Lehre objektorientierter Konzepte auf der Basis von Smalltalk und Java

Michael Böhm, Guido Pfeiffer, Jörg Raasch

Fachhochschule Hamburg

Abstract. Als primäres Ziel der Anfängerausbildung in Informatik wird die Vermittlung konzeptioneller Inhalte der Objektorientierung angesehen, ohne daß die handwerklichen Programmierkenntnisse vernachlässigt werden sollen. Smalltalk mit den beiden sprachkonzeptuellen Grundcharakteristiken Minimalität und Orthogonalität bietet den Freiraum, die Lehre von Anfang auf objektorientiertes Denken zu konzentrieren. Das Erlernen von Java kann darauf aufbauend relativ zügig erfolgen. Es werden Erfahrungen, Vorteile und Probleme eines kombinierten Smalltalk/Java-Lehrkonzeptes diskutiert.

1. Einleitung

An der Fachhochschule Hamburg konnten seit über 10 Jahren Erfahrungen in der Lehre objektorientierter Konzepte gesammelt werden [1]. Zu Beginn wurde für die Programmier-Grundausbildung C++ verwendet, was angesichts der Schwierigkeit dieser Sprache nicht ganz unproblematisch war. 1996 wurde im Studiengang ‚*Softwaretechnik*' eine Kombination von Smalltalk (1. Semester) und C++ (2. Semester) eingeführt. Die Umstellung auf Smalltalk sollte eine stärkere Betonung konzeptioneller Inhalte ermöglichen. Das Beibehalten von C++ war eine Konzession an die hohe Marktbedeutung von C++. Nach dem erfolgreichen Vordringen von Java wurde dann ab Sommersemester 1999 im Studiengang Softwaretechnik die Sprache C++ durch Java ersetzt, Smalltalk im ersten Semester jedoch erhalten. In einem zweiten Informatik-Studiengang an der Fachhochschule Hamburg *‚Technische Informatik'* wurde die Grundausbildung gänzlich auf Java umgestellt.

Eine Kombination von Smalltalk und Java als programmiersprachliche Basis der Informatikausbildung ist recht selten. In Java wurden einige der besten Konzepte von Smalltalk integriert. Trotz einiger gravierender Unterschiede gibt es Gemeinsamkeiten in beiden Sprachen, so daß man fragen kann, ob nicht die volle Konzentration auf eine einzige Sprache effektiver wäre. Wir haben jedoch einige Gründe dafür, Studierende primär über die Sprache *‚Smalltalk'* in die Objektorientierung einzuführen. Hierüber und die gemachten Erfahrungen wird im folgenden berichtet.

2. Vorteile von Smalltalk für das Erlernen objektorientierter Konzepte

Die beiden sprachkonzeptuellen Grundcharakteristiken von Smalltalk sind *Minimalität* und *Orthogonalität*. Die wichtigsten Gründe dafür, Smalltalk als erste Ausbildungssprache zu verwenden, sind [1]:

