Prof. Dr.-Ing. Klaus-Peter Löhr, Professor für Informatik an der Freien Universität Berlin seit 1985, davor an der Universität Bremen. Promotion und Habilitation an der Technischen Universität Berlin. Fachgebiete: Systemsoftware, Softwaretechnik, Middleware. Projekte in den Bereichen nichtsequentielle und verteilte Programmiersprachen, Sicherheitsmanagement für verteilte Objektsysteme, komponentenbasierte Entwicklung.

Prof. Dr. rer. nat. Horst Lichter hat Informatik an der Universität Kaiserslautern studiert. Anschließend war er als wissenschaftlicher Mitarbeiter in der Abteilung Software Engineering der ETH Zürich und der Universität Stuttgart tätig. Von 1993 bis 1995 war er Mitarbeiter der Schweizerischen Bankgesellschaft Zürich und danach bis 1998 Mitarbeiter des ABB Forschungszentrums Heidelberg. Seither ist er Professor für Informatik an der RWTH Aachen mit dem Schwerpunkt Software-Konstruktion und Qualitätssicherung.

Wir danken den Sponsoren der SEUH 2005:

ABB BOSCH dpunkt.verlag sd&m

Klaus-Peter Löhr · Horst Lichter (Hrsg.)

Software Engineering im Unterricht der Hochschulen

SEUH 9 – Aachen 2005

dpunkt.verlag

Klaus-Peter Löhr
lohr@inf.fu-berlin.de

Horst Lichter
lichter@cs.rwth-aachen.de

Herstellung: Birgit Bäuerlein
Umschlaggestaltung: Helmut Kraus, Düsseldorf
Druck und Bindung: Koninklijke Wöhrmann B.V., Zutphen, Niederlande

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

ISBN 3-89864-328-X
1. Auflage 2005

Ringstraße 19 b
69115 Heidelberg

Vorwort

Der vorliegende Tagungsband enthält die Beiträge zum 9. Workshop *Software Engineering im Unterricht der Hochschulen* (SEUH). 1992 von Jochen Ludewig ins Leben gerufen, findet dieser Workshop alle zwei Jahre statt. Er wird von der Gesellschaft für Informatik und dem German Chapter of the ACM gemeinsam mit der Schweizer Informatik-Gesellschaft durchgeführt. Ausbildung in Softwaretechnik an Universitäten und Fachhochschulen – wie ist sie in das Informatik-Studium eingebettet, wie sollte sie gestaltet werden, welche Erfahrungen gibt es? Dies sind die Themen, um die es bei den SEUH-Workshops geht.

SEUH gehört mittlerweile zum festen Bestandteil der Informatik-Tagungsreihen im deutschsprachigen Raum. Sein hoher Bekanntheitsgrad und seine zunehmende Beliebtheit als Forum für den Meinungs- und Erfahrungsaustausch haben der diesjährigen Veranstaltung die erfreuliche Anzahl von 32 eingereichten Beiträgen beschert, darunter viele von hoher Qualität. Das Programmkomitee hat 12 Beiträge zur Präsentation und Aufnahme in den vorliegenden Tagungsband ausgewählt; zusätzlich wurden 4 Kurzbeiträge angenommen. Für die einzelnen Sitzungen des Workshops sind die Beiträge wie folgt gruppiert:

- *Didaktik und Training*
- *Softwaretechnik-Praktika*
- *Fernlehre und Nahlehre*
- *PSP und XP*
- *Praktika mit externen Projekten*

Der überwiegende Teil der eingereichten Beiträge hat Softwaretechnik-Praktika und -Projekte zum Gegenstand. Auch zu einigen anderen Themen, die im Aufruf zur Einreichung genannt worden waren, gab es Beiträge. Bemerkenswert ist, zu welchen Themen keine Beiträge eingereicht wurden: Formale Methoden, modellgetriebene und komponentenbasierte Entwicklung, Invarianten der Softwaretechnik-Lehre. Das letztere Thema ist zugegebenermaßen schwierig; wir bedauern es sehr, dass keiner es gewagt hat, sich darauf einzulassen. (Hier gibt es also einen Merkpunkt für den nächsten Workshop ;-)

Ein wichtiges – und immer noch junges – Thema, *empirische und experimentelle Softwaretechnik*, war im Aufruf zur Einreichung *nicht* genannt worden. Walter Tichy, Universität Karlsruhe, hat sich dankenswerterweise bereit erklärt, im Rahmen eines eingeladenen Vortrags über *Ausbildung in empirischer Softwaretechnik* zu berichten.

Wir hoffen, dass der diesjährige SEUH-Workshop ähnlich erfolgreich wie seine Vorgänger verlaufen wird, und danken allen, die durch ihre Einreichungen dazu beigetragen haben. Den Mitgliedern des Programmkomitees gebührt diesmal besonderer

Dank, da sie wegen der zahlreichen Einreichungen mehr an Begutachtungsarbeit zu leisten hatten als ursprünglich vorgesehen:

- *Ruth Breu, Universität Innsbruck*
- *Harald Gall, Universität Zürich*
- *Wolfgang Hesse, Universität Marburg*
- *Horst Lichter, RWTH Aachen*
- *Jörg Raasch, Hochschule für Angewandte Wissenschaften Hamburg*
- *Günter Riedewald, Universität Rostock*
- *Silke Seehusen, Fachhochschule Lübeck*
- *Johannes Siedersleben, sd&m AG und Fachhochschule Rosenheim*
- *Debora Weber-Wulff, Fachhochschule für Technik und Wirtschaft Berlin*
- *Andreas Zeller, Universität Saarbrücken*

Um die Einreichungen zu verwalten, den Begutachtungsprozess zu unterstützen und das Tagungsprogramm zusammenzustellen, wurde das Konferenz-Verwaltungssystem Paperdyne eingesetzt, wodurch die Arbeit des Programmkomitees erheblich erleichtert wurde. Dank der Großzügigkeit unseres Softwaretechnik-Kollegen Volker Gruhn, Universität Jena, konnten wir Paperdyne unentgeltlich benutzen. Dank gebührt ferner unseren industriellen Sponsoren, ohne deren Unterstützung wir den Workshop nicht in einem so angenehmen Rahmen hätten durchführen können: Es sind dies die ABB AG, die BOSCH AG, der dpunkt.verlag und die sd&m AG. Frau Preisendanz vom dpunkt.verlag danken wir sehr herzlich für die gute Zusammenarbeit bei der Erstellung des Tagungsbands.

Dezember 2004

Klaus-Peter Löhr
FU Berlin
Programmkomitee

Horst Lichter
RWTH Aachen
Lokale Organisation

Das Ständige Organisationskomitee der SEUH besteht aus

Jochen Ludewig, Universität Stuttgart
Günter Riedewald, Universität Rostock
Andreas Spillner, Hochschule Bremen

Inhalt

Eingeladener Vortrag

Didaktik und Training

Softwaretechnik-Praktika

Fernlehre und Nahlehre

PSP und XP

Praktika mit externen Projekten

Kurzvorträge

Eingeladener Vortrag:

Ausbildung in empirischer Softwaretechnik

Walter F. Tichy
Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe
Postfach 6980,76128 Karlsruhe
tichy@ira.uka.de

Zusammenfassung

Empirische Untersuchungen sind aus der Softwareforschung nicht mehr wegzudenken. Zukünftige Software-Ingenieure sollten daher in der Lage sein, empirische Studien in der Softwaretechnik zu verstehen und in ihrer Aussagekraft einzuschätzen. Forscher sollten zusätzlich die Methodik des Experimentierens erlernen können. Aus diesem Grunde bietet die Universität Karlsruhe seit SS 2001 die Vorlesung „Empirische Softwaretechnik“ an. In dieser Veranstaltung werden sowohl wichtige Ergebnisse aus der empirischen Softwareforschung als auch die experimentelle Methodik behandelt. Aus Motivationsgründen ist die Vorlesung stark an Beispielstudien orientiert, an denen die verschiedenen Methoden wie Experimentaufbau, statistische Auswertung usw. erläutert werden. Die Konzeption dieser Vorlesung und Erfahrungen werden besprochen.

