Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

Vom Code zu den Anforderungen und wieder zurück: Software Engineering in sechs Semesterwochenstunden

Barbara Paech(1), Lars Borner(1), Jürgen Rückert(1), Allen H. Dutoit(2), Timo Wolf(2)
(1) Universität Heidelberg, Fakultät für Mathematik und Informatik, AG Software Engineering
Im Neuenheimer Feld 326, 69120 Heidelberg
{paech, lars.borner, juergen.rueckert}@informatik.uni-heidelberg.de
(2) Technische Universität München, Fakultät für Informatik, AG Angewandte Softwaretechnik
Boltzmannstraße 3, 85748 Garching b. München
{dutoit, wolft}@in.tum.de

Zusammenfassung

An der Universität Heidelberg wurde die Veranstaltung „Software Engineering" im Bachelor-Studiengang „Angewandte Informatik" erstmals mit einem großen Anteil praktischer Übungen durchgeführt. Die Herausforderung war, in Vorlesung und Übung eines Semesters mit 6 Semesterwochenstunden die Studierenden zu befähigen, an einem Softwaresystem realistischer Größe systematisch Änderungen durchzuführen. Die innovative Kernidee unseres Vorgehens ist, dass sich die Studierenden in der ersten Hälfte durch Reverse Engineering in ein existierendes, komplexes Softwaresystem einarbeiten und dieses dann in der zweiten Hälfte mit einem systematischen Entwicklungsvorgehen erweitern. In diesem Beitrag stellen wir unsere Vorgehensweise, das bearbeitete Softwaresystem und die verwendeten Werkzeuge sowie unsere Erfahrungen vor.

1 Einführung

Vor drei Jahren wurde an der Universität Heidelberg der Bachelor-Studiengang *Angewandte Informatik* eingeführt. Die Lehrveranstaltung „Software Engineering" (SWE) ist eine Wahlpflichtveranstaltung mit 6 Semesterwochenstunden (9 ETCS-Punkten) ab dem 4. Semester. Damit möglichst viele Studierende eine fundierte SWE- Ausbildung erhalten, sollen alle wesentlichen SWE-Inhalte in dieser Veranstaltung praxisnah vermittelt werden. Die Voraussetzung zur Teilnahme an der Veranstaltung ist eine erfolgreiche Teilnahme an der Informatik I Grundvorlesung, die die Grundlagen der Objektorientierung und der Programmiersprache C++ vermittelt. Ein Vorwissen auf dem Gebiet des SWE ist bei den TeilnehmerInnen so gut wie nicht vorhanden.

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

Die Ziele der Lehrveranstaltung stimmen mit den in [Lic 03] beschriebenen Zielen überein, d.h. Vermittlung von gesichertem Fachwissen im Bereich SWE, Anwenden des Fachwissens an realitätsnahen Problem- und Aufgabenstellungen, Reflexion über die Anwendung des Fachwissens und praxisnahe Ausbildung der Studierenden durch Kennenlernen von industriell benutzten Methoden, Sprachen und Werkzeugen (der ebenfalls genannte Einbezug der Industrie war nicht beabsichtigt).

Aufgrund der Kürze der Zeit liegt der Schwerpunkt in den praktischen Übungen auf dem eigentlichen Entwicklungszyklus, d.h. Softwarekontextgestaltung, Requirements Engineering, Architekturdefinition, Feinentwurf und Implementierung. Dabei sollen insbesondere auch die Qualitätssicherungsmaßnahmen der einzelnen Phasen erlernt werden. Das umfasst die verschiedenen Arten sowohl des dynamischen als auch des statischen Testens. Weiterhin ist auch die Vermittlung der „Soft Skills“ ([Lic 03], [Lew 01]) ein Anliegen.

In den SEUH-Bänden finden sich viele Vorschläge und Konzepte, wie SWE-Inhalte vermittelt werden können. Die meisten dieser Vorschläge beziehen sich allerdings auf Veranstaltungen, in denen für Vermittlung des Fachwissens bzw. praktische Übungen mindestens je ein Semester zur Verfügung steht (vgl. [Ble 99], [Dem 99], [Bot 01], [Spi 01]) oder auf Lehrveranstaltungen, die innerhalb eines Semesters durchgeführt werden, aber auf Kosten von zu vermittelnden Inhalten oder des Praxisbezugs (vgl. [Kle 01], [Bec 03], [Met 03]). Das Ziel an der Universität Heidelberg war es, Vermittlung von Fachwissen und Praxisbezug trotz der Kürze der Zeit gleichrangig zu behandeln.

In diesem Beitrag stellen wir die Grundidee unseres Vorgehens, das bearbeitete Softwaresystem und die verwendeten Werkzeuge (Abschnitt 2), die konkrete Durchführung (Abschnitt 3) sowie unsere Erfahrungen bei der Durchführung (Abschnitt 4) vor.

2 Vorgehensweise

Die konkrete Herausforderung ist, den Studierenden die folgenden Inhalte in 6 SWS zu vermitteln:

- SWE-Methoden zu Entwicklung, Qualitätssicherung, Weiterentwicklung und Wiederverwendung sowie Management.
- Ein komplexes Softwaresystem, da nur ab einer gewissen Komplexität SWE-Methoden sinnvoll eingeübt werden können.
- Eine möglichst durchgängige Werkzeugunterstützung, da komplexe Systeme ohne eine solche nicht sinnvoll zu bearbeiten sind.