- *Objektorientierung pur*. Anfänger sind von Beginn an gezwungen, in den Kategorien von Klassen, Objekten und Vererbung zu denken. Allein die Zweiteilung von Java in eine elementare, nicht objektorientiere C-Welt und eine Welt von Klassen und Objekten mit der Kopplung beider Welten über Wrapping-Techniken ist Anfängern nur schwer verständlich zu machen und kostet viel Zeit.
  Vorteil: volle Konzentration auf Objektorientierung und Zeitgewinn.
- *Minimale Syntax und Sprachkern*. Smalltalk gehört zu den syntaxärmsten Sprachen. Die Kenntnis von ein paar Syntaxregeln und einiger weniger Basiskonzepte wie Message Passing und Klassenkonzept reicht aus, um unmittelbar in die objektorientierte Softwareentwicklung einzusteigen. Ebenso ist der Sprachkern von Smalltalk minimal. Die gesamte Funktionalität der Sprache – selbst Komponenten, die traditionell zum Kern einer Programmiersprache gehören – sind in Bibliotheken verlagert. Folglich sind Anfänger ebenfalls von Beginn an gezwungen, wie Profis in und mit Bibliotheken zu arbeiten. Natürlich ist es zwingend notwendig, Anfängern zunächst nur einen winzigen Ausschnitt der gesamten Klassenbibliothek zu erklären, damit sie nicht orientierungslos vor dem erheblichen Umfang der Klassenbibliothek kapitulieren.
  Vorteil: großer Zeitgewinn.
- *Metasprachliche Eigenschaften*. Die metasprachlichen bzw. reflektiven Eigenschaften von Smalltalk sind äußerst gut ausgebaut, verglichen etwa mit denen von Java. Natürlich sind dies keine Themen der ersten Vorlesungsstunden. Jedoch dank der Einfachheit und Vollständigkeit der reflektiven Komponenten von Smalltalk können selbst Anfänger sehr frühzeitig mit Techniken von Frameworkerstellung und generischer Programmierung bekannt gemacht werden.
  Vorteil: konzeptionell anspruchsvolle Themen bei geringem Zeitaufwand.
- *Entwicklungssysteme*. Smalltalksysteme waren immer zugleich auch mächtige Entwicklungswerkzeuge, mit Browsern, Workspaces, Inspektoren, Debuggern etc. Natürlich erfordert jedes mächtige Entwicklungssystem eine gewisse Einarbeitungszeit. Die Erfahrung zeigte jedoch, daß die Studierenden – selbst diejenigen, die noch nie Kontakt mit einem Computer hatten – sich überraschend schnell in die Smalltalkwerkzeuge einarbeiteten und bereits nach kurzer Zeit produktiv arbeiten konnten.
  Vorteil: deutlicher Zeitgewinn.
- *Typisierung*. Ein Punkt wird bei der Beurteilung der Eignung von Programmierwerkzeugen für Lehre und Entwicklung – so auch bei der Frage Smalltalk oder Java – häufig kontrovers diskutiert, nämlich die Frage der *Typisierung*. Als typsicher werden Sprachen angesehen, auf deren Daten oder Objekte keine illegale Operationen angewandt werden können. In diesem Sinne sind Java und Smalltalk beide typsicher, im Gegensatz etwa zu C oder C++. Unterschiedlich ist alleine der Zeitpunkt der Typprüfung, welche (statisch) zur Übersetzungszeit

oder (dynamisch) zur Laufzeit erfolgen kann. Welches von beiden Verfahren nun der *richtige* Weg zu besserer Software ist, darüber wird manchmal kontrovers und aufwändig gestritten. Befürworter statischer Typisierung stellen didaktische Argumente für Programmieranfänger in den Vordergrund, während die Freunde dynamischer Typisierung eher die Vorteile der Flexibilität und schnelleren Entwicklungszeiten sehen. Unbestritten erkennt eine statische Prüfung von Signaturen viele Konsistenzfehler bereits frühzeitig, aber ebenso sollte man sehen, daß wirklich teure Fehler meist keine Signaturinkonsistenzen sind, sondern eher auf mangelhaftes Design zurückzuführen sind. Wägt man den Vorteil statischer Typsicherheit in Java gegen die damit verbundene Aufblähung der Sprache mit all ihren Folgeaufwänden ab, kann man durchaus zu einem relativierten Urteil über diesen Vorteil gelangen. Zudem kann in Java eine volle statisch überprüfte Typsicherheit nicht garantiert werden. Die recht häufig erforderlichen *statischen* Typecast-Operationen von einem allgemeinen zu einem spezialisierten Typ (z.B. von Object zu einer dedizierten Collection) müssen *dynamisch* verifiziert werden. Man findet sich damit in derselben dynamisch typisierten Welt wieder, die Smalltalk auschließlich anbietet.

Auch aus dem Blickwinkel der Typisierungsaspekte betrachtet bietet die Aufteilung des programmiersprachlichen Grundcurriculums auf die Sprachen Java *und* Smalltalk für die Studierenden Vorteile. Sie lernen die Vorzüge und Nachteile beider Ansätze aus eigener Anschauung kennen. In ihrer weiteren beruflichen Entwicklung können sie – sollten einmal diesbezügliche Fragen oder Entscheidungen anliegen – mit fundierten Argumenten aus eigener Erfahrung ideologiefrei diskutieren.

3. Zeitlicher Umfang und inhaltliche Schwerpunkte

Der Einfachheit halber sei hier ein Auszug aus der WEB-Seite eines der Autoren wiedergegeben, welche zeitlichen Umfang und stoffliche Schwerpunkte für die ersten beiden Semester wiedergibt [2].



Programmiertechnik (1. Semester)

Die Veranstaltung findet im ersten Semester Softwaretechnik statt.
Umfang: 6 SWS Vorlesung + 2 SWS Praktikum.