Methoden und Techniken zum Erreichen didaktischer Ziele in Software-Engineering-Praktika

Robert Stoyan, Martin Glinz
Institut für Informatik, Universität Zürich
Winterthurerstr. 190, CH-8057 Zürich, Schweiz
{stoyan, glinz}@ifi.unizh.ch

Zusammenfassung

Praktika an der Hochschule sind wirksamer Bestandteil der Software-Engineering-Ausbildung, sie sind aber auch eine Ausbildungsform, in der didaktische Herausforderungen und Möglichkeiten besonders intensiv zusammentreffen. Zu den didaktischen Zielen gehören eine hohe Motivation der Studierenden und Tutoren, eine wirksame und individuelle Leistungskontrolle, Praxisbezug sowie ein erfolgreicher Umgang mit unterschiedlichen Vorkenntnissen. Dieser Beitrag präsentiert Methoden und Techniken, um diese Ziele zu erreichen, und erläutert sie anhand von Beispielen. Aus diesen speziellen Maßnahmen werden Prinzipien für den Unterricht des Software Engineering extrahiert.

1 Einleitung

Software-Praktika, in denen die Prinzipien des professionellen Software Engineering in Projektform erlernt werden, sind seit längerem ein Standardbaustein in Informatik-Curricula an Hochschulen. In der SEUH-Reihe wurden in der Vergangenheit mehrfach Berichte über solche Praktika publiziert. In diesem Beitrag destillieren wir aus eigenen Erfahrungen und denen anderer Autoren eine Reihe von Methoden und Techniken zur Erreichung didaktischer Ziele, die für Software-Praktika besonders relevant sind. Aus Platzgründen beschränken wir uns auf die folgenden fünf Ziele:

1. Praxisnähe
2. Motivation der Teilnehmer
3. Motivation der Tutoren
4. Erfolg im Umgang mit unterschiedlichen Vorkenntnissen
5. Individualität und Wirksamkeit der Leistungskontrolle

Im nächsten Kapitel stellen wir das Software-Praktikum an der Universität Zürich als typisches Beispiel vor. Das darauf folgende Kapitel behandelt die Methoden und Techniken, gegliedert nach den didaktischen Zielen. Das Software-Praktikum dient hier als

durchgängiges Fallbeispiel, um die Anwendung unmittelbar zu veranschaulichen. Das letzte Kapitel diskutiert die Ergebnisse und verallgemeinert sie.

2 Fallbeispiel: Das Software-Praktikum an der Universität Zürich

Das Software-Praktikum an der Universität Zürich (kurz „SoPra“) ist ein Pflichtpraktikum im vierten Semester des Studiums der Wirtschaftsinformatik an der Universität Zürich. Die Studierenden sollen im SoPra lernen, in einem Team von 4-6 Personen ein Software-Projekt von den Anforderungen bis zur Präsentation des Produkts durchzuführen. Es nehmen je nach Jahrgang 80-160 Studierende teil, aufgeteilt in drei bis vier Klassen. Alle Teams bearbeiten die gleiche Aufgabe. Ein Teil des benötigten Codes wird zur Verfügung gestellt, um Reengineering zu üben. Das SoPra hat wöchentlich vier Stunden Präsenzzeit mit Anwesenheitspflicht. Voraussetzung sind Programmierkenntnisse in Java, die jedoch erfahrungsgemäß sehr unterschiedlich ausfallen, sowie der Besuch einer zweistündigen Vorlesung Software Engineering im Semester davor.

Für jede Durchführung des SoPra wird ein wissenschaftlicher Assistent als Praktikumsleiter bestimmt (2004 der Erstautor dieses Beitrags); ein Professor (der Zweitautor) trägt die Gesamtverantwortung für Konzept und Durchführung des SoPra. Zur Durchführung des Praktikums und zur Betreuung der Studierenden stehen dem Praktikumsleiter pro Klasse ein Obertutor und vier Tutoren zur Verfügung. Jeder Tutor ist für zwei Teams verantwortlich. Die Obertutoren bereiten gemeinsam mit dem Praktikumsleiter das gesamte Praktikum vor. Sie sind an der Auswahl der ihnen zugeordneten Tutoren aus dem Kreis der Bewerberinnen und Bewerber beteiligt, anschließend schulen sie diese in zwei Tagen Präsenzunterricht. Im Projekt haben Praktikumsleiter und Obertutoren die Rolle des Auftraggebers, die Tutoren die von Coaches, die Teams sind Firmen.

Die Praktikumsaufgabe im Sommersemester 2004 war die Erstellung der Software für ein Börsencafé, bei dem die Preise ausgewählter Speisen und Getränke sich je nach Nachfrage der Gäste ändern. Die Software muss die Erfassung der Bestellungen der Gäste, die Berechnung der variablen Preise, die Anzeige der aktuellen Preise und der Preisentwicklung sowie das Erstellen der Rechnungen für die Gäste unterstützen. Die erste Lektion beinhaltete Teamfindung und das erste Kundengespräch. Nach Anforderungsspezifikation, Oberflächenprototyp, erneuten Kundengesprächen, Modellierung, Architektur- und Detailentwurf, Reviews, Programmieren und Test endete das Praktikum mit der Wettkampfpräsentation aller Teams vor dem Kunden.

Der Aufwand des Praktikumsleiters betrug 1,3 Personenmonate für die Durchführung und 2 Personenmonate für die Weiterentwicklung der didaktischen Methoden und der Unterlagen des Praktikums. Seit der erstmaligen Durchführung des SoPra im Jahr 1996 sind mehr als ein Personenjahr Aufwand in die Entwicklung von Unterlagen, Aufgaben, dem minutiösen „Praktikumsdrehbuch“ und zur Verfügung gestelltem Code geflossen [Ryser1999].

Wir haben fast alle der nachstehend beschriebenen Methoden und Techniken im SoPra 2004 selbst mit Erfolg eingesetzt. Das Praktikum hat in dieser Form die beste Bewertung durch die Studierenden seit seinem Bestehen erhalten.

3 Methoden und Techniken

3.1 Didaktisches Ziel: Praxisnähe

Praxisnähe ist ein klassisches Ziel der Ausbildung im Software Engineering. Software-Praktika bieten hierzu viele methodische Möglichkeiten:

Echte Kunden, echte Anwendung: Die authentischste Praxisnähe gibt die Praxis selber – was jedoch nicht funktioniert, ist die Verbindung eines betreuten, nach Lernzielen konzipierten Praktikums im Hochschulbetrieb mit einem normalen kommerziellen Kundenprojekt. Erfolgreiche Modelle sind a) Entwicklung für Non-Profit-Kunden [Bothe2001], [Raasch1997]; b) kommerzielle Kunden kommen zur Abschlusslektion, die als Wettkampfpräsentation durchgeführt wird, und haben so die Gelegenheit, besonders gute Studierende kennen zu lernen (SoPra); c) für kommerzielle Kunden werden Aufgaben mit geringer Priorität gelöst [Forbrig1997].

Kundengespräche: Ob echt oder gespielt vom Dozenten, Kundengespräche sind ein Erlebnis im Vergleich zur üblichen Stoffvermittlung an Hochschulen. Zu einem wirklich wertvollen Lerninhalt werden sie jedoch dann, wenn auch professionelle Gesprächsstrategien gelehrt werden, beispielsweise das schrittweise Durchgehen von Geschäftsabläufen in einer Sprache, welche der Kunde versteht, das gezielte Nachfragen („Habe ich richtig verstanden, dass ... ?“) und natürlich das Stellen der Kernfrage „Wann werden Sie mit unserer Arbeit zufrieden sein?“.

Professionalität in der Form: In der Praxis müssen Ergebnisse auch äußerlich professionell sein. Dies wird im Hochschulbetrieb vielfach nicht verlangt. In schriftlichen Prüfungen beispielsweise wird nur die Konzentration auf den Inhalt trainiert: Solange die Abgabe gerade noch lesbar ist, zählt sie. In der Praxis ist diese Denkweise ein verhängnisvoller Fehler. Daher soll der Dozent diesen Unterschied hervorheben, und wenn er die Rolle des Kunden spielt, total unzufrieden sein, wenn diese Äußerlichkeiten der Professionalität nicht gegeben sind („Ich muss die Zusammenarbeit mit Ihrer Firma überdenken, wenn Sie solche Schmierpapiere produzieren!“).

Kundenorientiertes Denken: Wenn Studierende oder Mitarbeiter gefragt werden, was von einer Aufgabenliste für den Kunden wichtig sein könnte, schaffen es viele, sinnvoll Prioritäten festzulegen. Dennoch versagen die meisten in realen Situationen, da ihnen niemand sagt, dass dies jetzt eine Übung zum Priorisieren ist und nach welchen Kriterien sie das denn tun sollten. Solche Situationen können für Ausbildungszwecke leicht hergestellt werden: Den Teilnehmern wird nichts Besonderes gesagt, sie bekommen zu wenig Zeit für eine Aufgabe, und der Kunde wartet auf das Ergebnis.

