Von Analyse und Entwurf zur Programmierung in einem Semester:
UML, Java und deren Anwendung in Projektteams

Klaus Alfert
Software-Technologie, Informatik, Universität Dortmund
E-mail: alfert@ls10.cs.uni-dortmund.de

Wilhelm Hasselbring
Infolab, Tilburg University, Niederlande
E-mail: hasselbring@kub.nl

Arnulf Mester
Verteilte Systeme, Informatik, Universität Dortmund
E-mail: mester@acm.org

1 Einleitung

Im Softwarepraktikum im Grundstudium an der Universität Dortmund werden zur objektorientierten Modellierung BON [18] und zur Programmierung BETA [6] eingesetzt. Diese bisher aus didaktischen Gründen in Dortmund am Anfang gelehrten Sprachen spielen in der industriellen Praxis jedoch praktisch keine Rolle.

Die Unified Modeling Language (UML) wurde von der OMG als zukünftiger Industriestandard für die objektorientierte Modellierung akzeptiert. Java ist als 'besseres C++' insbesondere für die Entwicklung von Internet-Applikationen populär geworden [1].

In dem hier vorgestellten Kurs zur Softwareentwicklung mit der UML und mit Java, der im Sommersemester 1998 an der Universität Dortmund stattfand, sollten die Teilnehmer aufbauend auf den im Softwarepraktikum gewonnenen Kenntnissen die Sprachen UML (für Analyse und Entwurf) und Java (für Implementierung/Programmierung) bei der Durchführung kleinerer Projekte in Gruppen einsetzen. Nachfolgend erörtern wir die Veranstaltungskonzeption, den aktuellen Durchlauf sowie die dabei gewonnenen inhaltlichen und allgemeinen Erfahrungen.

2 Veranstaltungskonzeption

Der Informatikanteil des Kerninformatik-Grundstudiums an der Universität Dortmund folgt i.w. dem eher klassischen Kanon der Veranstaltungen Programmierung (1. Sem.), Rechnerstrukturen (2. Sem.), Datenstrukturen (3. Sem.) und Grundlagen der theoretischen Informatik (4. Sem.). Hinzu kommt ein Softwarepraktikum [17] nach dem 3. Semester und innerhalb eines Programmierkurses eine weitere Programmiersprache im 4. Semester. Als Programmiersprachen wurden für die ersten drei Semester BETA und ML eingesetzt.[1] Im Softwarepraktikum wurde zusätzlich zu BETA oder ML die Modellierungsnotation BON [18] verwendet. Als zusätzliche Wahlpflicht-Programmierkurse (2V+2Ü) sind für das 4. Semester C++, Prolog und Java im Angebot.

Ein solcher klassischer Grundstudiumskanon weist gerade am kritischen Anfang der Ausbildung einen sehr starken Schwerpunkt auf implementierungsnahen Denkweisen und Konzepten auf. Analyse und Entwurf treten in den Hintergrund (und werden erst nach dem 3. Semester als Fertigkeiten geübt) und werden auch vorher nicht 'passiv' (z.B. durch Reviews fertiger Entwürfe) vermittelt. Dies motiviert unser Experiment, einen Programmierkurs des 4. Semesters als eine Art Softwareentwicklungskurs zu gestalten, in dem die Teilnehmer den Bogen von Anforderungsanalyse über Entwurf bis hin zu Implementierungen schlagen und dabei die beiden aktuellen Techniken UML und Java anwenden lernen sollen.

Zielgruppe Bei den Teilnehmern dieses Durchlaufs waren sehr gemischte Vorkenntnisse vorhanden: bedingt durch die aktuellen Themen waren auch Interessenten aus verschiedenen anderen Semestern vertreten. Es konnten aber weder Java- noch UML-Vorkenntnisse vorausgesetzt werden: Die Mehrheit hatte Programmierkenntnisse in BETA und Modellierungskenntnisse in BON. Im Hauptstudium wird derzeit nur in der Stammvorlesung Softwaretechnologie die UML überblicksartig gelehrt; einige andere Veranstaltungen (i.w. die Projektgruppen) verwenden die UML und arbeiten sich jeweils selbständig ein. Die Veranstalter hatten aus verschiedenen Projekten Erfahrungen mit der UML und mit Java erworben.

Ziel Ziel der Veranstaltung war, den Teilnehmern erste Erfahrungen mit einem konkreten iterativen Softwareentwicklungsprozeß (von Anforderungsermittlung bis Implementierung) und den Hilfsmitteln UML und Java zu vermitteln. Dazu wurde die Veranstaltung in eine Vermittlungsphase (ca. 2/3 der Veranstaltungszeit) und eine Projektphase (das restliche 1/3 der Veranstaltungszeit, sowie die veranstaltungsfreie Zeit bis zum Anfang des nächsten Semesters) aufgeteilt.

[1] Ab WS 98/99 beginnt die Grundausbildung mit Java statt BETA.