Lernziel: Objektorientierte Methodik in Programmierung und Design.

Stoffübersicht:

Grundlagen der Programmierung:

- **Einführung**
  - Entwicklung von Smalltalk (35 Jahre Smalltalk)
  - Klassen, Objekte, Methoden, Message Passing
- **Basis-Sprachelemente**
  - Grundbegriffe (Literale, Variable, Zahlen, Nachrichtenarten)
  - elementare Kontrollstrukturen (Selektion, Iteration)
  - elementare Programmiertechniken
  - Basisklassen (Object, Boolean, Magnitude, String)

- **Vererbung**
  - Polymorphie, Generizität
  - Abstrakte Klassen und Methoden
  - Frameworktechniken
- **Collections und Streams**
  - Array, Interval, OrderedCollection, SortedCollection
  - Set, Bag, Dictionary
  - Streams, Iterationen in Collections
- **Blöcke**
  - Lexical Closures
  - Kontrollstrukturen und Blöcke
- **Elementare Algorithmen**
  - Sortieren, Suchen, Bäume
- **Testen**
  - Teststrategien (Unit Tests, Functional Tests, Regressionstests)
  - eXtreme Programming
- **GUI-Techniken**

Konzeptionelle Inhalte:

- **Struktur objektorientierter Programmiersprachen**
  - Wertsemantik, Referenzsemantik
  - Gleichheit, Identität, Kopieren
  - Virtuelle Maschine, Speicherverwaltung
  - objektorientierte Programmiersprachen im Vergleich
- **Reflections**
  - Metasprachliche Konzepte und Werkzeuge
  - Generische Techniken
  - perform:, doesNotUnderstand:, become:
- **Ausgewählte Designpattern**
  - Observer, MVC
  - Template, Iterator, Factory, Proxy
- **Designkonzepte**
  - Liskov Substitution
  - Aggregation, Delegation
  - CRC, Law of Demeter



Programmiertechnik (2. Semester)

Die Veranstaltung findet im zweiten Semester Softwaretechnik statt.
Umfang: 3 SWS Vorlesung + 1 SWS Praktikum.

Lernziel: Objektorientierte Softwareentwicklung in Java

Stoffübersicht:

- **Allgemeine Java-Konzepte**
  - Java und Smalltalk
  - JDKs, Entwicklungsumgebungen
- **Elementare Sprachelemente und Programmierung**
- **Typisierungskonzepte von Java (Cast-Problematik)**

- **Interfaces**
- **Standardklassen (Collections, Streams, Files, etc.)**
- **Widget-Bibliotheken (AWT, Swing) und GUIs**
- **Reflections in Java**

Auffallend ist das relativ große Zeitdeputat im ersten Semester von 6 Semesterwochenstunden (SWS) Vorlesung und 2 SWS Praktikum. Das Praktikum ist mit 4 SWS zweiwöchentlich bzw. vierwöchentlich (im 2. Semester) organisiert.

Inhaltlich werden im ersten Semester zwei Schwerpunkte gesetzt. Der eine umfaßt den notwendigen klassischen Kanon von Grundlagen der Programmierung, der zweite beinhaltet konzeptionelle Themen, hauptsächlich Designfragen und -Strategien aber auch allgemeine Konzepte von Programmiersprachen. Eine Auswahl der wichtigsten Entwurfsmuster steht ebenso auf dem Programm wie generische Techniken zur Frameworkerstellung. Das MVC-Pattern – ohnehin ursprünglich in Smalltalk entstanden – wird ausführlich behandelt, sowie die darauf basierende GUI-Technologie. Im 2. Semester steht allein Java als Programmiersprache im Vordergrund. Auf der konzeptionellen Ebene kommen nur noch Typisierungsfragen zusammen mit Interfaces hinzu.

Der Schwerpunkt der Lehre liegt deutlich auf den konzeptionellen Inhalten. Für die Vermittlung programmiertechnischer Grundlagen braucht nur relativ wenig Zeit veranschlagt zu werden – dank der Vorteile von Smalltalk. Ziel ist es schließlich Informatiker und nicht nur Programmierer auszubilden. Dies wird auch den Studienanfängern bereits sehr frühzeitig im ersten Semester verdeutlicht.