Im SoPra 2004 beispielsweise sollten die Teams aufgrund eines Kundengesprächs eine grobe Skizze der Benutzeroberfläche des Börsencafés anfertigen. Es war unmög-

lich, in der zur Verfügung gestellten Zeit die vollständige Oberfläche zu entwerfen. Aus Sicht des Informatikers ist die Administration der Artikel und Bestellungen zentral, weil sie u. a. das Klassenmodell ergeben, während die Gestaltung der Anzeigetafel für die Artikel mit variablem Preis sekundär ist. Für den Auftraggeber (den Wirt des Börsencafés) ist es umgekehrt: Von der Gestaltung der Anzeigetafel hängt der geschäftliche Erfolg eines Börsencafés und damit prinzipiell die Brauchbarkeit der Software ab. Teams, welche aus Zeitgründen die Preisanzeige weggelassen hatten, konnten dann erleben, wie ihr Kunde entrüstet feststellte, dass das Arbeitsergebnis für ihn wertlos sei. Durch dieses „Auflaufenlassen“ und anschließende Rückmeldung durch den Kunden (d. h. den Praktikumsleiter) hatten die Teilnehmer ein deutliches Aha-Erlebnis, und es gab viel spontanes Lob am Ende dieser Praktikumseinheit.

Was in der konkreten Projektsituation wichtig oder unwichtig ist, ist immer anders. Die gelernte Lektion ist jedoch, dass das Problem ohne besondere Aufforderung immer aus Kundensicht betrachtet und für den Kunden zufriedenstellend gelöst werden muss.

Große Software und Reengineering: Die Praxisnähe fortgeschrittener Praktika kann durch Aufgaben gesteigert werden, die den Überblick eines großen Systems erfordern. Hierzu sollte der gesamte Softwareentwicklungsprozess bereits erlernt und erprobt sein. Bewährte Möglichkeiten sind ein Reengineering-Projekt [Bothe2001] oder ein großes, immer weitergeführtes Projekt [Raasch1997]. In allen Praktika kann mit Reengineering konfrontiert werden, indem (undokumentierte!) Codeteile der Lösung vorgegeben werden (SoPra).

Harte Termine: Harte Termine sind in jedem Praktikum leicht implementierbar, indem Termine für die Abgabe von Zwischen- und Endprodukten gesetzt werden. Der Praktikumsleiter sollte dabei thematisieren, welche Optionen ein Team zur Einhaltung harter Termine hat (Mehrarbeit, Abstriche bei Funktionalität, Abstriche bei Qualität ...) und danach aufzeigen, welche Konsequenzen die getroffenen Entscheidungen haben.

Professioneller Werkzeugeinsatz: Zu viele Werkzeuge lenken vom eigentlichen Ziel der Ausbildung ab; wenige helfen, die Konzepte durch Anwendung zu lernen [Glinz1996]. Praxisübliche Werkzeuge für Modellierung, Codierung, Übersetzung und Fehlersuche sind in diesem Sinne richtig am Platz in jedem Software-Praktikum. Die obligatorische Verwendung eines Konfigurationsmanagement-Werkzeugs ist jedoch zweischneidig: Ohne das Chaos einmal selbst erlebt zu haben, sehen die Studierenden die Notwendigkeit des Konfigurationsmanagements nicht ein.

„Dirty Tricks“: Künstlich herbeigeführte Serverabstürze, korrupte Code Repositories usw. werden von [Dawson2000] empfohlen, um Praxisnähe herzustellen. Unserer Erfahrung nach kreieren Teilnehmer vergleichbare „unvorhersehbare“ Produktivitätsverluste bereits selber durch unklare Abstimmung im Team, vergessenes Abspeichern von Arbeitsergebnissen etc. Insgesamt erscheint ein sehr sparsamer Einsatz von „Dirty Tricks“, vorzugsweise bei sehr Fortgeschrittenen, sinnvoll.

Weitere Maßnahmen:

- Integration mit vorgegebenen Schnittstellen
- Projektmanagement: siehe Abschnitt „Leistungskontrolle“
- Wechselnde Ansprechpartner seitens des Kunden
- Änderungen und verspätete Präzisierung von Anforderungen

3.2 Didaktisches Ziel: Motivation der Teilnehmer

Über einen engen Praxisbezug wird bereits viel zur Motivation beigetragen. Folgende Punkte können die Motivation weiter steigern:

Wettkampf: Es ist eine der stärksten Motivationsmethoden, die Teams wettstrebend dieselbe Aufgabe lösen zu lassen. Der Wettkampf im SoPra 2004 war sehr motivierend: Einzelne Teams erstellten fernsehreife Vertriebsshows für die Abschlusspräsentation. Damit der Wettkampf und die Bestimmung des Siegerteams am Schluss auch Spaß macht, kam unter anderem ein „Applausometer“ zum Einsatz, ein Phonometer vom Physikinstitut, womit der Publikumsbeifall für die Präsentationen der Teams gemessen und in Wettkampfpunkte verwandelt wurde.

Lustige und bunte Anwendung: Die Aufgabe soll nicht nur Technisches enthalten, sondern auch Sichtbares und Gestalterisches. Die Aufgabe „Börsencafé“ im SoPra 2004 war beispielsweise viel motivierender, als es eine Verwaltungsanwendung gewesen wäre. Sie gab den Teilnehmern die Gelegenheit, sich in der professionellen Gestaltung einer auf die emotionalen Bedürfnisse der Endkunden zugeschnittenen Oberfläche und einer an Maßstäben professioneller Benutzbarkeit [Stoyan2004] orientierten Administrationsoberfläche zu üben. Da Benutzbarkeit ohnehin in fast jedem Informatik-Studiengang zu kurz kommt, in der Praxis aber maßgeblich über die Akzeptanz der Software entscheidet, sollten alle Gelegenheiten genutzt werden, dies zu lehren.

Gestaltungsfreiheit: Als der Kunde im SoPra 2004 lediglich betriebswirtschaftliche Ziele vorgab, hatten die Studierenden zunächst den Eindruck, dass es keine klaren Anforderungen gebe. Die Studierenden mussten darauf aufmerksam gemacht werden, dass der Kunde nichts von Informatik versteht und die Software daran messen wird, wie gut sie sein Geschäft unterstützt. Dann entwickelten die Teilnehmer jedoch viel Spaß am technisch freieren, aber betriebswirtschaftlich zielorientierten Gestalten. Technisch wurden vom Praktikum her Programmiersprache, Zielplattform und zu verwendende Codeteile vorgegeben. Das Ziel des Kunden lautete: „Wir möchten den Umsatz steigern.“ Die Teilnehmer mussten dann – in der Sprache des Kunden – erfragen, wie das denn geschehen soll. „Durch niedrigen Personalaufwand, indem häufig benötigte Abläufe sehr schnell durchzuführen sind. Weiterhin ...“

Natürlich können technische Vorgaben genauso praxisrelevant sein, wenn der Auftraggeber technisch versiert ist. Motivatorisch und für das Lernen per Überraschung empfehlen wir jedoch die betriebswirtschaftlichen Ziele.

Stau unausgesprochener Gefühle und ungelöster Probleme vermeiden: Zu den stärksten Motivationskillern gehören unausgesprochene, nicht angehörte oder nicht

beantwortete negative Gefühle. Wer motivierte Teilnehmer will, sollte sich nicht darauf beschränken, Probleme zu lösen, wenn darum gebeten wird oder diese offensichtlich werden. Das ist die zu erwartende Minimalleistung. Sie kostet jedoch keineswegs minimale Arbeit, weil gerade zum Beispiel Teamprobleme nur so lange einfache und gute Lösungen haben, wie sie nicht offensichtlich oder extrem störend sind [Stoyan2004]. Bewährt hat es sich hier, den Teilnehmern klar zu zeigen „Probleme sind willkommen, bitte berichten!“, dies jedoch kombiniert mit Einhaltung der „Eskalationshierarchie“, dass also der Ansprechpartner zuerst immer der Tutor und nicht der Praktikumsleiter ist. Indem dies im Laufe der Veranstaltung gelegentlich wiederholt wird, aber auch indem die Teilnehmer zur Halbzeit einen Feedbackbogen ausfüllen, werden negative Gefühle abgeschöpft, sodass sie nicht weiter wachsen. Eine typische Ursache für sich anstauende negative Gefühle sind mangelnde Leistungen bei Kommilitonen, wenn diese ohne Reaktion akzeptiert werden („Wozu bemühe ich mich dann?“). Siehe hierzu den Absatz „Teamleistung“ im Abschnitt „Leistungskontrolle“.

