Erfahrungen mit einem frameworkbasierten Softwarepraktikum[1]

Birgit Demuth*, Heinrich Hußmann*, Steffen Zschaler* und Lothar Schmitz**
Technische Universität Dresden (*), Universität der Bundeswehr München (**)

Wir berichten über ein Praktikumskonzept, welches als Ergebnis mehrerer Jahre der Softwaretechnologieausbildung im universitären Grundstudium entstanden ist. Gegenstand des Praktikums ist die Festigung und Anwendung objektorientierter Techniken, die in der Grundausbildung Softwaretechnologie vermittelt werden. Neben der Aneignung der softwaretechnologischen Grundfertigkeiten steht die praktizierte Teamarbeit im Mittelpunkt. Seit 1997 haben wir unser Konzept auf Java, UML und die CRC-Karten-Methode ausgerichtet. Eine weitere Besonderheit ist die praktische Übung der Wiederverwendung von Software-Artifakten, um zum einen praxisnahe Projektsituationen zu simulieren und zum anderen Anfängern im objektorientierten Design Hilfestellung auf dem Weg zum OO-Experten zu geben. Dazu wurde an der TU Dresden das Anwendungsframework *SalesPoint* für Verkaufsanwendungen entwickelt, das die gemeinsame Basis für eine Vielzahl von Teamaufgaben bildet. Aufgrund der Erfolgsrate im Wintersemester 1997/98 wird dieses Framework gegenwärtig in überarbeiteter Form sowohl an der TU Dresden als auch an der Universität der Bundeswehr München eingesetzt. Wir stellen das didaktische Konzept unserer Softwaretechnologie-Ausbildung, die Struktur und Anpassungsschnittstelle des Frameworks sowie Erfahrungen, Aufwände und Ergebnisse bei der Organisation und Durchführung des Softwarepraktikums dar.

1 Einleitung

Moderne Softwaretechnologie-Ausbildung ist im wesentlichen durch die Vermittlung objektorientierter (OO) Techniken geprägt. Ein Problem des objektorientierten Ansatzes ist dabei die bekannt lange Einarbeitungszeit. Um Studierende möglichst frühzeitig mit moderner, objektorientierter Programmiertechnik zu konfrontieren, werden gegenwärtig an der TU Dresden 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 werden Muster und Rahmenwerke (Frameworks) dargestellt; in den Übungen sammeln die Studierenden erste Erfahrungen mit diesen Techniken.

Im anschließenden Semester nehmen die Studierenden an einem Softwarepraktikum teil, in dem die Anfänger das erworbene Wissen in einem Softwareprojekt anwenden und gleichzeitig den Schritt vom „Programmieren im Kleinen“ zum „Programmieren im Großen“ vollziehen. 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

[1] Erscheint in: *Tagungsband des 6. Workshops „Software-Engineering im Unterricht der Hochschulen“*, Teubner-Verlag, 1999

und gleichzeitig ein innovatives Vorgehen zu vermitteln, haben wir uns entschlossen, die Praktikumsaufgaben auf einem Anwendungsframework aufzusetzen. Da gängige Frameworks meist zu komplex in der Einarbeitungsphase sind, wurde ein spezielles Framework eigens für dieses Praktikum ausgearbeitet. Die Studierenden werden aufgefordert, die Idee der Wiederverwendung objektorientierter Software in sinnvoller Weise zu praktizieren. Studierende der ersten Jahre neigen erfahrungsgemäß oft dazu, beim Programmieren "das Rad immer wieder neu zu erfinden". Zu dieser Haltung soll bewußt eine Gegenhaltung eingeübt werden. Außerdem wird durch das Framework eine Grundarchitektur vorgegeben, was den für Unerfahrene schwierigen OO Entwurf deutlich erleichtert. Das Framework ist durch Beschreibung und Anpassungsschnittstelle vergleichsweise gut zugänglich, so daß insgesamt eine relativ hohe Akzeptanz des Frameworks als sinnvolles Hilfsmittel erreicht werden kann.

Ein weiteres wichtiges didaktisches Ziel unseres Praktikumskonzeptes ist das Erlernen der Teamarbeit und das Erleben erster Projekterfahrungen. Die Studierenden bilden Projektgruppen von ca. 5 Teammitgliedern und arbeiten unter Anleitung je eines Betreuers pro Gruppe.