4. Erfahrungen mit Smalltalk im ersten Semester

Der recht umfangreiche Kanon inhaltlicher Themen des ersten Semesters kann trotz des relativ üppigen Zeitrahmens nur durch strikte Konzentration auf die wirklich wesentlichen Dinge bewältigt, und das heißt von den Studierenden auch rezipiert werden. Smalltalk mit seiner vergleichweisen minimalen und orthogonalen Struktur schafft die erforderlichen Freiräume dafür. Für technische Fragen der Programmierung (z.B. Syntax, elementare Kontrollstrukturen, etc.) braucht nur wenig Zeit aufgewendet zu werden.

Die starke Betonung von Design- und konzeptionellen Fragen (ca. zwei Drittel des Zeitrahmens im ersten Semester) kann insofern problematisch werden, als daß die handwerklichen Basisfähigkeiten vernachlässigt werden. Es besteht durchaus die Gefahr, daß die Studenten für eine gegebene Aufgabe zwar eine gute Modellierung skizzieren, jedoch nicht in der Lage sind, einfache technische Programmierprobleme wie Filehandling oder Stringverarbeitung zu meistern. Dieser Gefahr muß bewußt gegengesteuert werden. Daher wurde angestrebt, immer zwei Arten von Aufgaben parallel zu stellen:

- rein handwerkliche Aufgaben zum Training programmiertechnischer Fertigkeiten
- und konzeptionelle Modellierungsaufgaben

Bei Modellierungsaufgaben war entweder ein einziges Thema über das ganze Semester hinweg zu bearbeiten, mit von Termin zu Termin wachsendem Schwierigkeitsgrad. Oder es wurden kleinere Aufgaben aus unterschiedlichen Bereichen gestellt. Auf jeden Fall lag am Ende des Semesters eine umfangreichere Aufgabenstellung bearbeitet vor, welche handwerkliche als auch konzeptionelle Fertigkeiten erforderte und mit einer kleinen graphischen Benutzungsschnittstelle abgeschlossen war.

5. Erfahrungen mit Java im zweiten Semester

Im zweiten Semester steht ausschließlich Java-Technologie im Mittelpunkt. Bei konzeptionellen Fragen kann voll aus dem ersten Semester zitiert werden. Die Einführung in Java findet so statt, daß zunächst die programmiersprachlichen Ausdrucksmittel tabellarisch gegenüber gestellt werden, zusammen mit typischen Programmierbeispielen, implementiert in beiden Sprachen. Die erste praktische Java-Aufgabe besteht darin, eine bereits in Smalltalk bearbeitete und voll verstandene Aufgabe in Java nachzuimplementieren. Dabei sollen die wichtigsten Grundstrukturen der Sprache und die Entwicklungswerkzeuge studiert werden. Die Erfahrung zeigt, daß das Gros der Studenten in ca. einem Drittel der Semesterzeit zufriedenstellende Grundfertigkeiten in Java vorweisen kann. Die restliche Zeit dient der Vertiefung und Einübung der allerwichtigsten Java-Komponenten (s.o.). Bewußt nicht thematisiert werden Dinge wie Applets, Servlets oder Threads. Letztere werden am besten nach ausführlicher Vorbereitung im Zusammenhang mit Betriebssystemen behandelt.

6. Diskussion des kombinierten Smalltalk/Java-Konzeptes

Ein naheliegender Einwand gegen die Zweisprachlichkeit in der Grundausbildung könnte sein, daß bei Konzentration auf eine einzige Sprache, z.Zt. Java, ebenfalls Freiraum für die Vertiefung konzeptioneller Fragen entsteht und zudem die Java-Fertigkeiten stärker trainiert würden, was wiederum die kommerziellen Abnehmer der Studierenden womöglich begrüßen würden. Dem wäre entgegenzuhalten, daß Informatikausbildung nicht primär auf die aktuellen Tagesbedürfnisse der Industrie zugeschnitten sein sollte. Jedoch wesentlicher: natürlich lassen sich prinzipiell alle Designmöglichkeiten in Java realisieren und diskutieren – schließlich ist die Sprache turingmächtig. Aber es macht einen wesentlichen didaktischen Unterschied, ob Konzepte sauber und auf ihren essentiellen Kern reduziert vermittelt werden oder aufgrund von Sprachzwängen eher abschrecken, man denke etwa an Proxies oder Call-Back-Mechanismen bei der Eventverarbeitung. Jedenfalls ist unser Resumee nach zwei Semestern Grundausbildung:

- Objektorientiertes Denken und die darauf basierenden Entwurfsmuster sind für alle Anfänger bekanntermaßen die Hauptschwierigkeit. Die Minimalität und Flexibilität von Smalltalk erlauben es von Anfang an, sich voll auf diese Themen ohne hinderlichen Sprachballast zu konzentrieren.
- Wenn diese Konzepte erst einmal verinnerlicht sind, bedeutet das Erlernen der Sprachkonzepte von Java keinen großen Aufwand mehr und kann relativ zügig erfolgen.