Permanentes Feedback: Besonders motivierend wirkt es sich hier aus, wenn zusätzlich zum beschreibenden Feedback (mündlich bei mündlichen Leistungen, schriftlich bei schriftlichen) jede Abgabe benotet wird. Sinnvollerweise sollen die Noten auch in das Abschlussergebnis des Praktikums eingehen. So sehen alle, dass gute Arbeit erkannt und belohnt wird.

Arbeitstausch: Wenn Teams ihre Zwischenergebnisse tauschen und so weiterarbeiten sollen, erleben sie persönlich, wie wichtig Verständlichkeit von Konzeption bzw. Code ist. In dem Praktikum [Carls1993] entwickelten die Teams kooperativ Teile einer Software, hier steigerte es die Motivation. Insbesondere bei konkurrierenden Teams wird der Tausch von Arbeitsergebnissen, wie wir es im SoPra früher praktiziert hatten, jedoch leicht zum Motivationskiller: „Wozu haben wir uns dann angestrengt, wenn man uns unsere Ergebnisse wegnimmt?“

Tutoren müssen zur Lehrveranstaltung stehen: Selbst ein noch so kleines „Naserümpfen“ der Tutoren zu den didaktischen Methoden oder den Inhalten der Lehrveranstaltung spüren die Teilnehmer sofort, und das unterminiert die Glaubwürdigkeit und damit die Motivation. Daher soll einerseits im Vorstellungsgespräch beobachtet werden, ob die Bewerber für Tutorate wirklich den Sinn der Lehrveranstaltung verstehen. Andererseits soll ihnen Gelegenheit zu Kritik und Verbesserung gegeben werden. So gehört es im SoPra zur Schulung der Tutoren, diese hierzu zu befragen und ihre Punkte dort wo möglich auch gleich in das Praktikum einzubauen. Auch kann das wöchentliche Vorbereitungstreffen der Tutoren bewusst als Gelegenheit zum „Dampf ablassen“ genutzt werden. Diese psychohygienische Maßnahme hilft, ihren Frust dorthin zu lenken, wo er hingehört: zum Praktikumsleiter, anstelle zu den Teilnehmern. Wenn innere Ablehnungen bekannt werden, dann können sie beantwortet werden.

Permanente Beschäftigung in der Präsenzzeit: Wenn Teilnehmer während längerer Zeit nicht selbst aktiv sind, lenken sie ihre Aufmerksamkeit auf andere Dinge, Motivation und Lerneffekt sinken, und sie werden potenzielle Störenfriede.

3.3 Didaktisches Ziel: Motivation der Tutoren

Den Tutoren die Augen öffnen, was alles zu ihren Aufgaben gehört, und ihre Arbeit immer wieder persönlich prüfen und wertschätzen sind sicher das Wichtigste für eine hohe Motivation. Wie nachfolgend dargestellt, kann dies bewirken, dass Tutoren ihre Aufgabe als sehr nützlich wahrnehmen und als ihren ersten professionellen Einsatz in einer Führungsposition erleben.

Die Kontrolle ihrer Arbeit zeigt den Tutoren, dass ihre Bemühungen wahrgenommen werden und der Praktikumsleitung nicht egal sind. Sie sollte nur teilweise den Obertutoren überlassen werden. Diese sollten möglichst bereits Erfahrung in der Lehre haben, damit sie die Arbeit der Tutoren beurteilen können. Einfach präsent zu sein und irgendetwas im Übungsraum zu tun, gibt dem Praktikumsleiter eine unauffällige Möglichkeit, die Arbeit der Tutoren wahrzunehmen. Das häufigste Problem ist Passivität in Situationen, die nicht per se eine Aktion der Tutoren erfordern, zum Beispiel bei Programmierübungen. Wenn Passivität akzeptiert wird, so führt dies zunächst zu Bequemlichkeit und dann bald zum Gefühl, überflüssig zu sein. Wie soll aber jemand, der die Mehrheit der Zeit (scheinbar) nicht gebraucht wird, motiviert sein? So mussten in früheren Durchführungen des SoPra die Teilnehmer manche Tutoren erst in der Cafeteria suchen, wenn sie Fragen hatten. Seitdem wurde eine feste Regel eingeführt für die Arbeit der Tutoren: „Ständiger Kontakt mit der Gruppe." Dies wurde kombiniert mit der Erwartungshaltung, nicht nur auf Anfrage zu helfen, sondern auch gelegentlich Teilnehmer mit irgendetwas anzusprechen, und nicht nur bei Hausaufgaben, sondern auch bei Präsenzübungen Feedback zu geben. So hatten die Tutoren spürbar mehr Gefühl, nützlich zu sein, und somit mehr Motivation.

„Über Probleme sprechen, besonders über menschliche!" ist ein anderes Arbeitsprinzip. Wenn im SoPra Tutoren oder Obertutoren über (Team-)Probleme berichten, bekommen sie stets Anhörung und Anerkennung. Es wird bewusst getrennt zwischen der Erwartung, Probleme ständig zu berichten, und der Annahme, dass sie in der Lage sind, die meisten selber zu lösen. Während die Leitung somit ein genaueres Bild über den Zustand des Praktikums erhält, profitieren die Tutoren, indem sie ihre Rolle ganz anders erleben. Aus Sicht des Projekts für den Kunden sind sie ein Coach, dies gehört zur traditionellen Tutorenrolle. Aus Sicht der hier dargestellten Arbeitsweise und der Tutor-Obertutor-Praktikumsleiter-Hierarchie sind sie Führungskräfte wie in einer Firmenorganisation, in der Mitarbeitern auf allen Ebenen viel Eigenverantwortung zugestanden wird.

Bewerbungsverfahren: Vorstellungsgespräche, eventuell vorab eine Beschreibung relevanter Qualifikationen in einer Bewerbungsmail, bewirken das Gefühl „Wir sind ausgewählt und dürfen diesen Job tun". Im SoPra wurde die Erfahrung gemacht, dass gar nicht so sehr der Inhalt, sondern die Form von Bewerbungsmails sehr viel über die Motivation der Bewerber aussagt. Um qualifizierte und motivierte Tutoren auswählen zu können, braucht es viele Bewerber. Es wurde beobachtet, dass die Anzahl der Bewerber stark davon abhängt, wie gelungen das Praktikum das letzte Mal war.

Anerkennung als Studienleistung: ECTS-Punkte sind eine kostenfreie Form der Bezahlung. Die SoPra-Tutoren erhielten für ihre Leistung vier ECTS-Punkte. Dies ist wahrscheinlich auch ein wesentlicher Grund dafür, dass wir die Tutoren aus genügend vielen Bewerbern aussuchen konnten.

Ein zusätzliches kleines Gehalt: Dies hatte im SoPra den positiven Motivationseffekt, dass das Gefühl eines formalen Arbeitsverhältnisses entstand.

3.4 Didaktisches Ziel: Erfolg im Umgang mit unterschiedlichen Vorkenntnissen

Es gibt eine Reihe von Methoden, mit unterschiedlichen Vorkenntnissen der Teilnehmer konstruktiv umzugehen, Maßnahmen zum Ausgleich zu ergreifen oder zum Nacharbeiten fehlender Vorkenntnisse zu motivieren.

Druck: Gleich in der ersten Stunde einen konstruktiven Druck aufbauen:

- Das erwartete Niveau an Vorkenntnissen objektiv beschreiben („Sie müssen selbstständig eine einfache Programmieraufgabe lösen können, z. B. ...“).
- Eine grobe aber quantifizierte Schätzung geben, wie viel Zeit die Teilnehmer bei unterschiedlichen Vorkenntnissen für die Lehrveranstaltung einplanen müssen („sehr grob ca. ein Tag plus Präsenzzeit, wenn Vorkenntnisse erfüllt, ansonsten ...“).
- Zeigen, wie die personenbezogene Ergebniskontrolle erfolgt und was die Konsequenzen sind. (Beispiel: Projektplan mit Namen, wer was macht. Der Tutor wird zu den Abgaben Fragen an einzelne Personen stellen.) Hinweisen auf die soziale Kontrolle durch das Team, ob man diesem zur Last fallen will.
- Benennen, was Teilnehmer tun müssen, die im Kurs verbleiben wollen, aber die Vorkenntnisse nicht erfüllen („Folgendes Java-Buch nehmen und inklusive der dortigen Übungen bis Kapitel ... durcharbeiten, hierfür mindestens ... Zeit einplanen.“).

