die Aktion überhaupt erwünscht ist. Will man z. B. einen Knopf realisieren, muss er auf das Klicken mit der Maus reagieren und mit false antworten, da er nicht verschoben, aber informiert werden will.

Generell wurde das Interaktionsbrett von Studierenden in Praktika sehr positiv aufgenommen, da man sehr einfach zeichnen kann. Das Einüben der Maussteuerung ist dabei deutlich komplizierter und gehört in den zweiten Teil der Lehrveranstaltung. Aktuell wird diskutiert, ob die absichtlich gewählte minimale Darstellung von ausschließlich schwarzen Linien und der Verzicht auf einen Hintergrund, durch sinnvolle Erweiterungen ergänzt werden sollen. Die Weiterentwicklung findet dabei durch Studierende statt, die weitere Beispiele erstellen.

Processing

Processing [@Pro] wurde als Sprache und Entwicklungsumgebung entwickelt, um kreativen Leuten einen sehr einfachen Zugang zu den Themen Visualisierung, interaktive Animation und Simulation zu ermöglichen. Die zugehörigen Arbeiten wurden am Massachusetts Institute of Technology von Ben Fry und Casey Reas gestartet. Mittlerweile gibt es im Internet eine große Unterstützung des Ansatzes mit vielen Tutorials und quelloffenen Beispielen. Processing wird in vielen Design-Studiengängen, u. a. auch im Media and Interaction Design-Studiengang an der Hochschule Osnabrück für kleinere und größere Projekte eingesetzt.

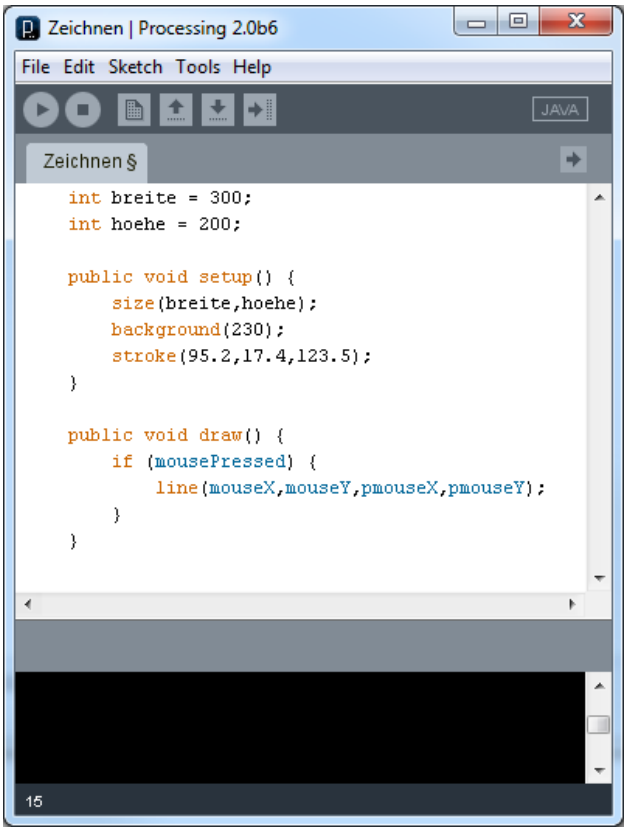

Abb. 2: Malprogramm in Processing

Sehr interessant ist aus didaktischer Sicht, dass viele Studierende, die sich bewusst für einen künstlerischen Studiengang entschieden haben, durch Processing einen schnellen und oft begeisternden Einstieg in die Programmierung finden. Es wird wieder auf die unmittelbare Visualisierung und recht einfache Interaktionsmöglichkeiten gesetzt. Anders als bei „Spielsprachen" findet die Programmierung im Wesentlichen in Java statt, wobei die Entwicklungsumgebung es ermöglicht, den Objektbegriff in den Hintergrund zu stellen. Dies soll mit dem vollständigen Programm in Abb. 2, mit dem ein Nutzer mit gedrückter Maustaste auf einer Fläche zeichnen kann, veranschaulicht werden.

