Vorkurse in objektorientierter Programmierung: Lösungsansatz für einen leichteren Einstieg in die Informatik

Marc Berges

Fachgebiet Didaktik der Informatik
TU München
85748 Garching - Germany
Web: http://www.ddi.tum.de
Email: berges@tum.de

Peter Hubwieser

Fachgebiet Didaktik der Informatik
TU München
85748 Garching - Germany
Web: http://www.ddi.tum.de
Email: peter.hubwieser@tum.de

Zusammenfassung: Die Studienanfänger der Informatik haben in Deutschland sehr unterschiedliche Grundkenntnisse in der Programmierung. Dies führt immer wieder zu Schwierigkeiten in der Ausrichtung der Einführungsveranstaltungen. An der TU München wird seit dem Wintersemester 2008/2009 nun eine neue Art von Vorkursen angeboten. In nur 2,5 Tagen erstellen die Teilnehmer ein kleines objektorientiertes Programm. Dabei arbeiten sie weitestgehend alleine, unterstützt von einem studentischen Tutor. In dieser Arbeit sollen nun das Konzept der sogenannten „Vorprojekte“ sowie erste Forschungsansätze vorgestellt werden.

1 Einleitung

Die Studenten, die jedes Jahr an den deutschen Hochschulen ihr Informatikstudium beginnen, starten mit sehr unterschiedlichen Kenntnissen in der Programmierung. Insbesondere das Wissen in der objektorientierten Programmierung (OOP) differiert sehr stark. Dies führt immer wieder zu großen Schwierigkeiten in den Einführungsveranstaltungen des ersten Semesters. Einige Studenten verfügen bereits über Programmierkenntnisse in imperativen Sprachen und müssen sich an das objektorientierte Paradigma erst gewöhnen. Andere kommen komplett ohne Vorkenntnisse und müssen zu den vielen neuen Konzepten, die vermittelt werden, auch noch eine Programmiersprache erlernen. An der Technischen Universität München wird daher seit dem Wintersemester 2008/2009 ein völlig neuer Weg beschritten. Vor dem Semesterbeginn wird ein freiwilliger Vorkurs für Studienanfänger angeboten. Unter der Leitung eines studentischen Tutors lernen die zukünftigen Studenten in sehr kurzer Zeit die einfachsten Grundlagen der OOP, ohne dabei einen Vortrag zu hören. Der Fokus liegt unter anderem auf der Überwindung der Angst vor dem selbstständigen Programmieren.

2 Beschreibung der Vorprojekte

Das in der Einleitung beschriebene Problem der Heterogenität der Studienanfänger bezüglich der Programmierkenntnisse ist auch an der TU München stark vorhanden. Im Zuge der Einführung der Studienbeiträge wurden vom Fachgebiet Didaktik der Informatik Vorkurse in objektorientierter Programmierung eingeführt.