Eine dermaßen akkurate Ansage ist eher ungewöhnlich und löst bei den Studierenden ein spürbares Erwachen aus. Es ist die faire Information, die sie brauchen, um Tendenzen zur Aufwandsminimierung zu überwinden und ihre erfolgreiche Teilnahme verantwortlich planen zu können. Nur wenn der Hinweis zu Beginn erfolgt, besteht noch real die Chance, dass fehlendes Vorwissen aufgearbeitet wird.

Pairprogramming: Dieses Prinzip vom eXtreme Programming [Beck2000] sieht vor, dass zwei Entwickler gemeinsam an einem Rechner programmieren: Der eine hat die Tastatur und erklärt, was er tut, der andere fragt nach und gibt Feedback. Ein Semester lang angewendet, wird es helfen, Unterschiede auszugleichen, ein bis zwei Pairprogramming-Sitzungen haben eine normierende Wirkung: Jeder weiß nachher, wo das Gruppensoll liegt. Im SoPra war die Reaktion der Teilnehmer auf Pairprogramming leicht positiv: Schlechte Programmierer sind interessiert daran, einzelne gute haben Spaß am Lehren. Die Mehrheit der guten empfindet es als kleine Last.

Theorie und Praxis zusammen lehren: Bei theoretischen Kenntnissen ist es das Beste, diese gleichzeitig mit dem Praktikum zu bieten, so werden zum Beispiel bei [Kerer2004] Praktikum und Vorlesung zum selben Thema im gleichen Semester gehal-

ten. Oft geht dies jedoch aus stundenplantechnischen Gründen nicht. Ein weiteres Problem ist, dass zu Beginn noch zu wenig des erforderlichen Basiswissens verfügbar ist.

An der Universität Zürich erwerben die Studierenden das Grundwissen in Form einer zweistündigen Vorlesung über Software Engineering im Semester vor dem SoPra. Um dieses Wissen wieder in den Vordergrund zu holen und mit dem notwendigen Spezialwissen zu ergänzen, verwenden wir im SoPra ein angeleitetes Literaturstudium und haben damit gute Erfahrungen gemacht. Über eine Web-Lernplattform werden für jede Praktikumswoche im Schnitt zehn Seiten Software Engineering Inhalte bereitgestellt, welche genau auf die jeweilige Praktikumseinheit abgestimmt sind. Zu Beginn des Praktikums mehr, gegen Ende weniger. Diese sind vor der Praktikumseinheit zu lesen und enthalten auch den Übungsablauf. Dies wurde kombiniert mit einer deutlichen Erwartungshaltung, dass jeder sich vorbereitet, und einer Gestaltung der Präsenzzeit, welche eine hinreichende Vorbereitung erzwingt. In früheren Durchführungen des SoPra hatten wir die zu den Praktikumseinheiten gehörende Theorie zu Beginn der Veranstaltung verkürzt vorgetragen, weil die meisten Studierenden nicht ausreichend vorbereitet zu den Praktikumsterminen kamen. Dies hatte den Nachteil, dass vorbereitete Studierende demotiviert wurden und nicht vorbereitete nur einen gekürzten Ersatz für das nicht Gelesene erhielten.

Diese Vorträge wurden im SoPra 2004 nach dem Prinzip „Studierende können selber lesen“ auf Organisatorisches und auf Motivation reduziert. Den Obertutoren fiel es nicht leicht, anstelle von Theorie Motivation zu vermitteln – eine nicht alltägliche Aufgabe im Studium. Wenn der Praktikumsleiter über eigene Praxiserfahrung verfügt, kann er gelegentlich den Motivationsteil der Einleitung übernehmen und persönliche Erfahrungen aus der Praxis zu dem Thema der Praktikumseinheit einbringen.

3.5 Didaktisches Ziel: Individualität und Wirksamkeit der Leistungskontrolle

Die Leistung erfolgt im Team, aber den Leistungsnachweis bekommt jeder Teilnehmer einzeln. Das ist der voreingebaute Widerspruch von Unterrichtsformen mit Teamarbeit. Die Ernsthaftigkeit des Problems der Leistungskontrolle zeigt [Kerer2004]. In dem berichteten Großpraktikum waren zwei Aspekte der Leistungskontrolle ersichtlich, die Benotung und die Entlarvung von Betrügern, zwei der drei größten Aufwände des gesamten Praktikums. Wie kann eine treffsichere individuelle Leistungskontrolle bei moderatem Aufwand erfolgen?

Tutorenschulung: Bei Kundengesprächen, Reviews und Konzeptionsarbeiten mit viel Kommunikation merken Tutoren, wer die aktiveren und wer die passiveren Teilnehmer sind und wer die Leistung grob verfehlt. Sie haben jedoch in der Regel zu wenig Erfahrung und manchmal auch keinen Mut, genügende von nicht mehr genügender Leistung zu trennen, und auch nicht die Führungsfähigkeiten, um auf mangelhafte Leistungen richtig zu reagieren. Sie müssen jedoch die Aufgabe der Leistungsbewertung erfüllen, weil sie am Ort des Geschehens sind. Für kommunikative Praktikumsaufgaben in der Präsenzzeit kann unserer Erfahrung nach eine gute Leistungskontrolle durch

Schulung der Tutoren und Aussprechen einer klaren Erwartungshaltung erzielt werden. Für SoPra besteht diese Schulung aus einfachen Rollenspielen, in denen im Tutorenkreis Übungen des Praktikums durchgeführt werden. Selbstverständlich müssen alle Tutoren das Praktikum kennen, also selber bereits absolviert haben. Einige spielen passive Teilnehmer, dies ist das typische Leistungsproblem. Teamprobleme wie Dominanz Einzelner können gleich mittrainiert werden. Der Tutor muss nun von vorher besprochenen Reaktionsmöglichkeiten eine angemessene anwenden. Beispiele: rein beschreibendes Feedback, aktives Eingreifen („Ihr beide habt euch dieses Mal kaum beteiligt, bitte übernehmt in der kommenden Woche die ...“) oder sogar eine Anweisung mit Benennung einer Konsequenz („Wenn du einen Leistungsnachweis willst, ...“). Als Einleitung der Schulung beschreiben die Tutoren Probleme, die sie als frühere Kursteilnehmer erlebt haben, und sagen, wie sie sich Lösungen vorstellen. Diese Schulung wird nur wirksam, wenn die Erwartungshaltung über das Semester aufrechterhalten wird. Dies erfolgt, indem Praktikumsleiter und Tutoren beobachtete Teamprobleme und deren Lösungen diskutieren. Der Praktikumsleiter fragt immer wieder nach Problemen („Bitte sagt, wo es drückt, ich höre euch zu!“). Wenn Tutoren nicht über Probleme berichten, ist dies ein klares Zeichen, dass sie ihre Aufgaben der Leistungskontrolle und als Teamarbeitsmentor nicht wahrnehmen.

Projektplan: Beim Programmieren ist individuelle Leistungskontrolle schwierig, da dies meist außerhalb der Präsenzzeit geschieht. Hier sollte das Team notieren, wer was gemacht hat. Schlechte Karten hat, wer einfache, unschuldige Fragen zu angeblich seinem Teilergebnis nicht beantworten kann. Der Ort, an dem ohnehin Aufgaben zugeordnet werden, ist der Projektplan. Besprechungsprotokolle eignen sich auch sehr gut [Forbrig1997]. Der Plan soll Aufgaben, geschätzte Aufwände und Namen enthalten, und nicht nur die zukünftigen, sondern auch die erledigten Aufgaben sind jede Woche zu aktualisieren. Zu Beginn solcher Praktika wird Projektplanung nicht immer sinnvoll sein, denn wie soll ein einzelnes Klassendiagramm, welches im Team erarbeitet wurde, Verantwortlichen zugeordnet werden? Hier ist aber auch die im Punkt „Tutorenschulung“ beschriebene kommunikative Kontrolle noch wirksam, etwa beim Review des Klassendiagramms. Spätestens ab Feinentwurf und Programmierung wird der Projektplan zum wirksamen Kontrollinstrument.

Leistungskontrolle per Prüfung: [Kerer2004] machte nach mehreren Versuchen die Praktikumsteilnahme optional und verlagerte die Leistungskontrolle auf die Prüfung zum Schluss. Nach den berichteten Erfahrungen war dies bei 600 Teilnehmern und eindeutig technischem Lernziel die beste Lösung.

Zählen von LOC: In [Gehrke2002] wird empfohlen, für alle Kursteilnehmer die LOC (Lines of Code) zu zählen, um Trittbrettfahrer zu vermeiden.