Phasen Die *Vermittlungsphase* bestand aus einer vorgeschalteten Sitzung zur allgemeinen und sprachunabhängigen Wiederholung und Terminologieabstimmung zu OO-Konzepten und dann je einer wöchentlichen Vorlesung (je ca. 2V) zu UML und zu Java. Beide Vorlesungen waren nur lose gekoppelt. Dies begründet sich darin, daß es auch eine Reihe von Teilnehmern gab, die sich nur isoliert für das Thema UML interessierten. Zusätzlich zu den Vorlesungen wurde von externen Referenten über den industriellen Einsatz von Java am Beispiel Swing und Beans, sowie in einem Forschungskolloqium über Delegation in Java berichtet. Den Abschluß der Vermittlungsphase bildeten Videos zum Thema: OOPSLA'96-Tutorial zu UML von den drei Amigos und Sun Developers Conference zu Java von James Gosling.

In der *Projektphase* sollte auf Grundlage der Vermittlung Transferleistung erbracht werden und es wurden in Kleingruppen kleinere Softwareprojekte mit Analyse, Entwurf und Implementierung durchgeführt. Diese wurden durch die Veranstalter kontinuierlich und ausführlich betreut.

Elektronische Unterstützung Für die Modellierung wurde Rational Rose 98 eingesetzt (Fachbereichsinstallation und 30-Tage-Testlizenzen für 'zu Hause'). Zusätzlich wurde auf Together/J für den heimischen Einsatz hingewiesen. Für die Programmierung setzten wir Sun JDK 1.1.3 ff., Sun Java Workshop 2.0 und (für den heimischen Einsatz) IBM Visual Age for Java 1.0 ein. Die Veranstaltung wurde durch ein eigenes internes Web und eine lokale Newsgroup unterstützt. Das Web konnte im Java-Anteil auf den Java-Programmierkurs im vorherigen Sommersemester aufbauen. Eine vorab zum Selbstkostenpreis erhältliche CD stellte den Teilnehmern das aktuelle JDK für Linux und Windows, Swing, das Java-Tutorial, das Emacs Java Development Environment und eine Evaluierungsversion von Rose 98 zur Einsparung eigener Downloadzeiten zur Verfügung.

Evaluierung Am Ende der Vorlesungen wurde eine formale Feedbackaktion durchgeführt. Während der Projektphase wurden kontiniuierlich informell Meinungsbilder von den Arbeitsgruppen eingeholt.

Die Ziele und Inhalte der einzelnen Veranstaltungseinheiten werden in den folgenden Abschnitten vorgestellt.

3 Vorlesung: UML

Die Unified Modeling Language (UML) kann als Integration und Erweiterung der Notationen für die Booch-Methode [2], für OMT [16] und für OOSE [12] angesehen werden. Es wird ausdrücklich nur eine Notation definiert; kein Vorgehensmodell. Die Notation wurde in der Vorlesung in den folgenden Blöcken vorgestellt:

1. Anwendungsfallbeschreibungen zur Anforderungsmodellierung (einschließlich OOSE-Softwareentwicklungsprozeß) (2 Sitzungen)

2. Klassendiagramme zur Beschreibung von Struktur (3 Sitzungen)

3. Modellierung von Dynamik (2 Sitzungen): Zustandsübergangsdiagramme; Sequenzdiagramme; Kollaborationsdiagramme; Aktivitätsdiagramme

4. Modellierung des Systems (1 Sitzung): Paketdiagramme; Komponentendiagramme; Verteilungsdiagramme;

5. Stereotypen und das UML-Metamodell (1 Sitzung)

Klassendiagramme stellen den Kernbereich der UML dar und werden daher in Lehrbüchern auch häufig als Startpunkt gewählt. Uns erschien es andererseits nicht sinnvoll, mit Klassendiagrammen zu beginnen, sondern mit Blick auf den Entwicklungsprozeß mit Anwendungsfallbeschreibungen (Use Cases) zu starten.

In der traditionellen strukturierten Analyse [5] werden üblicherweise Datenflußdiagramme zur Beschreibung der (funktionalen) Anforderungen eingesetzt. In einigen älteren Ansätzen zur objektorientierten Modellierung, wie z.B. OMT-1 [16], werden Datenflußdiagramme auch mit Klassendiagrammen (Beschreibung von Struktur) und Statecharts (Beschreibung von Dynamik) kombiniert. Häufig wird vorgeschlagen, Klassendiagramme auch zur initialen Anforderungsanalyse einzusetzen. Wir sind allerdings nicht der Ansicht, daß sich Klassendiagramme als guter, *erster* Ausgangspunkt zur gemeinsamen Analyse der Anforderungen mit den Anwendern an ein neues Informationssystem eignen, bevor ein funktionales Modell konstruiert wird: auch wenn die zukünftigen Anwender ein Klassendiagramm (glauben zu) verstehen, ist damit die *Funktionalität* eines neuen Systems i.a. noch nicht ermittelt. Anwender erwarten üblicherweise eine gewisse Funktionalität von einem Informationssystem und nicht eine bestimmte Struktur, in der relevante Daten zu speichern sind oder ein System strukturiert wird.