Wir beschreiben zunächst Inhalte und Zielsetzung des Lehrkonzeptes für das Fach Softwaretchnologie (Abschnitt 2) sowie Anwendungsbereich, Architektur und Anpassungsschnittstelle des verwendeten Frameworks (Abschnitt 3). Das frameworkbasierte Praktikum wurde erstmals im Wintersemester 1997/98 durchgeführt. Basierend auf den dabei gewonnenen Erfahrungen überarbeiteten wir das Framework und setzen es im Wintersemester 1998/99 im Softwarepraktikum gleichzeitig an der TU Dresden und an der Universität der Bundeswehr München ein. In Abschnitt 4 berichten wir über die Praktikumsorganisation, Aufwände, Ergebnisse und Erfahrungen. Eine Herausforderung für die Lehrenden ist die ständige Anpassung der Lehrinhalte an die rasche technische Entwicklung, speziell der von Java im Umfeld von modernen Softwaretechnologien. Deshalb geben wir zum Schluß einen Ausblick auf die weitere Entwicklung unseres Ausbildungskonzeptes.

2 Softwaretechnologie im Grundstudium

Die Pflichtlehrveranstaltung Softwaretechnologie (2-stündige Vorlesung, 2-stündige Übung) ist im zweiten Studienjahr angesetzt. Die Studierenden haben deshalb zu Beginn der Lehrveranstaltung bereits grundlegende Kenntnisse der Programmierung sowie von Datenstrukturen und Algorithmen. Ziel der Lehrveranstaltung Softwaretechnologie ist es, die besondere Problematik der Entwicklung großer und komplexer Softwaresysteme begreifbar zu machen, in den heutigen Stand der (objektorientierten) Softwaretechnologie einzuführen und verständlich zu machen, in welcher Weise diese Technologie zur systematischen und wirtschaftlichen Konstruktion hochqualitativer Softwaresysteme beitragen kann.

Es ist eine sinnvolle Ergänzung für diese Lehrziele, daß eine objektorientierte Programmiersprache (hier Java) bis zum flüssigen praktischen Umgang mit der Sprache gelehrt wird. Der Eindruck eines "Programmierkurses" wird allerdings bewußt vermieden: Die objektorientierte Programmiersprache dient als Hilfsmittel zur Veranschaulichung der Konzepte und zum praktischen Experimentieren, der

Großteil des erlernten Wissens soll jedoch übertragbar auf andere Programmiersprachen bleiben. Aus diesem Grund nimmt neben der objektorientierten Programmiersprache eine objektorientierte Modellierungsnotation (Unified Modeling Language UML) einen gleichwertigen Rang ein. Außerdem wird die objektorientierte Programmiersprache verwendet, um die Konzepte von Klassenbibliotheken, Application Frameworks und Entwurfsmustern praktisch erfahrbar zu machen.

Die Programmiersprache Java wurde nicht wegen ihres hohen Aktualitätsgrades und auch nicht im Glauben an eine baldige, massenhafte Verbreitung in der Softwareindustrie eingesetzt: Nach einigen Jahren der Ausbildung anhand von C++ wurde auf das konzeptionell sauberere Java umgestellt, weil diese Sprache einfach erheblich leichter zu unterrichten ist. Positiv wirkt sich dabei aus, daß auf allen für die Studenten relevanten Plattformen kostenfreie bzw. kostengünstige Java-Entwicklungsumgebungen verfügbar sind.

Die Lehrveranstaltung umfaßt 9 wesentliche Lehreinheiten von unterschiedlichem Umfang. Der ungefähre Umfang (in Unterrichtsstunden) ist in folgender Übersicht jeweils nach dem Titel der Einheit angegeben.

1. *Herstellung großer Softwaresysteme (2):* Typische Problematik großer Softwareprojekte, Themenübersicht zum Software-Engineering, Phasenmodelle

2. *Objektorientierung (2):* Grundkonzepte der Objektorientierung, Einordnung gegenüber anderen Programmier- und Modellierungsparadigmen, Historischer Überblick

3. *Objektorientierte Analyse (6):* CRC-Karten, Statische Modellierung mit UML, Dynamische Modellierung mit UML

4. *Objektorientierter Entwurf (8):* Softwarearchitekturen, Architekturbeschreibung mit UML, Realisierung von UML-Entwürfen in Java und anderen Sprachen, Datenstrukturentwurf

5. *Architektur interaktiver Systeme (4):* Ereignisgesteuerter Programmablauf, Model-View-Controller-Architektur, Benutzungsoberflächen am Beispiel Java AWT

6. *Wiederverwendung (4):* Klassenbibliotheken, Application Frameworks am Beispiel Java AWT, Entwurfsmuster

7. *Parallele Kontrollflüsse (2):* Nebenläufigkeit, Prozesse, Threads, Synchronisation.