Kontrolle der Teamleistung: Wenn Teamarbeit zu lernen mit zu den Zielen des Praktikums gehört, ist es berechtigt und notwendig, auch dessen Ergebnisse zu kontrollieren. Dies ist an sich einfach. Im SoPra wurde direkt vorgegeben, dass zwingend in jeder Praktikumseinheit alle Ergebnisse vom Team abgegeben werden müssen, damit die Teammitglieder ihren Leistungsnachweis erhalten. Die Tutoren geben Lob und erklären Mängel, bei größeren Problemen muss das Team noch mal abgeben.

Der damit vorprogrammierte Frust bei unterschiedlichen Leistungen im Team ist der schwierigere Teil. Dieser kann vorab mit dem Vergleich zur Industriepraxis und Definition als Lernziel in Praxisnähe verwandelt werden. Damit dies zum vollwertigen Praktikumsinhalt wird, müssen Teamarbeitskompetenzen auch unterrichtet werden, siehe z. B. [Raasch1997]. Speziell bezüglich Leistungskontrolle wurde im SoPra gelehrt, was getan werden kann: Jeder hat die Eigenverantwortung abzuschätzen, ob er mit seinen Teammitgliedern zum Ziel kommt. Wenn während der Arbeit die eigene Leistung oder die eines Teammitglieds ungenügend ist, offen darüber sprechen und nach Abhilfe suchen sowie nach Aufgaben, welche ein solches Teammitglied dennoch übernehmen kann. Wenn das Team das Problem nicht lösen kann, dem Tutor Bescheid sagen. Wenn Tutor und Obertutor das Problem nicht lösen können, gelangt das Problem an die Praktikumsleitung, die im Extremfall Teams neu zusammenstellen oder überprüfen kann, ob der Betreffende die Voraussetzungen für das Praktikum erfüllt. Das liegt so weit auf der Hand, es wird jedoch erst dann wirksam, wenn es während des Praktikums mehrfach, genau in den Lektionen, in denen erfahrungsgemäß Probleme auftauchen, sowohl Teilnehmern als auch Tutoren gesagt wird.

Eine interessante Beobachtung war, dass Teammitglieder einerseits die Trittbrettfahrer fast nie auffliegen lassen, andererseits jedoch dankbar sind, wenn Tutoren und Kursleitung ihre Aufgabe wahrnehmen, solche Probleme zu vermeiden oder gegebenenfalls die betreffenden Personen zu identifizieren und fair zu handhaben.

Differenzierung innerhalb der Teamleistung: Mit Hilfe der bislang beschriebenen Methoden kann eine faire Beurteilung erfolgen, ob Teilnehmer den Kurs bestanden haben. Um differenzierte Noten zu vergeben verwendet [Forbrig1997] eine Teamnote mit Auf- und Abschlägen je nach messbaren individuellen Leistungen. Dort wird auch auf Erfahrungen anderer mit verschiedensten Teambenotungsstrategien verwiesen.

Plagiate vermeiden geht vor bestrafen: Dem Kopieren von Code kann durch mehrere Maßnahmen entgegnet werden: Wettkampfgeist hilft gegen Kopien unter den Teams. Der entstehende Geheimhaltungseffekt ist so stark, dass im SoPra bei einem angeordneten wechselseitigen Code-Review der Teams alle sorgfältig darauf achteten, den zu reviewenden Code so auszuwählen, dass keine für den Sieg entscheidenden Produktmerkmale von der anderen Gruppe einzusehen waren. Die Reviews an sich konnten zum Glück ohne Beeinträchtigung konstruktiv durchgeführt werden.

Kleine Modifikationen der Aufgabenstellung mit möglichst durchgängigen Auswirkungen auf den Code helfen gegen das Kopieren von Lösungen aus früheren Semestern – wer es dennoch tut, hat zumindest eine herausfordernde Übung zum Reengineering bewältigt und damit ebenfalls für seinen Leistungsnachweis gearbeitet.

Teamarbeit an sich ist auch bereits eine Methode, um Teilnehmer vom Kopieren abzuhalten – hier wirkt soziale Kontrolle. In den Untersuchungen von [Kerer2004] hat sich die Anzahl der Plagiate verachtfacht, während die Anzahl der Abgaben sich nur vervierfacht hatte, als von Teamarbeit auf Einzelarbeit umgestellt wurde.

Schließlich gibt es gute Werkzeuge zur Erkennung von Plagiaten [Prechelt2002]. Während dies in der Literatur gerne empfohlen wird und technisch gut funktioniert, ist es didaktisch gesehen eine Maßnahme, die nur Verlierer kreiert: gescheiterte Teilneh-

mer sowie Dozenten, die Aufwand und Unannehmlichkeit der Diskussion mit den Betroffenen tragen müssen. Zum Überprüfen der Wirksamkeit der vorigen „weichen" Maßnahmen jedoch sehr zu empfehlen!

Good Guy & Bad Guy: Die Tutoren geraten durch die Notwendigkeit der Leistungsbeurteilung zwangsläufig in einen Konflikt zwischen zwei Rollen: Coach vs. Kontrolleur. Dieser Konflikt lässt sich durch ein „Good Guy / Bad Guy"-System mildern: Der Praktikumsleiter als Bad Guy übernimmt die Vertrauen mindernden Aufgaben wie das Stellen expliziter, unangenehmer Kontrollfragen oder das Sanktionieren von Fehlverhalten. Der Tutor als Good Guy sucht nach dem Positiven oder verhält sich zumindest unschuldig.

4 Diskussion

In den beschriebenen Methoden und Techniken für Praktika lassen sich einige generelle Prinzipien des Unterrichts von Software Engineering an Hochschulen erkennen:

Praxisnähe lässt sich auch in Lehrveranstaltungen an Hochschulen herstellen. Ob die Teilnehmer eine Lehrveranstaltung als praxisnah empfinden, hängt unserer Erfahrung von folgenden Kriterien ab, die sich in Praktika besonders gut umsetzen lassen:

- Lernhandeln: Die Veranstaltung bietet viele Möglichkeiten, selber zu tun.
- Überraschung: Möglichst viele dieser Übungen sind ein neues oder im Ausbildungskontext überraschendes Erlebnis für die Teilnehmer. Gegenbeispiel: Programmieren würde niemand als besondere Praxiserfahrung hervorheben.
- Authentizität & Nutzen: Die jeweiligen Lerninhalte sind für die Teilnehmer erkennbar praxisrelevant. Der Dozierende erklärt, welchen Nutzen das zu Lernende im späteren Berufsleben haben wird. Authentizität entsteht insbesondere, wenn Dozierende über eigene Praxiserfahrung verfügen und über diese berichten können. Letztlich kommt man um die Erkenntnis nicht herum, dass nur solche Personen Software Engineering authentisch und praxisnah unterrichten können, die selbst in der Praxis Software Engineering betrieben haben oder betreiben.

Eine Umsetzung dieser Kriterien ist nicht nur in Praktika möglich. Ein schönes Beispiel ist das von Lichter beschriebene Workshop-Seminar [Lichter2003].

Mit der Praxisnähe ergeben sich auch viele Möglichkeiten der *Motivation*. Wenn zudem das Lernziel professionelle Softwareentwicklung ist, liegt es nahe, auch die dort angewendeten Managementtechniken zu Motivation und Führung in der Lehre einzusetzen.

Mangelnde oder heterogene *Vorkenntnisse* sind in jeder Lehrveranstaltung problematisch. Bei Ausbildungsformen mit Teamarbeit ist das Problem einerseits verschärft, indem fehlende Vorkenntnisse einzelner Studierender den Erfolg ganzer Teams gefährden können. Andererseits besteht speziell im Software Engineering das störendste Defizit (zumindest nach unseren Erfahrungen in Zürich) meist bei den Programmierkenntnissen. Da Programmierung im Softwareentwicklungsprozess erst weiter hinten kommt,

eröffnet sich die Möglichkeit, fehlende Programmiervorkenntnisse gezielt auszugleichen.

Gleichzeitige Vermittlung von Theorie und Praxis trägt ebenfalls zum Ausgleich fehlender oder heterogener Vorkenntnisse bei.

Leistungskontrolle birgt im Software Engineering eine besondere Herausforderung, da unterschiedlichste Fähigkeiten gelehrt werden, in denen die Leistung nicht auf dieselbe Art geprüft werden kann. Theoretisches Wissen, Projektplanung, Konzeption, Programmierung, Teamarbeit, Präsentation, Kundengespräche etc. spannen ein Feld auf, welches sich jeder einzelnen Prüfungsmethode entzieht: Wissensabfrage, Lösen schriftlicher Aufgaben, Durchführung von Experimenten, Beobachtung der Arbeit der Studierenden über ein Semester, Präsentationen, Rollenspiele, Kontrolle der Teamleistung, Kontrolle der Einzelleistung – sie reichen alleine alle nicht aus. Vielmehr ist eine sehr differenzierte Leistungskontrolle unter Nutzung aller Kanäle erforderlich, auch derjenigen, die mehr Aufwand kosten. Erschwerend kommt hinzu, dass im Software Engineering hohe Studierendenzahlen, oft der gesamte Studienjahrgang, typisch sind.

