Unterschiede im Lernerfolg von Schülerinnen und Schülern in Abhängigkeit von der zeitlichen Reihenfolge der Themen (OOP-First bzw. OOP-Later)

Albrecht Ehlert, Oberstufenzentrum Informations- und Medizintechnik, ehlert@oszimt.de
Carsten Schulte, Freie Universität Berlin, schulte@inf.fu-berlin.de

Abstract: Unter den vielen fachdidaktischen Vorschlägen zum Unterrichten der Objektorientierten Programmierung (OOP) im Informatik-Anfängerunterricht werden bei dieser empirischen Untersuchung zwei Wege verglichen, der sog. OOP-First-Einstieg mit dem OOP-Later-Einstieg. Dabei werden zwei Schulklassen ein Schuljahr lang gleichartig in Informatik beschult (nur die zeitliche Reihenfolge der Themen variiert) mit der zentralen Forschungsfrage:
Sind (signifikante) Unterschiede bei den Lernerfolgen festzustellen?
In diesem Beitrag werden einige ausgewählte Ergebnisse dargestellt.

1 Einführung

Obwohl das objektorientierte Paradigma seit über einem Jahrzehnt in den Informatik-Unterricht in Schulen und Hochschulen Einzug gehalten hat, gibt es bis zum heutigen Tag eine offene Diskussion über angemessene didaktisch-methodische Konzepte zur Vermittlung des Paradigmas. Besonders diskutiert wird die Frage, ob die Idee der „objects first“ sinnvoll ist oder nicht. In dieser Diskussion werden viele verschiedene didaktische Dimensionen angesprochen: Angemessene Werkzeuge, Unterrichtsbeispiele, Reihenfolge und Schwerpunkte der Themen, Lehr- und Lernformen (vgl. [Di07], [Ba08], [BK03], [Sp05]). Die Vielfalt und innere Abhängigkeit der diskutierten didaktischen Aspekte verhindert, dass die zentrale Frage „objects first“ oder „objects later“ zu lösen ist. Hier fehlen eine Definition der verfolgten Ansätze sowie eine Abschätzung, welche Aspekte den zentralen Unterschied ausmachen.

In diesem Artikel werden wir zunächst eine solche Definition analytisch entwickeln und durch eine empirische Analyse verschiedener Lehrbücher überprüfen. Anschließend werden aufbauend auf dem so definierten Kernunterschied die beiden Ansätze in einem empirischen Experiment mit zwei Schulklassen in Bezug auf den Lernerfolg verglichen. Dazu werden wir den Aufbau, die Durchführung und die Analyse einer entsprechenden empirischen Studie beschreiben. Anschließend werden einige Ergebnisse in Bezug auf die Konsequenzen für die Schulpraxis und auf weiterführende fachdidaktische Forschungsfragen interpretiert.

2 Von objects-first zu OOP-First

objects first vs. objects later. Barnes und Kölling [BK03] umreißen den „objects-first-Schlachtruf“ folgendermaßen: „Ein Student kann als erste Aktivität ein Objekt erzeugen und seine Methoden aufrufen! Weil Benutzer Objekte direkt erzeugen und manipulieren können, können Konzepte wie Klassen, Objekte, Methoden und Parameter ohne weiteres direkt diskutiert werden, bevor die erste Zeile Java-Quelltext betrachtet werden muss.“ ([BK03], deutsche Ausgabe von 2003, S. 18) Ob diese Idee sinnvoll ist, ist sehr umstritten. So gab es auf der Mailingliste der SIGCSE 2004 einen ausführlichen Streit um Vor- und Nachteile des objects-first-Vorgehens, der hinterher von Fachdidaktikern ausgewertet wurde (vgl. Li06). Es wurde deutlich, dass tatsächlich viele unterschiedliche didaktische Dimensionen angesprochen werden und dass viele der Argumente auf individueller Anschauung und Lehr-Erfahrung beruhen.
In der Zwischenzeit hat es bereits einige empirische Studien gegeben, die den objects-first-Ansatz vergleichend untersuchen. Decker [De03] kommt zum Schluss, dass der objects-first-Ansatz überlegen ist. Allerdings hat sie den Ansatz mit einem sogenannten „Weniger Objekte“-Ansatz verglichen und vor allem bezüglich der so vermittelten objektorientierten Kenntnisse untersucht – da ist es wenig verwunderlich, dass in der Gruppe, in der mehr Zeit für Objektorientierung aufgewendet wurde auch mehr Wissen zum Thema entstanden ist. Andererseits hat sie auch längerfristige Vorteile in nachfolgenden Veranstaltungen festgestellt, und dass die Betonung der Objektorientierung keine Nachteile in den prozeduralen Wissensbeständen ergeben hat. Reges [Re06] kommt dagegen in einem empirischen Vergleich zu dem Schluss, dass die traditionelle Art des Unterrichtens dem objects-first-Ansatz deutlich überlegen sei: Bessere Leistungen, bessere Zufriedenheit der Lernenden, bessere Bewertung des Lehrenden, weniger Abbrecher. Allerdings vergleicht er von ihm selbst durchgeführte Kurse mit denen seines Vorgängers, den die Universität gefeuert hat, um ihn mit der ausdrücklichen Maßgabe anzustellen, die Anfangskurse didaktisch zu verbessern. Da sein Vorgänger im Gegensatz zum ihm einen objects-first-Ansatz vertreten hat, sieht er hier die Hauptursache – es könnte allerdings auch an anderen Dingen liegen: Möglicherweise ist das didaktische Konzept insgesamt besser, der Lehrende fähiger, die Studenten besser (oder besser motiviert), die gestellten Anforderungen geringer, etc.
Diese Studien machen verschiedene Anforderungen an eine empirische Untersuchung des Ansatzes deutlich (und liefern dazu gute Anregungen): A) Wie kann der objects-first-Ansatz angemessen mit einer Alternative verglichen werden? Und B) Welches sind geeignete Messzeitpunkte und Vergleichskriterien?
Ein Aspekt scheint besonders bedeutsam: Was ist der genaue Unterschied der objects-first-Idee im Vergleich zum „traditionellen Unterrichten“?