8. *Software-Qualitätssicherung (1):* Prozeßintegrierte Software-Qualität, Test und Integration

9. *Projektmanagement (1):* Grundzüge der Projektplanung, Versionskontrolle, Dokumentation

An der Universität der Bundeswehr sind in Vorbereitung des Praktikums die Teile 1-4 und 6 (in vergleichbarem Stundenumfang) in die Vorlesung „Einführung in die Informatik III“ integriert.

3 Framework

Das im Softwarepraktikum eingesetzte Anwendungsframework *SalesPoint* unterstützt die Entwicklung von Verkaufssimulationen sowohl von einfachen Automaten als auch von großen Kaufhäusern. Typische Anwendungen sind z.B. eine Wechselstube, in der Geld in andere Währungen getauscht werden kann, ein Postschalter, der Briefmarken und ein wohldefiniertes Angebot an Dienstleistungen bietet, eine Versandhausagentur oder ein Restaurant. Alle Anwendungen aus diesen Bereichen haben die folgenden gemeinsamen Eigenschaften:

- Ein Verkaufsstand (*SalesPoint*) bietet Artikel bzw. Leistungen aus einem festgelegten Katalog an. Für jeden Artikel gibt es im Katalog einen Eintrag, welcher den Namen, den Preis und ggf. andere relevante Eigenschaften speichert. Ein Bestand ist eine Multimenge von realen Artikeln aus einem Katalog. Beispiele für Bestände sind die Posten auf einem Bestellformular, die Ware in einem Automaten, in Ladenregalen oder im Warenkorb eines Kunden.
- Geld wird als Spezialfall dieser Terminologie aufgefaßt. Der Katalog ist in diesem Fall die Währung. Die Währung beschreibt die Menge gültiger Banknoten und Münzen sowie deren Werte. Der Inhalt einer Geldbörse oder einer Kasse ist ein Geldbestand.
- Der wichtigste Anwendungsfall eines Verkaufsstandes ist es zu handeln (verkaufen oder kaufen). Solche Transaktionen werden als atomar angesehen, d.h. sie werden entweder vollständig oder gar nicht ausgeführt. In einer Verkaufsstelle gibt es weitere Transaktionen bzw. Hintergrundvorgänge, wie z.B. Warenbestellung, Inventur und die Aktualisierung von Katalogen.

Entsprechend diesem Anwendungsverständnis unterstützt *SalesPoint* die Entwicklung von Verkaufssimulationen durch die Bereitstellung von

- Datenverwaltungsklassen für Kataloge (*Catalog*), Bestände (*Stock*), Währungen (*Currency*) und Geldbestände (*MoneyBag*) auf Basis der Objektserialisierung;
- generischen Formularen (*FormSheet*) und Menüs (*MenuSheet*) zur Interaktion mit dem Benutzer sowie grundlegende Standardformulare;
- Transaktionsunterstützung (*Transaction*) einschließlich Protokollierungsfunktion (*Log*);
- vorgefertigten Algorithmen für die Stückelung eines Wertes in einen Bestand (z.B. für Wechselgeld oder Briefmarken eines bestimmten Gesamtwertes).

In Auswertung des Einsatzes von *SalesPoint* im Wintersemester 1997/98 wurde das Framework überarbeitet. Die Wartung beschränkte sich dabei im wesentlichen auf eine Vereinfachung der Klassenstruktur. Abbildung 1 beschreibt die gegenwärtige Architektur des Frameworks aus logischer Sicht, die zentralen Klassen und deren Beziehungen. Das Framework wurde in Java (Sprache und JDK Version 1.1) implementiert.

