In: Volker Claus (Hrsg). Informatik und Ausbildung, GI-Fachtagung '98, 30.3. - 1.4.1998, Stuttgart. Springer, Informatik Aktuell 1998 (S. 43-51)

Programmieren als Handwerkszeug im ersten Semester

Eva Hornecker
Forschungszentrum artec
Universität Bremen

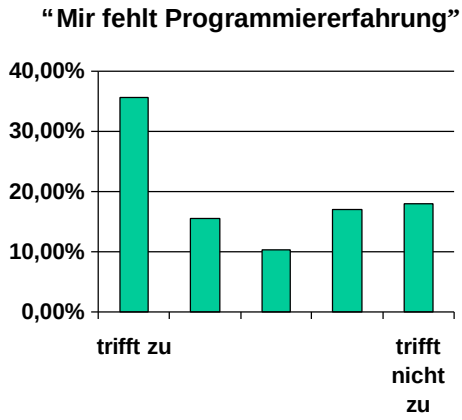
Ich verfolge in meinem Beitrag zwei Gedankenlinien zur Verbesserung der Lehre. Zum einen stelle ich das Problem der Heterogenität (Programmieranfänger und Studierende mit Vorerfahrung) dar, analysiere die speziellen Schwierigkeiten der **Programmieranfänger** und zeige Wege auf, mit dieser Situation produktiv umzugehen. Zum anderen stelle ich die Möglichkeit vor, mit Hilfe einer **Projektphase** im ersten Semester Motivation zu wecken, Gruppenerlebnisse und Teamfähigkeit zu fördern sowie inhaltliches Lernen in einer konzentrierten Arbeitsphase zu intensivieren.

Dieser Beitrag beruht im wesentlichen auf meiner Diplomarbeit [Ho95], in der ich Inhalte und Ablauf des ersten Semesters an der Technischen Hochschule Darmstadt analysierte und neu konzipierte. Dazu begleitete ich eine Veranstaltung [Kre96] (u.a. Fragebogenaktion, Interviews). Viele Vorschläge orientierten sich zwar an konkreten Gegebenheiten, dennoch läßt sich vieles verallgemeinern. Insbesondere die Idee einer Projektphase wurde umgesetzt und mittlerweile als voller Erfolg gewertet.

1 Programmieren: Implizit verlangtes Vorwissen oder explizit gelehrtes Handwerkszeug?

Entgegen dem allgemeinen Versprechen, daß für das Studium der Informatik nur Interesse an Mathematik und analytisch-logisches Denkvermögen Voraussetzung seien, erleben Programmieranfänger als implizit verlangtes Vorwissen jedoch Programmierkenntnisse. Dies fängt an mit der Fachsprache der Lehrenden, mit den „Fach"gesprächen unter Studierenden, mit Aufgabenstellungen, die erst interpretiert werden müssen, und endet mit objektiv feststellbaren Schwierigkeiten beim Program-mieren-Lernen, mit den Aufgaben und damit bei Scheinerwerb und Studien-motivation.

Trotz großen Bemühens der Lehrenden in der von mir begleiteten Veranstaltung gaben die Programmieranfänger in der Fragebogenaktion an, schlechter mit ihrer Zeit auszukommen, fanden, daß der Stoff zu schnell behandelt werde, und zweifelten stärker an ihrer Studienwahl, als dies der Rest tat [Ho95, S. 67 - 69]. Sie stimmten insbesondere der Aussage „Mir fehlt Programmiererfahrung" fast ausnahmslos zu. Die Aussage, „Mir fehlen Grundlagen für die Informatik-Vorlesung", wurde von 15 der 22 Programmieranfänger bejaht, von 121 der 177 Nicht-Anfänger jedoch abgelehnt (die Korrelation betrug -0,4). Dies zeigt deutlich, daß Programmierkenntnis als wichtige Grundlage empfunden wird.