Unterschiede von objects first und objects later. In einer internationalen Studie, an der auch viele deutsche Informatiklehrerinnen und -lehrer teilgenommen haben, wurden drei Varianten des objects-first-Begriffs unterschieden [BS07]:

1. Objekte benutzen: Wie oben im Zitat angedeutet, werden zunächst vorhandene Objekte benutzt und manipuliert bevor implementiert wird.
2. Klassen schreiben: Von Anfang an werden Klassen definiert, implementiert und instanziiert, um das Paradigma zu vermitteln. Frühe Programmiererfahrungen mit dem Objektorientierten Paradigma stehen im Mittelpunkt.

3. Konzepte: In Bezug auf die Diskussion in Deutschland könnte man auch vom objektorientierten Modellieren sprechen. Zunächst werden die abstrakten und generellen Ideen des Paradigmas vermittelt, wobei die objektorientierten Modelle im Mittelpunkt stehen.

Gemeinsam ist den verschiedenen Varianten, dass das objektorientierte Paradigma an den Anfang gerückt wird. Die folgende Abbildung veranschaulicht diesen Zusammenhang am Beispiel der Steuerstruktur Iteration:

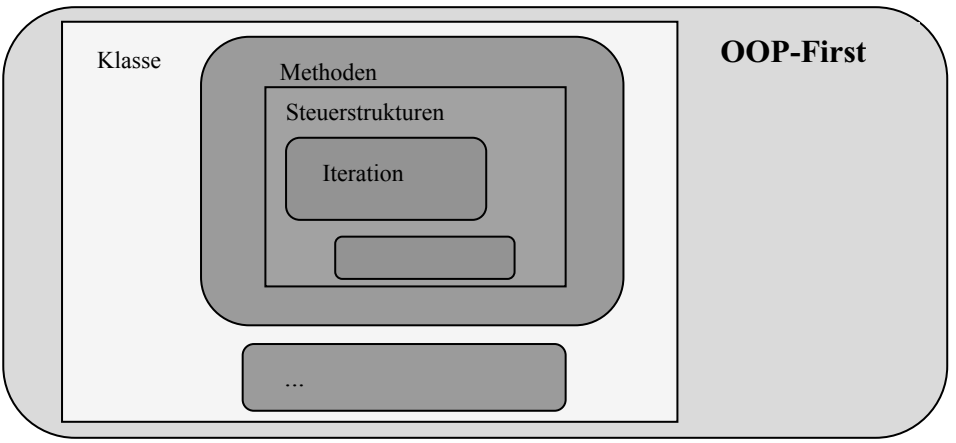

Abbildung 1: Thema Iteration beim OOP-First-Ansatz

Zuerst werden mit Hilfe von Klassen Objekte erzeugt und benutzt. Dabei werden entweder vorgefertigte Klassen benutzt, Klassen implementiert und instanziiert, oder Klassen modelliert, implementiert und instanziiert. Ja nach Variante werden also mehr oder weniger stark die im Klassen-Konzept enthaltenen Themen wie z.B. Attribute und Methoden allgemein eingeführt, um dann später diese zu konkretisieren (Methode -> Steuerstruktur -> Sequenz, Selektion, Iteration).

Dagegen wird beim objects-later-Ansatz in der Objektorientierten Programmierung auf vorhandenes Wissen aus der Prozeduralen Programmierung zurück gegriffen:

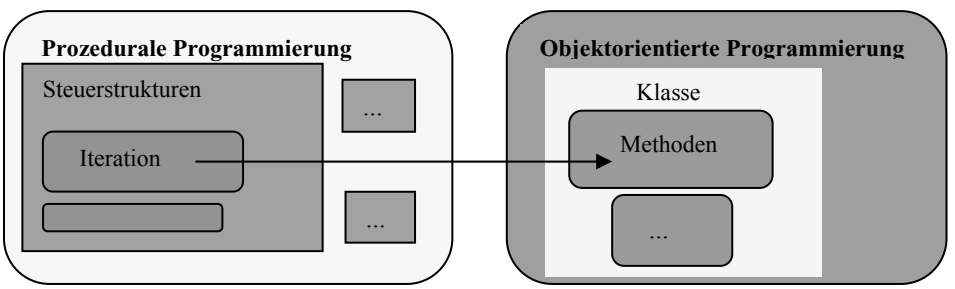

Abbildung 2: Thema Iteration beim OOP-Later-Ansatz

Die Abbildung 2 veranschaulicht, dass beim objects-Later-Ansatz beim Thema „Methoden einer Klasse“ z.B. auf das vorhandene Wissen über die Steuerstruktur Iteration aufgebaut werden kann.
Bei diesem Ansatz gibt es für den Unterrichtenden Freiheitsgrade, es stellt sich nämlich für ihn im Verlauf des Informatik-Unterrichts die Frage: Wann ist der Punkt gekommen, um von der prozeduralen Sichtweise auf die objektorientierte Sichtweise umzusteigen?