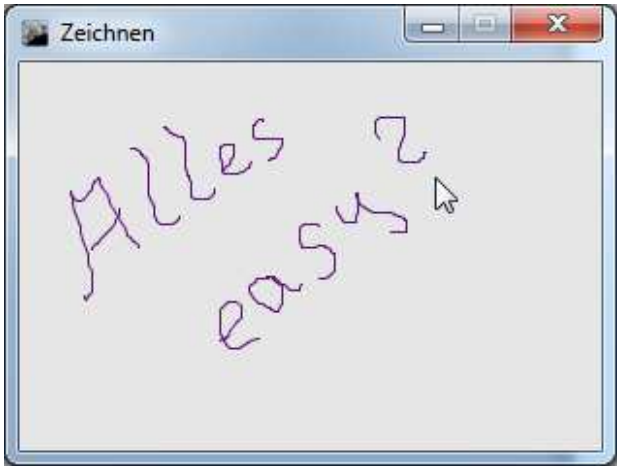

Abb. 3: Malprogramm in Benutzung

Abb. 3 zeigt das entstehende Beispielfenster, das auch leicht als ausführbares Programm, Java-Applet oder als auf Android lauffähiges Programm exportiert werden kann.

Zentral nutzt Processing zwei Funktionen (sogenannt im Processing-Sprachgebrauch), `init()` und `draw()`, wobei `init()` einmal am Programmanfang und `draw()` danach in einer Endlosschleife aufgerufen werden. Zentral in Processing ist die große Menge von Funktionen, die zur Erzeugung und Verarbeitung graphischer Elemente bereits zur einfachen Nutzung vorliegen, wodurch die eigentliche Processing-Sprache definiert wird. Dazu gehören auch viele Varianten, Farbtöne anzugeben und detailliert Fonts auszuwählen. Im obigen Beispiel sieht man, wie die Hintergrundfarbe als eine Möglichkeit mit einem `int`-Wert festgelegt wird. Mit `stroke()` wird die Farbe für die nächsten gemalten Objekte festgelegt. Dabei ist die Grundidee ähnlich wie beim Malen selbst, dass man durch Funktionen festlegt, wie gemalt wird und dies dann für alle folgenden Malaktionen gilt.

Das Beispiel zeigt auch, dass in Processing einfach einige „negative Magie" genutzt wird, um das Programmieren zu erleichtern. Es gibt eine Boolesche Variable `mousePressed`, mit der der Zustand der Maus abgeprüft werden kann. Weiterhin befindet sich die aktuelle Mausposition in den Variablen `mouseX` und `mouseY` sowie die unmittelbar vorherige Position in den Variablen `pmouseX` und `pmouseY`.

Processing ist keine prozedurale Sprache wie C, da die Klassen der Java-Klassenbibliothek genutzt werden und auf deren Objekte mit Methoden zugegriffen wird. Es entsteht so eine Mischung aus Funktionen und Methodenaufrufen. Die Entwicklungsumgebung enthält keinen Debugger. Semantisch programmiert man in Processing eine Erweiterung einer Klasse `processing.core.PApplet`, in die der Code aus dem Editor eingebaut wird. Man kann so auch direkt in Processing neue Klassen selbst entwickeln, die dann zu lokalen Klassen der großen umgebenden Klasse werden und direkt auf die Processing-Funktionen zugreifen können. Alternativ kann man in Processing auch echte neue Klassen schreiben, muss dann aber selbst eine Referenz zur Nutzung der Processing-Funktionen aufbauen.