Eine damit einhergehende Schwächung des Selbstvertrauens und Erhöhung der Abbruchquoten konstatieren verschiedene Studien (z.B. [Kay etal 89]). Leider sind die ohnehin wenigen Frauen häufiger in dieser Situation, die ihren Studienerfolg gleich zu Beginn gefährdet. In dem von mir untersuchten Jahrgang befanden sich nur 14 männliche Programmieranfänger, während sechs von 14 befragten Frauen in dieser Situation waren – also beinahe die Hälfte. Auch in Anzahl und Art der bekannten Sprachen gab es deutliche Unterschiede. C oder C++ kannte ein Viertel der männlichen Teilnehmer, aber keine einzige Frau. Die für Programmieranfänger entstehenden Probleme finden sich in dem relativ hohen Anteil an Wiederholerinnen wieder (fünf von 14 Frauen im Vergleich zu 19 von 175 Männern).[1]

Die befragten Tutoren bestätigen die Einschätzung: „die ohne Vorkenntnisse sind hemmungslos überfordert", „manche haben schon bei der Variablendeklaration Probleme", sie bräuchten mehr Zeit [Ho95, S. 52]. Andererseits stellten sie fest, daß Anfänger „sich leichter mit der Objektorientierung tun" als die sog. 'Hacker'. Wurden die Konzepte gut erklärt, fiel ersteren der Entwurf und die Codierung von anspruchsvollen OO-Konzepten überraschend leicht. Die imperativen Anteile jedoch überforderten sie.

Das algorithmische (imperative) Denken ist offenbar schwerer zu verstehen als viele Begriffe und Konzepte, es stellt eine 'fremde Welt' dar. Eine Welt, die aber die Denkweisen und Methoden der Informatik lange geformt hat und uns selbstverständlich geworden ist.

1.1 Vorwissen hilft beim Aufbau mentaler Modelle

Es ist eine gewisse Menge an Vorkenntnis nötig, um das implizit Gesagte zu verstehen und Zusammenhänge erkennen zu können. Das erste Semester wird oft als zusammenhangslos empfunden. Erst im weiteren Verlauf entsteht ein „Prozeß wachsenden Verstehens" [Ho95, S. 100], in dem sich ein Bild entwickelt, in dem auch die Lehrinhalte des ersten Semesters ihren Platz finden. Dies bedeutet zwar,

[1] Auch wenn die Anzahlen zu klein sind, um nach strenger Statistik als verallgemeinerbar zu gelten, zeigen sie einen deutlichen Trend. Zudem decken sich meine Ergebnisse mit mehreren anderen Studien.

daß Studierende ohne Vorkenntnisse später nicht schlechter als andere sein werden, fordert ihnen aber größere Frustrationstoleranz, größere Ausdauer und viel Energie ab. Wer diese nicht besitzt, bricht das Studium höchstwahrscheinlich ab, während es anderen (mit evtl. viel schlechteren analytischen Fähigkeiten) gelingt, die Minimalanforderungen zu erfüllen. Zwar scheitern auch sog. 'Hacker' an fehlender Theoriefähigkeit, meist aber erst gegen Ende des Grundstudiums. Zudem können viele dieses Defizit durch ihre Fähigkeit „irgendwie" eine Lösung zu „basteln", kompensieren.

Diese Phänomene lassen sich zum Teil mit dem psychologischen Konstrukt der 'Mentalen Modelle' [Dut94] erklären. Dutke schreibt, daß Erinnern und Wiedergeben ein konstruktiver Prozeß ist. Rekonstruktion erfordere aber eine strukturierte Form des Gedächtnismaterials, sog. Schemata [Dut94, S. 24]. Schematisches Wissen sei daher nötig, um Analogien überhaupt erkennen zu können. Verstehen ist ein Prozeß, in dem Kongruenz zwischen Neuem und bereits organisiertem Wissen hergestellt werde [Dut94, S. 44]. „Die Planung und Durchführung von Handlungen (*erfordert*) die Existenz eines wenigstens bruchstückhaften mentalen Modells vom System." [Dut94, S. 147] Ohne ein solches Modell ist Wiedererkennen und Handeln nur er-schwert möglich. Genau dies ist die Situation von Computeranfängern.