Generell stellt sich die Frage, wie kommunikative Fähigkeiten in Informatik-Studiengängen besser geschult und sogar zum Gegenstand von Prüfungen gemacht werden können. In anderen Disziplinen, zum Beispiel in der Gesangs- und der Theaterausbildung, gehören solche Prüfungen zum Alltag.

Wir haben uns in diesem Beitrag vor allem auf Erfahrungen an der Universität Zürich sowie ausgewählte Veröffentlichungen anderer Autoren gestützt. Natürlich gibt es eine Menge weiterer Literatur zu diesem Thema, insbesondere zur generellen Frage didaktischer Ziele und deren Erreichung. Eine umfassende Übersicht würde jedoch den Rahmen dieser Veröffentlichung bei weitem sprengen.

Wir nehmen in diesem Beitrag eine andere Perspektive ein, als dies bei Berichten über Software-Praktika sonst der Fall ist: Nicht zum Praktikum wird beschrieben, welcher didaktische Nutzen entsteht, sondern zu didaktischen Zielen wird genannt, welche Methoden und Techniken zur Erreichung dieser Ziele beitragen.

Im Aufruf zur Einreichung von Beiträgen für die SEUH 2005 wird als Themenschwerpunkt unter anderem „Invarianten der Softwaretechnik-Lehre (Verfallsdatum > 2050)" genannt. Wir hoffen, dass diejenigen von uns, die im Jahr 2050 noch leben, dann verifizieren können, dass einige der in unserem Beitrag zusammengetragenen Erkenntnisse zu diesen Invarianten gehören.

Danksagung

Wir danken allen, die an der Entwicklung und Durchführung des Software-Praktikums an der Universität Zürich beteiligt waren und sind. Unser besonderer Dank richtet sich an Silvio Meier, Christian Seybold, Nancy Merlo-Schett und Philippe Schürmann für den Erfahrungsaustausch, Harald Gall und Gerald Reif für die konstruktiven Hinweise zu diesem Artikel, sowie nicht zuletzt unseren Tutoren und Studierenden für unzählige kompetente Rückmeldungen.

Literatur

[Beck2000] K. Beck: eXtreme Programming explained: Embrace Change. Reading: Addison-Wesley.

[Bothe2001] K. Bothe, U. Sackowski: Praxisnähe durch Reverse Engineering-Projekte: Erfahrungen und Verallgemeinerungen. In H. Lichter, M. Glinz (Hrsg.): *Software Engineering im Unterricht der Hochschulen, SEUH7 – Zürich 2001*. Heidelberg: dpunkt.verlag, S. 11-21.

[Carls1993] H. Carls, J. Raasch: Der Unterschied zwischen Theorie und Praxis – Vorlesung und Praktikum „Software-Engineering“. In J. Raasch, T. Bassler (Hrsg.): *Software Engineering im Unterricht der Hochschulen, SEUH'93*. Stuttgart: Teubner, S. 105-114.

[Dawson2000] R. Dawson: Twenty Dirty Tricks to Train Software Engineers. *22nd International Conference on Software Engineering*, Limerick, Irland, 2000, p. 209-218.

[Forbrig1997] P. Forbrig: Probleme der Themenwahl und der Bewertung bei der Projektarbeit in der Software Engineering Ausbildung. In P. Forbrig, G. Riedewald (Hrsg.): *Software Engineering im Unterricht der Hochschulen, SEUH'97*. Stuttgart: Teubner, S. 45-52.

[Gehrke2002] M. Gehrke et al.: Reporting about Industrial Strength Software Engineering Courses for Undergraduates. *24th International Conference on Software Engineering*, Orlando, Florida, p. 395-405.

[Glinz1996] M. Glinz: The Teacher: "Concepts!" The Student: "Tools!"; On the Number and Importance of Concepts, Methods, and Tools to be Taught in Software Engineering Education. Proc. Third International Workshop on Software Engineering Education, Berlin. *Softwaretechnik-Trends* **16**,1 (1996).

[Kerer2004] C. Kerer, G. Reif et al.: ShareMe: Running A Distributed Systems Lab For 600 Students With 3 Faculty Members. Akzeptiert zur Publication in *IEEE Transactions on Education. http://www.infosys.tuwien.ac.at/reports/bin/abstract.pl?report=TUV-1841-2003-27*

[Lichter2003] H. Lichter et al.: Erfahrungen mit einem Workshop-Seminar im Software-Engineering-Unterricht. In J. Siedersleben, D. Weber-Wulff (Hrsg.): *Software Engineering im Unterricht der Hochschulen, SEUH 8 - Berlin 2003*. Heidelberg: dpunkt.verlag, S. 89-100.

[Prechelt2002] L. Prechelt, G. Malpohl and M. Philippsen: Finding Plagiarisms among a Set of Programs with JPlag. *Journal of Universal Computer Science* **8**, 11 (November 2002), p. 1016-1038.

[Raasch1997] J. Raasch, A. Sack-Hauchwitz: Kooperation, Kommunikation, Präsentation: Lernziele im Software-Engineering-Projekt. In P. Forbrig, G. Riedewald (Hrsg.): *Software Engineering im Unterricht der Hochschulen, SEUH'97*. Stuttgart: Teubner, S. 34-44.

[Ryser1999] J. Ryser, M. Glinz: Konzipierung und Durchführung eines Software-Praktikums – Ein Erfahrungsbericht. In B. Dreher, C. Schulz, D. Weber-Wulff (Hrsg.): *Software Engineering im Unterricht der Hochschulen, SEUH'99*. Stuttgart: Teubner. S. 55-68.

[Stoyan2004] R. Stoyan (Hrsg.): Management von Webprojekten: Führung, Projektplan, Vertrag. Berlin: Springer.

Vier Formen der Erfahrungsvermittlung im Studium

Kurt Schneider
FG Software Engineering, Universität Hannover
Welfengarten 1, 30167 Hannover
Kurt.Schneider@Inf.Uni-Hannover.de

Zusammenfassung

Software Engineering ist eines der Fächer, in denen Wissensvermittlung nicht ausreicht. Bei Software-Qualität oder im Software-Entwurf ist darüber hinaus Erfahrung erforderlich – und Erfahrung schließt eigenes Erleben und dadurch erzeugte Gefühle ein.

Wie kann aber Erfahrung in diesem Sinne vermittelt werden? Schon lange werden verschiedene Praktikumsformen diskutiert, die dies leisten sollen. Im vorliegenden Beitrag stelle ich vier Ansätze zur gezielten Vermittlung von Erfahrungen kurz vor. Sie werden danach verglichen und auf ihre Tauglichkeit für die Software-Engineering-Lehre hin diskutiert.

1 Einführung

Im Software Engineering gibt es zahlreiche Techniken und Methoden, die Studenten einfach kennen müssen. Dieses Wissen zu vermitteln, ist eine Aufgabe der Software-Engineering-Lehre. Aber solches Wissen reicht bekanntermaßen nicht aus, um praktische Projekte zu bearbeiten. Vielmehr sind Fertigkeiten und Erfahrungen erforderlich, die das Wissen ergänzen müssen.

Software Engineering insgesamt und mehrere seiner Teilgebiete haben einige Eigenschaften, die für besonders „erfahrungslastige“ Disziplinen typisch sind [Sch 02]:

- Technisches Wissen und wirtschaftlich-pragmatisches Urteilsvermögen müssen zusammenkommen.
- Die Rahmenbedingungen und Traditionen der Anwender sind zu beachten. Das scheinbar Optimale ist nicht immer vermittelbar.
- Es gibt keine klaren Kriterien für eine beste Lösung. Gangbare Lösungen müssen in der Diskussion erarbeitet werden.