Da Processing im Wesentlichen in Java geschrieben ist, kann man die zugehörigen Klassenbibliotheken auch in andere Programme einbauen. In [Kle12] wird technisch beschrieben, wie man Processing aus BlueJ heraus nutzen kann, wobei durch die besondere Art der Ausführung der Debugger nicht zur Verfügung steht. Funktional muss der Entwickler für angeklickte Elemente selbst berechnen, welches Element angeklickt wurde, eine Funktionalität, die beim Interaktionsbrett schon gegeben ist. Weiterhin unterstützt Processing keine Eingabe in Konsolenfenstern.

Die vorherigen Abschnitte haben klar gezeigt, welche didaktischen Defizite Processing enthält, weshalb man es mit guten Gründen nicht in einer Programmiereinführung einsetzen sollte. Dem muss man die weite Verbreitung von Processing in kreativen Bereichen gegenüberstellen, in denen es auch gelingt, nicht programmieraffinen Personen die Begeisterung an der Entwicklung beizubringen. Ob dieser Ansatz auch den systematischen späteren Ein- oder Umstieg in die objektorientierte Programmierung erlaubt, müssten längerfristige Untersuchungen zeigen.

Zwischenfazit zu Java

In den letzten Abschnitten wurde der klassische imperative Lehrweg dem konsequenten Ansatz Objects First gegenüber gestellt und aus Sicht des Autors die Vorteile in der durchgeführten Lehre vorgestellt. Diese Sichtweise spiegelt sich in der neu entstehenden Literatur zu Java [HMG11] [Ull11] [Far12] nur selten wieder [Wu10], es wird maximal ein Ansatz „Objects early" versucht, der „schnell" etwas zu Alternativen und Schleifen zei-

gen will, um dann in Objekte und Klassen einzusteigen. Da hier auch am Anfang Programme mit Strukturen geschrieben werden, die konsequent der objektorientierten Vorgehensweise widersprechen, kann aus didaktischer Sicht hier kein Unterschied zum ursprünglichen Vorgehen gesehen werden.

Aus Sicht der doch recht kleinen Gruppe der „Objects first"-Vertreter stellt sich die Frage, ob der Ansatz ein Irrtum ist und man gegen Windmühlen kämpft oder ob man weiterhin bei der Behauptung „die Erde ist rund" bleiben soll. Die weiteren Abschnitte zu Java zeigen eventuell einen anderen Weg, dass man den „Spaß an der Entwicklung" herstellen soll und die Zeit mit den Folgeerfahrungen den Weg zur guten Programmierung ebnet.

Zusammengefasst bietet Java sehr viele Möglichkeiten zu einem spannenden Einstieg in die Programmierung, wobei man die anfänglich genannten sprachlichen Probleme beachten und didaktisch umschiffen muss.

Fazit

Etwas verallgemeinert kann man die dargestellten Erfahrungen in der Form zusammenfassen, dass die Auswahl der ersten Programmiersprache im Studium wahrscheinlich kein wesentlicher Faktor für den Studien- und den Berufserfolg von Studierenden ist. Begeisterte und begeisternde Dozenten mit guter fachlicher Unterstützung und einfach zu nutzenden Werkzeugen können als Erfolgsfaktoren bestimmt werden. Dies bedeutet auch, dass Dozenten zumindest in den ersten Semestern ein gemeinsames Konzept in der Anwendung von Programmiersprachen finden sollten, was häufiger nicht der Fall ist.

Möchte man nach dem ersten Semester eine Programmiergrundlage legen, die recht einfach erweitert werden kann und Studierenden nebenbei recht viele Nebentätigkeiten in Hochschulen oder Unternehmen ermöglicht, sollte die Auswahl auf eine objektorientierte Sprache fallen. Dabei gibt es Indikatoren auf Basis der Literatur und der gemachten Erfahrungen, dass ein konsequenter Einstieg mit dem Erlernen von Objekten und Klassen den Vorteil hat, dass diese Abstraktionsebene sehr früh eingeübt wird.