Die Frage, wann und in welchem Maße objektorientierte Programmierung Inhalt einer Einführungsveranstaltung in der Informatik sein muss, wird in diversen Arbeiten ausgiebig behandelt [WH1, DT1]. Um möglichst früh in die theoretischen Konzepte einzusteigen, ist ein gewisses Handwerkszeug nötig, damit diese Konzepte auch praktisch erprobt werden können. Diese Werkzeuge können nicht im Rahmen der Einführungsveranstaltung in die Informatik vermittelt werden.
Das Institut für Informatik hat deswegen bereits der Einführungsveranstaltung ein Praktikum angegliedert, in dem die Konzepte praktisch erprobt werden können. Damit dieses Praktikum nicht zu einem bloßen Programmierkurs verkommt, ist es nötig, einige Programmierkenntnisse in das Studium mitzubringen. Dies wiederum soll aber kein Aufnahmekriterium für das Informatikstudium sein. Es muss also sichergestellt werden, dass einfaches Programmieren von den Studienanfängern beherrscht wird, was den Rahmen für die Vorprojekte bildet.
Ein weiteres Problem ist die Überschneidung solcher Vorkurse mit der jeweiligen Einführungsveranstaltung. Auch in unseren Vorprojekten sollten keine wesentlichen Inhalte vorweg genommen werden. Der erste Schritt in diese Richtung wurde bereits mit der Konzeption der Arbeitsblätter übernommen. Zum zweiten wurden die Veranstaltungen auf 2,5 Tage reduziert. Wie sich herausstellte, war das genug Zeit, um erste Programmieraufgaben zu lösen, aber sicherlich zu wenig, um ganze Konzepte zu vermitteln.
Das Konzept der Vorprojekte war von Beginn an mehrschichtig aufgebaut. Zum einen soll den beginnenden Studenten die Angst vor dem selbstständigen Lernen und Arbeiten genommen werden, zum anderen sollen die Studenten möglichst mit einem Grundlevel an Programmierkenntnissen in das Studium starten. Um nicht eine komplette Einführungsveranstaltung zu ersetzen, sollen die Studienanfänger nur die wirklich notwendigsten Konzepte gezeigt bekommen. Das selbstständige Ausprobieren steht ganz zentral im Mittelpunkt. Das besondere an den Vorprojekten ist, dass die angehenden Studenten keinen Vortrag zu hören bekommen. Ihnen werden lediglich vier Arbeitsblätter[1] gegeben, mit denen sie sich ihre ersten Programmierkenntnisse erwerben sollen. Damit hat jeder die Chance, seinen individuellen Lernweg zu beschreiten.
Als ganz zentrales Element wird jeder Gruppe, die aus acht bis zehn Studenten besteht, ein studentischer Tutor zugewiesen, der für Fragen zur Verfügung steht. Diese werden zumeist im Rahmen des etablierten Tutorenbetriebs am Institut für Informatik der TU München angeworben. Dies hat entscheidende Vorteile. Diese Tutoren haben eine didaktische Schulung bekommen [SB1] und verfügen über einige Erfahrung in der Lehre, da sie in den Basisvorlesungen eingesetzt werden. Dadurch sind sie mit den Problemen der Studienanfänger sehr vertraut. Außerdem hat sich der enge Kontakt der angehenden Studenten zu erfahrenen Studenten als sehr wichtig erwiesen.
Ein weiteres zentrales Element ist die Individualität des Lernweges. Durch die unterschiedlichen Vorkenntnisse ist eine möglichst individuelle Betreuung unbedingt notwendig. Diese fördert das selbstständige Umsetzen des Gelernten und ermöglicht so ein optimales Lernen. In der „Cognitive Load Theory“, u.a. vorgestellt in [CB1], wird davon ausgegangen, dass effektives Lernen nur dann möglich ist, wenn die Kapazität des Arbeitsgedächtnisses möglichst optimal ausgenutzt wird. Dabei ist es vor allem wichtig, dass die Elemente Muster aus ähnlichen Inhalten bilden. Das praktische „Ausprobieren“, das in den Vorprojekten ganz besonders gefördert wird, unterstützt diese Musterbildung und vereinfacht so die Übernahme ins Langzeitgedächtnis.

1 http://vmhub1.informatik.tu-muenchen.de/publikationen/material/vorprojekte.pdf

Damit die individuelle Hilfe auch auf optimalem Niveau erfolgen kann ist es nötig, möglichst homogene Gruppen zu bilden. Dazu werden die Teilnehmer bei der Anmeldung um eine Selbsteinschätzung ihrer Programmierkenntnisse gebeten.
Es kann zwischen folgenden Kategorien gewählt werden:

1 – keine Programmierkenntnisse

2 – schon mal programmiert

3 – schon mal objektorientiert programmiert

2.1 Konkrete Zahlen der ersten beiden Durchgänge

In Tabelle 1 sind die Anmeldezahlen, sowie die Verteilung auf die „Kenntnis"-Gruppen tabellarisch aufgelistet. Außerdem wird der Anteil der Teilnehmer an der Anzahl der Studienanfänger in den Bachelor-Studiengängen Informatik und Wirtschaftsinformatik dargestellt. Der Rückgang der Anmeldungen ist auf zwei Gründe zurückzuführen. Erstens sind die Studierendenzahlen leicht gesunken. Außerdem wurden die Studienanfänger der Bioinformatik nicht mehr explizit angeschrieben, da die Einladung im Rahmen der Immatrikulation an der TU München stattfindet. Die Bioinformatiker sind aber an der LMU München eingeschrieben. Trotzdem haben sich weniger Teilnehmer als im ersten Jahr angemeldet.