In 1992 durchgeführten Interviews zu einem Repetitorium zur Informatik I sagten die u. a. befragten Programmieranfänger, das Repetitorium habe sie „gerettet", nachdem die eigentliche Vorlesung „unverständlich" war [Ho92]. Auch hier half sicherlich, die Inhalte schon einmal gehört zu haben. Das Interview mit Steffi (ein Semester nach der Informatik I) scheint mir typisch für die Situation von Anfängern und den Prozeß wachsenden Verstehens. Sie sagt: „Einige Dinge konnte ich ziemlich gut lösen, Pro-grammieren war das Problem, Features zu schreiben ist das Schlimmste." Die Theorie verstand sie von Anfang an gut, mit dem Entwurf kam sie gegen Ende des ersten Semesters gut zurecht. Sie war in der Übung immer in der Rolle derjenigen, der alles erklärt wurde, ohne zur Diskussion beitragen zu können. Für die Nachklausur nahm sie Nachhilfe und bestand gut: „ Das war der Durchbruch zum Programmieren. Ich wußte früher immer auf Anhieb: ich kann das nicht, ich finde keinen Ansatz." Mittlerweile kennt sie den Kontext, so daß sie weiß, was (implizit) von ihr verlangt ist und wie sie vorgehen muß. Diesen „Durchbruch" erlebt sie intensiv und berichtet mir begeistert, daß sie „ziemlich schnell" C gelernt habe. [Ho95, S. 77]

Mit funktionalen Programmiersprachen zu beginnen, um die Niveaus anzugleichen, wie es in Hamburg geschieht, mildert das Problem nur zum Teil. Auch hier berichten die Veranstalter [Hamb], daß den Programmieranfängern „vieles detailliert erklärt werden mußte, da diesen das Verständnis für den Programmablauf fehlte". Eine von mir befragte Studentin berichtete, im ersten Semester habe ihr vermutlich ihr mathe-matisches Vorwissen den Einstieg erleichtert. Dies machte sie selbstbewußter. Als im zweiten Semester imperativ programmiert werden sollte, wurde es aber auch für sie schwierig. Sie beklagt ein „Erfahrungsdefizit", fühlt sich auch im vierten Semester noch unsicher. Ihr Erfahrungsdefizit betreffe vor allem die

Fehlersuche, die System-bedienung und den Feinablauf beim imperativen Programmieren. Objektorientierter Entwurf macht ihr deutlich mehr Spaß.

Der Versuch, Vorkenntnisse durch wenig bekannte Paradigmen zu nivellieren, ist abhängig von den aktuellen Moden. Wie obiges Beispiel zeigt, löst jedoch auch das funktionale Vorgehen das Problem nicht zufriedenstellend. Man darf nicht glauben, nur durch den Wechsel der Sprache Programmieranfänger und -erfahrene auf eine Stufe zu bringen. Die Untersuchung von [LGG92] zeigte im Vergleich von Vorwissen und Leistungserfolg von Studierenden, die Smalltalk bzw. C++ Kurse besuchten, daß die Kenntnis von prozeduralen und funktionalen Sprachfamilien positiven Einfluß auf die Leistung hatte; je mehr Sprachfamilien jemand kannte, desto größer der Vorteil (auch wenn es nicht dasselbe Paradigma betraf).