Trotzdem ist es sehr sinnvoll, Datenmodelle (spezifiziert durch Klassendiagramme) für die Anforderungsanalyse mit den zukünftigen Anwendern zu diskutieren, um z.B. zu überprüfen, ob die Modellierer konzeptionelle Fehler gemacht haben. Anwender verstehen Datenmodelle, in denen auf hohem Niveau Begriffe aus ihrem Arbeitsbereich verwendet werden, wenn sie durch einen Modellierer erklärt werden. Zur Modellierung der Funktionalität sind jedoch andere Modelle notwendig.

Inzwischen ist es etabliert, *Anwendungsfallbeschreibungen* zur Beschreibung der funktionalen Anforderungen in objektorientierten Analysemethoden, wie auch in der UML, einzusetzen. Diese Technik hat ihren Ursprung im OOSE-Softwareentwicklungsprozeß von Jacobson [12]. OOSE ist ein Ansatz zur objektorientierten Entwicklung, der sehr gut durch das CASE-Werkzeug Objectory unterstützt wird

[11]. Anwendungsfallbeschreibungen werden mit Akteuren in Beziehung gestellt. Die *Akteure* interagieren mit dem zu entwickelnden System. Sie sind nicht Teil des zu entwickelnden Systems. Die Akteure können Menschen in bestimmten Rollen oder technische Gerätschaften sein, die nicht genau beschrieben werden müssen. Die Anwendungsfallbeschreibungen beschreiben, *was* das zu entwickelnde System können soll. Anwendungsfälle sind sequentielle Folgen von Transaktionen, die zunächst informell beschrieben werden. Zusätzlich gehören zum Anforderungsmodell *Schnittstellenbeschreibungen* (z.B. Prototypen für graphische Benutzungsschnittstellen).

Als Konsequenz dieser Überlegungen haben wir zunächst Anwendungsfallbeschreibungen zur Anforderungsmodellierung vorgestellt und in diesem Zusammenhang den Objectory-Entwicklungsprozeß eingeführt, bevor die weiteren Notationssprachen der UML vorgestellt wurden.

Da die Studenten deutschsprachige Literatur bevorzugen, wurde zum schnellen Einstieg in die UML neben [3, 15] insbesondere [8] empfohlen. Für einen fundierteren Einstieg wurde auf [10, 13] verwiesen.

4 Vorlesung: Java

Die Programmiersprache Java [1] ist eine Entwicklung von Sun, die aufgrund ihrer Portabilität (Ausführung auf virtuellen Maschinen), ihrem einfachen Aufbau (Einfachvererbung, Garbage Collection) und ihrer Unterstützung für robuste Programme (keine Pointer, Exceptions, Schnittstellen, strenge Typisierung) in kurzer Zeit eine weite Unterstützung sowohl in der Industrie als auch in der Wissenschaft gefunden hat. Entgegen dem kolportierten häufigsten und populärsten Einsatz in Form von Applets auf WWW-Seiten, war unser Ziel, Java als vollständige Programmiersprache für eigenständige Applikationen vorzustellen.

Im Gegensatz zu anderen Programmiersprachen ist Java erst einmal für den Unterricht unhandlich: zwar ist die eigentliche Sprache übersichtlich und kompakt, mithin schnell zu erlernen. Ein produktives Arbeiten mit einer Programmiersprache setzt aber auch ein einsetzbares Wissen über die verwendbaren Abstraktionen und Bibliotheken voraus. Im Gegensatz zu anderen Programmiersprachen sind diese Bibliotheken sehr umfangreich und vor allem in allen Sprachimplementierungen standardisiert verfügbar. Um diesen Produktivitätsvorteil im Sinne der Teilnehmermotivation aber auch der Produktqualität auch ausschöpfen zu können, ist für den Unterricht eine sinnvolle Auswahl von zu behandelnden Bibliotheken zu treffen, aber auch die Struktur und Funktionalität aller Bibliotheken überblicksartig zu vermitteln (auf JDK 1.2-Sachstand mit Bezug auf 1.1-Sachstand, da kein Werkzeug

1.2-fähig war). Insbesondere waren auch Idiome sinnvoller Bibliotheksnutzung vorzustellen (z.B. Enumeration- und Observer-Nutzung).

Die Gesamtstruktur der Java-Vorlesung war deshalb zweigeteilt. Zu Beginn haben wir die Sprachkonstrukte von Java in den Vordergrund gestellt. Anschließend haben wir wesentliche Klassenbibliotheken vorgestellt:

1. Einführung in Java (0,7 Sitzungen): Historie; Sprachüberblick; imperative Sprachelemente en detail
2. OO-Strukturen (1,3 Sitzungen): Klassen, Kapselung, Sichtbarkeit; Vererbung, Polymorphie, Overloading: Schnittstellen vs. abstrakte Klassen
3. Nebenläufigkeit (1 Sitzung): Threads, Synchronisation, Deadlocks
4. Bibliotheken:
   - Überblick und Standardbibliotheken lang/io/util (1 Sitzung)
   - GUI-Programmierung: AWT und Swing (1 Sitzung),
   - Netzprogrammierung (Sockets, Datagramme, Web-Objekte, RMI, Corba, Verzeichnisdienst) und Sicherheitskonzepte (1 Sitzung)
   - Java Beans: Komponentenbasierte Softwareentwicklung allgemein, Beans-Definition, Systemkonstruktion (2 Sitzungen)