Tabelle 1: Aufschlüsselung der Teilnehmerzahlen

Semester	Anmeldungen	Gruppe 1	Gruppe 2	Gruppe 3	Anteil an Studienanfängern
WS08/09	200	114	42	44	53%
WS09/10	135	80	30	25	39%

3 Die Arbeitsblätter

Da in den Vorprojekten auf Vorträge jeglicher Art verzichtet wird, bekommen die verteilten Arbeitsmaterialien und die Tutoren eine noch gewichtigere Rolle. Auf das Tutorensystem wurde bereits im vorherigen Abschnitt ausführlich eingegangen. In diesem Abschnitt sollen nun die Arbeitsblätter näher betrachtet werden. Zunächst müssen allerdings einige theoretische Grundlagen geklärt werden.

3.1 Programmieren

Damit der Rahmen der Arbeitsblätter festgelegt werden kann, muss zunächst geklärt werden, was Programmieren lernen eigentlich heißt. Nach Boyer, Langevin und Gaspar heißt programmieren lehren, die Studenten zu befähigen, die Anforderungen zu analysieren und die benötigten Algorithmen zu ermitteln. Diese müssen dann in einer Programmiersprache umgesetzt und auf Korrektheit überprüft werden können [BLG1]. Ordnet man also die Kompetenz des objektorientierten Programmierens in die Taxonomie von

Anderson/Krathwohl [AK1] ein, so lassen sich die kognitiven Elemente der Wissensdimension in die Kategorie Factual Knowledge einordnen [BLG1].

3.2 Minimalkonzept der Objektorientierung

Sucht man nach Definitionen zur Objektorientierung findet man eine kaum zu überblickende Anzahl. Es gibt die verschiedensten Herangehensweisen. Henrik Christensen beschreibt in [HC1] das Problem der verschiedenen Definitionen und Herangehensweisen sehr genau. Hier werden die gefundenen Definitionen in verschiedene Perspektiven eingeteilt. Die „language centric perspective“ stellt die Klasse und das Objekt als Elemente zur Konstruktion von Software in den Mittelpunkt. Die „model centric perspective“ sieht Objekte immer im Zusammenhang mit einem Modell. Die letzte Perspektive, die „responsible centric perspective“ stellt die Interaktion der Objekte in den Vordergrund.
Der Kern der meisten Definitionen bleibt aber der gleiche. Hier soll zunächst die Definition von Hares [HS1] exemplarisch genannt werden:

> *„An object is, in terms, composed of two types of information, the data and logic information components. An object does not have to have both types of information simultaneously, and can contain data and logic or just logic. The object must contain logic, because, as we shall see, the data component cannot be accessed except via the logic component. The data component is optional.“*

Aus den verschiedenen Definitionen haben wir die wichtigsten Kernelemente herausgenommen. Ein Programm, welches der OOP entspricht, muss aus mindestens einer Klasse bestehen. Zudem müssen Objekte instanziiert werden. Ein weiterer sehr wichtiger Aspekt ist die Datenkapselung. Daraus resultiert, dass zumindest eine Methode in der Klasse enthalten sein muss.
Für die Konzeption unserer Arbeitsblätter haben wir uns an unserem Minimalkonzept und der „language centric perspective“ orientiert. Diese beiden Elemente bilden mit den nötigen Elementen der Programmiersprache Java die Inhalte der vier Arbeitsblätter, die an die Studenten verteilt werden. Im nächsten Abschnitt werden die Arbeitsblätter nun im Detail beschrieben.

3.3 Aufbau der Arbeitsblätter

Die benötigte Information für die Programmierprojekte wird auf vier Arbeitsblätter komprimiert. Die vier Elemente des Programmierenlehrens aus [BLG1], nämlich „Anforderungsanalyse“, „Algorithmenermittlung“, „Implementierung“ und „Prüfung auf Korrektheit“, geben der Rahmen der Arbeitsblätter vor. Die Prüfung auf Korrektheit der jeweiligen Implementierungen wird allerdings nicht explizit thematisiert, da die Zeit der Vorkurse zu knapp ist. Außerdem würde dies zu sehr dem Inhalt der Einführungsveranstaltung vorweggreifen.
Auf dem ersten Arbeitsblatt werden die groben Ideen hinter der Objektorientierung erklärt. Dabei liegt der Fokus im Verstehen des Objektbegriffs. Attribute und Methoden bilden die zentralen Inhalte. Konzepte wie Vererbung und Polymorphie werden gänzlich ausgelassen. Diese würden ebenso den Inhalten der Einführungsveranstaltung zu stark vorweggreifen und den engen Zeitrahmen sprengen.
Nach dem theoretischen Teil über Objekte sollen die Teilnehmer ihre frisch erworbenen Kenntnisse anhand des Programms ObjektDraw einsetzen. ObjectDraw wurde von Martin Papst zur Unterstützung der Einführung in die Objektorientierung an der Schule entwickelt. Hier können Grafiken als Objekte erzeugt werden und durch Methodenaufrufe