Eine wirkliche fundierte Untersuchung zeigt sich als schwierig, da hier mehrere Herausforderungen der empirischen Sozialforschung zusammenkommen, um mögliche Hypothesen überprüfen zu können. Oftmals haben Studierende bereits Kenntnisse der Programmierung aus der Schule oder haben sich diese teilweise selbst beigebracht. Dabei zeigt sich immer wieder, dass die Programmierausbildung in den Schulen ein extremst unterschiedliches Niveau hat. Häufiger sind Fälle zu beobachten, dass Lehrer, die natürlich ebenfalls auf der Suche nach dem richtigen Ansatz zur Programmierausbildung sind, zu sehr merkwürdigen Programmierrichtlinien und Programmierstilen führen. Natürlich ist auch das andere Extrem vertreten, bei denen wesentliche Teile von Programmierveranstaltungen an der Hochschule für Studierende dank ihrer Schul- oder evtl. betrieblichen Ausbildung trivial werden. Die Frage, ob ein Informatik-Unterricht mit Inhalten der Studiensemester in der Schule überhaupt sinnvoll ist, sei dabei am Rande gestellt.

Eine Analyse des Erfolgs eines Ansatzes ist noch schwieriger, da er eigentlich nur durch die intensive Analyse der Fähigkeiten von Studierenden ermöglicht wird. Da aber die damit verbundene intensivere Betreuung bereits wesentlichen Einfluss auf den Lernerfolg haben kann, wird eine Beurteilung oder gar ein Vergleich von Ergebnissen sehr schwierig.

Es stellt sich so die Frage, ob statt einer direkten Beobachtung Indikatoren gefunden werden können, die auf den Programmiererfolg schließen lassen. Dies könnte durch die Verbindung von Entwicklungswerkzeugen mit statischen Analysewerkzeugen geschehen, die z. B. messen, wie oft ein Compiler aufgerufen wird und wie oft Fehler z. B. in gleichen Zeilen dabei gefunden werden. Diese Analysewerkzeuge dürfen dabei die eigentliche Nutzung nicht beeinflussen. Vor diesem Ansatz ist aber dann zu klären, ob die Indikatoren wirklich geeignet sind. Der Erfahrungsbericht bietet einige Fragen und Hypothesen, die genauer zu untersuchen sind.

- Da die am meisten angewandten Sprachen objektorientiert sind, stellt sich die Frage, ist ein Start mit C noch sinnvoll?
- Wenn man objektorientiert beginnt, sollte man dann rigoros auf Objects first setzen?
- Ist Java als Einführungssprache geeignet, obwohl sie nicht konsequent objektorientiert ist?
- Welchen Motivationsschub kann aus der Möglichkeit schnell einfache Animationen zu erstellen gezogen werden? Gibt es vergleichbar erfolgreiche Ansätze, wie der Einsatz von kleinen Robotern?
- Welche Bedeutung hat die Auswahl von Entwicklungsumgebungen?
- Wie sieht eine kontinuierliche Integration der Programmierausbildung in die weiteren Lehrveranstaltungen der ersten Semester aus? Ist es z. B. sinnvoll parallel eine weitere Sprache wie JavaScript in einer Einführung zur Web-Technologie zu lehren?

- Welche Bedeutung hat das Engagement der Lehrenden? Ketzerisch formuliert, sind die vorherigen Fragen nachrangig, wenn das echte Bemühen die Programmierung näher zu bringen, deutlich wird?

Zusammengefasst wird die Suche nach dem richtigen Weg in der Programmierausbildung sicherlich wie in der Vergangenheit auch, zu mehrfach neu gefundenen und dann auch wieder konsequent verworfenen Wegen führen.

Der Autor dankt Dr. Cheryl Kleuker und Prof. Dr. Frank Thiesing für die Diskussion und Kommentierung einer Vorversion dieses Artikels.

Literatur

Web-Seiten vom Stand 1.11.2012 betrachtet.