Gleichartige Themen bei beiden Ansätzen. Diese Diskussion zeigt, dass der wesentliche Unterschied vom objects-first-Ansatz zum objects-later-Ansatz in der Reihenfolge der Lernthemen – und nicht in den verwendeten Werkzeugen oder Unterrichtsmethoden – zu suchen ist (Wobei natürlich je nach Reihenfolge Werkzeuge anders in den Unterricht eingebettet werden können bzw. müssen).
Daher können für beide Ansätze gleichartige Themen benannt werden [Eh07]**.** Dazu wurden verschiedene Programmier-Lehrbücher (mit den verschiedensten Ansätzen) im Hinblick auf die einzelne Programmier-Themen verglichen. Im Ergebnis finden sich immer wieder gleichartige Kapitel bzw. Sequenzen, kleine Unterschiede ausgenommen: Mal firmieren die Steuerstrukturen im Kapitel „Anweisungen“, mal entdeckt man die fundamentalen Datentypen im Unterkapitel „Variablen und Konstanten“ und manchmal findet sich ein Zusatzthema, z.B. „Testen“ (vgl. z.B. [KS07][Er05][Ba08]).
Diese gleichartigen Themen sind:
- Einstieg in die OOP, Klasse und Objekt
- Variablen (bzw. Attribute), Konstanten und fundamentale Datentypen
- Steuerstrukturen: Sequenz, Iteration und Selektion
- Prozeduren (bzw. Operationen, Methoden, Funktionen, Botschaften)
- Komplexe(re) Datentypen
- Vererbung etc.
- Assoziation etc.

Obwohl beide Wege auf den ersten Blick sehr unterschiedlich sind, enthalten sie doch im Endeffekt die gleichen Themen, z.B. die Implementierung von Klassen und die Erzeugung von Objekten, genauso wie die Themen Daten- und Steuerstrukturen. Nur die Sichtweise ist eine andere:
1. Das objektorientierte Paradigma steht von Anfang an im Mittelpunkt des Anfänger-Programmier-Unterrichts (OOP-First), die Themenfolge ist oft wie folgt:
- Klasse und Objekt
- Attribut (inkl. Datentypen)
- Methode (inkl. Steuerstrukturen)
- Vererbung
- Assoziation

2. Das objektorientierte Paradigma baut auf das prozedurale Paradigma auf (OOP-Later), die Themenfolge ist oft wie folgt:
- Variable, Konstante, einfache Datentypen
- Steuerstrukturen: Sequenz, Selektion, Iteration
- Prozedur (Methode)
- Klasse und Objekt

Die Vorgehensweisen, die auf den ersten Blick so unterschiedlich wirken, sind aber in der Summe ihrer Themen gleich, nur die zeitliche Abfolge der Themen ist eine andere!

OOP-First und OOP-Later. Im Folgenden soll der Begriff **OOP-First** verwendet werden, wenn im Anfangsunterricht gleich mit dem Begriff Objekt bzw. Klasse eingestiegen wird und der Schwerpunkt eher in der Programmierung bzw. Implementierung als in der Modellierung liegt (dies ist auch die Abgrenzung zu objects first, OOM und OO-First). Instruktionspsychologisch lässt sich hier im Sinne einer Top-Down-Argumentation positiv anführen, dass den Lernenden die Ziele und Anwendungsmöglichkeiten schnell deutlich werden, auf die sie hin arbeiten. Dies könnte sich emotional günstig auswirken.
Dagegen wird der Begriff **OOP-Later** gesetzt, wenn ein Teil der prozeduralen Themen vor dem Einstieg in die OO-Themen erfolgt und auch hier wieder der Schwerpunkt eher in der Programmierung bzw. Implementierung liegt [Eh07]. Instruktionspsychologisch könnte man mit dem Vorteil eines systematischen Wissensaufbaus argumentieren.

Mit Hilfe dieser Definition kann nun (unter Fokussierung auf OOP-First und OOP-Later und der dort enthaltenen Programmier-Komponente) der objects-first-Ansatz mit dem objects-later-Ansatz – unter dem als zentral herausgearbeiteten Aspekt der unterschiedlichen Sequenzierung von Lerninhalten – verglichen werden.

3 Empirische Studie zum Vergleich von OOP-First und OOP-Later

In diesem Abschnitt werden ausgewählte Aspekte einer größeren empirischen Studie vorgestellt und diskutiert. Wir konzentrieren uns auf den Vergleich in den Lernergebnissen. Die zentrale Forschungsfrage lautet: Gibt es nach einem Jahr (signifikante) Unterschiede im Lernerfolg der Schülerinnen und Schüler der beiden Klassen? Kann man diese aufschlüsseln nach einzelnen Themengebieten?