beeinflusst werden. Desweiteren werden alle Objekte durch Objektkarten dargestellt und können so sehr gut beobachtet werden.
Als Aufgabe sollen die Teilnehmer ein Fußballfeld oder ein anderes Sportfeld grafisch darstellen. Dabei sollen immer wieder die Veränderungen der Objekte beobachtet werden. Das erste Blatt beinhaltet außerdem die Aufgabentexte zu den drei Schwierigkeitsgraden.
Auf dem zweiten Blatt steht die Einführung in die Entwicklungsumgebung BlueJ im Vordergrund. BlueJ bietet einen einfachen visuellen Zugang zu den Objekten [BBK1]. Bei den Aufgaben auf Blatt 2 haben wir uns an den Vorschlägen von Kölling und Rosenberg aus [KR1] orientiert. In einem ersten Schritt sollen die Teilnehmer den Umgang mit BlueJ anhand des Beispielprojekts Shapes erproben. Danach sollen sie aus dem Aufgabentext die Klassen, Attribute und Methoden extrahieren und in BlueJ anlegen. Um die Orientierung im Quellcode zu vereinfachen, ist dem zweiten Blatt eine Beispielklasse angefügt.
Im dritten Blatt folgt nun der Einstieg in die Java-Programmierung. Dazu sind auf dem Blatt die wichtigsten Java-Konstrukte erklärt. Außerdem wird die Syntax grafisch dargestellt (siehe Abbildung 1). Die Darstellung entstammt [MW1]. Mit den Elementen auf dem dritten Blatt lassen sich bereits die Methoden und Attribute der Klassen anlegen.

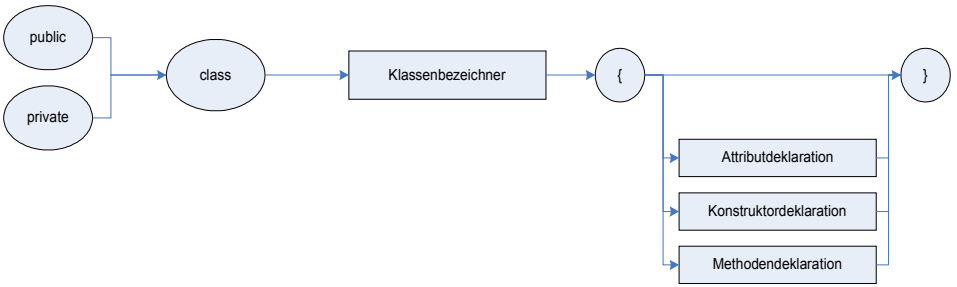

Abbildung 1: Grafische Darstellung eines Java-Konstrukts (class)

Das Füllen der Methodenrümpfe ist Inhalt des vierten Arbeitsblattes. Zu Beginn werden den Teilnehmern zwei Möglichkeiten gezeigt, Algorithmen zu modellieren. Danach werden die fehlenden Java-Konstrukte erklärt. Damit haben die Teilnehmer alle wichtigen Sprachelemente zusammengefasst dargestellt, um die Aufgabe zu lösen.

4 Einschätzung der Teilnehmer

Neben der Qualität des Codes, der von den Teilnehmern erstellt wurde, ist die Selbsteinschätzung von großer Bedeutung. Eine der zentralen Aufgaben der Vorprojekte ist es, die Angst vor dem selbstständigen Programmieren zu nehmen. Ein weiteres Ziel, das wir erreichen wollen, ist, dass die angehenden Studenten sehr früh daran gewöhnt werden, sich weiterführende Informationen selbstständig zu besorgen. Dafür muss ihnen natürlich der Weg dorthin gezeigt wird. Dies wird durch die individuelle Betreuung mit Hilfe der Tutoren gewährleistet.