Da die Teilnehmer bereits alle Programmiererfahrung aus den einführenden Vorlesungen besitzen und kleinere Projekte innerhalb des Softwarepraktikums realisiert haben, war die Einführung in die Sprache Java sehr kompakt. Ein substantieller Teil beschränkte sich darauf, die konkrete Syntax für die vorgestellten Konstrukte zu präsentieren. Ausgehend von diesem Gerüst wurden dann wesentliche Aspekte der Objektorientierung in dem Abschnitt über Klassen vorgestellt und diskutiert. Dabei war es uns wichtig, daß die Möglichkeiten einer objektorientierten Sprache für erweiterbare und robuste Programme bewußt angewandt werden. So ist ausführlich auf das Konzept der Schnittstellen und den Unterschied zu abstrakten Klassen eingegangen worden.

Ein für die meisten Teilnehmer neuartiges Konzept wurde im Kapitel über Nebenläufigkeit vermittelt. Zwar bietet die Sprache BETA die Möglichkeit mit Koroutinen zu arbeiten, dies wurde aber in der Programmierausbildung nicht nennenswert behandelt. Da Java Multithreading direkt unterstützt, sind die wichtigen Konzepte für nebenläufige Programme auf der Basis von Threads und Monitoren ausführlich vorgestellt worden.

Nach den Sprachkonzepten selbst wurde dann den Fokus auf die Klassenbibliotheken gelegt. Da objektorientierte Klassenbibliotheken sehr groß und komplex werden können, haben wir uns auf eine Übersicht über die Bibliotheken, Auszüge aus

den zusammen mit der Sprache definierten Bibliotheken und ausgewählte weitere Bibliotheken beschränkt. Einfache, kaum strukturierte Bibliotheken haben wir den Teilnehmern zum Selbststudium überlassen. Als wesentliche Bibliotheken haben wir zum einen die beiden GUI-Bibliotheken AWT und Swing, sowie die verschiedenen zur komfortablen Netzprogrammierung ausgewählt. Den Abschluß bildete die Vorstellung des Konzeptes der Java Beans, die eine komponentenorientierte Software-Entwicklung ermöglichen und (fast) alle wesentliche Techniken von Java in sich vereint.

Als Literatur haben wir neben der Sprachdefinition [1] das Java-Tutorial von Sun [4] empfohlen (u.a. wegen der den aktuellen Stand wiedergebenden und kostenfreien Web-Fassung), für die nebenläufige Programmierung zusätzlich [14], für fortgeschrittene objektorientierte Konzepte das Design-Patterns-Buch von Gamma et. al. [9]. Im lokalen Web verwiesen wir auf aktuelle Entwicklungen und Informationen, u.a. auch zu nicht vertieften oder behandelten Themen (z.B. Programmierstil/konventionen), wie auch auf die 'üblichen' Seiten (z.B. Javasoft, Java bei IBM, Java World).

5 Projekte

Ziele der Projektphase waren, daß die Teilnehmer Erfahrungen im Aufstellen von Anforderungen und deren Konkretisierung mit Kunden (hier: Veranstalter in der Rolle der Kunden), im Entwurf, dessen Iteration, Diskussion und Review, in der Implementierung, sowie in Teamarbeit erhalten. Diese Erfahrungen sollen die Teilnehmer in die Lage versetzen, weitere Projekte im Hauptstudium problemloser abwickeln und dort Erfahrungen mit größeren Systemen sammeln zu können. Fachlich sollte die Modellierung mit UML und die Implementierung mit Java eingeübt werden. Außerdem dienten die Projekte den Teilnehmern dazu, eine (für den weiteren Studienverlauf allerdings nicht relevante) Bestätigung der erfolgreichen Teilnahme erhalten zu können.

Vorzulegende Arbeitsergebnisse einer solchen erfolgreichen Projektdurchführung sind eine werkzeuggestützte UML-Modellierung und Dokumentation (der Analyse- und der Entwurfsphase), eine daraus abgeleitete funktionierende und stabile Implementierung in Java, eine ausreichende Implementierungskommentierung (Text einschl. javadoc), eine aussagefähige Web-Präsentation (und ggfs. Demo) (die nach Abschluß des Kurses mit den Namen der Teilnehmer in das World-Wide-Web gestellt werden soll), sowie die Teilnahme am Projektmarkt, d.h. einer kurzen öffentlichen Vorstellung der Projektergebnisse. Dies alles muß innerhalb des aktuellen Semesters (einschließlich der vorlesungsfreien Zeit) abgeschlossen werden.

Die Projektphase bestand aus Themenwahl, Analyse, Entwurf, Implementierung

und der formalen Abnahme.

