1 Verkaufsanwendungen auf Basis des Anwendungsframeworks SalesPoint

1.1 Frameworkbasiertes Softwarepraktikum an der TU Dresden

Wir berichten über ein Ausbildungsprojekt an der Fakultät Informatik der TU Dresden, in dem erstmals Java in einem Praktikum zur arbeitsteiligen Softwareentwicklung eingesetzt wurde. Im Rahmen eines einsemestrigen komplexen Softwarepraktikums für Informatik-Studenten im Grundstudium wurde auf Basis eines gegebenen Anwendungsframeworks von 21 studentischen Projektteams jeweils eine Verkaufsanwendung entwickelt. Die Ähnlichkeit der Praktikumsaufgaben sowie das gleiche Projektumfeld fundieren unsere Erfahrungen mit Java als objektorientierter Entwicklungssprache.

Ein Problem des objektorientierten Ansatzes ist die bekannt lange Einarbeitungszeit. Um Studierende möglichst frühzeitig mit moderner, objektorientierter Programmiertechnik zu konfrontieren, wurden in der Grundvorlesung Softwaretechnologie (2. Semester, Pflichtfach) objektorientierte Analyse (OOA) und objektorientierter Entwurf (OOD) mit CRC-Karten und Unified Modeling Language (UML) sowie Java als Implementierungssprache eingeführt. Anhand der Klassenbibliothek von Java wurden Muster und Rahmenwerke (Frameworks) dargestellt; in den Übungen sammelten die Studierenden erste Erfahrungen mit diesen Hilfsmitteln. Die inhomogenen Vorkenntnisse ergaben Schwierigkeiten, die z.T. erst im anschließenden Softwarepraktikum überwunden werden konnten: Anfänger mußten den Schritt vom „Programmieren im Kleinen“ zum „Programmieren im Großen“ vollziehen, die Erfahreneren sich auf das neue Paradigma einstellen.

In der Praxis ebenso wie in längerlaufenden Projekten an der Universität besteht die erste Aufgabe von Hinzukommenden darin, sich in das bereits

bestehende Programmsystem einzuarbeiten, an dessen Weiterentwicklung sie mitarbeiten sollen. Um diese Situation zu simulieren und gleichzeitig ein innovatives Vorgehen zu vermitteln, haben wir uns entschlossen, die Praktikumsaufgaben auf einem gemeinsamen Framework aufzusetzen, das eigens für diese Zwecke ausgearbeitet wurde. Auf diese Weise konnten die Studierenden in einem angemessen begrenzten und realistischen Rahmen zuvor erlernte Techniken der objektorientierten Softwareentwicklung anwenden, die Teamarbeit erlernen sowie die Idee der Wiederverwendung von vorhandener Software praktizieren.

1.1.1 Fachlichkeit und Funktionalität

Verkaufsanwendungen werden als Anwendungen verstanden, die Transaktionen über Katalogen und Beständen ausführen und verwalten. Ein **Katalog** ist eine Liste von Gegenständen und deren Eigenschaften. Jeder Katalogeintrag enthält eine Bezeichnung (z.B. Name oder Bestell-Nummer) und im kaufmännischen Bereich einen Wert. Je nach Anwendung kommen weitere Eigenschaften wie Farbe, Maße, Hersteller usw. hinzu. **Bestände** beziehen sich jeweils auf einen bestimmten Katalog und geben zu jedem Katalogeintrag im einfachsten Fall an, wie oft der Gegenstand im Bestand vorhanden ist. Besondere Bestände sind **Geldbestände**, wie sie in Kassen vorkommen und wie wir sie im Geldbeutel haben. Der zugehörige Katalog wird als **Währung** bezeichnet.

Die bisher entwickelten Verkaufsanwendungen sind:

- Fahrkartenautomat
- Wechselstube
- Postschalter
- Getränkemarkt
- Duty-Free-Shop
- Kino
- Foto-Service
- Restaurant-Betrieb
- Versandhaus-Agentur
- Schulbibliothek

Allen Anwendungen sind im wesentlichen folgende Anwendungsfälle gemeinsam:

- „Verkaufstransaktionen“, z. B. Geldtausch, Kauf, Tausch, Rückkauf, Auftragsverwaltung, Ausleihvorgänge
- Tagesabschluß und Gewinnermittlung
- Aktualisieren von Katalogen
- Bestellungen bei Zulieferern

Die Anwendungsfälle sind zum Teil automatisierte Geschäftsprozesse, die durch einen definierten Anfang und ein definiertes Ende gekennzeichnet sind. In diesem Sinne können die realisierten Verkaufsanwendungen als **Workflow-Anwendungen** betrachtet werden.

Zur Illustration einer Verkaufsanwendung sei ein Getränkemarkt betrachtet. Im folgendem ist ein Ausschnitt aus der Praktikumsaufgabenstellung gegeben.

Entwickeln Sie eine Software für den Betrieb eines Getränkemarktes. Dabei seien zunächst nachfolgende Anforderungen bekannt:

Kunden kaufen Getränke Flaschen- bzw. Trägerweise. Es gibt eine Vielzahl von Getränkesorten und Flaschengrößen. Verzahnt mit dem Getränkeverkauf ist der Flaschen- bzw. Trägerverleih. Flaschen sowie Träger gibt es in verschiedenen Größen. Das Sortiment an Getränken und Pfandgut wechselt ständig. Ladenhüter werden aus dem Sortiment genommen. Es muß rechtzeitig nachbestellt werden. Dabei sind die Lieferfristen der Großhändler (Tage bis wenige Wochen), die maximale eigene Lagerkapazität und der aktuelle Lagerbestand zu beachten. Sämtliche Transaktionen müssen protokolliert werden.
Berücksichtigen Sie folgende typische Anwendungsfälle:

- Kauf von Getränken
- Ausleihe von Flaschen und Trägern
- Rückgabe von Pfandgut
- Bestellungen bei Großhändlern
- Tagesabschluß und Umsatzermittlung
- Verändern der Bestände