Die Fragebögen sind in sechs Teile aufgeteilt und wurden anonym ausgefüllt. Im Folgenden sollen die einzelnen Teile näher beschrieben und ausgewertet werden. Die dargestellten Ergebnisse beziehen sich dabei auf den Durchgang 2009/2010.

4.1 Personendaten

Der Fragebogen erfasst in seinem ersten Abschnitt Personendaten wie Alter und Geschlecht. Besonders das Geschlechtsmerkmal ist zur Untersuchung unter Gender-Aspekten sehr wertvoll.
In einer ersten Untersuchung wurde geprüft, ob der Anteil der weiblichen Teilnehmer sich vom Anteil an den Studienanfängern unterscheidet. Es hat sich gezeigt, dass sich die Verteilung von weiblichen und männlichen Teilnehmern in etwa an den Zahlen der Studienanfänger orientiert. Es ist also kein signifikanter Unterschied zwischen den Anzahlen von Teilnehmerinnen an den Vorprojekte und weiblichen Studienanfängern festzustellen.
Außerdem werden der Studiengang und die Vorkenntnisse erfasst. Um bei der Codeauswertung die Vorkenntnisse einordnen zu können, wird nach der Schulausbildung Informatik gefragt. Dieses Merkmal wird vor allem im nächsten Jahr interessant, wenn in Bayern der „Doppelte Jahrgang“ die Gymnasien verlässt. Es wird dann Schüler geben, die bereits eine schulische Grundausbildung in Informatik durchlaufen haben, und solche, denen diese Ausbildung fehlt. Hier ergeben sich interessante Forschungsfelder.
Desweiteren wird bei den Personendaten auch noch die Herkunft erfragt. Dabei gibt es die Möglichkeiten, das Bundesland anzugeben oder das Land, aus dem man kommt. Auch hier soll eine Einschätzung der informatischen Vorbildung stattfinden.

4.2 Veranstaltung

Der zweite Teil des Fragebogens beschäftigt sich mit der Veranstaltung an sich. Hier werden Merkmale wie Zufriedenheit mit Ankündigung, Organisation, Ablauf und Gruppengröße abgefragt. Diese Merkmale dienen in erster Linie der Verbesserung der Durchführung und werden deswegen hier nicht näher betrachtet.

4.3 Blätter

Im Abschnitt „Blätter“ soll vor allem die Verständlichkeit der Arbeitsblätter eingeschätzt werden.
In der ersten Frage sollen die Teilnehmer den Informationsgehalt der Arbeitsblätter auf einer Skala von 1–5 (hoch – niedrig) einstufen. 84% haben den Informationsgehalt mit 1–3 eingestuft, wobei der Hauptanteil mit 40% bei Stufe 2 liegt.
Die zweite Frage zielte auf die Verständlichkeit der Blätter. Hier ging die Skala von 1–5 (gut – schlecht). Die Verteilung der Antworten ist genau die gleiche wie bei der vorherigen Frage.
Die Ausführlichkeit der Erklärungen ist Inhalt der dritten Frage. Hier ging die Skala von 1–5 (nicht ausführlich – zu ausführlich). Auch hier haben 88% zwischen 1–3 bewertet. Der Hauptanteil liegt hier mit 61% aber auf 3 – „ausführlich“.
In der Gesamtheit sind die Blätter also durchweg positiv bewertet worden. Dies deckt sich auch mit den Ergebnissen der Codeanalyse, die in Abschnitt 5 präsentiert wird.

4.4 Tutor

Der Tutor stellt einen ganz zentralen Aspekt des Konzepts dar. Von ihm hängen sowohl der Erfolg der Teilnehmer wie auch die Selbstständigkeit der Teilnehmer ab. Zum Tutor wurden zwei Fragen gestellt.
Die erste Frage zielt darauf ab, wie hilfreich die Anwesenheit empfunden wurde. Die Skala reicht von 1–5 (sehr – gar nicht). Die Antworten waren zu 95% im Bereich 1–3, wobei 65% die Stufe 1 angaben. Die Präsenz eines Tutors ist also von den Teilnehmern als extrem wichtig eingestuft worden. Dies deckt sich auch mit unseren Beobachtungen während der Durchführung.
Dies wird auch durch die zweite Frage gestützt, die auf die Verständlichkeit der Erklärungen des Tutors abzielt. Hier reicht die Skala von 1–5 (verständlich – nicht verständlich). 92% gaben die Stufen 1–3 an, wobei der Hauptanteil von 60% bei Stufe 1 lag.