- Deshalb sind qualitative Unterschiede hinsichtlich der Java-Expertise nach dem zweiten Semester gegenüber nur in Java ausgebildeten Studierenden kaum feststellbar.

Zudem ist es ein nicht zu unterschätzender Bildungsvorteil für Anfänger, mit zwei teilweise zwar ähnlichen aber doch auf unterschiedlichen Voraussetzungen basierenden Sprachen konfrontiert zu werden. So kann bereits früh im Studium die wichtige Erfahrung gemacht werden, daß objektorientiertes Denken nicht an eine bestimmte Sprache gebunden ist. Und ferner kann die Zweissprachlichkeit einer leicht möglichen falschen Einschätzung vorbeugen, daß eine einzige, gerade im Mittelpunkt des allgemeinen Interesses stehende Programmiersprache das Maß aller Ding sei.

Auf zwei weitere möglichen Probleme sei noch hingewiesen, eines der Lernenden und eines der Lehrenden. So können schwächere Studierende mitunter durchaus Probleme bekommen, zwei Ausbildungssprachen in einem Jahr zu verkraften. Besonders beim Wechsel nach dem ersten Semester kann es Schwierigkeiten geben: gerade stellen sich die ersten Erfolge in der einen Sprache ein und dann – so das Gefühl der betroffenen Studierenden – geht es wieder von vorne los.

Das Problem für die Lehrenden auf der anderen Seite sollte ebenfalls erwähnt werden. Einen Programmierkurs für zwei Sprachen zu konzipieren erfordert zunächst den Erwerb der notwendigen Expertise, sodann Vorbereitung, harmonische Abstimmung des Materials etc. Insgesamt ist dies zweifellos mit erheblich höherem Aufwand verbunden als sich auf eine einzige Sprache zu beschränken.

7. Lehre objektorientierter Konzepte im weiteren Studium

Die Vermittlung objektorientierter Konzepte gilt nach den ersten beiden Semestern als weitgehend abgeschlossen. Danach stehen andere Lehrinhalte im Vordergrund. Beispielhaft sei aber noch das Verhalten der Studierenden in zwei Folgeveranstaltung erwähnt. In den höheren Semestern, im Projektstudium, bei Studien- und Diplomarbeiten herrscht bei den Studierenden ohnehin eine geringe Neigung vor, das Thema der Programmierwerkzeuge überzudiskutieren. Das Bewußtsein über Vor- und Nachteile ist ausreichend geschult. Eingesetzt wird ohne ideologische Scheuklappen das, was dem Projekt aktuell am meisten nutzt.

In *Algorithmen und Datenstrukturen* (3. Semester) spielen objektorientierte Designfragen eine wesentlich größere Rolle als dies früher der Fall war. Die Programmiertechnik selber tritt in den Hintergrund. Sie wird einfach vorausgesetzt. Die Wahl des Sprachwerkzeugs wird freigestellt. Die Erfahrung ist: leicht mehrheitlich wird von vornherein Smalltalk gewählt. Die anderen beginnen zunächst mit Java. Sobald gegen Mitte des Semesters die Zeit knapp wird, schwenken die meisten zu Smalltalk zurück. Diese Verhalten zeigt, daß die angestrebte Expertise in beiden Programmiersprachen auch erreicht und konkret angewandt wird.