Die Aufgabenstellung zum Getränkemarkt wurde von drei Projektteams in jeweils eine Java-Anwendung umgesetzt. Eines der Projekte ist mit einer „interaktiven“ Benutzer-Demonstration im World Wide Web (WWW) verfügbar (http://www.inf.tu-dresden.de/~rs27/Anleitung/). Für den Verkäufer gibt es dort beispielsweise ein Kundentransaktionsfenster (Abbildung 1), über welches alle Transaktionen realisiert werden, die direkt mit dem Kunden in Verbindung stehen. Dazu gehört neben dem Verkauf von Artikeln die Rücknahme von Pfandgut. Zum Verkauf eines Artikels ist dieser auszuwählen und die Anzahl einzustellen. Dabei werden nur Artikel angeboten, die auch tatsächlich am Lager sind. Auf die gleiche Weise können weitere Artikel ausgewählt werden. Der Verkauf wird mittels Klick auf die Taste `Einkauf` eingeleitet. Es öffnet sich ein Fenster mit der Angabe des zu zahlenden Preises (aufgeschlüsselt nach Preis und Pfandwert). Der Verkauf kann dann bestätigt werden. Das Pro-

gramm aktualisiert automatisch den Lagerbestand und den Kassenbetrag. Die Rückgabe von Pfandgut erfolgt analog durch Klick auf Rückgabe.

Abbildung 1: Kundentransaktionsfenster eines Getränkemarktes

1.1.2 Projektmanagement und -verlauf

Das Java-basierte Softwarepraktikum wurde im Wintersemester 1997/98 unter strengen zeitlichen Restriktionen durchgeführt. Zunächst wurden 22 studentische Projektteams mit 3 bis 6 Teammitgliedern gebildet. Jedes Team erhielt die Aufgabe, eine der genannten Verkaufsanwendungen unter Verwendung des Frameworks *SalesPoint* objektorientiert zu entwikkeln. Jede einzelne Aufgabenstellung wurde maximal dreimal vergeben. Die konkreten Anforderungen an die Verkaufsanwendung waren verbal spezifiziert. Das Softwareprodukt sollte eine reine Java-Anwendung werden und über eine grafische Oberfläche verfügen. Nach Ablauf des Praktikums (14 Wochen) konnten 21 Teams ihr Java-Projekt im Rahmen einer öffentlichen Veranstaltung erfolgreich präsentieren. Die parallele Bear-

beitung und gemeinsame Präsentation gleicher Aufgabenstellungen wirkte als Konkurrenzsituation und belebte die Diskussionen.

Jedem Team war ein Betreuer zugeteilt, der neben der softwaretechnologischen Betreuung und Beratung die Rolle des Kunden bzw. Auftraggebers für das Softwareprojekt übernahm. Die Ergebnisse jeder Phase wurden im Rahmen von Pflichtkonsultationen mit den Betreuern besprochen und in einer Entwicklungsdokumentation pro Projektteam zusammengefaßt. Auf diese Weise konnte der Arbeitsfortschritt der Projektteams kontrolliert und ggf. in die Projektorganisation eingriffen werden. Zusätzlich zur persönlichen Betreuung der studentischen Teams wurde das Praktikum durch Informationen (Organisatorisches, Hilfestellungen u.a.) im WWW [3] unterstützt.

Teamorganisation

Die Bildung der Projektteams erfolgte nach dem Prinzip der Chefprogrammiererorganisation. Einem Studenten wurde die Funktion des Chefprogrammierers übertragen, die anderen Gruppenmitglieder fungierten als Mitarbeiter. Der Chefprogrammierer übernahm die Rolle des Teamleiters und war in erster Linie für die Kommunikation mit dem Praktikumsbetreuer, die Teamorganisation, für alle wichtigen Entwurfsentscheidungen sowie die qualitäts- und termingerechte Erstellung des Gesamtsystems (Softwareprodukt und Dokumentationen) verantwortlich. Die anderen Teammitglieder teilten sich in die Aufgaben bzw. Rollen des Assistenten, des Sekretärs sowie der Programmierer. In Ergänzung zu diesem allgemeinen Verständnis wurde von jedem Student verlangt, daß er unabhängig von seiner Rolle im Team einen Teil der Implementation eigenständig übernahm. Im Ergebnis des Entwurfes, an dem alle Teammitglieder beteiligt sein sollten, erfolgte die Festlegung, wer für die Implementation welcher Klassen bzw. Teilsysteme verantwortlich ist. Die konkrete Arbeitsteilung in der Gruppe wurde durch das Team festgelegt und im Projektplan festgeschrieben. In Einzelfällen erfolgte während der Projektbearbeitung ein Rollentausch im Team.

Aufwände und Ergebnisse

Der durchschnittliche wöchentliche Arbeitsaufwand pro Teammitglied wurde von den Studenten auf knapp 10 Stunden geschätzt, wobei zwei bis drei Stunden auf Teamabsprachen, der Rest auf eigene Teilaufgaben entfallen. Dabei ist zu beachten, daß die konkreten Angaben stark differieren. Die Angaben der Projektteams reichen von minimal 4.5 bis maximal 19

Stunden durchschnittlichen Gesamtaufwand pro Teammitglied und Woche.

Eine qualitative Bewertung der präsentierten Softwareprojekte erfolgte durch die Teams selbst. Die meisten Teams sahen ihre Aufgabenstellung als voll erfüllt (5 mal) bzw. erfüllt mit geringen Abstrichen (13 mal) an. Abstriche gab es an der Funktionalität, der beschränkten Wiederverwendbarkeit der Lösung und an der Robustheit der Java-Anwendung. Ein weiteres Team bewertete ihr Praktikum als übererfüllt.