4.5 Selbsteinschätzung

Der fünfte Abschnitt ist der wohl mit Abstand interessanteste. Hier wurde nach der Selbsteinschätzung der gelernten Inhalte gefragt.
Die erste Frage ermittelt, wie viel die Teilnehmer glauben gelernt zu haben. Hier reicht die Skala von 1–5 (viel – wenig). Die Ergebnisse ballen sich mit 88% wieder in den Stufen 1–3, wobei diesmal der größte Anteil mit 57% auf Stufe 2 entfällt.
Mit der zweiten Frage sollten die Teilnehmer ihre hinzugewonnenen Javakenntnisse einschätzen. Die Skala war hier umgedreht von 1–5 (nicht vorhanden – selbstständiges Programmieren). 41% der Teilnehmer haben sich in Stufe 3 eingeordnet. Immerhin 11% trauten sich nach der Veranstaltung zu, selbstständig zu programmieren.
Die letzte Frage zielt auf das Verständnis des objektorientierten Konzepts. Hier reicht die Skala von 1–5 (verstanden – nicht verstanden). Wie auch die Codeanalyse im nächsten Abschnitt zeigen wird, wurden die Grundkonzepte der Objektorientierung von 95% der Teilnehmer verstanden. Dennoch ist diese Zahl mit Vorsicht zu genießen, da sowohl der Tutor als auch die Entwicklungsumgebung, die bereits einen großen Teil vorgegeben hat, eine gewichtige Rolle gespielt haben dürften.

5 Auswertungsmöglichkeiten des Codes

In einem ersten Schritt haben wir den Code der Teilnehmer analysiert, die ohne Vorkenntnisse in die Veranstaltung gegangen sind. Dies geschah vor allem deswegen, da hier offensichtlich eine sehr homogene Gruppe bezüglich der Vorbedingungen vorliegt. Um sicherzustellen, dass die richtigen Ergebnisse untersucht werden, haben wir die Fragebögen mit Nummern versehen. Diese Nummern sollten von den Teilnehmern als Kommentar in den Quellcode geschrieben werden. Dadurch ist die Anonymität gewährleistet und die Vorbedingungen wie Geschlecht, Herkunft und Vorkenntnisse können im Nachhinein zugeordnet werden.
Für den Erfolg der Vorprojekte gibt es zwei Kriterien. Der erste ganz wichtige Punkt, der untersucht werden muss, ist die Frage, ob das Konzept überhaupt angenommen wurde. Ein Anhaltspunkt ist der Fragebogen, der im vorherigen Abschnitt ausführlich besprochen wurde.
Ein anderer Aspekt ist die Frage, welche und wie viele Teile der Arbeitsblätter umgesetzt wurden. Zunächst sollte festgehalten werden, dass von 37 Projekten, die untersucht wur-

den, nur zwei nicht compilierbar waren. Einschränken muss man diese Aussage nur durch die Tatsache, dass die benutzten Entwicklungsumgebungen eine große Hilfe beim Erstellen von compilierbarem Code sind.
Weitaus interessanter ist die Tatsache, dass 26 von 37 Projekten eine sinnvolle Ausgabe auf vorher bestimmte Testszenarien gaben. Dafür wurden die Programme mit zwei Eingaben getestet, einem Normalfall und einem Spezialfall. Der Spezialfall wurde dabei weit weniger berücksichtigt als der Normalfall.
Zusätzlich soll noch betrachtet werden, wie die Teilnehmer die Konzepte, die auf den Arbeitsblättern angesprochen wurden, verwendet haben.
Diese Konzepte sind im Einzelnen:

