Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen
SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.

Can you Java? Ein Erfahrungsbericht

Ulrike Jaeger
Studiengang Software Engineering
FH Heilbronn
ulrike.jaeger@fh-heilbronn.de
http://www.se.fh-heilbronn.de/~jaeger

Zusammenfassung

Dieser Erfahrungsbericht schildert Erfahrungen mit der Grundausbildung in Programmiersprachen. Soweit es ging, konnten Studenten aktiv lernen. Auch die Leistungskontrolle sollte einen aktiven Charakter bekommen: gleichzeitig von den Studenten aktiv erlebt und gestaltet werden, ihnen aber auch Feedback geben und am Ende benotet werden. Dabei erwies sich die klassische Klausur als ungeeignet. Der Beitrag schildert die ersten Schritte in Richtung aktiv erlebter Feedback-Veranstaltungen.

1 Programmiersprachen im Studiengang Software Engineering

Der Studiengang Software Engineering an der FH Heilbronn wurde im Wintersemester 1996 gegründet. Er ist bewußt als praxisorientiertes Fachhochschulstudium mit Betonung auf ingenieursmäßige Herangehensweise konzipiert. Objektorientierung, Teamarbeit und projektbezogenes Denken werden in vielen Fächern von Anfang an vermittelt. Das Curriculum für Programmierung bietet in den ersten beiden Semestern jeweils 4+2 SWS "*Programmiersprache I*", die am Ende gemeinsam in einer Klausur geprüft werden. Im Hauptstudium gibt es eine 4-stündige Vertiefungsvorlesung "*Programmiersprache II*".

1.1 Erste Erfahrungen

1996 bis 1998 wurde im ersten Jahr C++ als Einführungssprache gewählt. Die Vertiefung im Hauptstudium bot dann eine Java-Vorlesung nach [Eckel00].

C++ erwies sich als zu speziell und umfangreich für eine Einführung in die Programmierung. Versuche, ein weitere, nicht kommerzielle Sprache [Pemberton87] parallel zur Relativierung mit zu unterrichten, verwirrte die Studenten. Besser erwies sich ein an Modula angelehnter Pseudocode [Appelrath95]. Die vielbeschworene Praxisrelevanz von C++ traf zumindest für unsere Industriepraktika nicht zu: kaum ein Student arbeitete an Projekten in C++.

Nach dem Praxissemester war der Wiedereinstieg sehr mühsam. Die C++ Kenntnisse waren schon wieder halb vergessen, die Java Vorlesung baute aber nach

Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen
SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.

[Eckel00] darauf auf. Statt das vergessene C++ als Referenz für alles, was in Java anders ist zu verwenden, beschlossen wir, die Reihenfolge umzudrehen. Im Wintersemester 1999/2000 starteten wir mit Java als erste Programmiersprache.

1.2 Anpassung der Lehrinhalte

Durch den Wechsel haben sich einige Vorlesungen verändert. Die Grundvorlesung mit ihren zweimal 4+2 Stunden basierte nun auf Java. Im 4. Semester gibt es eine zweistündige Vorlesung "Programmiersprachen II", die hauptsächlich C++, aber auch andere Sprachen streift, sowie eine eigene Vorlesung zum Thema "Entwicklungsumgebungen". Das schafft die Möglichkeit, sich in den ersten beiden Semestern ganz auf Java in einfachster Umgebung zu konzentrieren.

2 Aktives Lernen

Wir wollten in dieser ersten Programmiervorlesung keine Sprach-Spezialisten ausbilden, sondern Java lediglich als Vehikel zum Verständnis von Modellier- und Programmierprinzipien benutzen.

2.1 "Java Day"

Im ersten Semester wurden über die Woche verteilt seminaristische Vorlesungen mit Übungsterminen am Rechner kombiniert (4+2). Im zweiten Semester gab es stattdessen einen sogenannten *„Java Day"* pro Woche, an dem in sich lockerer Folge Theorie und Übungen am Rechner abwechselten. Das ermöglichte uns, Fragen, Varianten und Lösungsideen der Studenten sofort praktisch zu untersuchen.

2.2 Dokumentendatenbank

Zur Verbreitung von Material, Folien, Aufgaben und Lösungen wurde eine *Lotus Notes™ Domino* Dokumentendatenbank eingerichtet. Sie bot vollständige Information, leichte Verwaltung und geeignete Stichwortsuche. Im Gegensatz zu früheren Modellen über ftp-Server waren die Studenten dieses Mal mit der Bereitstellung von Unterlagen sehr zufrieden. Die Bedienung war attraktiv, die Stichwortsuche wurde eifrig genutzt. Einzig die Verfügbarkeit des Servers litt unter der Last der Zugriffe.