Java versus C++

Die Entscheidung für Java als Implementierungssprache im Fach Softwaretechnologie und für das Praktikum wurde nach mehrjährigen Lehrerfahrungen mit C++ getroffen. Unser Ziel war und ist, den Studierenden objektorientierte Techniken am Beispiel einer weit verbreiteten und praxisrelevanten objektorientierten Programmiersprache zu vermitteln. Im Rahmen unseres Studienplanes Informatik bedeutet das, in einem Semester neben softwaretechnologischem Grundwissen und objektorientierten Analyse- und Entwurfstechniken das „Handwerkzeug“, die Programmierung in einer objektorientierten Sprache zu lehren. Aufgrund der Komplexität und der dadurch erschwerten Erlernbarkeit von C++ genügte die zur Verfügung stehende Zeit früher lediglich dazu, daß die Studenten am Ende des Semesters C++-Programme lesen und einfache Klassen schreiben konnten. Eine umfassende Ausbildung in C++ und damit gute Vorbereitung auf das anschließende Softwarepraktikum war nicht möglich. Die im hier beschriebenen Praktikum verwendete Sprache Java bietet dagegen aufgrund eines eleganteren und saubereren Sprachentwurfes in kürzerer Zeit die Möglichkeit, objektorientierte Programmierung zu erlernen. Neben technischen Aspekten sahen wir in der freien Verfügbarkeit von Java auf vielen Plattformen für die Studierenden die großartige Möglichkeit, praktische Java-Übungen nicht nur im Rechnerlabor der Fakultät, sondern auch auf privaten PC durchzuführen. Dazu kommt die allgemeine Begeisterung für Java als „Internet-Sprache“, die die Motivation für das Softwarepraktikum verbessern half.

1.1.3 Technische Architektur

Die Anwendungen wurden auf unterschiedlichen Plattformen entwickelt. Das schließt Unix-Plattformen (Solaris, Ultrix, Linux), Windows95/NT, Mac OS und Amiga OS ein. Zur Sicherung der Kompatibilität mit dem verwendeten Framework mußten die Java-Anwendungen auf Basis von

JDK 1.1.x implementiert werden. Die Klassen des Frameworks SalesPoint standen im WWW zur Verfügung.

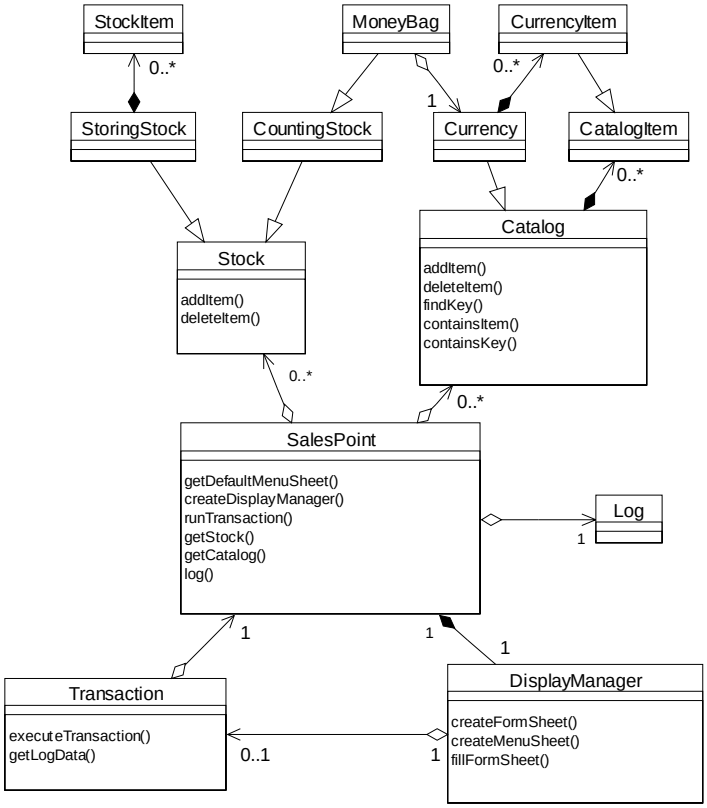

Abbildung 2: Ausschnitt aus dem Klassendiagramm von *SalesPoint*

Architektur von SalesPoint

Das Framework *SalesPoint* (siehe Abbildung 2) unterstützt die Entwicklung von Verkaufsanwendungen, indem es zu den genannten Verkaufsbegriffen und Anwendungsfällen vorgefertigte Komponenten, wie z.B. das Stückeln eines Geldbestandes, bereitstellt. Neben fertigen Komponenten enthält das Framework viele Hook-Methoden, die durch anwendungsspezifische Algorithmen redefiniert werden können.

Zentrale Komponente ist der Verkaufsstand (Klasse *SalesPoint*), der die Verkaufsstelle, den Schalter oder den Automaten repräsentiert. An ihm

werden sowohl Bestände (*Stock*) und Kataloge (*Catalog*) als auch Transaktionen (*Transaction*), Ein- und Ausgaben sowie Protokolldateien (*Log*) angebunden. Diese können direkt über Methoden von *SalesPoint* angesprochen werden. Durch das Anlegen eines neuen SalesPoint-Objektes wird der Verkaufsstand gestartet. In einem System kann es beliebig viele Verkaufsstände geben, die auf gemeinsame Kataloge und Bestände zugreifen können. Sinnvoll erscheint das zum Beispiel in einem Warenhaus, wo es mehrere Kassen geben kann.

Kataloge und Bestände verwalten die Daten der Verkaufsanwendung und sind auf Basis der Java-Datenstrukturen (*java.util.**) und des Mechanismus der Objekt-Serialisierung implementiert. Die Klasse *DisplayManager* ist Bestandteil eines Framework-Packages, das Komponenten, insbesondere Formulare (*FormSheet*), zur Realisierung von „Verkaufsoberflächen“ zur Verfügung stellt. Für die Implementierung der darin vorgefertigten Nutzeraktionen wurden innere Java-Klassen und das AWT-Package benutzt. Die Menü- und Transaktionssteuerung wurde mit Hilfe von Threads implementiert.

Bei der Vermittlung der objektorientierten Entwurfsmethodik als auch im Prozess der Framework-Entwicklung wurde großer Wert auf das Verständnis bzw. die sinnvolle Anwendung von Entwurfsmustern [2] gelegt. Die von Frameworks geforderte Flexibilität wird im wesentlichen durch die geeignete Plazierung von Einschubmethoden (Hook-Methoden) erreicht, durch deren Überschreiben das Verhalten der Objekte verändert und damit die vorgefertigten Softwarebausteine an spezielle Anforderungen anpaßt werden können [4]. Dieses für Frameworks typische Entwurfsmuster (Template Method) kam auch in der Anwendung von *SalesPoint* zum Tragen. Das Muster wird beispielsweise verwendet, um den Kern einer Transaktion zu definieren. Der Programmierer muß lediglich die Methode *Transaction.executeTransaction()* redefinieren, welche von einem wesentlich umfangreicheren (Framework-)Algorithmus aus aufgerufen wird. Dieser sorgt dafür, daß

- Rollback–Informationen gesichert und gegebenenfalls wieder restauriert werden
- die Transaktion beim *DisplayManager* an- und abgemeldet wird
- eine ordnungsgemäße Fehlermeldung ausgegeben wird, wenn in *executeTransaction()* eine Ausnahme auftritt.