Design der Studie. Zwei parallele Schulklassen wurden ein Jahr lang in Informatik beschult. Die Themen waren gleich, die Themenreihenfolge war aber unterschiedlich. Das zugrunde liegende Forschungsdesign der empirischen Studie ist ein kausales, welches zum Ziel hat, Zusammenhänge zwischen Variablen aufzudecken. Dazu wurde ein Experiment durchgeführt, indem sich die Eingangsvariablen (fast) nur durch die unterschiedliche Reihenfolge der Themen unterschieden. Über eine geschichtete Stichprobenauswahl wurden einzelne Schüler-Merkmale (wie Geschlecht, Alter, etc.) schon von vornherein in ihrem richtigen Verhältnis im Sample repräsentiert. Für jedes Thema wurden Fachkompetenzen formuliert und daraus Aufgaben abgeleitet, die als Indikatoren für das Erreichen der einzelnen Kompetenz dienen. Die Programmiersprache war durch schulinterne Vorgaben auf Java festgelegt, der OOP-Einstieg (ein Thema von neun Themen) erfolgte mit BlueJ [BK03], alle Programme für die anderen Themen wurden mit dem JavaEditor [Rö09] erstellt. Durch ständige Absprachen der beiden Unterrichtenden wurde dafür gesorgt, dass die Themen gleichartig und gleichlang unterrichtet wurden. Die Gleichartigkeit der Klassen (im Hinblick auf das mathematische Verständnis) wurde durch den selben Mathematik-Lehrer in beiden Klassen überprüft. In einer Vorstudie (Schuljahr 2006/2007) wurde das „Setting“ der Studie erprobt, damit in der Hauptstudie tatsächlich bis auf die Themenreihenfolge alle anderen Variablen möglichst gleichartig sind.

Der Lernerfolg wurde am Ende des Schuljahres mittels eines Tests (Post-Test bzw. Vergleichstest) über ca. 120 Minuten ermittelt: OOP-abhängige und OOP-unabhängige Themen waren dabei mit ca. 50% vertreten. Für den Vergleich eventueller Unterschiede in der Nachhaltigkeit wurde ein vergleichbarer Test 8 Wochen nach dem letzten Test (also nach den Sommerferien 2008) als Follow-Up-Test bzw. Nachhaltigkeitstest geschrieben.
Die verschiedenen Fragebögen wurden aus den Forschungsfragen entwickelt und überarbeitet bzw. über die Vorstudie evaluiert.

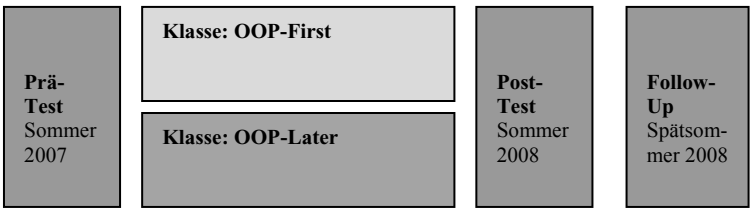

Abbildung 3: Forschungsdesign (Hauptstudie)

Durchführung der Studie. Die Studie wurde mit zwei Schulklassen in der Einführungsphase des Beruflichen Gymnasiums am OSZ Informations- und Medizintechnik (www.oszimt.de) in Berlin durchgeführt. Damit gelten die folgenden Rahmenbedingungen:

- die Schülerinnen und Schüler haben in der Regel keine Programmier-Erfahrungen und sind eher leistungsschwach,
- die Schülerinnen und Schüler sollen neben der Modellierung auch relativ schnell real programmieren (hier: Java),
- die Schülerinnen und Schüler sollen in einem Schuljahr (à 3h / Woche) von Datentypen, Steuerstrukturen bis zur Vererbung in der OOP sowohl imperative als auch objektorientierte Sichtweisen kennengelernt haben.

Für die Schülerinnen und Schüler ist die Einführungsphase in der gymnasialen Oberstufe das erste Schuljahr (11. Klasse) an der Schule. Daher konnten vorher die Klassen für den Vergleich nach Geschlecht (in der Regel männlich), Alter (ca. 17 Jahre), Vorbildung (in der Regel Realschulabschluss) und Mathematik- und Deutsch-Note gleichartig zusammengesetzt werden.
Die Vorstudie lief im Schuljahr 2006/2007, die Hauptstudie ist nach dem Schuljahr 2007/2008 beendet worden.
Die Themen in beiden Klassen waren gleichartig, nur die Reihenfolge der Themen war eine andere. Die insgesamt neun Themenblöcke wurden aus den übereinstimmenden Themen der Programmier-Fachliteratur gewonnen (s. Seite 4), es kam natürlich noch der Einstieg in die IDE hinzu. Es wird dabei mit Absicht von Themen und nicht von Modulen gesprochen, da die Art und Weise der Vermittlung der einzelnen Themen auch abhängig ist von den vorher unterrichteten Themen.