Im *Software-Engineering* (3. und 4. Semester) wird die semantische Modellierung der Anforderungen an das System mittels objektorientierter Methoden vermittelt. Im begleitenden Praktikum werden in Gruppenarbeit kleinere Projekte aus Technologie-

bereichen bearbeitet, die gegenwärtig in der Wirtschaft intensiv erkundet und in innovativen Anwendungen auch benutzt werden. Typische Beispiele sind: Nutzung eines Palm-PDA als Zugangsgerät für ein Data-Warehouse, das auf einem Windows-Server lokalisiert ist. Oder: Zugriff auf ein Informationssystem durch ein WAP-Mobiltelefon. Entscheidend bei diesen Projekten ist, dass in der begleitenden Vorlesung die zu benutzenden Technologien bewußt kaum erwähnt, geschweige denn erläutert wurden. Wie in Projekten in der Praxis sollten sich die Studierenden in diese Technologien einarbeiten. Die Einarbeitung erfordert stets eine erhebliche Grundkompetenz in objektorientierten Programmiersprachen, Sprachkonzepten und Denkweisen. Die Studierenden haben aufgrund ihrer in den ersten Semestern erworbenen Kompetenz eine entscheidende Grundlage für die erfolgreiche und kurzfristige Einarbeitung in die neuen Themen erworben. Dies ist nur möglich, weil den Studierenden bereits von Beginn des Studiums an Konzepte anstelle von vordergründigen Produktkenntnissen vermittelt werden.

8. Typische Meinungen von Studenten

Im Hinblick auf eine Evaluierung unseres Konzeptes haben wir Studierenden am Ende des vierten Semesters einen Fragebogen vorgelegt. Es seien hier nur wenige Äußerungen von Studierenden zu einigen der Fragen wiedergegeben, um die in diesem Beitrag formulierten Ansichten aus Sicht der Lernenden zu illustrieren. Für eine umfassende, statistisch signifikante Auswertung war ohnehin die Grundgesamtheit zu klein.

Frage: *Halten Sie die Kombination von Smalltalk und Java für gelungen?*
Antworten:

(1) „Ja, da Smalltalk sehr gelungen ist für den Einstieg und Java später durchaus das Mittel der Wahl ist."

(2) „Ja, die Idee gefällt mir. Außerdem ist es immer gut, alles von unterschiedlichen Perspektiven kennen zu lernen. Und die Sprachen haben es auch verdient, gelernt zu werden."

Frage: *Würden Sie es begrüßen, wenn noch eine weitere Programmiersprache (z.B. C++) gelehrt würde?*

(1) „Ja, würde ich begrüßen, zumindest C++ auf freiwilliger Basis."

(2) „Nein. Zumindest nicht, wenn es sich um eine weitere objektorientierte Sprache handelt. Man würde nur eine neue Syntax lernen und keine neuen Konzepte."

Frage: *Halten Sie Ihre handwerklichen Programmierfähigkeiten für ausreichend oder vermissen Sie etwas?*

(1) „Halte ich für ausreichend. Das muss ich sowieso ständig neu oder weiter lernen. Und das Erlernen einer Programmiersprache darf kein Problem sein."

(2) „Eigentlich ausreichend. Es fehlen nur noch Details, die man sich aber auch selber aneignen kann, z.B. Servlets, Applets, ..."

Zusammenfassung

Objektorientierte Konzepte lassen sich zweifellos mit unterschiedlichen Programmiersprachen erfolgreich vermitteln. Unser Ziel, die konzeptionellen Inhalte der Objektorientierung sowie Designfragen in den Vordergrund der Informatikausbildung zu stellen, ohne das kleine Einmaleins handwerklicher Fähigkeiten zu vernachlässigen erscheint uns über den Weg von Smalltalk am effektivsten erreichbar zu sein. Wichtig erscheint uns auch, daß der Einstieg in die objektorientierte Methodik direkt am Studienanfang erfolgt. Die Denkgewohnheiten der Studierenden können so bereits sehr frühzeitig positiv geprägt werden. Erfahrungsgemäß lassen sich schlechte Gewohnheiten später nur noch schwer korrigieren.

Zusammenfassend läßt sich sagen, daß der höhere Aufwand für Lernende und Lehrende gerechtfertigt ist. Die Studierenden haben am Ende des zweisemestrigen Grundkurses solide handwerkliche Programmierkenntnisse in Smalltalk und in Java erworben, und sie haben bereits sehr früh im Studium die Umsetzung objektorientierter Konzepte in zwei verschiedenen Sprachen kennengelernt.

Literatur

1. Böhm, M., Freytag, J., Owsnicki-Klewe, B., Pfeiffer, G., Raasch, J.: Objektorientierung in der Informatikausbildung auf der Basis von Smalltalk. Informatik-Spektrum 20:335-343, 1997
2. http://www.informatik.fh-hamburg.de/~pfeiffer/