C1) **Anordnung von Attributen, Konstruktoren und Methoden:** Auf dem zweiten Arbeitsblatt gibt es eine Beispielklasse, die die Anordnung der einzelnen Elemente demonstriert.
C2) **Initialisierung der Attribute mit Standardwerten:** Im Text wird explizit darauf hingewiesen, dass bereits im Konstruktor die Attribute initialisiert werden sollten. Darüber hinaus sollten generell alle Attribute initialisiert werden. Eine besondere Herausforderung stellen dabei die Arrays dar.
C3) **Definition eines eigenen Konstruktors mit Parametern:** Im Abschnitt über Konstruktoren wurde auch darauf eingegangen, dass dieser überladen werden kann.
C4) **Verwendung eines Rückgabewertes bei Methoden:** Haben die Teilnehmer das „return"-Statement benutzt oder haben sie die Rückgabe mit globalen Variablen realisiert?
C5) **Verwendung von Parametern im Methodenkopf:** Haben die Teilnehmer Parameter im Methodenkopf benutzt oder haben sie nur globale Variablen verwendet?
C6) **Korrekte Verwendung von Zugriffsmodifikatoren:** Nicht alle Methoden und Attribute müssen public sein. Die Datenkapselung ist ein zentrales Element der Objektorientierung.
C7) **Verwendung von Feldern:** Felder wurden auf dem Arbeitsblatt extra hervorgehoben.
C8) **Verwendung von „this":** Der Gebrauch von sprechenden Attributnamen zwingt häufig dazu, die gleichen Bezeichner für Attribute und Parameter zu benutzen.
C9) **Verwendung der „main"-Methode:** Zu Beginn des Projektes sollten alle Studenten mit BlueJ arbeiten. Hier ist der direkte Zugriff auf die Objekte ohne eine „main"-Methode möglich. Am Ende des Projektes sollten die Teilnehmer dann ihren Code ohne BlueJ ausführen. Dazu ist die Implementierung einer „main"-Methode nötig.
C10) **Korrekte Verwendung von Entscheidungs-Konstrukten:** Die Teilnehmer sollten das „if"-Konstrukt korrekt mit einem „else"-Zweig benutzen, sofern dieser nötig ist. (Nicht mit zwei „if"-Konstrukten)
C11) **Verwendung von Wiederholungsschleifen:** Auf den Arbeitsblättern wurden drei verschiedene Schleifenarten vorgestellt. Diese sind „while-do", „do-while" und „for".

Wie man in Abbildung 2 sehen kann, gibt es einige Konzepte, die den Teilnehmern leichter gefallen sind, und andere, die kaum verwendet wurden. So hat nur ein Teilnehmer einen eigenen Konstruktor (C3) verwendet. Ebenso wurden kaum Rückgabewerte (C4) oder Parameter (C5) in Methoden benutzt. Dies deutet darauf hin, dass der Einsatz

globaler Variablen wohl einfacher umzusetzen ist. Die globalen Variablen wurden benutzt, ohne auf den Blättern erwähnt zu werden.
Das wohl einfachste Konzept sind die Wiederholungsschleifen (C11), die von fast allen Teilnehmern verwendet wurden. Insgesamt betrachtet wurden aber doch mehr als die Hälfte der Konzepte von den meisten Teilnehmern umgesetzt.