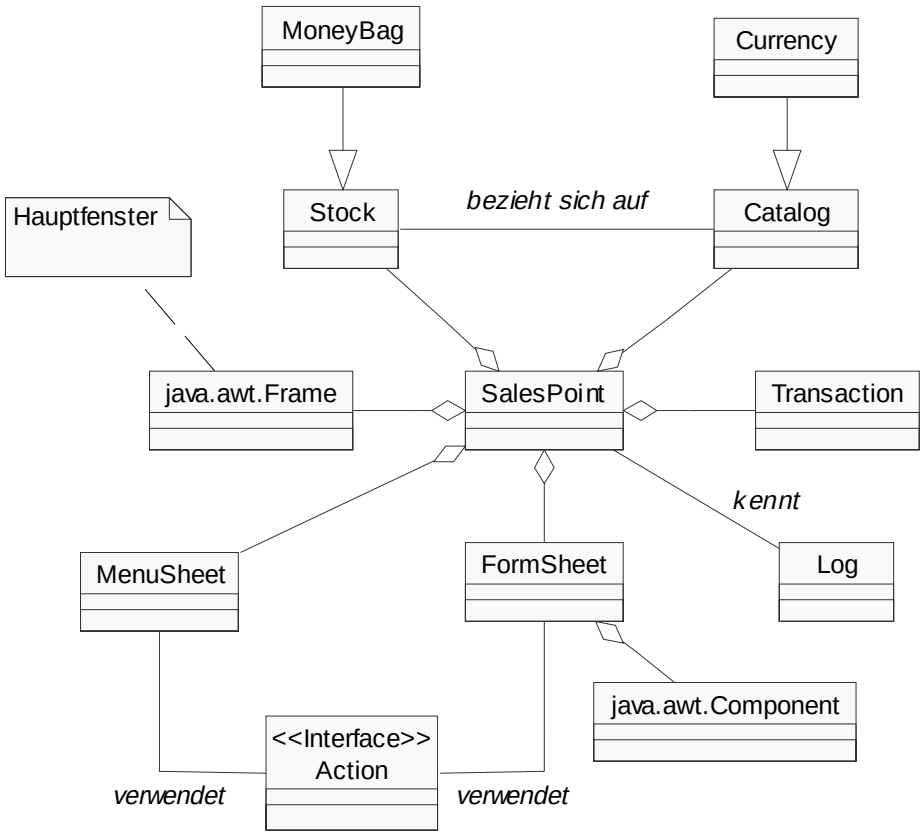

Abbildung 1: Architektur von *SalesPoint*

Eine wichtige Frage beim Entwurf, der Verwendung und des Verständnisses eines Frameworks ist dessen Anpassungsschnittstelle an konkrete Anwendungen. Das Framework *SalesPoint* wurde unter Berücksichtigung bewährter Muster [1] entworfen und kann durch folgende Techniken flexibel an die Bedürfnisse des Benutzers angepaßt werden:

- spezialisierte Unterklassen (z.B. werden Menüaktionen als Implementation des Interfaces *Action* realisiert);
- Überschreiben von Hook-Methoden (z.B. wird der Rollback-Mechanismus einer Transaktion durch drei Hook-Methoden implementiert: *saveState()* zur Sicherung des Zustandes vor einer Transaktion, *restoreState()* zum Rücksetzen aller durch die Transaktion durchgeführten Änderungen im Fehlerfall und *clearStateInfo()* zur Freigabe eventl. belegter Ressourcen);
- Parametrisierung von Methoden (z.B. werden bei der Erzeugung eines neuen Bestandes Parameter zur Angabe des zugehörigen Kataloges, der konkreten Implementation und ggf. des Stückelungsalgorithmus übergeben).

Das überarbeitete Framework ist frei verfügbar [2]. Die Dokumentation besteht im wesentlichen aus mit *javadoc* generierten Klassenbeschreibungen als Referenz

sowie einem Tutorial zur effektiven Einarbeitung in das Framework. Das Tutorial behandelt am Beispiel eines Fahrkartenautomaten die wesentlichen Elemente der Anpassungsschnittstelle des Frameworks (Black-Box-Gebrauch) sowie einige interne Strukturen und Prinzipien (White-Box-Gebrauch). Neben dem Quellkode der Fahrkartenautomatenanwendung steht ein Applet zu Demonstrationszwecken zur Verfügung.

4 Praktikumsorganisation und Erfahrungen

Die im folgendem beschriebene Praktikumsorganisation ist im Ergebnis mehrerer Jahre entstanden. In den Studienjahren vor unserem Java- und frameworkbasiertem Konzept experimentierten wir mit verschiedenen C++-Praktikumsprojekten. Das Ergebnis dieser Projekte war oft nicht befriedigend. Zum einen brachte das Erlernen von C++ unter engen zeitlichen Restriktionen erhebliche Probleme mit sich, zum anderen bestätigte sich die Erfahrung, daß für Anfänger der objektorientierte Entwurf *from scratch* ein schwieriger Schritt ist.