Die OOP-First-Klasse startete gleich mit den OOP-Fachtermini wie Klasse, Objekt, Attribut und Methode:

Reihenfolge	Thema
1	T1: Einführung in die OOP (mit BlueJ)
2	T2: Einführung in den Java-Editor, Attribute (Variablen), fundamentale Datentypen, Steuerstruktur Sequenz
3	T3: Klasse und Objekt
4/5*	T4: Methoden
5/4*	T5: Steuerstruktur: Selektion (inkl. Struktogramm)
6	T6: Steuerstruktur: Iteration (inkl. Struktogramm)
7	T7: Komplexere Datentypen: Arrays und Strings
8	T8: Vererbung (inkl. abstrakte Klassen und abstrakte Methoden)
9	T9: Assoziationen (inkl. Klassenattribute und -methoden)

Abbildung 4: Reihenfolge der OOP-First-Themen
(*linke Zahl: Reihenfolge Vorstudie, rechte Zahl: Reihenfolge Hauptstudie)

Die OOP-Later-Klasse fokussierte sich dagegen erst einmal (ca. ½ Schuljahr) auf prozedurale Themen inklusive den Daten- und Steuerstrukturen:

Reihenfolge	Thema
1	T2: Einführung in den Java-Editor, Variablen (Attribute), fundamentale Datentypen, Steuerstruktur Sequenz
2	T5: Steuerstruktur: Selektion (inkl. Struktogramm)
3	T6: Steuerstruktur: Iteration (inkl. Struktogramm)
4	T7: Komplexere Datentypen: Arrays und Strings
6/5*	T1: Einführung in die OOP (mit BlueJ)
7/6*	T3: Klasse und Objekt
5/7*	T4: Methoden
8	T8: Vererbung (inkl. abstrakte Klassen und abstrakte Methoden)
9	T9: Assoziationen (inkl. Klassenattribute und -methoden)

Abbildung 5: Reihenfolge der OOP-Later-Themen
(*linke Zahl: Reihenfolge Vorstudie, rechte Zahl: Reihenfolge Hauptstudie)

Bei der letzten Darstellung erkennt man schon, dass Lehren aus der Vorstudie gezogen wurden: Das Thema „Prozeduren“ wurde jetzt nicht mehr vor dem OOP-Einstieg als „statische Methoden“, sondern erst nach dem OOP-Einstieg als „Methoden“ unterrichtet.

Prä-, Post- und Follow-Up-Test. Für den Vergleich des Lernerfolgs wurde in beiden Klassen nach einem Jahr der selbe Test (Vergleichstest bzw. Post-Test) über 120 Minuten geschrieben. In der Vorstudie wurden die fünf Themen Daten- und Steuerstrukturen, Struktogramme, Methoden, „statische OOP“, UML und das Zusatzthema „dynamische OOP“ abgefragt. Beim letzten Thema erhielten die Schüler Quelltext eines Anwendungsprogramms, in dem verschiedenen Objekte erzeugt und auf ihnen verschiedene Methoden aufgerufen wurden. Die Schülerinnen und Schüler sollten dann die dynamische Entwicklung der Objektzustände nachvollziehen können. In der Hauptstudie wurden alle neun unterrichteten Themen einzeln abgefragt inklusive zweier Zusatzthemen („dynamische OOP“ und OOM: aus einer Problembeschreibung sollten Klassen modelliert werden). OOP-abhängige (UML, Klasse, Vererbung etc.) und OOP-unabhängige Themen (Datentypen, Steuerstrukturen, Struktogramme etc.) waren dabei mit jeweils ca. 50% vertreten. Maximal 30% waren Multiple-Choice-Aufgaben.

4 Ergebnisse

Vorstudie. Bei einem ersten Probedurchlauf im Schuljahr 2006/2007 waren die objektiven Lernergebnisse (ermittelt durch den Post-Vergleichstest am Ende des Schuljahrs) bei der OOP-Later-Klasse teilweise signifikant besser, gerade im Hinblick auf die OOP-Themen! [ES07]

Thema	Punkte	OOP-First	OOP-Later
Daten- und Steuerstrukturen	20	61%	59%
Methoden	20	67%	64%
Struktogramme	20	60%	64%
„Statische OOP“	20	72%	79%
UML-Klassendiagramme und -Objektdiagramme	20	**72%***	**85%***
Zusatz: „Dynamische OOP“	10	**50%***	**67%***
Gesamt	110	65%	70%

Abbildung 6: Ergebnisse des Vergleichstests (Post-Test) in der Vorstudie 2006/2007
(*signifikante Ergebnis-Unterschiede sind fett gedruckt)

Auffällig war auch, dass die OOP-First-Klasse mit den Themen schneller vorangekommen ist, so dass ein kurzes Projekt das Schuljahr abschloss. Dies kann ein Vorteil des OOP-First-Wegs sein, evt. aber auch die Erklärung dafür, warum die Schüler schlechter abgeschnitten haben. Vielleicht „verführt“ der OOP-First-Einstieg zu einem schnelleren Vorgehen. Eine andere Erklärung ist, dass Schüler das zuletzt Gelernte am Besten abrufen können. Dies waren bei der OOP-Later-Klasse ja die OOP-Themen!