1.2 Programmieranfängern helfen UND Erfahrenen Neues anbieten

Die typischen Anfängerfehler beim Programmieren wurden bisher nur punktuell, isoliert untersucht. Als Hauptursachen werden dort semantische Mißinterpretation von Sprachkonstrukten und Mißverstehen der Plankomposition genannt. Pläne sind dabei Muster bzw. Lösungsraster für Teilziele (Schleifen, typische Befehlsfolgen). Plankomposition bezeichnet die Verschachtelung und Mischung von Plänen. Insbesondere die Schleifenkonstruktion bereite Probleme (Termination und Initialisierung), sowie die Fehlerbehandlung. [Ebra94] Anfänger lesen Programme zudem linear, zeilenweise, ohne Strategie und spekulieren zeilenweise über deren Wirkung, während ‘Experten’ geistig den Programmablauf simulierten [WieFix93]. ‘Experten’ treffen dabei explizite Verbindungen zwischen Codestücken und Teilzielen und erkennen typische Muster im Grobablauf, entwickeln eine ‘mentale Landkarte’. Davies bescheinigt den Anfängern eher strategische als wissensbasierte Schwierigkeiten [Dav93], sie hätten ein fragmentarisches, fragiles, noch schlecht nutzbares Wissen.

Diese Ergebnisse liefern einige Anregungen für die Unterstützung von Programmieranfängern, denn die nötigen Strategien und Heuristiken könnten explizit gelehrt werden. Alle meine Interviewpartner beklagten z.B. nie Fehlersuchstrategien erlernt zu haben. Die „mentale Simulation“ von Programmen als Lesestrategie, Üben der Plankomposition, Entwurf und Strukturierung mit Schnittstellen sind Vorgehensweisen, deren explizite Thematisierung Allen zugute kommen kann, da sie Automatismen reflektieren helfen.

Der Veranstalter besitzt Verantwortung für die Programmieranfänger, aber auch für die Studierenden mit Vorkenntnissen. Lehrkonzeption und -inhalte sollten beiden Gruppen gerecht werden. Auffälliges Phänomen der von mir untersuchten Veranstaltung war, daß den meisten Anfänger die OO-Konzepte leichter fielen als das Programmieren. Eigene Beobachtungen und andere Praxisberichte [BiGr93] zeigen zudem, daß ein Beginnen mit dem imperativen Programmieren i.d.R. dazu führt, daß die Nicht-Anfänger sich langweilen, das Neue regelrecht verpassen, es nicht als wirklich Neu erkennen und schlecht umlernen. Bei den ersten Informatikein-

führungen mit objektorientierten Sprachen an der THD war die Konzeption: erst imperativ programmieren lehren, dann im Laufe der Zeit OO einführen.

Das von mir propagierte 'kombinierte Top-Down/Bottom-Up Vorgehen' bedeutet dagegen, die abstrakten Konzepte nacheinander und sauber vorzustellen (Klassen – Teil-von-Hierarchien – Vererbung – Polymorphie), jeweils anschließend in ihre Benutzung (auch programmiertechnisch) einzuführen und das imperative Programmieren von Codeteilen erst im Laufe der Zeit einzuführen. Die Reihenfolge der Lehrinhalte in dieser Weise umzudrehen, wurde nach der weitgehenden Umsetzung von allen begrüßt [HenSch97, S. 1]. Anfänger profitieren dabei von Erfolgserlebnissen und verstärktem Selbstbewußtsein, sammeln Vorwissen und entwickeln ein mentales Modell dessen, was ein Programmsystem ist. Nicht-Anfänger werden durch die Einführung in (moderne) Konzepte und das 'Programmieren im Großen' motiviert und erkennen sie klarer, können dadurch besser umlernen.

2 Programmieren in einer Projektphase

Das Programmieren hat in seinem Stellenwert im Studium eine seltsame Zwitterstellung. Es ist nicht das Ziel des Studiums, aber es ist ein Handwerkszeug, das jeder sauber beherrschen sollte. Wird es in den Mittelpunkt gestellt, dominiert es das Denken. Wird es gar nicht thematisiert, kann der falsche Eindruck entstehen, es sei kein notwendiges 'Handwerk', aber auch der Eindruck, es würde sowieso schon vorausgesetzt! Wird die Einübung zu lange hinausgezögert, wächst die Hemmschwelle immer weiter an.