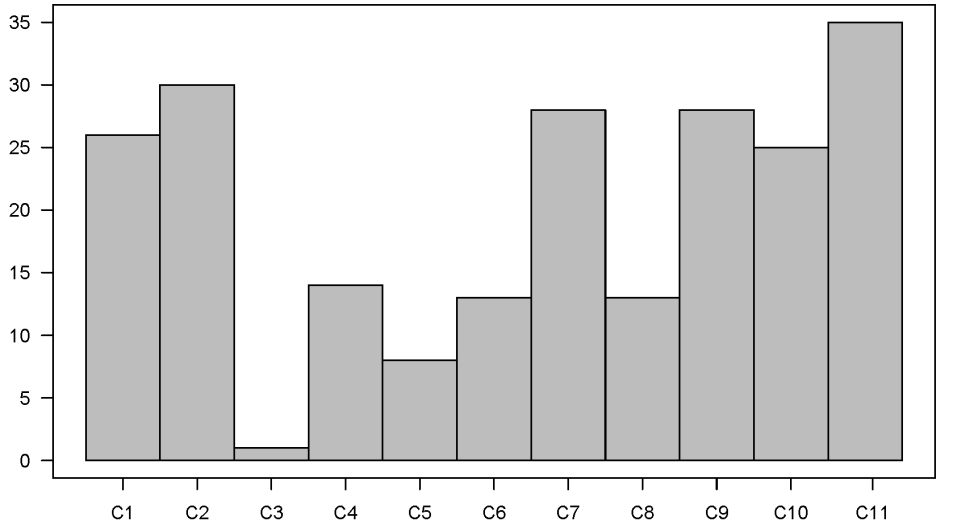

Abbildung 2: Häufigkeit der Konzepte

Der zweite wichtige Punkt ist die Auswirkung der Ergebnisse auf den Studienverlauf. Sind also die Teilnehmer ohne Vorkenntnisse im Laufe der Einführungsveranstaltung besser als diejenigen, die nicht an den Vorprojekten teilgenommen haben? Dieser Frage werden wir in den nächsten Semestern zusammen mit einer Untersuchung des „Doppelten Jahrgangs“ nachgehen.

6 Ausblick

Die allgemeine Resonanz der Teilnehmer an den Vorkursen ist durchweg positiv. Betrachtet man die Ergebnisse, die von den Teilnehmern produziert wurden, so kann man sagen, dass trotz geringer Vorkenntnisse von nahezu allen Teilnehmern ein lauffähiges Programm produziert wurde. Die Ergebnisse der ersten beiden Runden haben schon einige interessante Fragen aufgeworfen.
Im nächsten Jahr können wir die gewonnen Ergebnisse dann in einem besonders spannenden Szenario anwenden. In Bayern wird dann der sog. „Doppelte Jahrgang“ die Schulen verlassen. Hier treffen im gleichen Jahr Schüler aufeinander, die bereits mehrere Jahre Informatikunterricht gehabt hatten, wie auch solche, die noch keine oder kaum Vorkenntnisse haben [HP1]. In diesem Forschungsfeld lassen sich hoffentlich Ergebnisse erarbeiten, die auf Lernstrategien in der Informatik Rückschlüsse zulassen.

Literatur

[BBK1] J. Bergin, K. Bruce, M. Kölling: Objects-Early Tools – A Demonstration, SIGCSE 2005, St. Louis – Missouri – USA, 2005

[KR1] M. Kölling, J. Rosenberg: Guidelines for Teaching Object Orientation with Java, ITiCSE 2001, Canterbury – UK, 2001

[MW1] H. Müller, F. Weichert: Vorkurs Informatik - Der Einstieg ins Informatikstudium, B.G. Teubner Verlag – Wiesbaden, 2005

[WH1] H. M. Walker: The Role of Programming in Introductory Computing Courses, ACM Inroads Vol.1 No.2, 2010

[DT1] I. Donchev, E. Todorova: Object-Oriented Programming in Bulgarian Universities' Informatics and Computer Science Curricula, Informatics in Education 2008, Vilnius, 2008

[CB1] M. E. Caspersen, J. Bennedsen: Instructional Design of a Programming Course – A Learning Theoretic Approach, ICER 2007, Atlanta – Georgia – USA, 2007

[HC1] H. B. Christensen: Implications of Perspective in Teaching Objects First and Object Design, ITiCSE 2005, Monte de Caparica – Portugal, 2005

[HS1] J.S. Hares, J. D. Smart: Object orientation: technology, techniques, management and migration, John Wiley & Sons Ltd, Chichester, 1994

[HP1] P. Hubwieser: Functions, Objects and States: Teaching Informatics in Secondary Schools, 2nd International Conference ISSEP, Springer, Berlin, 2006

[SB1] K. Bender, M. Steinert: Ein handlungsorientiertes, didaktisches Training für Tutoren im Bachelorstudium der Informatik, HDI 2008 – 3. Workshop des GI-Fachbereichs Ausbildung und Beruf/Didaktik der Informatik, Universitätsverlag Potsdam, Potsdam, 2008

[AK1] L. W. Anderson, D. R. Krathwohl: A Taxonomy for Learning, Teaching, and Assessing – A Revision of Bloom's Taxonomy of Educational Objectives, Longman, 2001

[BLG1] N. R. Boyer, S. Langevin, A. Gaspar: Self Direction & Constructivism in Programming Education, SIGITE 2008, Cincinnati – Ohio – USA, 2008