2.3 Interaktives Programmieren

Für die Grundlagenvorlesung suchten wir Literatur, die weder andere Sprachkenntnisse voraussetzt, noch einen Crashkurs für die legendären 21 Tage bietet, und nicht nur eine reine Beschreibung der Sprache Java ist. Wir entschieden uns für den Ansatz von Lynn Andrea Stein [Stein00].

Das Konzept dieses Buches geht von Interaktiver Programmierung aus. Es baut die Vorstellung vom Programmieren auf dem Paradigma des gegenseitigen Vertrags

von Teilnehmern auf. Solche Teilnehmer können Objekte sein, auf tieferen Ebenen dann Methoden mit ihren Schnittstellen. Das Denken in Vereinbarungen, Garantien und Ausnahmen läßt sich aus alltäglichen Situationen ableiten und führt schnell auf Interaktionsmodelle und Protokolle hin.

Zusätzlich wurde [Bishop99] empfohlen und im zweiten Semester auch [Eckel00] und [Gamma96] für die Diskussion von *Design Patterns*. Alle Studenten wurden ermutigt, für sich persönlich weitere Bücher zu prüfen.

Die Programmierung wurde eng mit der gleichzeitigen vierstündigen Vorlesung "*Grundlagen der Informatik*" verzahnt. Im ersten Semester geht es um den Algorithmusbegriff und Modularisierung. Im zweiten Semester werden dann Interaktion, Ereignisse mit Verteilten Systemen gekoppelt.

Wir benutzten keine komfortable Entwicklungsumgebung. Zur Modellierung wurde zwar die *UML*-Notation verwendet, aber ebenso oft intuitive Skizzen mit Strichmännchen und Eigenkreationen der Studenten. Wir wollten die Studenten vorerst an das Notieren selbst gewöhnen, aber noch nicht auf eine bestimmte Notation einschwören.

Schon in den ersten Wochen wurden die Studenten mit den *JDK Packages* von Sun™ konfrontiert. Lediglich für komfortables I/O haben wir die üblichen Vereinfachungen für die Studenten angeboten, alles andere war Original *JDK1.2*. Das anfangs mühsame Lesen und Verstehen von Dokumentationen wurde einfacher, als mitten im Semester auch die Folien vollständig auf Englisch umgestellt wurden. Offensichtlich wurde danach eher in Englisch "gedacht".

2.4 Didaktische Konzepte

Angeregt durch einen Gastaufenthalt an der Universität von Amsterdam, die bei ihren Medizin-Informatikern das *Problem Based Learning PBL* [PBL97] einsetzt, wollten wir die Studenten so aktiv wie möglich beteiligen. Persönlich Erlebtes wird eher verstanden und besser behalten als passiv Gehörtes. *PBL* geht außerdem davon aus, daß die Studenten sehr wohl in der Lage sind, Lösungen und Antworten selbst zu erarbeiten. Obwohl wir mit 45 Studenten deutlich überbelegt waren, versuchten wir, die Studenten zum Diskutieren, Erklären und Ausprobieren zu ermutigen. Grundsätzlich wurden die Übungsstunden aufgeteilt, so daß es zwei Großgruppen im Programmierpraktikum gab. Zusätzlich bildeten sich Kleingruppen von 3-4 Teilnehmern, die Aufgaben und den Stoff während der Vorlesungen bearbeiteten, im Plenum vortrugen und diskutierten. Ganz nebenbei wurde damit auch das Arbeiten im Team, Vortragen und Moderieren geübt.

Im zweiten Semester wurden die Kleingruppen strukturiert. Die Teilnehmer hatten (wechselnde!) Rollen:

- Mediator:
  - vertritt die Gruppe nach außen
  - spricht Schnittstellen mit anderen Gruppen ab

- entscheidet bei Konflikten innerhalb der Gruppe

- Designer:
  - prüft, ob es bereits fertige Teillösungen gibt
  - entwirft die Struktur der Lösung
  - entwirft die Struktur der Lösung

- Hacker:
  - testet fragliche Varianten
  - zeigt generelle Machbarkeit von Einzelproblemen

Die Vorlesungen waren nicht immer für das gesamte Plenum gedacht. Es gab Sonderveranstaltungen nur für Designer (z.B. *Observer Design Pattern*) oder nur für Hacker (z.B. *Reflection* in Java). Diese Spezialisten hatten dann die Aufgabe, das Gelernte den anderen Gruppenmitgliedern zu erklären.