Themenwahl Die interessierten Teilnehmer sollten noch am Ende der Vermittlungsphase selbstständig Arbeitsgruppen bilden und sich ein Projektthema wählen. Dazu konnten sie eines aus einer im Projektweb vorbereiten Liste (siehe Anhang, im Web jeweils mit einer Seite Erläuterungen) oder auch selbst in Abstimmung mit den Veranstaltern ein eigenes Thema wählen. Die vorbereiteten waren mehr oder weniger detailliert, mehr oder weniger anspruchsvoll oder interessant. In jedem Fall sollten die Teilnehmer diese an ihre Gruppengröße und Vorerfahrungen anpassen. Die Arbeitsgruppengröße war ab drei Personen vorgesehen, um Projekte (und damit Entwürfe) einer gewissen Größe und Komplexität bearbeiten zu können. Zudem mußten die Veranstalter hier dafür sorgen, daß die Projektgröße so an die Gruppengröße angepaßt wurde, daß entsprechende Effekte größerer Gruppenprojekte erfahrbar werden, aber auch ein solches Projekt in realistischem Zeitbedarf lösbar ist. Idealerweise sollten alle Gruppen innerhalb von zwei Vollzeitwochen ein vollständiges Projekt bearbeiten können, sofern für die Vermittlungsphase keine Nacharbeit mehr erforderlich ist und dort in (der dort auch propagierten) Eigeninitiative schon eigene kleine Modellierungs-, Implementierung- und Werkzeugerfahrungen gesammelt wurden.

Analyse und Entwurf

Die Analyse und der Entwurf ging im Wechselspiel zwischen Gruppenarbeit und Reviewsitzungen mit jeweils zwei Veranstaltern vonstatten. Dabei wurden die Gruppen durch entsprechende gegenseitig abgestimmte Termin- und Zwischenergebnisvereinbarungen 'getaktet', teilweise mehrmals wöchentlich. Die Reviews beinhalteten Diskussionen über die Analysen und Entwürfe, sowie deren (teilweise auch werkzeuggestützte) Dokumentation. Jede Gruppe konnte dabei in ihrem eigenen Tempo voranschreiten.

Implementierung und Abnahme Die Implementierung wurde ohne gemeinsame Sitzungen nur mit einem Beratungsangebot per email begleitet. Die formale Abnahme bestand aus der Übergabe der endgültigen Fassungen der obigen Dokumentation und Software.

Konkrete Projekte Im konkreten Durchlauf bildeten sich vier Gruppen: 1. Ein teilnehmereigener Vorschlag für ein Auftragsverwaltungssystem für wärmebehandelnde Betriebe. 2. Projekt 'Time-Tracker': Verteilte (möglicherweise auch Offline-) Erfassung eigener Projekte und Aktivitäten und der für sie aufgewendeten Arbeitszeit, sowie Online-Berichtswesen. 3. Projekt 'Flexibler Grapheditor für allgemeine graphbasierte Strukturen'. 4. Projekt 'Netzmanagement-Unterstützung'. Sonderfall: hier war die Implementierung aus dem letzten Programmierkurs Java schon vorhanden und es war ein Reverse Engineering dazu zu leisten.

6 Erfahrungen

6.1 Verhalten/Probleme der Teilnehmer mit der UML

Bei den Modellierungen durch die Studenten war es sehr auffällig, daß die ersten Modelle sehr implementierungsnahe Entwürfe und nicht problemnahe Beschreibungen der zu lösenden Anforderungen waren. Eine Ursache für dieses häufig anzutreffende Phänomen dürfte daher rühren, daß sich Informatiker primär für Lösungen und nicht für die zu lösenden Probleme interessieren. Es ist für eine erfolgreiche Softwareentwicklung allerdings notwendig, zunächst zu betrachten, *was* das zu entwickelnde System können soll, bevor untersucht wird, *wie* das zu entwickelnde System realisiert werden kann. Hier scheint auch in der Informatikgrundausbildung ein gewisses Umdenken erforderlich zu sein.

Entsprechend dem OOSE-Softwareentwicklungsprozeß [12] sind Problembereichs- und Analysemodelle, zusätzlich zu den Anwendungsfallbeschreibungen, die zentralen Modelle, die im Analyseprozeß zur Anforderungsbeschreibung erstellt werden sollten. Ein *Problembereichsmodell* enthält Objekte mit Gegenstück im realen Problembereich bzw. Konzepte, mit denen ein zu entwickelndes System umgehen soll. Das Problembereichsmodell dient auch als Grundlage für das *Analysemodell*, das eine erste Strukturierung des zu entwickelnden Systems auf hoher Abstraktionsebene liefert [12]. Implementierungsaspekte sollen im Analysemodell noch vernachlässigt werden; das Analysemodell ist aber als ein erster, grober Entwurf anzusehen. Das *Entwurfsmodell* dient in OOSE zur Verfeinerung und Anpassung der Anforderungs- und Analysemodelle an die Systemumgebung (Programmiersprache, Datenbank, Netzwerk etc). Dieses Vorgehensmodell wurde in der Vorlesung vorgestellt, es war jedoch notwendig diese Aspekte in den Projekten erneut zu diskutieren, um die dazu notwendigen Einsichten bei den Studenten zu erreichen.

Das Phänomen, daß die ersten Modellierungen durch die Studenten sehr implementierungsnahe Entwürfe und nicht problemnahe Beschreibungen der zu lösenden Anforderungen waren, sollte auch für die Informatikgrundausbildung ein gewisses Umdenken zur Folge haben. Bei den Studierenden muß Interesse für die zu lösenden Probleme geschaffen werden; nicht nur für die Lösungen. Für eine erfolgreiche Softwareentwicklung ist es notwendig, zunächst zu wissen, *was* erreicht werden soll, bevor der Weg zum Ziel gesucht wird.