Auch ein Top-Down orientiertes Verfahren muß also in absehbarer Zeit das Programmieren einüben. In meiner Diplomarbeit schlage ich dazu ein zwei- bis dreiwöchiges Projekt gegen Ende des ersten Semesters oder zu Beginn der vorlesungsfreien Zeit vor. In Kleingruppen soll ein Problem vom Entwurf bis zum Programm bearbeitet werden. Dieses Konzept wurde an der THD mit großem Erfolg erprobt (WS 95/96, WS 96/97) und zur Fortführung empfohlen [HenSch97]. Für das Projekt wurde eine Stunde des dreistündigen Programmierpraktikums 'ausgekoppelt'. Notwendig, bzw. sinnvoll ist dabei eine intensive Betreuung durch Tutoren sowie ein Abschluß mittels Gruppen-Reviews, in denen das Verständnis aller Teilnehmer diskursiv geprüft wird, bzw. mit schriftlichen Projektberichten.

Noch ein weiterer Aspekt spricht für Projekte im Grundstudium. Soziale Kompetenzen werden zunehmend als wichtiges – und bisher vernachlässigtes – Ziel des Informatikstudiums benannt. Die meisten Curricularvorschläge enthalten Veranstaltungen, die solchen Kompetenzen förderlich sein sollen oder sie explizit einüben, jedoch erst im Hauptstudium. Zu diesem Zeitpunkt sind viele Studierende bereits durch ihr Grundstudium geprägt worden. Die im Grundstudium üblichen traditionellen Veranstaltungsformen Vorlesung, (Vorrechen-)Übung und Einzelpraktikum können aber nicht als Vorbereitung auf Teamarbeit bezeichnet werden. Im Gegenteil können sie zur Vereinzelung beitragen und vorhandene Fähigkeiten verkümmern lassen. Soziales Lernen und das Üben von Teamarbeit

sollte daher bereits im Grund-studium beginnen. Projektpraktika bieten aus fachdidaktischer Sicht den Vorteil, innerfachliche und extrafunktionale Qualifikation zu verbinden.

2.1 Durchführung und Auswertung der stattgefundenen Projekte

Im WS 95/96 bestand die Aufgabe in der Entwicklung eines Interpreters für ein „Subset“ der Sprache EIFFEL in EIFFEL. Jede Gruppe wurde während der drei Wochen insgesamt mindestens 4,5 Stunden individuell betreut. Der Rechnerraum war von 10 bis 18 Uhr betreut. Geprüft wurde von den Veranstaltern in Reviews. Die Projektnote wurde mit der Klausurnote zusammen gerechnet. Die Teilnehmer konnten den Leistungsumfang des Programms selbst bestimmen, was Einfluß auf die mögliche Note hatte [Sto96, HenSch97].

Im WS 96/97 programmierten die Erstsemester in Vierer- oder Sechsergruppen in zwei Wochen einen automatischen Reversi-Spieler, d.h. wahlweise entweder die Komponente ‘Spieler’, ‘Monitor’ (Verwaltung, Schiedsrichter) oder die graphische Oberfläche. Für den Test wurden Dummy-Module zur Verfügung gestellt. Die Rechnerräume wurden von 9 bis 18 Uhr betreut. Das Projekt wurde mit einem öffentlichen Turnier abgeschlossen, das als Integrations- und Funktionstest diente. Herausforderung war dabei die Ausgestaltung einer guten Spielstrategie. Jede Gruppe mußte (anstelle Review) einen ca. zehnseitigen Projektbericht erstellen, der oft Tage-buchform annahm. [Gä97, Matt97, Proj96]

Projekte regen soziales Lernen an und konfrontieren mit allen Problemen der Projektarbeit. Sie sind somit eine wichtige Vorbereitung auf die Praxis: Zeitplanung, Koordination, Zusammenarbeit, innere Konflikte, Aufgabenteilung usw. Die Konzentration auf ein Thema für zwei bzw. drei Wochen führte zu einem tieferen Verständnis sowohl für das imperative Programmieren wie für die Objektorientierung [Sto96], bewirkte Routineerwerb und Einschätzung der eigenen Leistungsfähigkeit. Die Teilnehmer im WS 95/96 empfanden die Existenz eines gemeinsamen Ziels trotz der schwierigen Aufgabe als sehr motivierend. Da es möglich war, den Leistungs-umfang des Programms selbst zu bestimmen, waren die Teilnehmer nicht überfordert, aber herausgefordert. Es wurden präzisere und konkretere Fragen an die Tutoren gestellt als sonst, deshalb machte auch diesen die Betreuung mehr Spaß. [Sto96]