Ein weiteres wichtiges Muster ist Abstract Factory. Es wird beispielsweise angewandt, um einen Standard-Katalog zu erzeugen, wenn ein noch nicht vorhandener Katalog angefordert wird, oder um die konkrete *Dictionary*-

Implementation eines Katalogs bzw. Bestandes zu erzeugen. Das Bridge-Muster dagegen erlaubt die Wahl zwischen verschiedenen Implementierungen von Beständen und Katalogen.

Architektur der Verkaufsanwendungen

Die entwickelten Verkaufsanwendungen wurden entsprechend der Aufgabenstellung als einfache Anwendungen auf Basis von *SalesPoint* entwikkelt. Sie verfügen über keine Client-Server-Architektur, die Datenverwaltung basiert auf dem Konzept der Objektserialisierung von JDK 1.1. Fortgeschrittene Java-Technologien, wie z.B. die JDBC- oder RMI-Programmierung wurden aufgrund der sehr begrenzten Entwicklungszeit und der Tatsache, daß die Entwickler Studierende des 3. Semesters waren, nicht angewendet.

Ein Projektteam realisierte einen netzwerkfähigen Duty-Free-Shop auf Basis der Socket-Programmierung in Java (*java.net*). Diese Architektur (Komponentendiagramm siehe Abbildung 3) beruht auf der Idee, daß ein zentraler Verkaufsstand (*DutyPoint* als Spezialisierung von *SalesPoint*) alle Kataloge und Bestände verwaltet und verschiedene Kassen (*CashPoint*) starten kann. Dabei muß die Kasse nicht wissen, ob sie lokal zum *DutyPoint* oder auf einem anderen Rechner läuft. Dazu wurden spezielle Transfer-Klassen entwickelt, die entweder eine lokale oder eine Netzwerkverbindung zwischen *DutyPoint* und *CashPoint* realisieren.

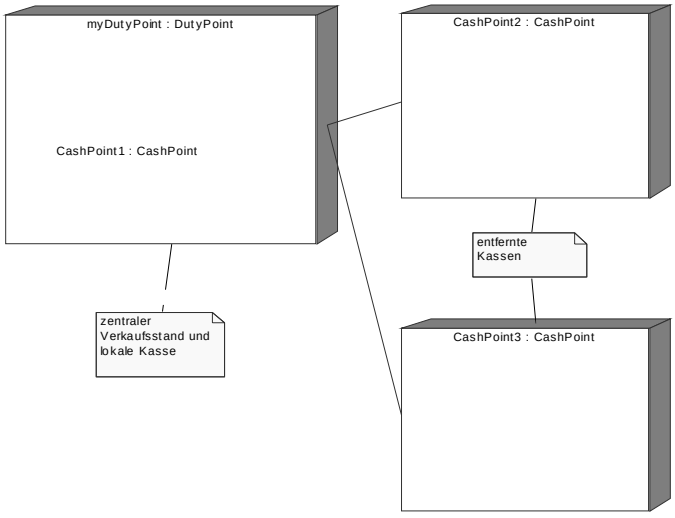

Abbildung 3: Komponentendiagramm eines netzwerkfähigen Duty-Free-Shops

1.1.4 Mengengerüste

Nach Abschluß des Praktikums wurde ein speziell für die durchgeführten Java-Projekte entwickelter Fragebogen von den Studenten ausgefüllt, der neben qualitativen auch quantitative Wertungen zum Grad der Wiederverwendung des Frameworks und zu einfachen Metriken enthält. Aus den Ergebnissen der Befragung sowie der softwaretechnischen Messung von Metriken (vgl. Tabelle 1) lassen sich folgende Aussagen ableiten:

- Der Grad der Wiederverwendung einzelner Komponenten des Frameworks wurde von den Projektteams mit „gar nicht“ (1) bis „vollständig“ (5) bewertet. Die Auswertung ergab, daß die Klassen *SalesPoint*, *Transaction, Display*, deren abgeleitete Klassen sowie die Datenverwaltungskomponente (u.a. *Stock* und *Catalog*) fast vollständig wiederverwendet wurden (Bewertung 4 und 5). Als Gründe für die Nichtanwendung vorgefertigter Bausteine nannten die Studenten, daß die Klassen entweder nicht benötigt wurden (z.B. Verwaltung verschiedener Währungen mit *Currency*) oder daß diese zu kompliziert bzw. unkomfortabel in der Anwendung waren.
- Die Komplexität der Entwürfe für die Verkaufsanwendungen differiert von 9 bis 153 Klassen. Durchschnittlich wurden 40 Klassen implementiert. Die große Differenz in der Anzahl der implementierten Klassen erklärt sich aus
  - der unterschiedlichen Komplexität einzelner Klassen (1 bis 59 Methoden pro Klasse);
  - dem Umfang der durch das Projektteam präzisierten Aufgabenstellung (Anforderungsspezifikation);
  - dem Grad der Wiederverwendung von Framework-Klassen;
  - der Anwendung bzw. Nichtbeachtung von Entwurfsmustern, um die Verkaufsanwendung selbst wiederverwendbar zu gestalten.
- Die Vererbungshierarchien der implementierten Verkaufsanwendungen sind durch eine durchschnittliche maximale Tiefe von 3 und eine durchschnittliche maximale Breite von 8 bis 9 charakterisiert. Die Klassen besitzen durchschnittlich 6 Methoden. Die größte Breite von Klassenhierarchien entstand durch die Spezialisierung von Transaktionen.
- Der implementierte Java-Code (LOC: Lines Of Code) pro Verkaufsanwendung (ohne Framework-Code) weist analog zur Anzahl der Klassen eine große Schwankungsbreite auf (689 bis 10404 LOC). Im Durchschnitt wurden pro Anwendung 3045 LOC implementiert. Die im Umfang größeren Verkaufsanwendungen sind durch eine gegen-