Hauptstudie. In der Hauptstudie 2007/2008 wurde der Vergleichstest umgestaltet. Die einzelnen unterrichteten Themengebiete finden sich jetzt besser wieder:

Thema	Punkte	OOP-First	OOP-Later
T2: Datentypen und Steuerstruktur Sequenz	10	6,8	7,0
T5: Steuerstruktur Selektion	10	6,3	7,1
T6: Steuerstruktur Iteration	10	5,8	6,0
T7: Arrays und Strings	10	**5,7***	**3,7***
T4: Methoden	10	6,1	4,8
T1/T3: Einführung in die OOP / Klasse und Objekt	20	14,6	14,9
T8: Vererbung	10	7,6	6,9
T9: Assoziation	10	**6,2***	**4,9***
Zusatz: Dynamische OOP	10	7,1	7,0
Zusatz: Objektorientierte Modellierung (OOM)	10	6,7	5,9
Gesamt	110	72,9	68,2
Mathematik-Note (Punkte)		*8,8*	*7,8*

Abbildung 7: Ergebnisse des Vergleichstests (Post-Test) in der Hauptstudie 2007/2008
(*signifikante Ergebnis-Unterschiede sind fett gedruckt)

Das etwas bessere Gesamt-Abschneiden der OOP-First-Klasse könnte einfach durch die besseren Mathematik-Leistungen dieser Klasse erklärt werden, eine Korrelation zwischen Mathematik- und Informatik-Note voraussetzend. Der Unterschied bei dem Thema „Methoden“ ist zwar auffällig, aber nur die Ergebnisse bei den Themen „Arrays

und Strings“ und „Assoziation“ sind signifikant unterschiedlich. Vor einer Bewertung dieser Leistungs-Unterschiede sollen aber erst einmal die Ergebnisse des Nachhaltigkeits- bzw. Follow-Up-Tests (ca. 8 Wochen später) dargestellt werden:

Thema	Punkte	OOP-First	OOP-Later
T2: Datentypen und Steuerstruktur Sequenz	10	6,4	7,2
T5: Steuerstruktur Selektion	10	6,5	7,4
T6: Steuerstruktur Iteration	10	5,7	5,4
T7: Arrays und Strings	10	3,6	3,8
T4: Methoden	10	5,3	4,6
T1/T3: Einführung in die OOP / Klasse und Objekt	20	13,4	13,0
T8: Vererbung	10	5,8	6,0
T9: Assoziation	10	2,5	1,6
Zusatz: Dynamische OOP	10	5,3	5,1
Zusatz: Objektorientierte Modellierung (OOM)	10	5,3	6,3
Gesamt	110	59,8	60,4

Abbildung 8: Ergebnisse des Nachhaltigkeitstests (Follow-Up-Test) in der Hauptstudie 2007/2008
(Es gibt keine signifikanten Ergebnis-Unterschiede)

Hier fallen die deutlichen Einbrüche der Ergebnisse der OOP-First-Klasse auf, sowohl im Endergebnis als speziell auch bei den vorher signifikant unterschiedlichen Themen-Ergebnissen. Bei der Ursachenforschung wurde schnell klar: Beim Vergleichstest waren die Klassen von den beiden beteiligten Informatik-Lehrern sehr unterschiedlich vorbereitet worden! Daher sind die deutlich objektiveren Ergebnisse jene des Nachhaltigkeitstest (auf den überhaupt nicht vorbereitet wurde) und sollen deshalb im Folgenden diskutiert werden.

Ergebnis1. Es gibt keine Unterschiede zwischen dem OOP-First- und dem OOP-Later-Vorgehen bei der Nachhaltigkeit! Sowohl was das Gesamtergebnis als auch was die einzelnen Themen anbelangt, ergeben sich keine signifikanten Unterschiede. Beide Vorgehensweisen produzieren auf lange Sicht vergleichbare Lernerfolge.

Die Ergebnisse für beide Klassen können gemittelt, quantitativ geordnet und dann in drei Leistungsbereiche aufgeteilt werden:

Thema	Punkte	Beide Klassen
T5: Steuerstruktur Selektion	10	7,0
T2: Datentypen und Steuerstruktur Sequenz	10	6,8
T1/T3: Einführung in die OOP / Klasse und Objekt	20 (10)	13,2 (6,6)
T8: Vererbung	10	5,9
Zusatz: Objektorientierte Modellierung (OOM)	10	5,8
T6: Steuerstruktur Iteration	10	5,6
Zusatz: Dynamische OOP	10	5,2
T4: Methoden	10	5,0
T7: Arrays und Strings	10	3,7
T9: Assoziation	10	2,1
Gesamt	110	60,3

Abbildung 9: Geordnete Ergebnisse des Nachhaltigkeitstests in der Hauptstudie 2007/2008

Wenn man die Ergebnisse im Hinblick auf OOP-abhängige und OOP-unabhängige Themen betrachtet, fällt auf, dass sich die beiden Themenarten in allen drei Leistungsbereichen wiederfinden.

Ergebnis2. Die Themen der OOP sind nicht schwerer als die prozeduralen Themen! In jeder Themenart finden sich leichtere und schwerere Themen wieder.