4.1 Praktikumsdurchführung

Das Java-Softwarepraktikum wurde erstmalig an der TU Dresden im Wintersemester 1997/98 unter strengen zeitlichen Restriktionen durchgeführt. Zunächst wurden 22 studentische Projektteams mit ca. 5 Teammitgliedern gebildet. Jedes Team erhielt die Aufgabe, eine Verkaufsanwendung unter Verwendung des Frameworks *SalesPoint* objektorientiert zu entwickeln. Jede einzelne Aufgabenstellung wurde maximal dreimal vergeben. Die konkreten Anforderungen an die Verkaufsanwendung waren verbal spezifiziert. Das Softwareprodukt sollte eine Java(1.1)-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 Bearbeitung und gemeinsame Präsentation gleicher Aufgabenstellungen wirkte als Konkurrenzsituation und belebte die Diskussionen.

Jedem Team war ein Betreuer zugeteilt, der neben der softwaretechnologischen Betreuung 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 [2] unterstützt.

Die Bildung der Projektteams erfolgte nach dem Prinzip der Chefprogrammiererorganisation. Einem Studenten wurde die Funktion des Chefprogrammierers übertragen, die anderen Gruppenmitglieder fungierten als Mitarbeiter. Von jedem Student wurde 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 Festle-

gung, 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.

Im Wintersemester 1998/99 wird das Softwarepraktikum an der TU Dresden in fast unveränderter Organisation wiederholt [2] und an der Universität der Bundeswehr München in an deren Studienbedingungen (Trimester, geringere Teilnehmerzahl) angepaßter Form erstmalig durchgeführt.

4.2 Softwareentwicklungsprozeß

Die Vorgabe der zeitlichen Verteilung der Projektphasen Praktikumseinweisung/Projektplanung (2 Wochen), Analyse (2 Wochen), Entwurf (2 Wochen), Implementation/Test (4 Wochen) sowie Abnahme/Wartung (4 Wochen) hatte empfehlenden Charakter. Dabei wurde bewußt eine angemessen 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. In der Analysephase hatten die Studierenden zunächst die Aufgabe, sich in das Objektmodell des Frameworks einzuarbeiten, um anschließend die Projektspezifikation frameworkbezogen zu analysieren. Dazu mußten die Studenten zunächst das Framework verstehen. Die Projektdurchführung nach dem Wasserfallmodell ist jedoch oft unrealistisch: nach den Erfahrungen aus dem Praktikum waren im Durchschnitt zwei bis drei Iterationszyklen erforderlich. Durch den fest vorgegebenen Endtermin des Praktikums blieb für eine qualifizierte Abnahme/Wartung der entstandenen Anwendung wenig Zeit (durchschnittlich zweieinhalb Wochen). Diese Phase 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.

Randbedingungen des Entwicklungsprozesses 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 CRC-Karten sowie UML-Anwendungsfall-, Sequenz- und Klassendiagramme. Die Projektteams hatten jedoch insbesondere hinsichtlich eingesetzter Entwicklungsplattformen, Entwicklungsumgebungen, Tools sowie ihrer Projektorganisation Freiräume bei der Ausgestaltung ihres Softwareentwicklungsprozesses.

4.3 Aufwände, Ergebnisse und Erfahrungen

Zu Abschluß der Projekte im Januar 1998 beantworteten alle Teams einen Fragebogen zur quantitativen und qualitativen Bewertung des Praktikums. Die ermittelten Metriken geben ein Bild über die Komplexität der Anwendungen und die Produktivität von Anfängern im objektorientierten Umfeld:

- 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;
  - dem Grad der Wiederverwendung von Framework-Klassen;
  - der Anwendung bzw. Nichtbeachtung von Entwurfsmustern, um die Anwendungsklassen 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 gegenüber der Aufgabenstellung zusätzlich projektierte Funktionalität gekennzeichnet.
- Durchschnittlich implementierte jeder Studierenden 677 LOC. Dabei wurde der durchschnittliche wöchentliche Arbeitsaufwand pro Teammitglied von den Studenten auf 8 bis 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.
- Zur Abschätzung der studentischen Projektproduktivität 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. Es ist jedoch zu beachten, daß die Projektproduktivität 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.
- 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*, 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. Unter Berücksichtigung des Umfangs des ver-

wendeten Frameworks (3741 LOC) verdoppelt sich in etwa die Produktivität der Studierenden.