Im Folgenden wird zuerst das Gesamtkonzept, dann das bearbeitete Softwaresystem und danach die Entwicklungsumgebung vorgestellt.

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen,
SEUH 9, Aachen, dpunkt.verlag, Heidelberg

2.1 Gesamtkonzept

Unsere innovative Kernidee ist eine Kombination von Reverse und Forward Engineering. Nach unserer Erfahrung können Studierende den Nutzen von Modellierung, Qualitätssicherung und wartbarem Code nicht erkennen, solange sie nicht mit wirklich großem Code in Berührung gekommen sind. Daher entstand die Idee, die praktischen Übungen mit dem Kennenlernen eines großen Systems zu beginnen und darauf aufbauend nacheinander die höheren Abstraktionsebenen einzuführen. Dieses Vorgehen hat insbesondere den Vorteil, dass Testtechniken schon früh eingeübt werden können, da von Anfang Code zur Verfügung steht. Weitere Vorteile eines Reverse Engineering orientierten Vorgehens sind in [Bot 01] zusammengestellt. Genauso wesentlich ist aber, dass die Studierenden dann die erlernten Methoden selbstständig bei der Entwicklung anwenden. Deshalb erhalten die Studierenden in der zweiten Hälfte des Semesters die Aufgabe, das System um einige Funktionalitäten zu erweitern. Die Vorlesung liefert begleitend dazu die notwendigen Inhalte. Insbesondere werden in der ersten Hälfte die Modellierungstechniken (UML) und Qualitätssicherungstechniken (Review, Testen)

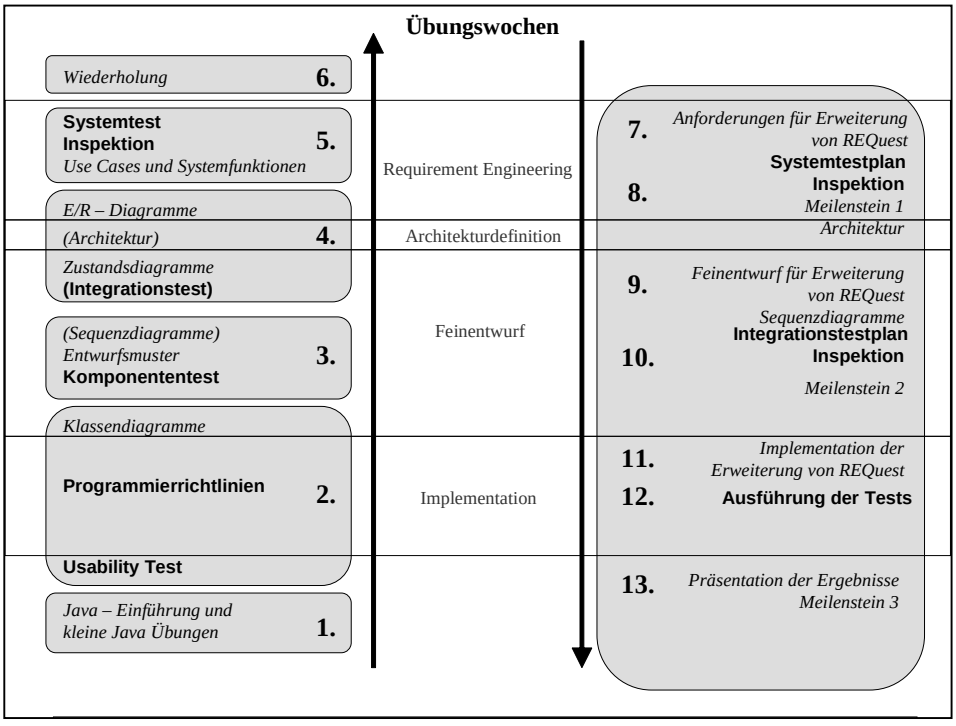

Abb. 1.: Abfolge der Themen

einzeln vorgestellt und eingeübt, während in der zweiten Hälfte diese Techniken zu einem durchgängigen Entwicklungsprozess verbunden werden.

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

Die Abfolge der Themen ist in Abbildung 1 verdeutlicht und wird im Abschnitt 3 näher erklärt. Klammern bedeuten, dass das Thema inhaltlich an diese Stelle gehört, aber aus Zeitgründen in die 2. Hälfte verschoben wurde. Qualitätssicherungsmaßnahmen sind fett gedruckt.

Die Bearbeitung der Reverse-Engineering-Aufgaben sowie der Forward-Engineering-Aufgaben erfolgt in Teams mit enger Betreuung durch je einen Mitarbeiter, um insbesondere auch die Kommunikations- und Koordinationsaspekte des SWE einzuüben.