über der Aufgabenstellung zusätzlich projektierte Funktionalität gekennzeichnet, wie z.B.
    - ein netzwerkfähiger Duty-Free-Shop mit einem zentralen Verkaufsstand und mehreren Kassen (Gruppe 1-4b; gefordert war ein Stand-alone-Duty-Free-Shop);
    - eine Schulbibliothek mit der Möglichkeit zur Formulierung mächtiger Suchausdrücke (Gruppe 1-9b; gefordert war die Realisierung einfacher Recherchen).
- Durchschnittlich wurden pro Studierenden 677 LOC implementiert. Die in Tabelle 1 gegebene Anzahl von Bearbeitern bezieht sich auf die erfolgreich am Praktikum teilgenommenen Studenten.
- Zur Abschätzung der „studentischen Projekteffizienz“ ist die Anzahl LOC pro Student und Stunde von Interesse. Diese Metrik schließt den gesamten Entwicklungsaufwand für das Softwareprodukt ein. Bei einer zugrunde gelegten realen Bearbeitungszeit von 12 Wochen ergeben sich durchschnittlich 5.9 LOC pro Student und Stunde. Dabei ist jedoch zu beachten, daß die Projekteffizienz der einzelnen Teams aufgrund der unterschiedlichen Größe der realisierten Anwendungen und der dazu benötigten Zeit von 0.9 bis 18.1 LOC pro Student und Stunde variiert.

Die Metriken vermitteln in ihrem Zusammenhang einen Eindruck davon, daß bei den einzelnen Projekten ein weitgehend ähnlicher Entwurfs- und Programmierstil erreicht werden konnte. Dieser beruht auf dem in den vorangegangenen Lehrveranstaltungen angewendetem Stil der Standard-Java-Klassenbibliotheken und dem Draft Java Coding Standard [4].

Team	Bear-beiter	Verkaufs-anwendung	LOC	Klassen	Komplexität der Klassen				
					Anzahl der Methoden			LOC	
					Durchschnitt	max	min	min	max
1-1a	5	Wechsel-	1664	17	5,2	19	1	7	389
1-1b	2	stube	1222	9	6,3	18	2	26	315
1-1c	5		1683	31	4,5	19	1	9	217
1-2a	5	Postschalter	2106	25	4,2	15	1	9	272
1-3a	5	Getränke-	1538	30	3,4	12	0	11	296
1-3b	4	markt	4381	53	7,2	43	1	7	425
1-3c	5		1898	20	6,7	20	1	7	226
1-4a	5	Duty-Free-	2987	42	4,8	21	1	12	530
1-4b	4	Shop	10404	110	8,6	51	0	3	838
1-5b	5	Kino	3147	34	6,2	38	2	15	265
1-5c	5		7976	74	8,1	59	1	11	753
1-6a	4	Foto-	765	10	4,9	11	2	8	166
1-6b	4	Service	2345	20	8,7	37	1	11	693
1-7a	1	Restaurant-	1507	25	5,2	13	1	9	232
1-7b	5	Betrieb	689	14	4,4	13	0	5	238
1-7c	5		4269	42	5,4	46	0	8	723
1-8a	5	Versand-	2618	39	5,8	31	1	8	404
1-8b	5	haus-	1173	16	4,2	23	1	14	170
1-8c	5	Agentur	1835	19	7,7	20	1	12	345
1-9a	5	Schul-	2874	63	3,6	19	1	7	205
1-9b	5	bibliothek	6859	153	4,4	47	0	4	391
Durch-schnitt	4,5		3045	40,3	5,7	27	0,9	9,7	385
		SalesPoint	3741	87	5	28	0	3	244

Tabelle 1: Metriken

1.2 Der Entwicklungsprozeß, die Methodik

Verschiedene Aspekte des Entwicklungsprozesses und der objektorientierten Methodik waren durch das Lehrkonzept und die zugrunde gelegte Lehrveranstaltung vorgegeben. Die Projektteams hatten jedoch insbesondere hinsichtlich eingesetzter Entwicklungsplattformen, Entwicklungsumgebungen, Tools sowie ihrer Projektorganisation Freiräume der Ausgestaltung .

Software-Entwicklungsmodell

Die Vorgabe der zeitlichen Verteilung der Projektphasen Praktikumseinweisung/Projektplanung (2 Wochen), Analyse (2 Wochen), Entwurf (2 Wochen), Implementation/Test (4 Wochen) sowie Wartung/Pflege (4 Wochen) hatte empfehlenden Charakter und wurde von den Projektteams

unterschiedlich umgesetzt. Dabei wurde bewußt eine kurze Bearbeitungszeit für OOA/OOD vorgeschlagen. Die Erfahrungen aus früheren Jahren besagen, daß die Studenten die Arbeit anfangs unterschätzen bzw. häufig die Stoßarbeit zum Ende des Semesters bevorzugen. Zum anderen ist eine Projektdurchführung nach dem Wasserfallmodell oft unrealistisch: nach den Erfahrungen aus dem Praktikum waren in Durchschnitt zwei bis drei Iterationszyklen erforderlich. Durch den fest vorgegebenen Endtermin des Praktikums blieb für eine qualifizierte Wartung der entstandenen Anwendung mehr oder weniger Zeit (durchschnittlich zweieinhalb Wochen). Die Wartungsphase wurde durch den Einsatz der Anwendung beim Betreuer und dessen sich daraus ergebenden Wünschen simuliert. In vielen Fällen wurde die für die Wartungsphase geplante Zeit jedoch für einen gründlichen Test der Anwendung benötigt.

Einige Gruppen haben ganz bewußt ein evolutionäres Entwicklungsmodell angewendet und während des Projektverlaufes Prototypen ihrer Verkaufsanwendung erstellt. Diese Prototypen wurden nicht nur von den Betreuern, sondern auch von potentiellen realen Kunden (Post, Getränkemarkt) oder anderen „Endnutzern“ (Freunde) getestet und auf Brauchbarkeit evaluiert.