Während des Turniers

Die anders geartete Aufgabe und das Finale mit einem Turnier im zweiten Durchlauf des Projekts verstärkten offenbar die gruppen-dynamischen Prozesse und erhöhten Spaß und Motivation. Ca. 250

Studierende nahmen teil. Vom Veranstalter offen-gelassene Schnittstellen-aspekte wurden in News-gruppen von den Teil-nehmern ausgehandelt.

Motivation und Einsatz waren hoch: „Als am Vorabend des Turniers abzusehen war, daß einige Spieler mit dem Austesten noch etwas Zeit benötigten, wurden kurzerhand die Pool-Räume die ganze Nacht hindurch für die Arbeit am Projekt offengehalten.“ [Gä97, S. 11] Noch während des Turniers wurden Strategien (Bewertungsfunktionen) verfeinert und Fehler behoben.

„Inforz: Wann seid ihr fertig geworden?
Gruppe Quark (Spieler): Gerade eben, nachdem wir verloren hatten. Ist ja K.O.-System. Ich teste gerade noch die besten Parameter aus. (...) Wir hoffen noch auf eine Looserrunde.“ [Gä97, S. 13,14].
Und im Notfall wurde auch selber gerechnet:
„Inforz: Ich denke,ihr habt so ein tolles Programm. Warum notiert ihr das denn hier alles noch per Hand?
Gruppe Verteilte Gehirne (Monitor): Ja das Programm ist augenscheinlich doch noch nicht so gut. Wir setzen auf eine hybride Konfiguration.“ [Gä97, S. 10].

Es gab sogar Teilnehmer aus dem dritten Semester, die rein aus Interesse am Projekt teil-nahmen. Durch das Turnier wurde das Zusammengehörigkeitsgefühl des Jahr-gangs gestärkt, „eine Herausforderung gemeinsam mit Freude und Motivation gemeistert zu haben“ [Gä97, S. 15].

Darüber hinaus lernten die Teilnehmer viel über Software-Entwicklung (Auszüge aus Projektberichten nach [Matt97]): „Wir machten zu Anfang des Projekts den Fehler, relativ schnell mit der Implementierung der Methoden zu beginnen. Außerdem hat sich (...) die Anwendung von Kommentaren im Programmtext durchgesetzt, da sonst nach einigen Tagen eigene Methoden nicht mehr verstanden werden.“. Die Wichtigkeit von Schnittstellen und ihrer Einhaltung, sowie das Verhältnis zwischen Programmlänge und Fehlerzahl wurden am eigenen Leib erfahren. Der Lerneffekt über Programmieren und Java sei in diesen beiden Wochen höher als im laufenden Semester gewesen.

Probleme entstanden durch langsame Compiler, Rechnermangel, fehlende Stühle, zuwenige Tutoren, unkomfortable Tools. Fast allen gefiel das Projekt, wenn auch der Arbeitsaufwand (für viele zehn- bis zwölfstündig plus Wochenende, etliche Nächte) zum Teil heftig kritisiert wurde. Als problematisch erachte ich zudem, daß die ursprüngliche Konzeption eines Entwurfs- und Programmierprojekts tendenziell zu einem Programmier-Parforce mutierte. Bei ungenügender Betreuung und Vor-bereitung ensteht Überforderung. Die Orientierung auf das Endprogramm verdrängt dann das prozeßhafte, reflektierende Lernen. Scheinkriterium sollte kein Endprodukt, sondern der Lernerfolg (und sei er aus Fehlern) sein.