2.2 Das zu bearbeitende System: Sysiphus

Wie oben erläutert sollen die Studierenden mit einem existierenden Softwaresystem arbeiten. Dazu muss natürlich einerseits der Code zugreifbar sein, andererseits soll aber auch eine ausreichende Dokumentation vorhanden sein, die eine Einarbeitung unterstützt. Letzteres machte insbesondere den Einsatz von Open-Source-Software nicht realistisch. Aus früheren Kooperationen hat die Universität Heidelberg Zugriff auf das System Sysiphus, das an der TU München entwickelt wird. Sysiphus unterstützt die Durchführung von SWE-Projekten. Durch die Verwendung von Sysiphus können die Studierenden die Methoden des SWE praktizieren und lernen gleichzeitig die Anwendungsdomäne des zu bearbeitenden Systems kennen. Das System ist auf Nachfrage erhältlich. Insbesondere wurde es für die Lehrveranstaltung ausgewählt, weil

- es für den Einsatz in der Lehre entwickelt wurde und die gleichzeitige Bearbeitung durch mehrere Teams unterstützt,
- die Anwendungsdomäne (nämlich das SWE) für die Studierenden wichtig ist (wenn auch noch nicht vertraut),
- es mit 100.000 LOC in über 1000 Klassen komplex ist,
- es ausreichend dokumentiert ist,
- es durch seine leicht verständliche Schichtenarchitektur gut erweiterbar ist,
- es ein Web-Interface bietet,
- es in der objektorientierten Programmiersprache Java geschrieben ist.

JAVA und Web-Interface ermöglichen den Studierenden, neuesten Technologien kennen zu lernen. Nachfolgend werden die Grundkonzepte beschrieben.

Das Werkzeug Sysiphus [Sys 04] unterstützt die Durchführung von Softwareprojekten, indem es die Erstellung von Modellen und Dokumenten mit der Kommunikation und der Erfassung von Entscheidungsbegründungen (Design Rationale) vereinigt. Abbildung 2 zeigt die Benutzungsoberfläche. Auf der linken Seite können Systemmodelle wie z.B. Anwendungsfälle erstellt und modifiziert werden. Auf der rechten Seite können Fragen, verschiedene Lösungsvorschläge, deren Evaluierungen, Argumentationen und Entscheidungen zu jedem Modellelement erstellt werden. Modellelemente und Rationalelemente sind miteinander verlinkt.

Sysiphus ist eine verteilte Client-Server-Anwendung und bietet mit REQuest [Dut 02] eine Web-basierte Benutzerschnittstellte sowie mit RAT [Wol 04] eine Java Swing-

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

basierte Benutzerschnittstelle als Client-Anwendungen an. Beide Client-Anwendungen arbeiten mit dem gleichen Server und können zeitgleich verwendet werden. Im Gegensatz zu anderen CASE-Tools unterstützt Sysiphus eine enge Integration der Kommunikation und Begründungserfassung in die Systemmodelle. Auftretende Fragen und Kommunikationsströme, wie sie aus Newsgroups bekannt sind, können direkt zu einzelnen Modellelementen wie z.B. Anwendungsfällen, nichtfunktionalen Anforderungen, Klassen oder Testfällen erstellt werden (siehe Abbildung 2). Somit werden die Modelle mit explizitem Wissen angereichert, welches bei der Verwendung von herkömmlichen Kommunikationsmedien wie E-Mail oder Telefon verloren gehen kann oder nicht für alle ProjektteilnehmerInnen erreichbar ist. Die EntwicklerInnen sind jederzeit in der Lage, von den Modellelementen zu den Kommunikationsströmen und Begründungserfassungen zurück zu navigieren. Die Nutzung der Verzahnung von Dokumentenentwicklung und Rational in der Lehre ist beschrieben in [Dut 04].

Sysiphus wurde für die Durchführung von Softwareentwicklungsprojekten (ARENA[1], Cargo Logistic[2]) sowie für die SWE-Lehre entwickelt. Des Weiteren dient Sysiphus als Grundlage von einzelnen, studentischen Praktika und Projekten. Da die

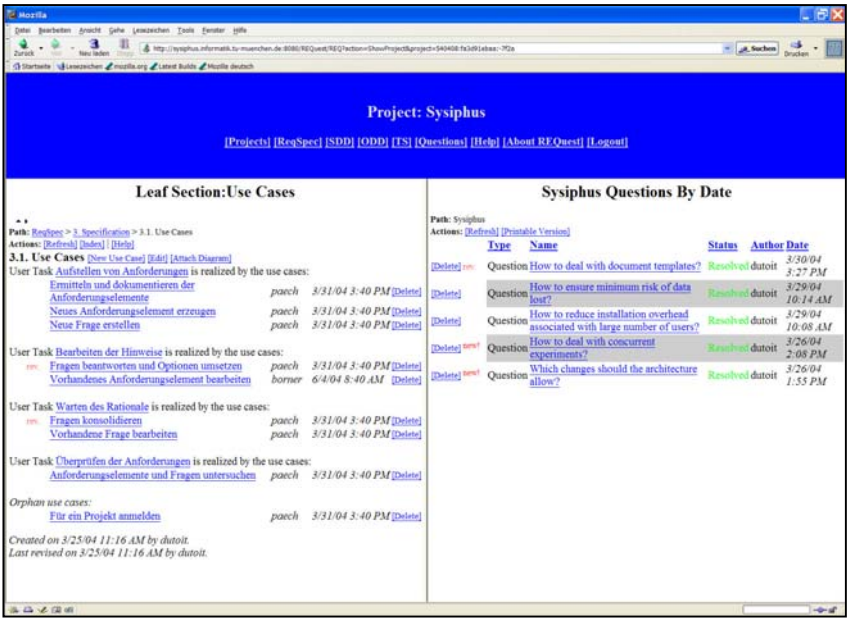

Abb. 2: Screenshot des bearbeiteten Systems

studentischen Projekte zeitlich sehr begrenzt sind, ist im Allgemeinen eine lange Einarbeitungszeit in das System nicht möglich. Deswegen sollten die Studierenden in der Lage sein, einzelne Teile des Systems verändern zu kön-