Der Abschlußbeleg mußte sowohl eine Entwicklungsdokumentation (mit während des Praktikums entstandenen Dokumenten) als auch eine Anwenderdokumentation enthalten. Die vollständige oder teilweise Bereitstellung der Dokumentationen im WWW war ausdrücklich gewünscht.

Spezifikationsmethodik

Randbedingungen der durch die Projektteams selbst gestalteten Entwicklungsprozesse waren der Einsatz der CRC-Karten-Methode in der Analysephase sowie die Verwendung von UML für die Notation der Analyse- und Entwurfsmodelle. Als typische Entwicklungsdokumente entstanden Data Dictionarys (als Ausgangspunkt für die CRC-Karten-Analyse), CRC-Karten sowie UML-Anwendungsfall-, Sequenz- und Klassendiagramme.

Frameworkbezogene Analyse

Ein interessantes softwaretechnologisches Problem ist die Einbindung des Anwendungs-Frameworks in den Softwareprozeß. Eine wichtige Forderung der Aufgabenstellung war, die Java-Anwendung unter weitgehender Wiederverwendung des Frameworks *SalesPoint* zu implementieren. Dabei sind zwei Herangehensweisen denkbar. Erstens läßt sich die Analyse der

Aufgabenstellung **unabhängig** vom Anwendungs-Framework durchführen. Die Anpassung des Modells an das Framework erfolgt in der Entwurfsphase und erfordert dann unter Umständen wesentliche Modelländerungen. Eine zweite Möglichkeit ist, sich zunächst in das Objektmodell des Frameworks einzuarbeiten, um anschließend die Projektspezifikation **frameworkbezogen** zu analysieren. Wir wählten aus Effizienzgründen die frameworkbezogene Analyse. Dazu mußten die Studenten zunächst das Framework verstehen. Hilfe boten die im WWW zur Verfügung stehenden Online-Dokumentationen und eine Beispielanwendung für einen Fahrkartenautomaten. Technische Fragen bzw. auftretende Probleme zum Framework konnten elektronisch oder auch persönlich mit dem Framework-Entwickler (Student des gleichen Studienjahres) diskutiert werden. In der Analysephase waren zunächst für die vorhandenen Klassen des Frameworks CRC-Karten zu erstellen (Reverse Engineering). Die Framework-CRC-Karten bildeten den Ausgangspunkt für die eigentliche CRC-Karten-Analyse der zu entwickelnden Verkaufsanwendung.

Entwicklungsumgebung und eingesetzte Tools

Als Entwicklungsumgebung auf Windows-Rechnern stand Symantec Café zur Verfügung. Eine ganze Reihe von Teams haben ihre Anwendung direkt mit dem Sun Java Compiler und dem JDK entwickelt. Auch innerhalb der Projektgruppen wurden unterschiedliche Entwicklungsplattformen und -umgebungen eingesetzt. Zur Dokumentation der implementierten Java-Klassen wurde der Java-Dokumentationsgenerator *javadoc* benutzt. Für die Notation von UML-Diagrammen kamen Demolizenzen, (vor allem Rational Rose) sowie diverse Grafikprogramme zum Einsatz. In Einzelfällen wurde für die Unterstützung des Projektmanagements MS Project verwendet. Das wichtigste Kommunikationstool für die Studenten waren Internet-Browser und Mail-Programme.

1.3 Kritische Bewertung der durchgeführten Java-Projekte

Die folgenden Aussagen resultieren sowohl aus der Entwicklung des Frameworks, aus Erfahrungen der Projektteams als auch aus Beobachtungen der Praktikumsbetreuer.

1.3.1 Vor- und Nachteile bzw. Stärken/Schwächen von Java

Die Meinungen Studenten zu Java sind unterschiedlich, insgesamt aber überwiegend positiv geprägt. Insbesondere werden die leichtere Erlernbarkeit sowie die sauberen Sprachkonzepte gegenüber C++ hervorgehoben. Kritik gibt es insbesondere hinsichtlich der Stabilität der virtuellen Java-Maschine (häufige, teilweise unerklärliche Seitenfehler) und des Entwicklungskomforts.

Aus softwaretechnologischer Sicht zeigt Java alle wesentlichen objektorientierten Prinzipien auf, vermeidet typische C++-Probleme und unterstützt nebenläufige Programmierung, Garbage Collection sowie eine ausgefeilte Ausnahmebehandlung. Das Java Development Kit (JDK) eignet sich dazu, interessante softwaretechnologische Aspekte wie Klassenbibliotheken, Frameworks, Entwurfsmuster und Programmierstile beispielhaft zu erläutern und damit die Idee der Wiederverwendung von Software-Artefakten anschaulich zu demonstrieren.

Die Portabilität von Java konnte grundsätzlich nachgewiesen werden. Die Lauffähigkeit aller, auf den unterschiedlichsten Plattformen entwickelten Klassen, Packages bzw. Anwendungen wurde zu Projektabschluß im Rahmen von Präsentationen auf einem Windows/NT-Rechner nachgewiesen. Einschränkungen zur Portabilität traten hinsichtlich des Verhaltens einiger ausgewählter, insbesondere von AWT-Klassen auf. Hier gibt es im Detail Unterschiede in den Java-Versionen auf heterogenen Plattformen, wodurch Anpassungen nötig wurden.

Eine weitere Kritik an den AWT-Klassen betrifft die Tatsache, daß der AWT-Thread nicht beendet wird, wenn das letzte Fenster geschlossen wird. Da dieser kein "Dämonthread" [1] ist, kann die gesamte Anwendung nicht automatisch beendet werden, sobald das letzte Fenster geschlossen wurde. Die Anwendung muß statt dessen explizit beendet werden. Auf Anfrage teilte SUN mit, dies sei eine bewußte Entwurfsentscheidung gewesen und kein Fehler. Abgesehen von der Tatsache, daß ein solches Verhalten der AWT - Bibliothek nicht gerade intuitiv erscheint, erschwert es die Arbeit für ein Framework zusätzlich. Während ein "einfaches Programm" relativ leicht entscheiden kann, wann es endet, ist es für das Framework praktisch unmöglich, aus den ihm zugänglichen Informationen sein Ende herauszulesen. Es ist daher bei Verwendung der AWT-Oberfläche (*AWTDisplayManager*) notwendig, den Aufruf von *System.exit(0)* explizit anzugeben, um das Programmende zu markieren. Weiterhin erschienen manche Teile von AWT inkonsistent (z.B. unterschiedliche Bedeutungen der *setBounds()*-Methode in *java.awt.ScrollPane*).