In der Umfrage, die im ersten Durchlauf nach der Projektphase stattfand, war das Projekt (im Vergleich mit Vorlesung, Übung, Praktikum und Sprechstunden) die beliebteste Veranstaltungsform und wurde von 95 aus 107 Befragten als „notwendig“

beurteilt. Hier führte es jedoch dazu, daß für die Klausur (deren Note durch das Projekt wesentlich verbessert werden konnte), deutlich weniger als normal gelernt wurde. Da sich das Klausurlernen auch von der Lernweise (mehr auswendig, mehr Pattern-Matching von Standardaufgaben, mehr Fleißaufgaben) vom Projektlernen (handlungsorientiert, diskursiv, individuell) unterscheidet, stellt dies ein Problem dar, wenn schriftliche Klausuren und Vordiplome gestellt werden müssen. Auch hier muß an die Anpassung der geprüften Inhalte gedacht werden, bzw. müssen die Schwer-punkte rechtzeitig bekannt gemacht werden. Ebenfalls überdacht werden sollte der Zeitpunkt von Klausuren, um Prioritätskonflikte zu vermeiden.

Lehrende wünschen sich meist verstehensorientiertes Lernen, fragen in Prüfungen oft aber nur nach Fakten, individuelle Lernergebnisse werden dabei kaum honoriert. [Entw/Entw91]. Wenn ein strategisches Vorgehen ihnen nötig erscheint, lernen Studierende strategisch [Rams87]. Konflikte zwischen Lehrstil und Prüfstil führen also entweder dazu, daß der Lehrstil nicht sein Ziel erreicht oder – wird er ernst genommen – der Lernende die Folgen zu spüren bekommt.

3 Anleitung und Hinleitung zum Projekt

Wird das Programmieren nun, wie ich es vorschlug, erst im Verlauf des Semesters erlernt und am Ende vertieft, so darf man nicht plötzlich Fähigkeiten verlangen, die nicht erlernt wurden. Begleitende Maßnahmen können das Projekt vorbereiten, Einführungskurse in die Rechnernutzung in den ersten Wochen das Praktikum ersetzen (vergl. [AGA96]). Dabei wird in kleinen, nach Vorkenntnissen getrennten Gruppen geübt. Die Anfänger erhalten dabei intensivere und längere Betreuung. Im WS 95/96 in Darmstadt wurde vom Veranstalter ein Stützkurs für Anfänger zum imperativen Programmieren angeboten [HenSch97, S. 2], der anhand aktueller Themenwünsche als Frage/Antwortspiel bzw. Präsenzübung ablief. Auch für den Teil des Praktikums, der weiterhin im Semester stattfindet, gibt es Ideen. Erste Program-mieraufgaben können das Programmverstehen üben, indem vorhandene Klassen mit sichtbarer Spezifikation in eigenen, einfachen Klassen verwendet werden, indem Beispielklassen analysiert und modifiziert werden, indem der Programmablauf in Beispielklassen nachvollzogen wird und Fehler gesucht werden. Aufgaben zur Übung der geistigen Simulation, zu Plankomposition und Fehlersuchstrategien können danach im Praktikum am Rechner erprobt werden. (vergl. [Ho95, S. 154]

Wird auf diese Weise zum Programmieren hingeleitet, so sind auch die Anfänger auf das Projekt gut vorbereitet – zusätzlich zur ohnehin helfenden zeitlichen Verlagerung. Mit dieser umfassenden Konzeption (die die Prüfung mit einschließt) läßt sich der Verantwortung für Alle gerecht werden. Für die Erfahrenen stellt das Projekt eine willkommene – und oft lehrsame! – Herausforderung dar, für die Anfänger ist es eine Phase intensiven, aber kommunikativen Übens.

Jedes Projekt wird ein neues Experiment mit neuen Erfahrungen sein. Nur wenn wir uns gegenseitig ernstnehmen, nicht im Trott stehenbleiben, dann wird Lehren und Lernen zum lebendigen Prozeß, der beiden Seiten Spaß bereitet.