[1] *http://arena.globalse.org/*
[2] *http://wwwbruegge.in.tum.de/SePrakt03*

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

nen, ohne das gesamte System verstehen zu müssen. Dies führte zu folgender Schichtenarchitektur (siehe Abbildung 3):

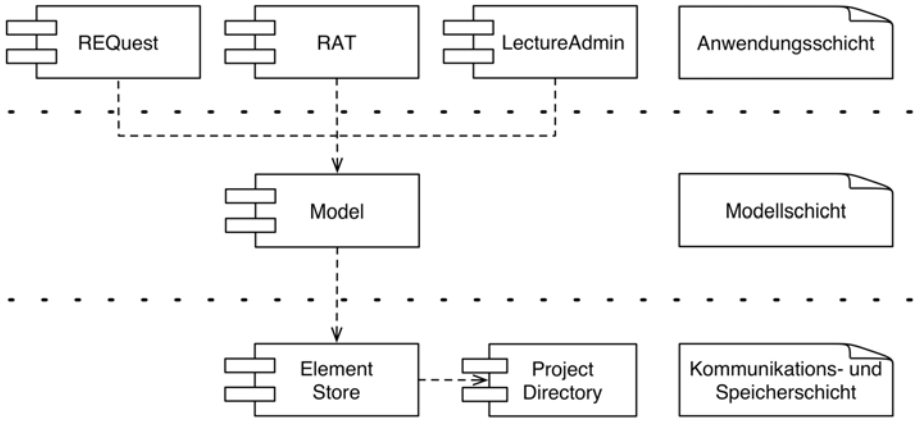

Abb. 3: Schichtenarchitektur von Sysiphus (UML-Komponentendiagramm)

Die Kommunikations- und Speicherschicht beinhaltet den ElementStore und das ProjectDirectory. Der ElementStore ist für die Speicherung der Systemmodell- und Kommunikationsdaten und für die Kommunikationsinfrastruktur von Sysiphus verantwortlich. Zur Authentifizierung von Benutzern und der Zuweisung von Benutzerrechten wird das ProjektDirectory verwendet. In der Modellschicht sind die SWE-Modelle und deren Beziehungen realisiert. Dies sind unter anderem UML-Elemente (z.B. Anwendungsfälle, Klassen, Komponenten), Testfälle, Elemente zur Modellierung von Dokumenten und Modelle zur Kommunikation und Begründungserfassung. Das Modell der Begründungserfassung basiert auf QOC (Question/Option/Criteria) [Mec 99]. Die Anwendungsschicht beinhaltet die Benutzeranwendungen REQuest, RAT und die LectureAdmin-Komponente. Die Anwendungen basieren auf der Modellschicht, welche eine Ortstransparenz der verteilten Architektur von Sysiphus realisiert. Zusätzlich haben wir folgende Anforderungen an das System gestellt: 1. Die Modellschicht und die Client-Anwendungen sollen bei laufendem Server veränderbar und austauschbar sein und 2. der Server soll parallel mit verschiedenen Versionen der Modellschicht und verschiedenen Versionen der Client-Anwendungen arbeiten können.

Durch die Realisierung dieser Anforderungen können verschiedene Gruppen die gleichen oder unterschiedliche Änderungen in der Modellschicht oder in den Client-Anwendungen vornehmen. Trotz dieser Änderungen können alle gleichzeitig mit dem gleichen Server und dem gleichen Datenbestand arbeiten. Bei Änderungen der Client-Anwendungen oder der Modellschicht benötigt der Server keinen Neustart. Dadurch wird der Installations-, Organisations- und Wartungsaufwand bei einer großen Studierendenanzahl während einer Lehrveranstaltung oder eines Praktikums verringert. Darüber hinaus müssen sich die Studierenden nicht in die komplizierte, verteilte Architek-

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

tur von Sysiphus einarbeiten. Je nach Wissensstand können sie leichtere oder schwerere Aufgaben in unterschiedlichen Abstraktionsbereichen der Architektur bearbeiten. Aufgrund der beschriebenen Eigenschaften eignet sich das System Sysiphus hervorragend für den Einsatz in Lehrveranstaltungen des SWE.