Der Follow-Up-Test bzw. Nachhaltigkeitstest war nicht identisch mit dem Post-Test, aber Aufgabe für Aufgabe vergleichbar. Der Prä-Test war ein stark verkürzter Post-Test und dient nur dem Nachweis, dass die beiden Klassen auf dem gleichen (tiefen) Ausgangsniveau starten. Da im Prä-Test nur die ersten 8 Themen abgefragt wurden, finden sich im folgenden Diagramm auch beim Post- und beim Follow-Up-Test nur die ersten 8 Themen wieder:

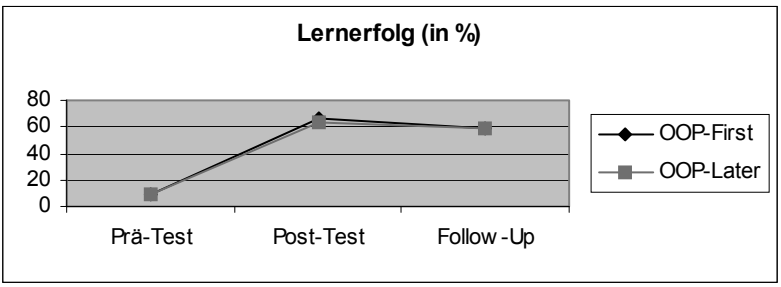

Abbildung 10: Lernerfolg in der Hauptstudie 2007/2008
(Bezogen auf die ersten 8 Themengebiete, Prä-Test erfolgte im nächsten Jahrgang parallel zum Follow-Up-Test)

Die Schülerinnen und Schüler kommen fast ohne Programmier-Vorkenntnisse in die Einführungsphase (9,8% richtige Antworten sind eher den Multiple-Choice-Aufgaben geschuldet). Nach einem Schuljahr erreichen sie ca. 64,6% des erwünschten Wissens, durch die Sommerferien vergessen sie (nur) ca. 5,8%!

Ergebnis3. Die Unterschiede in den Lernerfolgskurven sind nicht signifikant. Vom Gesamtergebnis her gesehen gibt es keine Unterschiede beim Post-Test und Follow-Up-Test (bei vergleichbarem Prä-Test) zwischen dem OOP-First- und dem OOP-Later-Vorgehen.

5 Interpretation, neue Forschungsfragen und Zusammenfassung

Interpretation. Das letzte Ergebnis könnte so interpretiert werden, dass, wenn man sich für den Informatik-Anfängerunterricht inkl. OOP-Einstieg nur genug Zeit nimmt (in diesem Fall ca. 100 Stunden), es unabhängig von den Lernstilen, den Lernmethoden, den Lerntools, der verwendeten Programmiersprache etc. den gleichen Lernerfolg gibt.

Auch wenn das zentrale Ergebnis ist, dass sich die Lernergebnisse mittelfristig nicht unterscheiden, so wird es doch bestimmt „lokale“ Unterschiede im Lernprozess geben. Dazu könnte nach jedem Thema eine Momentaufnahme gemacht werden:

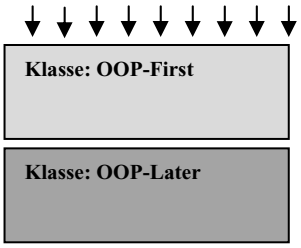

Abbildung 11: Mögliche Momentaufnahmen nach jedem Thema

Durch diese Prozessbeobachtung könnte sich dann herausstellen, dass jedes Vorgehen spezifische Vor- und Nachteile hat, z.B. bei den Themenübergängen oder bei der von den Schülern erlebten „Themenschwere“ (siehe dazu spätere Veröffentlichung).

Die Themen „Assoziation“ und „Arrays und Strings“ hatten den wenigsten Lernerfolg. Die Schwere dieser Themen könnte darin begründet sein, dass beim Thema „Arrays“ es zu einer Strukturierung eines Datentypen kommt (z.B. von int auf int-Feld) inkl. der Anwendung eines Algorithmus’ (sprich: Iteration, um die Feldstruktur zu durchlaufen). Wenn eine 1-n-Assoziation mit Hilfe eines Arrays implementiert wird, dann ist es logisch, dass das Thema „Assoziation“ von den Schülern auch als komplexer erlebt wird.

Die empirische Studie bestätigt (quasi nebenbei) auch noch einmal den Zusammenhang bzw. Unterschied von OO-first und OO-later: Es konnten neun Themengebiete definiert werden, die sowohl beim OOP-First-Ansatz als auch beim OOP-Later-Ansatz zur Geltung kamen. Nach einem Jahr sind in beiden Varianten alle 9 Themengebiete unterrichtet worden, nur die zeitliche Reihenfolge der Themen hat variiert.