6.2 Verhalten/Probleme der Teilnehmer mit Java

Die Teilnehmer hatten relativ wenig Probleme bei der Implementierung ihrer Projekte in Java. Die größten Probleme traten bei den komplexeren Strukturen innerhalb der Klassenbibliotheken auf. Dies betrifft zum einem die GUI-Bibliothek

Swing, die das Model-View-Controller-Konzept realisiert. Dies hat bei komplizierten Elementen wie der Klasse `JTree` zu Problemen geführt, da die sinnvolle Verwendung aus der API-Referenz kaum hervorgeht. Den Teilnehmern war zumeist die praktische Anwendung des MVC-Konzepts nicht geläufig, so daß hier Zusatzarbeit notwendig war.

Beim Einsatz von RMI zur verteilten Programmierung gab es eine Reihe von Verständnisproblemen. Sie beruhten darauf, daß die Entscheidung, welche Objekte auf welchen virtuellen Maschinen implementiert werden, und wie bzw. ob Objektkopien zwischen diesen Maschinen ausgetauscht werden, unklar waren. Auch hier fehlte eine konkretere Anleitung, wie man die Möglichkeiten der entfernten Objekte sinnvoll nutzen kann.

Zusammenfassend bleibt festzuhalten, daß die Beherrschung der sehr großen Klassenbibliothek von Java für die konkrete Verwendung von Java wesentlich ist. Da diese Bibliothek sich zur Zeit aber noch in einer Stabilisierungsphase (für JDK 1.2) befindet, gibt es kaum Literatur, die über eine Referenz der angeboten Klassen und sehr einfachen kleinen Beispielen hinaus gehen. Damit ist der Aufwand für die Bildung des notwendigen technischen Know-Hows für den Bau realistischer Anwendungen sehr hoch.

Unabhängig von Java war zu beobachten, daß trotz BETA in den vorhergehenden Semestern die objektorientierte Modellierung selbst einfachster dynamischer Datenstrukturen (z.B. Graphen) bei einer ganzen Reihe von Teilnehmern immer noch sehr von aus nicht-OO-Sprachen stammenden Repräsentationen geprägt ist (z.B. hier Adjazenzlisten über Knotennummern).

6.3 Werkzeugeinsatz

Für die Realisierung der Projekte und für das Selbststudium haben wir den Teilnehmern einige Java- und UML-Werkzeuge bereitgestellt. Dies waren Rational Rose 98 und Together/J für die Modellierung in UML, und IBM Visual Age for Java 1.0, der Sun Java Workshop 2.0 sowie die Kombination JDK 1.1.3 ff. und Emacs für die Implementierung in Java. Beim Einsatz dieser Werkzeuge beobachteten wir bei **Rational Rose (98)**: Die Java-Codegenerierung hat nicht funktioniert. Das Reverse-Engineering existierender Java-Anwendungen funktionierte gut. Z.T. konnten die Grafiken nicht im Sinne der Studenten editiert werden: die Werkzeugbedienung scheint ohne 'vernünftige' Dokumentation und Schulung zu komplex zu sein. Die 30-Tage-Testlizenz für die Arbeit 'zu Hause' ist für die Projekte wegen der Projektdauer nicht sinnvoll einsetzbar. **Together/J**: Es konnten nicht alle UML-Diagramme verwendet werden, die grafische Notation entsprach zusätzlich nicht der der drei Amigos. Die Bedienung des Programms war etwas gewöhnungsbedürftig und das Drucken der Dokumente nicht möglich. Daher ist Together/J von den Teilnehmern

kaum eingesetzt worden. **IBM Visual Age for Java 1.0, Sun Java Workshop 2.0**: Beide IDEs-Versionen unterstützen leider die visuelle Entwicklung von Swing nicht. **JDK 1.1.x und Emacs 19/20**: Beide Pakete sind einfach und schlicht, aber zuverlässig. Mit der Makropaket JDE (Java Development Environment) für den Emacs wird Java komfortabel (Klassenbrowser, Debugger-Integration, etc.) unterstützt. Im Gegensatz zu den IDEs wird die Arbeit mit mehreren Entwicklern durch Integration von Versionsverwaltungstools durch den Emacs unterstützt.

Unabhängig von den Veranstaltern haben einige Projektteilnehmer noch weitere IDEs ausprobiert, beklagten aber deren Instabilität. Die Werkzeugunterstützung für Java und UML ist zur Zeit nach unseren Erfahrungen noch nicht besonders gut.