2.3 Die Werkzeuge

Zur Bearbeitung des Sysiphus-Systems ist aufgrund seiner Komplexität eine praxisnahe Entwicklungsumgebung erforderlich. Aus früheren Veranstaltungen gab es in Heidelberg erste Erfahrungen mit der Rational-Suite. Diese erfordert aber die Einarbeitung in sehr unterschiedliche Tools und bietet keine Unterstützung für die in einem Praktikum relevanten Kommunikationsaspekte. Weitere Erfahrungen gab es mit der Anwendung REQuest, die wie oben beschrieben auf Sysiphus aufsetzt. Der Einsatz von REQuest bedeutet für die Studierenden, dass sie das System, was sie weiterentwickeln, auch selbst anwenden (und umgekehrt). Dies hat den Vorteil, dass sie sich gut in die Rolle der Nutzer versetzen können und aufgrund ihrer Erfahrungen motiviert sind, das System kennen zu lernen und weiterzuentwickeln. Der Nachteil ist allerdings, dass sie diese Nutzerrolle erst im Laufe der Vorlesung durch die Behandlung der einzelnen Techniken wirklich kennen lernen, d.h., dass die Anwendungsdomäne nicht von Anfang gut bekannt ist. Darüber hinaus besteht die Gefahr der Verwirrung, da die Ebene der Anwendung (d.h. das Einüben des Werkzeugumgangs) nicht klar getrennt ist von der Ebene der Entwicklung (d.h. das Einüben der SWE-Techniken). Die Erfahrungen dazu werden im Abschnitt 4 diskutiert.

Um den Studierenden den Einstieg in die Programmiersprache Java zu erleichtern, setzen wir das IDE Eclipse [Ecl 04] ein. Weiterhin wird JUDE [Jud 04] für die Erstellung von UML-Diagrammen verwendet. Verteiltes Arbeiten wird durch CVS ermöglicht. Als Kriterien für die Auswahl sind insbesondere die freie Verfügbarkeit und die Plattformunabhängigkeit der einzelnen Werkzeuge zu nennen.

3 Ablauf

Das SWE-Fachwissen wurde den Studierenden in einer wöchentlich stattfindenden Vorlesung am Montagvormittag (2 SWS) vermittelt. Die Anwendung des dort erworbenen Wissens stand im Mittelpunkt der Übungsstunden am Montagnachmittag (2 SWS) und Dienstagmittag (2 SWS). Dieser hohe Anteil der Praxis gegenüber der Wissensvermittlung ist für SWE unverzichtbar, da SWE sich nur durch praktische Anwendung der Methoden und Techniken erlernen lässt (vgl. [Ble 99], [Rys 99]).

Um den Studierenden einen gewissen, wenn auch künstlich erzeugten industriellen Projektkontext zu geben, wurden die Aufgaben in kleine Szenarien eingebettet. Die Szenarien beschreiben einen Katastrophenfall in einer Softwarefirma, bei dem aufgrund eines Hardwareproblems die wichtigsten Entwicklungsdokumente zerstört worden sind und nachträglich wieder erzeugt werden müssen. Dabei schlüpften die Studierenden in

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

die Rolle neuer Angestellte in einer Softwarefirma, die sich in das bereits vorhandene System Sysiphus einarbeiten müssen. Nachfolgend ist ein Beispiel einer solchen Szenariobeschreibung aus dem Übungsblatt zum Thema Entwurfsmuster eingefügt:

Um einige wichtige Dokumente aus dem Projekt nachträglich wieder erzeugen zu können, hatte unsere Firma eine Gruppe von Fachinformatikern in Ausbildung zu uns eingeladen. Die Aufgabe dieser Gruppe war es, aus den wichtigsten Quellcodestellen einige Klassendiagramme zu erzeugen. Die Fachinformatiker waren zwar alle Experten auf dem Gebiet der Programmiersprache Java, aber leider hatten sie sehr wenig Erfahrung mit Klassendiagrammen. Sie kannten kaum Entwurfsmuster, die häufig in unserer Firma eingesetzt werden. Daher waren die Diagramme inhaltlich korrekt, aber die Entwurfsmuster, die im Quellcode umgesetzt worden sind, wurden nicht richtig in den Diagrammen dargestellt. Ihre Aufgabe ist es nun, die fehlenden Entwurfsmuster in zwei dieser Klassendiagramme einzufügen.

Die 12 TeilnehmerInnen der Veranstaltung wurden in 3 Gruppen eingeteilt. Die Aufgaben wurden nahezu vollständig in Teamarbeit gelöst. Auf diese Weise waren die Studierenden gezwungen, miteinander zu kommunizieren und zu lernen, gemeinsam schwierige, komplexe und sehr umfangreiche Aufgaben zu lösen. Jedem Team wurde ein persönlicher Betreuer zur Seite gestellt, der in Inhaltsfragen als Ansprechpartner zur Verfügung stand und als Teamleiter fungierte.

Im Nachfolgenden wird kurz genauer auf die wichtigsten Inhalte der beiden Abschnitte des Semesters eingegangen, mit Hauptaugenmerk auf die häufig vernachlässigten Qualitätssicherungsmaßnahmen (in Abbildung 1 fett hervorgehoben). Die Folien sind unter *http://www.-swe.informatik.uni-heidelberg.de* herunterladbar. Übungsmaterialien sind auf Nachfrage erhältlich.