Der Entwicklungsaufwand wird u.a. durch die Existenz des Garbage Collectors in Java gegenüber C++ wesentlich geringer eingeschätzt. Dieser Effekt verstärkt sich bei der Anwendung von Frameworks.

Insgesamt blieb der Eindruck von Java als einer weitgehend ausgereiften Sprache, vor allem im Vergleich zu C++ zurück. Einen weiteren wesentlichen Fortschritt brachte mit der Einführung von inneren Klassen der Übergang von Java 1.0 auf 1.1. Innere Klassen erlauben eine wesentliche Verknappung von Programmcode und in ausgewählten Fällen eine bessere Lesbarkeit des Programms. Ein Beispiel für die sinnvolle Anwendung von inneren Klassen ist eine Implementierung des *ActionListener*-Interfaces zur Behandlung von Oberflächen-Ereignissen, vor allem dann, wenn die Implementierung des Ereignisses aus einer oder wenigen Quellzeilen (z.B. *System.exit(0*)) besteht. Der Haupteffekt besteht darin, daß der Code an der Stelle im Programm steht, wo er wirklich wichtig ist. Die Gefahr dabei ist, daß möglicherweise ähnliche Dinge doppelt codiert und nicht ausreichend abstrahiert werden.

1.3.2 Was haben wir gelernt? (Lessons learned)

Insgesamt kann das durchgeführte Softwarepraktikum als erfolgreich eingeschätzt werden. Im Gegensatz zu früheren Praktika, in denen C++ als Implementierungssprache eingesetzt wurde, war die Erfolgsrate der Projektgruppen wesentlich höher. Die Studenten haben dabei die Anforderungen an das Praktikum ziemlich eindeutig als „auf keinen Fall zu niedrig“ bewertet. Auch wenn die Java-Verkaufsanwendungen nicht unmittelbar zum Einsatz kommen, war das Interesse an der Aufgabenstellung eher positiv.

Für das Projektmanagement

Die Auswertung des Projektverlaufs ergab, daß der Einarbeitungsaufwand für das Framework relativ hoch gemessen am Gesamtaufwand für das Projekt war (durchschnittlich 27 %). Als Konsequenz daraus wird den Studenten im kommenden Praktikum (Winter 1998/99) zusätzlich ein Tutorial zum Framework angeboten, um den Einarbeitungsaufwand in *SalesPoint* zu senken.

Der individuelle Entwicklungsaufwand hing vom Engagement bei der Umsetzung der Aufgabenstellung, von den Vorkenntnissen in Java und objektorientierter Technologie sowie nicht zuletzt von der Teamfähigkeit der Studenten und ihrer Projektorganisation ab. In diesem Zusammen-

hang haben wir die Beobachtung gemacht, daß einige Studierende mit vermeintlichem Verständnis objektorientierter Konzepte basierend auf C++- und/oder Delphi/Pascal-Erfahrungen glaubten, das geforderte Java-Projekt *on the fly* zu absolvieren. In den meisten Fällen ging dies schief. Ein hoher Entwicklungsaufwand und ein qualitätsmindernder Softwareentwicklungsprozeß waren die Folge.

Eine bereits in früheren Praktika bestätigte Beobachtung betrifft die Verteilung des Entwicklungsaufwandes über die Gesamtzeit von 14 Wochen. Wie erwartet, waren die Endphasen (Implementierung und Wartung) die arbeitsintensivsten. Einige Gründe dafür wurden bereits genannt (Selbstüberschätzung, Mentalität des stoßweisen Arbeitens, Mängel in der Analyse).

Ein weiteres Problem war die zum Teil unterschiedliche Motivation von Studierenden innerhalb eines Projektteams. „Schwarze Schafe“ wurden erst in der Implementierungsphase erkannt und bedeuteten für den Rest der motivierten Teammitglieder in den späten Softwareentwicklungsphasen einen erhöhten Aufwand, um das Praktikum erfolgreich zu absolvieren. Insgesamt kann die Motivation der Teams jedoch als gut eingeschätzt werden. Die statistische Auswertung ergab, daß im Nachhinein 13 der 21 erfolgreichen Projektteams ihre kooperative Arbeit als gut bis sehr gut bewerteten. Dabei ist zu bedenken, daß viele Studenten als „Hacker-Naturen“ geprägt sind, die schnell ein eigenständiges Programm schreiben können, es aber erst noch lernen müssen, ein komplexes Problem in arbeitsteiliger Softwareentwicklung zu bewältigen. Die Absprache über Schnittstellen, die Kommunikation unter den Teammitgliedern, die Erziehung zur Arbeitsdisziplin und das Finden von Kompromissen war ein wertvoller Lernprozeß und manchmal schmerzhafte Erfahrung für die Studierenden.

Damit wurde wie schon in früheren Softwarepraktika bestätigt, daß die real praktizierte Teamarbeit die größten Probleme für die Studenten aufwirft, von deren Bewältigung der Erfolg des Softwareprojektes abhängt. Gerade weil die Teamarbeit so wichtig ist, werden wir versuchen, in Zukunft die bei der ersten Durchführung des Praktikums vorsichtig erprobte interdisziplinäre Zusammenarbeit mit den Arbeitspsychologen der Fakultät Maschinenbau stärker auszubauen. Ziel ist, den Studenten eine effektive Hilfestellung bei der Bewältigung von Teamproblemen zu geben.

Einen weiteren Grund für die hohe Praktikumserfolgsrate (neben dem Einsatz von Java) sehen wir in einer straffen und zeitlich strengen Führung des Praktikums. Erfahrungsgemäß brauchen die Studenten „Druck“, um anspruchsvolle Aufgaben termin- und qualitätsgerecht zu erfüllen. Der Druck war in unserem Fall der gesetzte und nicht verlängerbare Endter-

min der Projektbearbeitung sowie die durchgeführten Pflichtkonsultationen nach jeder Projektphase.