[AGA96] AGA AGA Projekt. Http://wwww.bs.cs.tu-berlin.de/studentisch/aga-aga/

[BiGr93] The Challenge of Introducing the Object-Oriented Paradigm, An Empirical Investigation of a Software Engineering Course. In: Structured Programming, (14,49), 1993

[Dav93] Simon Davies. Models and theories of programming stategy. In: International Journal of Man-Machine Studies 39. 1993 (S. 237 - 267)

[Dut94] Stephan Dutke. Mentale Modelle: Konstrukte des Wissens und Verstehens -- kognitionspsychologische Grundlagen für die Software-Ergonomie. Verlag für Angewandte Psychologie, Göttingen, 1994

[Ebra94] Alireza Ebrahimi. Novice programmer errors: language constructs and plan composition. In: International Journal of Human-Computer Studies 41, Sept. 1994 (S. 457 - 480)

[Entw/Entw91] Noel Entwistle, Abigail Entwistle. Contrasting forms of understandign for degree examination: the student experience and its implications. In: Higher Ecucation (22, 3), (S. 205 - 227). Kluwer Academic Publishers 1991

[Gä97] Felix Gärtner und Projektteilnehmer. Reversi meets Java – Ein Live-Bericht vom letzten Tag des Informatik I Projektes. In: Fachschaftszeitung am FB Informatik der THD „Inforz", Mai 97, (S. 9 - 16)

[Hamb] Yvonne Dittrich, Guido Gryczan, Universität Hamburg, E-Mail-Zitate 1994

[HenSch97] Wolfgang Henhapl, Ulrik Schroeder. Erfahrungsbericht über Grundzüge der Informatik I WS 95/96, 2. Überarbeitete Fassung: 11 Juni 1997. Fachbereich Informatik, Technische Hochschule Darmstadt

[Ho92] Eva Hornecker. Neue Wege in die Informatik I – eine Umfrage. in: Fachschaftszeitung am FB Informatik der THD „Inforz", Mai 92

[Ho95] Eva Hornecker. „Grundzüge der Informatik I" Didaktische Analyse des Übungsbetriebs – Anregungen zur Neukonzeption. Diplomarbeit THD 1995

[Kay etal 89] J. Kay, J. Lublin, G. Poiner, M. Prosser. Not even well begun: women in computing courses. In: Higher Education 18 (1989). 1989 (S. 511 - 527)

[Kre96] Christoph Kreitz. Projektaufgaben in Anfängerveranstaltungen: ein Mittel zur Förderung eines objektorientierten Programmierstils. In: Softwaretechnik-Trends 14:4, Dezember 96. (S. 30 - 43**)**

[LGG92] Ch. Liu, St. Goetze, B. Glynn. What Contributes to Successfull Object-Oriented Learning? In: ACM SIGPLAN Notices (27,10). Oct. '92, OOPSLA 1992 (S. 77 - 85)

[Matt97] Friedemann Mattern, THD, E-Mail vom 2.8.97

[Proj96] http://www.informatik.th-darmstadt.de/VS/Lehre/WS96-97/Info1/: Informationen zum Programmierprojekt – DI 1 (WS 96/97) Verteiltes Reversi-Spiel und: http://www.igd.fhg.de/~girschik/inf/gspord/impressionen1.html Photos von Martin Girschick

[Rams87] Paul Ramsden (Univ. of Melbourne) Improving Teaching and Learning in Higher Education: the case for a relational perspective. In: Studies in Higher Education (12, 3), 1987 (S. 275 - 286)

[Sto96] Julia Stoll. Grundzüge der Informatik I im Wintersemester 95/96 – oder: Was gab es dieses Mal besonderes? .In: Fachschaftszeitung am FB Informatik der THD „Inforz“, Juni 96. (S. 25 - 32)

[WieFix93] Susan Wiedenbeck, Vikki Fix. Characteristics of the mental representation of novice and expert programmers: an empirical study. In: International Journal of Man-Machine Studies 38. 1993 (S. 347 - 368), auch: CHI 96