6.4 Teilnehmerfeedback

Gegen Ende der Vorlesungszeit wurde von den Veranstaltern ein formales Feedback auf der Basis eines offenen Fragebogens durchgeführt. Leider haben sich nur 20 Studenten daran beteiligt. Die Projekte hatten noch nicht begonnen, so daß nur der Vorlesungsbetrieb begutachtet werden konnte. Besonders gut gefallen haben den Teilnehmern die Aktualität der Themen, die Praxisberichte und das umfangreiche interne Web-Angebot. Kritisiert wurde insbesondere die zu geringe Integration zwischen der UML- und Java-Vorlesung, sowie die Notwendigkeit von mehr Beispielen, um die Konzepte als auch die Integration von UML und Java zu verdeutlichen. Außerdem wurden sowohl der zu große Stoffumfang als auch die Auslassungen der Java-Vorlesung bemängelt: Hier prallen wohl die Vorstellungen der Aufgabe der Vorlesung als Präsentation der Konzeption von Java gegen eine praxisrelevante vollständige Bibliothekenvorstellung gegeneinander. Die Teilnehmer bemängelten auch, daß es keine (vorgefertigten) Programmierübungen parallel zur Vorlesung gab und konkrete Beschäftigung mit den besprochenen Themen der Eigeninitiative der Einzelnen überlassen blieb. Außerdem fand eine Reihe von Teilnehmern, daß der (geschätzte) Zeitaufwand für die (möglichst realistischen) Projekte für sie einen zu großen Aufwand darstellte.

6.5 Veranstalterresümee

Effektivität der eingesetzten Ressourcen Die bereitgestellte Newsgroup wurde nicht verwendet. Das interne Web fand jedoch viel Resonanz und hat sich auch für uns als vorteilhaft erwiesen, weil es neben den Standardmaterialien (wie Folienkopien und Übungsanregungen) einen zusätzlichen Vermittlungskanal für weitergehend Interessierte bereitstellte. Bei den Werkzeugen hat sich der anfängliche Selektionsaufwand aufgrund der oben beschriebenen Probleme nicht ausgezahlt. Die

bereitgestellten Sun-Accounts wurden nur spärlich genutzt: viele Arbeiten wurden (trotz der dort eingeschränkten Funktionalität) 'zu Hause' durchgeführt.

Betreuungsaufwand Unser zeitlicher Aufwand hielt sich für die Vermittlungsphase im vertretbaren Rahmen — trotz der notwendigen Neukonzeption des UML-Anteils und Reorganisation und Aktualisierung des Java-Anteils. Die Projektphase beanspruchte aber unerwartet mehr Aufwand: wir hatten nicht erwartet, daß die Besprechung von Analyse- und Entwurfsergebnissen, die Diskussion von sinnvolleren Alternativen, sowie die teilweise notwendige Wiederholung von Vorlesungsstoff so viel Zeit beanspruchen würde. Zudem machte sich hier die stark implementierungsnahe vorherige Ausbildung negativ bemerkbar. Außerdem hat keine der Gruppen die Option genutzt, ihr Projekt innerhalb von 2 Wochen kompakt abzuschließen. Nur die Hälfte aller Gruppen wurden in den letzten vier Wochen fertig.

Einflüsse auf einen weiteren Durchlauf Mit den jetzt gewonnenen Erfahrungen würden wir einen weiteren Durchlauf folgendermaßen gestalten: Beibehalten würden wir die Trennung in Vermittlungs- und Projektphase und in deren Trennung von Modellierung und Implementierung. Die Vermittlungsphase würde allerdings stark integriert sein: ein gemeinsames Projekt würde durchgängig besprochen und an diesem die Grundlagen der Modellierung und Implementierung quasi exkursionshaft vermittelt. Den Abschluß der Vermittlung würde ein gemeinsames kompaktes Review eines weiteren Projekts darstellen, das als Modellprojekt (u.a. für Ablauf und Dokumentation) für die nachfolgende Projektphase dient. Das lokale Web würde noch stärker in die Veranstaltungskonzeption mit eingebunden: zusätzliche Lese-/Übungsaufgaben sowie alternative Sichtweisen zu den Einzelthemen der Vorlesungen u.a. als Vorbereitung konkreter Teilproblemlösungen in späteren Projekten und vor allem der Gewinnung von Erfahrung in Umgang mit Werkzeugen und Bibliotheken. Weiterhin (über den Debugger JDB hinausgehendes) praxisrelevantes Material über den Test objektorientierter und nebenläufiger Systeme: dies wurde im aktuellen Durchlauf vernachlässigt. Die Projektphase ist durch das eingesetzte Coaching-Verfahren (notwendigerweise) sehr betreuungsintensiv. Für deren Verbesserung wäre es sinnvoll, daß die Studierenden schon frühzeitig Modellierungserfahrungen hätten sammeln oder Modellierungen hätten studieren können. Eine solche Orientierung an bewährten Entwürfen und Teilentwürfen (Analyse- und Entwurfsmusterorientierung, z.B. [7, 9]) ist allerdings im derzeitigen Dortmunder Kanon nicht erst im 4. Semester sinnvoll und würde zudem diesen Kurs stark überfrachten. Somit bleibt hier nur, die gruppenindividuellen Terminvereinbarungen einem stärkeren Struktur zu unterziehen.

Literatur

[1] ARNOLD, K. und J. GOSLING: *Java – Die Programmiersprache*. Addison-Wesley, 1996.

[2] BOOCH, G.: *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings, 2. Auflage, 1994.