2.5 Ein Beispiel

Über das zweite Semester hin wurde u.a. ein Taschenrechner entwickelt. Dazu benötigten sie die sehr hilfreiche Diskussion in [Stein00] über Entscheidungen, die von Verzweigungen bis zur Modellierung von entsprechenden Objekten führt. Als Informatik-Grundlagen boten wir die *Stack*-Datenstruktur für Argumente und Zwischenergebnisse, und Zustandsautomaten mit Ein- und Ausgaben.

In Kleingruppen arbeiteten die Studenten an immer anspruchsvoller werdenden Fassungen des Taschenrechners:

1. Fassung: ohne GUI, der arithmetische Ausdruck wird einfach als String von der Konsole gelesen. Einige Gruppen implementieren das Einlesen und Interpretieren des Ausdrucks (*IO-Interface*), andere Gruppen implementieren die Arithmetik selbst.

2. Fassung: einige Gruppen stecken ein *GUI* an das *IO-Interface*, dabei nutzen sie das Adapter Design Pattern.

3. Fassung: Schnittstellenanpassung, so daß die *GUI* und Arithmetikteile der einzelnen Gruppen beliebig gegeneinander austauschbar werden.

4. Fassung: Erweiterung des Taschenrechners um weitere Funktionen.

Nach der zweiten Fassung erarbeiteten die meisten Gruppen ein gründliches Redesign ihrer Lösung. Sie wußten wieviel Arbeit es macht, jedesmal von vorn anzufangen. Sie identifizierten logische Einheiten, die unverändert bleiben konnten, und begannen ernsthaft, sich mit den anderen Gruppen über die Schnittstellen zu unterhalten.

Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen
SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.

3 Feedback und Prüfen

Über den langen Zeitraum eines Jahres ging es uns zwar auch um handwerkliche Fähigkeiten in Java, aber vor allem um eine Erziehung zum objektorientierten und interaktiven Denken, das sich prinzipiell auf andere Sprachen und Probleme übertragen läßt. Es fragte sich, wie wir diese "Erziehung" prüfen konnten.

3.1 Aktives Feedback

Im ersten Semester wurde ein unbenoteter Schein aus drei kleineren Aufgaben gebildet:

- individuell: schriftliche Ausarbeitung zum Thema: Namen, Typen und Schnittstellen
- individuell: *online* Programmieraufgabe
- Kleingruppenarbeit: Erarbeiten von Prüfungsfragen, Musterlösungen und Begründung für die Frage (z.B. Was wird überprüft? Was soll gelernt werden?) und Präsentation im Plenum. Diese Aufgabe wurde einheitlich für die ganze Gruppe beurteilt.

Im zweiten Semester wünschten sich die Studenten zur eigenen Leistungskontrolle nun freiwillige Aufgaben ähnlich dem ersten Semester. Alle Aufgaben der Übungen waren Gruppenaufgaben, alle Musterlösungen stammten ausschließlich von den Studenten selbst.

Zur Klausurvorbereitung am Ende gab es wieder auf vielfachen Wunsch eine Session mit von den Studenten erarbeiteten und diskutierten Prüfungsfragen. Dieses Mal waren die Fragen an die spezielle Situation einer schriftlichen Klausur angepaßt. Es gab also sehr kleine Programmieraufgaben zur Analyse oder zum Ergänzen, *Multiple Choice* Fragen und Verständnisfragen. Die Studenten waren sehr engagiert und nutzten die Session zur intensiven Wiederholung und Diskussion. Irrtümer wurden selbständig korrigiert. Die Dozentin hatte wie schon im ersten Semester nur sehr selten etwas zu ergänzen.

3.2 Klausur

Am Ende des zweiten Semesters wurde der gesamte Stoff in einer schriftlichen Klausur geprüft. Hier zeigte sich ein ernstes Problem: während des gesamten Jahres waren die Studenten gewohnt, sich die Informationen aus dem Netz zusammenzusuchen. In der Klausur gab es plötzlich nur Bücher und Skripten als Hilfsmittel. Die Programmieraufgaben mußten nun auf einfache Beispiele reduziert werden, alle notwendigen Klassen mußten als lange Texte auf Papier zur Verfügung gestellt werden. Dieser Bruch im Denken und Handeln wurde von allen sehr beklagt.