Bottom up, vom Quellcode zu den Anforderungen

Wie in Abbildung 1 zu erkennen ist, wurde das Semester mit einem Crash-Kurs in Java begonnen. Um die Wissenslücken auf dem Gebiet der Java-Programmierung zu überbrücken, wurde innerhalb eines dreistündigen Vortrags Basiswissen über Java,, Java-Servlet und Eclipse vermittelt. Innerhalb der folgenden Woche erhielten die Studierenden zusätzlich zu den SWE-Aufgaben drei Hausaufgaben, in denen sie selbstständig Java-Programmieraufgaben steigender Komplexität lösen sollten.

Als Einarbeitung in die Oberfläche führten die Studierenden Usability-Tests anhand vorgegebener Szenarien aus. In der Vorlesung erfolgte an dieser Stelle auch eine erste Einführung in das Gebiet Qualitätssicherung.

Die zweite Übung beschäftigte sich mit dem Quellcode von Sysiphus. Neben der Ableitung von Klassendiagrammen aus dem Code lernten die Studierenden Programmierrichtlinien kennen und den Code damit zu verbessern.

In der dritten und vierten Übung sollten sich die Studierenden hauptsächlich mit den Modellierungstechniken und dem Integrationstest beschäftigen. Doch es wurde schnell klar, dass es weder in der Vorlesung noch in den Übungen möglich war, alle Dia-

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

grammarten ausreichend zu behandeln. Wir entschieden uns, die Übungen zum Thema Architektur, Sequenzdiagramme und Integrationstest in den zweiten Teil des Semesters zu verlegen. Das Thema Architektur erfordert einen gewissen Überblick, der in der zweiten Hälfte eher vorhanden ist. Sequenzdiagramme kann man am besten motivieren bei der Ableitung von Klassendiagrammen aus Use Cases, d.h. beim Forward Engineering, und Integrationstest baut auf Architektur und Sequenzdiagrammen auf.

In den Aufgaben zum Thema Modellierungstechniken erstellten die Studierenden einerseits aus dem Code Zustandsdiagramme und andererseits ergänzten sie ein vorgegebenes ER-Diagramm. Bei den Sequenzdiagrammen wurden zu Klassenbeziehungen die notwendigen Sequenzdiagramme erstellt. Diese Diagramme wurden zur Spezifikation, Implementierung und Durchführung von Integrationstestfällen verwendet.

In der dritten Übung wurden insbesondere auch Testfälle für Klassen spezifiziert, implementiert und ausgeführt. Die Testfälle wurden unter Verwendung der Äquivalenzklassenbildung in Kombination mit der Anweisungsüberdeckung erzeugt. Auf diese Weise bekamen die Studierenden einen ersten Überblick über die Komplexität des Systems und lernten parallel dazu eine der wichtigsten Arten der Qualitätssicherung kennen.

Die Techniken der Anforderungsspezifikation wurden in der fünften Übung nach dem in [Pae 03] beschriebenen aufgabenorientierten Vorgehen eingesetzt. Die Studierenden sollten zu gegebenen Aufgabenbeschreibungen zuerst vorhandene Use-Case-Texte ergänzen und ein Use-Case-Diagramm ableiten, und danach eigenständige Use-Case-Texte erstellen. Weiterhin hatten die Studierenden die Aufgabe, anhand einer vorgegebenen Checkliste die vorhandene Anforderungsspezifikation, die künstlich eingebaute Fehler enthielt, zu inspizieren. Die während der Inspektion gefundenen Fehler sollten im Anschluss selbstständig beseitigt werden.

Zum Abschluss der ersten Hälfte der Übungen erstellten die Studierenden aus den nun vollständigen und richtigen Anforderungen die Systemtestfälle. Da das zu testende System bereits vorhanden war, wurden die Systemtestfälle sofort implementiert und ausgeführt.

Top down: Die Änderungsaufgabe

Nachdem die Studierenden das Sysiphus hinreichend gut kennen gelernt hatten, erhielten sie die Aufgabe, Sysiphus so zu erweitern, dass ein Inspektor eines Anforderungsdokuments unterstützt wird, d.h. insbesondere syntaktische Probleme wie das Fehlen bestimmter Einträge oder Links von Sysiphus erkannt werden. Um diese Aufgabe zu lösen, mussten sie die erlernten Techniken in einem zusammenhängenden Prozess anwenden (Forward Engineering). Diese Vorgehensweise ist typisch für viele Praktika und soll hier nicht näher erläutert werden. Um auch hier die Qualitätssicherung zu betonen, wurde nach jeweils zwei Wochen eine Meilensteinabnahme durchgeführt, zu der die Studierenden die Ergebnisse des jeweils anderen Teams inspizierten. Hierbei lernten sie eine weitere Art der Inspektion kennen: die perspektiven-basierte Lesetechnik.

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

4 Erfahrungen

TeilnehmerInnen waren Studierende der Angewandten Informatik, Computerlinguistik, Mathematik und der Volkswirtschaftslehre. Dementsprechend hatten die TeilnehmerInnen sehr unterschiedliches Vorwissen, insbesondere auch Programmiererfahrung. Dies wurde bei der Gruppenzusammensetzung berücksichtigt. Eine Gruppe löste sich nach wenigen Wochen aufgrund der hohen Arbeitsbelastung auf (s.u.), so dass eine Neuzusammensetzung nötig war. Dies war aber ohne große Probleme möglich.