[3] BURKHARDT, R.: *UML – Unified Modeling Language: Objektorientierte Modellierung für die Praxis*. Addison-Wesley, Bonn, 1997.

[4] CAMPIONE, MARY und KATHY WALRATH: *The Java Tutorial 2nd Edition. Programming for the Internet.* Addison-Wesley, 1998. Online: `http://java.sun.com/docs/books/tutorial/`.

[5] DEMARCO, T.: *Structured Analysis and System Specification.* Prentice Hall, Englewood Cliffs, NJ, 1985.

[6] DOBERKAT, E.-E. und S. DISSMANN: *Einführung in die objektorientierte Programmierung mit BETA.* Addison-Wesley, 1996.

[7] FOWLER, M.: *Analysis Patterns: Reusable Object Models*. Addison-Wesley, Reading, MA, 1997.

[8] FOWLER, M. und K. SCOTT: *UML konzentriert.* Addison-Wesley, 1998.

[9] GAMMA, E., R. HELM, R. JOHNSON und J. VLISSIDES: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[10] HARMON, P. und M. WATSON: *Understanding UML: The Developer's Guide With a Web-Based Application in Java*. Morgan Kaufmann, 1998.

[11] HASSELBRING, W.: *Erfahrungen mit dem Einsatz von Objectory für vorlesungsbegleitende Übungen in der Softwaretechnik-Ausbildung*. Softwaretechnik-Trends, 16(2):10–15, Mai 1996.

[12] JACOBSON, I., M. CHRISTERSON, P. JONSSON und G. ÖVERGAARD: *Object-Oriented Software Engineering – A Use Case Driven Approach.* Addison-Wesley, Reading, MA, 1992.

[13] LARMAN, C.: *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*. Prentice Hall, 1998.

[14] LEA, DOUG: *Concurrent Programming in Java - Design Principles and Patterns.* Addison-Wesley, 1996.

[15] OESTEREICH, B.: *Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language*. R. Oldenbourg Verlag, München, 1997.

[16] RUMBAUGH, J., M. BLAHA, W. PREMERLANI, F. EDDY und W. LORENSEN: *Object-Oriented Modelling and Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[17] SCHMEDDING, D.: *Teamarbeit im Software-Praktikum – Ein Erfahrungsbericht.* In: *Software Engineering im Unterricht der Hochschulen SEUH'95*. Teubner, 1995.

[18] WALDÉN, K. und J.-M. NERSON: *Seamless Object-Oriented Software Architecture – Analysis and Design of Reliable Systems.* Prentice-Hall, 1995.

A Projektanregungen

1. Netzmanagement-Unterstützung: Gemeinsame Datenhaltung von Betriebs- und Bestandsdaten zur Netzmanagement und Inventarisierungsunterstützung. Betriebsdatenerhebung über definierbare Skripte oder SNMP-Polling.
2. Systembetriebsdatenanzeige: in einer Webseite frei parametrisierbare Verlaufsanzeigen-Applets. Periodische Datenbeschaffung über SNMP.
3. Raumeinrichtungsplaner
4. Flexibler Grafikeditor für allgemeine graphbasierte Strukturen
5. StateChart-Editor
6. Mindmap-Editor
7. Metaplan-Vorbereitungswerkzeug: Vorbereitung/Dokumentation von Pinnwandarbeit.
8. Unterstützung formularbasierten Arbeitens: Erfassung in definierbare Eingabefelder in eingescannten oder per PS definierten Papierformularen.
9. Time Tracker: Verteilte (möglicherweise auch Offline-)Erfassung eigener Projekte und Aktivitäten und der für sie aufgewendeten Arbeitszeit, sowie Online-Berichtswesen.
10. 'personal/group organizer'-Bestandteile (mit Unterprojekten)
11. Dokumentkommentierungswerkzeug: Annotationen (Grafik, Text) zu vorhandenem Druckfassungsdokument (PS, PDF) anfügen.
12. Autorennen: nicht-grafische Fassung, wo es um Fahrtstrategie und Taktik geht. Reale Wettrennen über dynamische Komponenteneinbindung und Web-Anzeige.
13. Visualisierung von Komponentenkopplungen bei Systembeschreibungen
14. Zugriff/Koordination verteilter Literaturstellendaten und Glossare
15. Werkzeug zum Ordnen von Ideen/Gedanken/Material
16. Online-Raumreservierung: Web-basierte Besprechungsraum- oder Ressourcenbuchung
17. Time/Control-Zeitabrechnungs-Client und Business Logic (2 von 3 tiers): Entwurf der Business Logic und Nachimplementierung der umfangreichen GUI in Java eines kommerziellen verbreiteten Zeitabrechnungssystems.
18. Unterstützung virtueller Gruppen: verteiltes Autoren/ Übersetzerteam
19. Experten/Projektdatenbank im Web: interne Web-Erfassung/Pflege sowie Web-Recherche von Expertisenkontakten innerhalb einer Organisation.
20. Problem Tracking System in Java: vgl. GNU GNATS
21. Konferenzmanagementsystem: Web-basiertes Management des Einreichungs-, Review- und Benachrichtigungsprozesses einer wissenschaflichen Konferenz.