Einschränkungen der Gültigkeit. Mögliche Fehl- oder Überinterpretationen sind wie folgt möglich:

- Nicht entschieden werden kann über den Einfluss der gesetzten Randbedingungen, z.B. bei Wahl anderer Werkzeuge, Programmiersprachen etc. Trotzdem ist es schon ein Erfolg, einen systematischen Vergleich geschafft zu haben. Darauf können Folgeuntersuchungen aufbauen.
- Gegebenenfalls gibt es „lokale“ Unterschiede im Prozess (siehe oben). Auch ist bisher nur die Schülergesamtheit betrachtet worden. Vielleicht gibt es aber in Teilgruppen (z.B. sehr gute oder sehr schlechte Schüler) signifikante Unterschiede beim Lernerfolg in Abhängigkeit vom Vorgehen.
- Die Studie bezieht sich auf OOP nicht auf objects first, wobei der Unterschied marginal sein mag (Hinweis: Auch bei der zusätzlichen Modellieraufgabe gab es keinen Unterschied). Größere Unterschiede könnten sich ergeben, wenn OOM-First mit OOP-Later verglichen wird. Dies ist dann aber eher eine Frage der Gewichtung von Lernzielen als eine didaktisch-methodische Frage.
- Die Testfragen waren überwiegend (79%) aus dem Anforderungsbereich II, der Anforderungsbereich III fehlte gänzlich. Quelltext-Entwicklung erfolgte auf dem Papier, nicht am Rechner.
- Die Probandenzahl der beiden Schulklassen betrug 40 Schülerinnen und Schüler.

Neue Fragestellungen. Aus der Studie lassen sich weitere Fragen entwickeln, z.B.:

- Was passiert, wenn einzelne Variablen geändert werden, z.B. die Programmiersprache (Java) oder das OOP-Einstiegstool (BlueJ)?
- Was passiert, wenn das sequentielle OOP-First-Vorgehen auf ein spiralförmiges OOP-First-Vorgehen umgestellt wird?

Zusammenfassung. Die Fragestellung braucht in Zukunft nicht mehr zu lauten, mit welchem Paradigma die Lehrerin bzw. der Lehrer in den Informatik-Anfängerunterricht einsteigt, da bezogen auf ein Schuljahr (mit ca. 100-120h) die Klassen fast das Gleiche lernen. Der Unterrichtende muss sich vielmehr Gedanken machen, warum bestimmte Themen aus welchen Gründen von den Schülern als schwer erlebt werden bzw. mangelhaft gelernt werden und warum bestimmte Themenübergänge Schwierigkeiten bereiten. Auf den Informatik-Unterricht des OSZ IMT bezogen könnte man z.B. das Thema „Assoziationen“ im ersten Schuljahr gänzlich herausnehmen und mit der gewonnenen Zeit das vertiefende (sprich: längere) Unterrichten von „Arrays und Strings“ bewirken.

Literaturverzeichnis

[Ba08] Balzert, Helmut: Java, der Einstieg in die Programmierung
W3L GmbH 2008, 2. Auflage, ISBN 978-3-868-34000-6

[BK03] Barnes D.J., Kölling M.: Java lernen mit BlueJ
Pearson Education Deutschland, 2003, 3. Auflage, ISBN 0-13-197-629X

[BS07] Bennedsen, Jens; Schulte, Carsten: What does “Objects-First” Mean? An International Study of Teachers’ Perceptions of Objects-First. In: Lister, R. und Simon, Hrsg. (2007), Koli Calling 2007, Finland, ACS., http://crpit.com/confpapers/CRPITV88Bennedsen.pdf

[De03] Decker, A.: A tale of two paradigms.
J. Comput. Small Coll. 19, 2 (Dec. 2003), 238-246.

[Di07] Diethelm, Ira: Strictly models and objects first, Dissertation
Kassel, Universität, 2007, urn:nbn:de:hebis: 34-2007101119340

[Eh07] Ehlert, Albrecht: Studie: Objects-First- und Objects-Later-Einstieg
Praxisband der 12. Fachtagung "Informatik und Schule - INFOS 2007", Seite 17 – 20, Herausgeber: Stechert P., Universität Siegen 2007, ISBN 978-3-936533-23-1

[Er05] Erlenkötter, Helmut: C++ - Objektorientiertes Programmieren von Anfang an
Rowohlt Tb.; Auflage: Erw. N.-A. (Mai 2005), ISBN 978-3499600777

[ES07] Ehlert A.., Schulte C.: Learners Views on Objects-First and Objects-Later -
Results of an Exploratory Study, Report on the 11th Workshop TLOOC at ECOOP 2007
Herausgeber: Börstler J., Hadar I., ISBN 978-3-540-78194-3

[KS07] Krüger Guido, Stark Thomas: Handbuch der Java-Programmierung
Addison-Wesley, München; November 2007, ISBN 978-3827323736

[Li06] Lister, R., Berglund, A., Clear, T., Bergin, J., Garvin-Doxas, K., Hanks, B., Hitchner, L., Luxton-Reilly, A., Sanders, K., Schulte, C., and Whalley, J. L. 2006. Research perspectives on the objects-early debate. SIGCSE Bull. 38, 4 (Dec. 2006), 146-165

[Re06] Reges, S. 2006. Back to basics in CS1 and CS2. In Proceedings of the 37th SIGCSE, SIGCSE '06. ACM, New York, NY, 293-297

[Rö09] Röhner, Gerhard: JavaEditor, ’09, http://lernen.bildung.hessen.de/informatik/javaeditor/

[Sp05] Spolwig, Siegfried.: Karel D. Robot – Der Delphi Karel, OSZ Handel, Berlin, 2005
http://www.oszhandel.de/gymnasium/faecher/informatik/delphi_karel/index.htm (3.5.09)