Im Laufe des Semesters stellte sich heraus, dass die Vermittlung des Lehrstoffs „bottom up“ sowohl Vor- als auch Nachteile mit sich bringt. Ein deutlicher Vorteil war, dass die Studierenden sich gleich am Anfang mit dem Quellcode beschäftigen konnten. Dies war ohne SWE-Vorwissen möglich. Durch die kurze Einführung in die Programmiersprache Java und die kleinen Übungen konnten die Studierenden langsam und geführt den Quellcode von Sysiphus kennen lernen. Wie erhofft, war die Erstellung der Modelle anhand des Codes relativ einfach. Insbesondere ist es hilfreich, wenn die Studierenden in einem ersten Schritt vorhandene Modelle ergänzen, bevor sie sie dann selbstständig erstellen. Weiterhin kann das Bewusstsein für das Thema Testen von Anfang an geschaffen werden, da von Beginn an ausführbarer Code vorliegt. Beim Forward Engineering konnte dann von Anfang an jeweils die Testplanung parallel zur Spezifikation durchgeführt werden.

Die Vermittlung der SWE-Phasen in umgekehrter Reihenfolge hatte aber den Nachteil, dass die Studierenden relativ bald unter den vielen Einzeltechniken „den Wald vor lauter Bäumen“ nicht mehr sahen und die Zusammenhänge der Techniken erst in der zweiten Hälfte entdeckten. So entstand nach einigen Wochen ein Motivationsproblem, das nur durch erhöhten Betreuungsaufwand abgefangen werden konnte.

Die Verwirrung zwischen den Ebenen Anwendung und Entwicklung war nicht so groß wie befürchtet. Allerdings waren die Studierenden mit dem Usability-Test am Anfang überfordert, da sie zu wenig von der Domäne SWE wussten. Bei der nächsten Durchführung sollen die Studierenden zuerst explorativ mit dem System umgehen, bevor sie einen systematischen Usability-Test durchführen.

Wie oben erwähnt, mussten einige der Übungen, die für den ersten Teil des Semesters geplant waren, in die zweite Hälfte verschoben werden. Die anfängliche Befürchtung, dass dadurch für die Studierenden nicht genügend Zeit bleiben könnte, das erworbene Wissen noch einmal zu üben vor der Durchführung der Änderungsaufgabe, erwies sich als unbegründet.

Sysiphus hat sich als Werkzeug bewährt, da es problemlos die Bearbeitung der gleichen Aufgabe in verschiedenen Teams unterstützt, so dass einerseits in jedem Team eine gemeinsame Version zur Verfügung stand und andererseits, z.B. bei Ergebnispräsentationen, online zwischen den Versionen gewechselt werden konnte.

Die Rationale-Fähigkeiten des Werkzeugs haben sich bewährt, um den Studierenden bei der Einarbeitung Zusatzinformation zur Verfügung zu stellen, z.B. verschiedene Architekturalternativen und Begründungen für die Auswahl der spezifischen Architek-

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

tur. Sie wurden aber nicht zur Kommunikation zwischen den Studierenden genutzt. Im Laufe des Semesters wurde deutlich, dass die mit dem Rationale verbundene Review-Funktionalität auch zur direkten Kommunikation zwischen Betreuer und Studierenden genutzt werden kann. Der Betreuer kann für die Aufgabenbearbeitung spezifische Fragen festlegen, die von den Studierenden beantwortet werden müssen. Dies ermöglichte, sie bei der Bearbeitung der Dokumente „in die richtige Richtung“ zu lenken. Weiterhin konnten die Betreuer den Studierenden mit dem Werkzeug sofort strukturiert Feedback zu ihren Lösungen geben bzw. Lösungen hinterfragen.

Am Schluss des Semesters bewerteten die verbliebenen 8 Studierenden die Lehrveranstaltung. Dabei zeigte sich, dass alle Studierenden viel oder zu mindestens etwas Spaß an den gestellten Aufgaben hatten, auch wenn der Nacharbeitungsaufwand mit durchschnittlich 7 angegebenen Stunden pro Woche doch recht hoch war. Einem Großteil der TeilnehmerInnen war die Verwendung von Sysiphus als Übungsobjekt für den Lernerfolg wichtig (3) bzw. sogar sehr wichtig (2). Eine Person war der Meinung, dass die Verwendung von Sysiphus eher hinderlich für den Lernfortschritt gewesen ist. Die Arbeit in der Gruppe wurde von 3 Studierenden als sehr wichtig und von weiteren 3 Studierenden als wichtig eingeschätzt. Hingegen empfanden 2 Studierende die Arbeit in der Gruppe als störend.

Zusammenfassend kann gesagt werden, dass die Lehrveranstaltung ein Erfolg gewesen ist und bei den Studierenden großen Anklang gefunden hat. Die Studierenden haben trotz der kurzen Zeit gelernt, selbstständig Änderungen an einem großen Softwaresystem durchzuführen. Der Nachbearbeitungsaufwand ist mit 7 Stunden eine Stunde höher, als von uns geplant, aber sicherlich nicht höher als bei anderen, auf praktische Übungen ausgerichteten SWE-Veranstaltungen. Der Betreuungsaufwand war allerdings sehr hoch und ist sicherlich nur bei diesen geringen Studierendenzahlen zu leisten.