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 Team bewertete ihr Praktikum als übererfüllt.

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 wurde den Studenten im Praktikum 1998/99 zusätzlich eine Einführungsvorlesung zum Framework angeboten. Neben diesem Angebot ist es aber wichtig, die Studierenden zur aktiven Auseinandersetzung mit dem Framework zu bewegen. An der Uni der Bundeswehr mußten deshalb die Studenten als ersten Meilenstein zum Praktikumsschein den Beispiel-Fahrkartenautomaten zu einem einfachen (Zigaretten-) Automaten modifizieren. Eine andere Variante der Einarbeitung in das Framework war die Erstellung von CRC-Karten für das Framework (TU Dresden).

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 Zusammenhang 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.

Ein weiteres Problem war die zum Teil unterschiedliche Motivation von Studierenden innerhalb eines Projektteams. „Schwarze Schafe“ wurden häufig 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 waren 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.

Neben dem Einsatz von Java und Framework sehen wir in einer straffen und zeitlich strengen Führung des Praktikums eine wichtige Voraussetzung für den Erfolg

der Projekte. Erfahrungsgemäß brauchen die Studenten „Druck“, um anspruchsvolle Aufgaben termin- und qualitätsgerecht zu erfüllen. Der Druck war in unserem Fall der gesetzte und nicht verlängerbare Endtermin der Projektbearbeitung sowie die durchgeführten Pflichtkonsultationen nach jeder Projektphase.

Die Organisation der Praktikumsbetreuung ist eine wichtige Frage im Gesamtkonzept. Den Großteil der unmittelbaren Betreuung der Projektgruppen (im Wintersemester 1997/98 ca. 320 Stunden) übernehmen Studierende höherer Semester, die selbst bereits erfolgreich das Softwarepraktikum absolviert haben. Erfahrungsgemäß steigt die Betreuungsqualität, wenn die Tutoren über mehrere Jahre in die Arbeit eingebunden werden. Neben dem technischen Knowhow benötigen sie vor allem soziale Kompetenzen, um bei Teamproblemen rechtzeitig helfend in die Projektorganisation eingreifen zu können. Das Angebot an die Studierenden, bei Bedarf Arbeitspsychologen an der Universität zu konsultieren, fand keine Resonanz. Neben dem Betreuer jeder Projektgruppe steht der Framework-Entwickler bei (programm)technischen Problemen als Ansprechpartner zur Verfügung.

5 Ausblick

Das vorgestellte Praktikumskonzept für die Softwaretechnologie-Ausbildung im universitären Grundstudium hat sich bewährt. Das läßt sich nicht zuletzt an der hohen Erfolgsrate und Produktivität der Studierenden ablesen. Um die Effizienz der studentischen Softwareentwicklungsprozesse weiter zu verbessern, denken wir über die Weiterentwicklung des Frameworks einschließlich des Findens zusätzlicher Dokumentationshilfen nach. Gegenwärtig werden beispielsweise die Methoden der Framework-Klassen unter dem Aspekt ihrer empfohlenen Redefinition klassifiziert (*never*, *sometimes*, *always*). Sinnvolle Erweiterungen des Frameworks könnten sein:

- datenbankbasierte Verwaltung von Katalogen und Beständen;
- Bereitstellung von geschachtelten Transaktionen;
- Unterstützung von verteilten Verkaufsanwendungen durch Nutzung von RMI oder Corba.
- Realisierung von Verkaufskomponenten (Java Beans).

Die gegenwärtigen und in Zukunft denkbaren Aufgabenstellungen gestatten eine fast beliebige Komplexität, die wiederum Einfluß auf die Anforderungen an das Framework hat. Das ist ein interessantes Experimentierfeld für neue Software- und Java-Technologien, die insbesondere für die vertiefende Softwaretechnologie-Ausbildung im Hauptstudium gefragt sind.

Literatur

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

[2]TU Dresden, Fakultät Informatik: Softwaretechnologie-Praktikum. http://www.inf.tu-dresden.de/TU/Informatik/ST2/ST/praktikum.htm