Als Alternative wäre eine *online* Klausur denkbar gewesen. Es gibt aber in unserem Haus keinen hinreichend großen Rechnerraum. Eine Aufspaltung in Gruppen

hätte unvergleichbare Nebenbedingungen geschaffen, die nicht benotet werden können.

4 Ausblick

4.1 Nächste Schritte

Für die Zukunft haben wir deshalb einige Veränderungen geplant:

- Verwendung eines *Lotus Notes™ Learning Space*, der freiwillige Leistungskontrolle, Feedback durch halbautomatische Tests besser unterstützt, Eigenbeteiligung der Studenten im Learning Space attraktiver macht
- Keine Klausur mehr, nur noch vorlesungsbegleitende, unbenotete Scheine
- Für spezielle, fachübergreifende Aufgaben: *Board of Advisors*: Dozenten anderer Fächer fungieren als Berater für bestimmte Fragestellungen. Beispiele:
  "Verteilte Systeme" erklärt Protokolle für die *thread*-Programmierung
  "Grundlagen des Software-Engineering" berät bei Notationen und Analyse
  "Organisations-Psychologie" berät bei Gruppenproblemen

Um die Studierenden eher auf die objektorientierte, verteilte Anwendungswelt vorzubereiten, werden wir in Zukunft bereits im ersten Semester die folgenden Prinzipien beachten:

- Verteiltes Programmieren von Anfang an
- Design Patterns von Anfang an
- Zweigleisige Aufgaben: Gleichzeitig Top Down und Bottom Up Aufgaben stellen, z.B. Bausteine der Programmiersprache einüben, aber auch GUI-unterstützte Anwendungen zum Konfigurieren anbieten (nach [Stein00]).

4.2 Erkenntnisse

Der Umstieg auf Java war eine Gelegenheit, die Vorlesung neu zu konzipieren. Die positive Resonanz liegt nicht nur an der deutlich klareren Sprache Java selbst, sondern auch an den begleitenden Maßnahmen.

Der Stoff wird weniger im Frontalunterricht vermittelt, sondern anhand von konkreten Aufgaben durch Fragen der Studenten erschlossen. Dieses didaktische Prinzip ist seit längerem allgemein anerkannt, aber die Umsetzung scheitert allzuoft an den Realitäten einer Hochschule. Bei uns zum Beispiel war es sehr schwierig, sowohl einen Hörsaal als auch einen Rechnerpool für den Javatag im zweiten Semester freizuhalten.

Die Veranstaltung war das gesamte Jahr über von Neugierde und Experimentierfreude der Studenten geprägt. Scheinaufgaben, Gruppenarbeit und Organisation wurden einhellig gelobt. Die vorgeschriebene Klausur war dann der Einbruch der trockenen Hochschulwirklichkeit in ein lebendiges Experiment. Zwar bestanden fast alle Stu-

denten, aber die Noten blieben im mittleren Bereich. Es fragt sich, ob diese Noten der wirklichen Lernleistung entsprechen Die Leistung sollte auf dieselbe Art überprüft werden wie der Stoff bewältigt wird: lebendig, kreativ und diskussionsfreudig. Wir plädieren deshalb für unbenotete Scheine, deren Bestandteile während der Veranstaltung entstehen.

Andere Hochschulen können sicher Ähnliches berichten, wenn sie aktives Lernen verwirklichen wollen. Seit vielen Jahren wisssen wir alle, was man besser machen könnte, aber allein die angespannte Personal- und Raumsituation genügt schon, um gute Ansätze im Keim zu ersticken.

Literatur

[Appelrath95] Hans-Jürgen Appelrath, Jochen Ludewig: Skripum Informatik — eine konventionelle Einführung. Teubner, 1995.

[Bishop99] Judy Bishop: Java lernen. Anfangen, Anwenden, Verstehen. Addison Wesley, 1999.

[Eckel00] Bruce Eckel: Thinking in Java, Prentice Hall, 2nd Edition, 2000.

[Gamma96] Erich Gamma: Design Patterns, Addison Wesley, 1996.

[PBL97] PBL Conference 97: Integrity, Innovation, Integration, 3 - 6 December 1997, Australian Catholic University, Macauley Campus, Brisbane.

[Pemberton87] Steven Pemberton: An Alternative Simple Language and Environment for PC's, IEEE Software, Vol. 4, No. 1, Januar 1987, pp 56-64.

[Stein00] Lynn Andrea Stein: Interactive Programming in Java, Morgan Kaufman, to appear (http://www.mkp.com/ipij/)