[@Blu] BlueJ, http://www.bluej.org/

[@Ecl] Eclipse, http://www.eclipse.org/

[@Gre] Greenfoot, http://www.greenfoot.org/door

[@Int] Interaktionsbrett, http://home.edvsz.hs-osnab-rueck.de/skleuker/querschnittlich/InteraktionsbrettMID/index.html

[@Kara] Programmieren lernen mit Kara, http://www.swisseduc.ch/informatik/karatojava/index.html

[@Karo] Java Karol, http://www.schule.bayern.de/karol/jdownload.htm

[@Pro] Processing, http://processing.org/

[@Scr] Scratch, http://scratch.mit.edu/

[@SDL] Simple DirectMedia Layer, http://www.libsdl.org/

[BK09] D. J. Barnes , M. Kölling, Java lernen mit BlueJ: Eine Einführung in die objektorientierte Programmierung, 4. Auflage, Pearson Studium, 2009

[Boe07] J. Boerstler, Objektorientiertes Programmieren - Machen wir irgendwas falsch?, Didaktik der Informatik in Theorie und Praxis 2007, http://subs.emis.de/LNI/Proceedings/Proceedings112.html, Seiten 9-20, 2007

[Bol08] D. Boles, Programmieren spielend gelernt mit dem Java-Hamster-Modell, 4. Auflage, Vieweg+Teubner, Wiesbaden, 2008

[Bol09] D. Boles, Threadnocchio – Einsatz von Visualisierungstechniken zum spielerischen Erlernen der parallelen Programmierung mit Java-Threads, in U. Jaeger, K. Schneider (Hrsg.), Software-Engineering im Unterricht der Hochschulen 2009, Seiten 131-144, dpunkt.verlag, Heidelberg, 2009

[Bra08] J. Brauer, Grundkurs Smalltalk - Objektorientierung von Anfang an: Eine Einführung in die Programmierung, 3. Auflage, Vieweg+Teubner, Wiesbaden, 2009

[Far12] J. Farell, Java Programming, 6. Auflage, Course Technology, Cengage Learning, USA, 2012

[Fla08] D. Flanagan, The Ruby Programming Language, O'Reilly, USA, 2008

[HMG11] C. Heinisch, F. Müller-Hofmann, J. Goll, Java als erste Programmiersprache, 6. Auflage, Vieweg+Teubner, Wiesbaden, 2011

[HSRM10] Hochschule RheinMain, Modulhandbuch Angewandte Informatik, 2010

[Jad06] M. C. Jadud, An Exploration of Novice Compilation Behaviour in BlueJ, PhD Thesis University of Kent, 2006 (http://kar.kent.ac.uk/14615/1/An_Exploration_of_Novice_Compilation_Behaviour_in_BlueJ.pdf)

[Kle12] S. Kleuker, Technischer Bericht, http://home.edvsz.hs-osnab-rueck.de/skleuker/querschnittlich/BlueJUserManualMID.pdf

[KS12] P. Kinnunen, B. Simon, My program is ok – am I? Computing freshmen's experiences of doing programming assignments, Computer Science Education, 22:1, Seiten 1-28, 2012

[MFRW11] L. Ma, J. Ferguson, M. Roper, M. Wood, Investigating and improving the models of programming concepts held by novice programmers, Computer Science Education, 21:1, Seiten 57-80, 2011

[RTH11] S. Ruby, D. Thomas, D. Heinemeier Hansson, Agile Web Development with Rails, 4. Auflage, Pragmatic Programmers, USA, 2011

[SZ07] A.Schmolitzky, H. Züllighoven, Einführung in die Softwareentwicklung - Softwaretechnik trotz Objektorientierung? In: A. Zeller and M. Deininger (Hrsg.), Software Engineering im Unterricht der Hochschulen, Seiten 87-100, 2007

[Ull11] C. Ullenboom, Java ist auch eine Insel, 9. Auflage, Galileo Computing, Bonn, 2011

[Wu10] C. T. Wu, An introduction to object-oriented programming with Java, 5. Auflage, Mc Graw Hill, USA, 2010