5 Zusammenfassung und Ausblick

In diesem Beitrag haben wir eine Möglichkeit vorgestellt, wie man eine SWE-Lehrveranstaltung strukturieren kann, um sowohl SWE-Wissen als auch -Praxis innerhalb eines Semesters vermitteln zu können. Dabei lernten die Studierenden im ersten Schritt ein bestehendes komplexes Softwaresystem durch Reverse Engineering zu verstehen und im zweiten Schritt in dieses System Änderungen durch Forward Engineering einzubauen. Auf diese Weise ließ sich insbesondere Qualitätssicherung früh thematisieren.

Die Lehrveranstaltung wird im kommenden Sommersemester wiederholt. Dazu wird den Studierenden ein Prozesshandbuch zur Verfügung stehen, das das Forward-Engineering-Vorgehen und die einzelnen Techniken erklärt. Weiterhin wollen wir die Rationale-Funktionalität des Werkzeugs verstärkt einsetzen. Die Studierenden sollen in der Lage sein, ihre getroffenen Entscheidungen und die betrachteten Alternativen für andere sichtbar zu machen. Dies soll insbesondere die Selbstreflexion fördern [Lew 01].

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

Danksagung

Wir danken Philipp Häfele für seinen Einsatz bei der Gruppenbetreuung und allen TeilnehmerInnen für die konstruktive Mitarbeit.

Literatur

[Bec 03] P. Becker-Pechau, W.G Bleek, H. Züllighoven: Integration agiler Prozesse in die Softwaretechnik-Ausbildung im Informatik-Grundstudium. In: SEUH ‘03, dpunkt.verlag 2003, S. 8–21.

[Ble 99] W.G Bleek, G. Gyczan, C. Lilienthal, M. Lippert, S. Roocks, H. Wolf, H. Züllighoven: Von anwendungsorientierter Softwareentwicklung zu anwendungsorientierten Lehrveranstaltungen der Werkzeug & Material-Ansatz in der Lehre. In: SEUH ’99, Teubner, 1999, S. 9–20.

[Bot 01] K. Bothe, U. Sacklowski: Praxisnähe durch Reverse Engineering – Projekte: Erfahrungen und Verallgemeinerungen. In: SEUH ’01, dpunkt.verlag, 2001, S. 11–21.

[Dem 99] B. Demuth, H. Hußmann, S.Zschaler, L. Schmitz: Erfahrungen mit einem frameworkbasierten Softwarepraktikum. In: SEUH ’99, Teubner, 1999, S. 21–30.

[Dut 02] A.H. Dutoit, B. Paech: Rationale-based use case specification, Requirements Engineering Journal, vol. 7, S. 3–19, 2002.

[Dut 04] A.H. Dutoit, T. Wolf, B. Paech, B. Borner, J. Rückert: Using Rationale in Software Engineering Education, in submission, 2004.

[Ecl 04] Eclipse *http://www.eclipse.org*, 2004.

[Jud 04] JUDE *http://objectclub.esm.co.jp/Jude/*, 2004.

[Kle 01] N. Kleiner, S. Sarstedt: Einsatz von Standardprozessen bei der Gestaltung von Lehrveranstaltungen, In: SEUH’01, dpunkt verlag, 2001, S.99–108.

[Lew 01] C. Lewerentz, H. Rust: Die Rolle der Reflexion in Softwarepraktika. In: SEUH ’01, dpunkt.verlag, 2001, S. 73–86.

[Lic 03] H. Lichter, R. Melchisedech, O. Scholz, T. Wiler: Erfahrung mit einem Workshop-Seminar im Software Engineering-Unterricht. In: SEUH ’03, dpunkt.verlag, 2003, S. 89–100.

[Mec 99] A. MecLean, R.M. Young, V. Bellotti, T. Moran: Questions, options, and criteria: Elements of design space analysis. Human-Computer Interaction, 6(11):201–250, 1999.

[Met 03] A. Metzger: Konzeption und Analyse eines Softwarepraktikums im Grundstudium, In: SEUH’03, dpunkt verlag, 2003, S.41–48.

[Pae 03] B. Paech, K. Kohler: Task-driven requirements in object-oriented development, In: J. Leite, J. Doorn, (eds.): Perspectives on Requirements Engineering, Kluwer Academic Publishers, 2003.

[Rys 99] J. Ryser, M. Glinz: Konzipierung und Durchführung eines Software-Praktikums – Ein Erfahrungsbericht. In : SEUH ’99, Teubner, 1999, S. 55–68.

[Spi 01] A. Spillner: Erfahrung mit einem Werkzeug zur Projektunterstützung: In: SEUH ‘01, dpunkt.verlag, 2001, S. 45–54.

[Sys 04] Sysiphus *http://wwwbruegge.in.tum.de/Sysiphus*, 2004.

[Wol 04] T. Wolf, A.H. Dutoit: A Rationale-based Analysis Tool, In 13th International Conference on Intelligent & Adaptive Systems and Software Engineering 2004 (IASSE'04), p. 209–214.