Für den Entwicklungsprozeß

Ein Vergleich der Metriken zwischen dem Framework *SalesPoint* und einer „durchschnittlichen“ Anwendung (siehe Tabelle 1) zeigt, daß der Programmieraufwand für die Verkaufsanwendungen durch den Einsatz eines „Halbfabrikats“ erheblich reduziert werden konnte.

Die Anwendungsentwicklung auf Basis eines Frameworks und einer mächtigen Standard-Klassenbibliothek wie JDK erlaubt zugleich eine kurzfristige Realisierung von Prototypen und damit ein evolutionäres Vorgehen in der Softwareentwicklung .

Die frameworkbezogene Analyse hat sich für die Rahmenbedingungen des Praktikums bewährt. D.h., im Fall eines fest vorgegebenen Frameworks, unter strengen zeitlichen Restriktionen und bei Entwicklern, die wenig Erfahrung im objektorientierten Entwurf haben, ist die Kenntnis und das Verständnis wesentlicher Framework-Klassen (siehe Abbildung 2) im Hinblick auf die termin- und qualitätsgerechte Erstellung des Softwareprodukts sehr nützlich. Eine vom Einfluß eines konkreten Frameworks unabhängige Analyse stellt höhere Anforderungen an das Abstraktionsvermögen der Entwickler und ist mit dem Ziel des Findens einer besonders kreativen Lösung der bessere Weg. Aus diesem Grund eignet sich die unabhängige Analyse für ein Praktikum in der vertiefenden Ausbildung.

Der Einsatz von Symantec Café als Entwicklungsumgebung für die Praktikumsprojekte war ausreichend, wenn auch nicht immer befriedigend. Im Vergleich zu anderen Entwicklungsumgebungen, die wir später getestet haben (Visual Age for Java, PowerJ) erscheint Symantec Café nicht so ausgereift. Neben einigen Compiler-Problemen ist bei Symantec Café z.B. der Fakt ärgerlich, daß das Konsolenfenster nach Programmende nicht bestehen bleibt, sondern einfach verschwindet.

Das Zeichnen von UML-Modellen war unbefriedigend, geschweige das Generieren von Java-Code aus UML-Spezifikationen. Aufgrund der Unreife des zum Zeitpunkt des beschriebenen Praktikums verfügbaren CASE-Tools Rational Rose (Version 4.0), wurde Rational Rose nur sehr eingeschränkt verwendet. Die Alternative waren Grafik-Programme oder im Einzelfall die Eigenentwicklung eines Tools, welches UML-Notationen aus Java-Code erzeugt (was nicht im Sinne des Praktikumsziels lag).

1.4 Zusammenfassung, Ausblick in die Zukunft

Wir haben ein Praktikumskonzept für die universitäre Softwaretechnologie-Ausbildung entwickelt, welches für die arbeitsteilige Softwareentwicklung neben modernen objektorientierten Techniken (UML, CRC-Karten, Entwurfsmuster) Java als Implementierungssprache sowie die Wiederverwendung von Java-Code in den Mittelpunkt stellt. Im Vergleich zu früheren C++-basierten Praktika ohne Anwendungs-Framework können wir auf eine hohe Erfolgsrate der studentischen Projektteams verweisen.

Aufgrund der damit verbundenen überwiegend positiven Erfahrungen mit dem Java-basierten Softwarepraktikum ist geplant, das Praktikum im Wintersemester 1998/99 in ähnlicher Form zu wiederholen, und es ebenso im Herbst 1998 an der Universität der Bundeswehr München erstmals durchzuführen. Das verwendete Framework wird zuvor aufgrund der gewonnenen Erfahrungen überarbeitet und erweitert. Mit dem erprobten und verbessertem Framework sowie unseren Projekterfahrungen hoffen wir, daß die Studenten im Wintersemester 1998/99 ihre Projektaufgaben wiederum termingerecht sowie in gleicher Qualität und besser erfüllen können.

In vertiefenden Lehrveranstaltungen zur Softwaretechnologie sowie im Verlauf des Hauptstudiums (z.B. Großer Beleg, Diplomarbeit) erhalten die Studenten später Gelegenheit, ihre entwerferischen Tätigkeiten gegenüber dem Framework-basiertem Ansatz auszubauen und zu vervollkommnen.

1.5 Kurzvorstellung der Autoren

Birgit Demuth ist wissenschaftliche Mitarbeiterin an der Fakultät Informatik der TU Dresden und beschäftigt sich mit Fragen der Software- und Datenbanktechnologie (demuth@inf.tu-dresden.de). Im Rahmen des beschriebenen Projektes war sie Lehrverantwortliche für das Softwarepraktikum.

Lothar Schmitz ist wissenschaftlicher Mitarbeiter an der Universität der Bundeswehr München (lothar@informatik.unibw-muenchen.de). Im Studienjahr 1996/97 vertrat er die Professur Softwaretechnologie an der Fakultät Informatik der TU Dresden. Im Zusammenhang mit dem darauffolgenden Softwarepraktikum entwickelte er die Idee der Einbeziehung von Anwendungs-Frameworks in studentische Software-Projekte.

Steffen Zschaler ist Informatik-Student an der Fakultät Informatik der TU Dresden und Entwickler des Java-Frameworks *SalesPoint* (zschaler@inf.tu-dresden.de). Während des Praktikums hat er die Studenten in technischen Fragen zum Framework betreut und die Wartung des Frameworks übernommen.

Literatur

[1] Arnold, Kent; Gosling, James: The Java Programming Language. Second Edition. Addison-Wesley, 1997

[2] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995

[3] TU Dresden, Fakultät Informatik: Softwaretechnologie-Praktikum. http://www.inf.tu-dresden.de/TU/Informatik/ST2/ST/praktik97/prakti97.htm, Wintersemester 1997/98

[4] Draft Java Coding Standard: http://gee.cs.oswego.edu/dl/html/javaCodingStd.html

[5] Pree, Wolfgang: Komponentenbasierte Softwareentwicklung mit Fraumeworks. Dpunkt Verlag, 1997