Zum Beispiel braucht ein Software-Architekt diese Fähigkeiten, um zu einer befriedigenden Lösung zu gelangen. Aber auch Qualitätsbeauftragte müssen beurteilen können, welche Maßnahmen durchsetzbar sind. Nahe liegend mag scheinen, Studierende in „echte Projekte“ in Wirtschaftsunternehmen zu stecken, damit sie dort ihre Erfahrungen machen. Diese gute Idee hat aber leider auch einige Nachteile:

- Nicht an jedem Hochschulstandort sind auf Dauer ausreichend viele Praktikumsplätze vorhanden. Firmen scheuen oft davor zurück, ständig für wenige Wochen neue Praktikanten einzulernen.
- Die Pflicht, einen Praktikumsbericht zu schreiben, ersetzt nicht Betreuung und bietet kaum Unterstützung zur eigenen Reflexion. Es kann also durchaus gelingen, dass Studierende zwar „in der Praxis“ arbeiten und dort auch an Vorgängen teilhaben, die durchaus typisch sind – dies aber gar nicht richtig wahrnehmen. Wie sollen sie auch erkennen, was nützliche Erlebnisse sind, und diese in nutzbare Erfahrung übersetzen?
- Schließlich führen Praktika in den neuen Bachelor-Studiengängen zu einem Engpass im ohnehin sehr kurzen Studienplan. Wenn in sechs Semestern die Berufsbefähigung erworben werden soll, kann man sich kaum leisten, zwei davon für Bachelorarbeit und Praktikum aufzuwenden. Irgendwann muss ja auch das nötige Grundwissen vermittelt werden.

An der Universität Hannover wurde daher das Praktikum aus dem Studienplan genommen – wenn auch mit ausdrücklichem Bedauern. In der Industrie ist man sich der Bedeutung von *Erfahrung* sehr bewusst. Es gibt in vielen großen Firmen Initiativen, die den Wissens-, aber auch Erfahrungsaustausch explizit anstreben und durch Projekte unterstützen [Joh 99; Con 00; Din 00]. Für DaimlerChrysler war ich am SEC-Projekt beteiligt, in dem auf mehreren Ebenen der Austausch und die Nutzung von Erfahrungen betrieben wurde [Lan 99].

Unsere Definition für Erfahrung war:

$$\textit{Erfahrung} =_{Def} \textit{Beobachtung + Gefühl + Schlussfolgerung}$$

Eine Beobachtung kann nur machen, wer an einem Vorgang beteiligt ist oder ihm zumindest beiwohnt. Erfahrung kann nicht aus der Theorie abgeleitet werden. Wichtig ist dabei das Gefühlsmoment: Wird die Beobachtung durch Euphorie – oder aber durch ein Gefühl der Enttäuschung, der Frustration oder des Ärgers – begleitet, so prägt sich die Beobachtung tiefer ein und wird mit Assoziationen angereichert. Die Schlussfolgerung (oft mit sehr hypothetischem Charakter) macht aus dem Eindruck des Erlebten eine Lehre, die sich in der Zukunft wiederverwenden lässt. Wird Erfahrung weitergegeben, so wird sie häufig auf diese Schlussfolgerung (als Verhaltensregel oder Mahnung) verkürzt; dann fehlen ihr aber die Einprägsamkeit des Gefühls und die Authentizität einer konkreten Beobachtung.

Wie gibt man aber (vollwertige) Erfahrungen im Studium weiter? An der Hochschule wird die *Wissensweitergabe* ja über Vorlesungen, Übungen und Seminare bewältigt, Praktika richten sich schon stärker an das praktische Können und die Erfahrung. Ich möchte im vorliegenden Beitrag vier verschiedene Ansätze vorstellen, die im Studium explizit (auch) die Weitergabe von *Erfahrungen* bewirken sollen. Dann vergleiche ich die Kernideen und arbeite heraus, wofür sich welche Form gut eignet und wann weniger. Daraus ergeben sich einige Empfehlungen, wie man Erfahrungen (nicht Wissen) im Studium handhaben kann. Der Beitrag ist als Diskussionsimpuls zu verstehen.

2 Simulation eines ganzen Software-Projekts: SESAM

Das Projekt SESAM wurde vor über zehn Jahren am Lehrstuhl von Prof. Jochen Ludewig an der Universität Stuttgart gestartet [Dra 95]. Ihm liegt die Idee zugrunde, ein interaktives Simulations- und Lehrspiel für Software-Projektleiter zu entwickeln. Dabei sollte ein gesamtes Software-Projekt simuliert werden, nur der Projektleiter selbst war real und interagierte mit dem simulierten „Rest“ seines Projekts. SESAM wurde schon in den frühen 90er-Jahren in der Lehre eingesetzt [Sch 94]. Es wird auch heute noch weiterentwickelt [ASE 04].

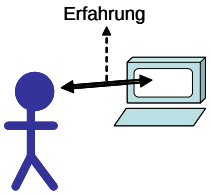

Abb. 1.: SESAM: Erfahrung erwächst aus Interaktion mit dem simulierten Projekt

Im Kern braucht SESAM ein umfassendes dynamisches Modell des simulierten Entwicklungsprojekts [Dei 94]. Es muss auf „korrektes“ (das heißt: der Vorlesung entsprechendes) Verhalten des Projektleiters reagieren, aber auch auf Verstöße und Fehler ein plausibles Verhalten zeigen. So ein Modell ist schwer zu validieren, und die Modellentwicklung ist selbst eine spannende Forschungsaufgabe. Rodriguez et al. beziehen sich auf einen anderen Simulationsansatz, aber mit ähnlicher Absicht [Rod 04].

Im Hinblick auf die Erfahrungsvermittlung versucht SESAM, in einem simulierten und absolut nicht realen Umfeld authentische Erfahrungen zu vermitteln. In erster Näherung soll es ja keine Rolle spielen, dass die Reize aus dem Projekt nur durch den Computer vermittelt werden. Der Projektleiter soll *beobachten*, wie sich seine Handlungen und die Eigendynamik des simulierten Projekts auswirken. Versäumt er zum Beispiel, ein Review anzuberaumen, so hat er später kaum eine Möglichkeit, die Qualität des Entwurfs zu beurteilen. Dies ist so unerfreulich wie in einem entsprechenden realen Projekt. Auch die *Gefühlsantwort* kann daher ähnlich und vergleichbar intensiv

sein. Wird nun noch die gleiche *Schlussfolgerung* gezogen, dann ist Erfahrung in allen drei Aspekten vom Modellentwickler an den spielenden Projektleiter vermittelt worden.

Wir haben bei den ersten Einsätzen von SESAM am Rande bemerkt, dass die computerbasierte Simulation nicht unbedingt die authentischsten Gefühle hervorrief. Solange der Simulator noch nicht zur Verfügung stand, hatten wir das Modell schon (mit Excel-Unterstützung) „manuell" benutzt [Dei 94]. Die von Hand errechneten Reaktionen wurden von Assistenten den Spielern in definiertem Wochenraster überbracht. Menschliche Betreuer als Informationsüberbringer führten offenbar zu einer stärkeren gefühlsmäßigen Identifikation mit dem simulierten Projekt und erhöhten offenbar die Glaubwürdigkeit der simulierten Informationen aus Spielersicht. Dagegen wirkt (wieder nach Aussage der damaligen Studenten) die reine Simulation am Rechner eher steril und unecht – keine gute Voraussetzung für authentische Gefühle oder Erfahrungen.

In den letzten Jahren hat SESAM hier sicher Fortschritte gemacht, doch mir geht es hier um eine allgemeinere Beobachtung zu Simulationen: Eine „zu glatt und zu schnell" laufende Simulation ist wohl generell eher kontraproduktiv für die Erfahrungsvermittlung. Wenn das simulierte Projekt in wenigen Minuten abgewickelt werden kann, wird allein dadurch Planung fast unmöglich, und Situationen wie banges Warten werden verhindert. Auch kann sich nicht in Minuten dieselbe Spannung aufbauen, wie wenn man auf die Antwort drei Tage warten muss. Spannung, banges Warten und dergleichen bräuchte man für starke, nach der obigen Definition vollwertige Erfahrungen. Einen anderen Anspruch kann aber auch die schnell laufende Simulation durchaus einlösen: Nämlich festzustellen, ob man das gelernte *Wissen* des Projektleiters schnell reproduzieren kann. Damit kann man richtiges Verhalten üben und falsches erkennen.

3 Erfahrung pur, obwohl man etwas anderes tut: Agile Hour

Agile Methoden sind in den letzten Jahren viel diskutiert worden. Besonders eXtreme Programming von Kent Beck hat großen Zuspruch, aber auch extreme Ablehnung ausgelöst [Bec 00]. Inzwischen hat die Kontroverse an Schärfe verloren; man versucht nun, die Stärken von agilen und anderen Ansätzen zu verbinden. Dazu gibt es interessante Bücher, zum Beispiel [Boe 03]. Für viele Hochschulen gehört Grundwissen über agile Methoden heute zum Handwerkszeug des Software-Ingenieurs. Aber wieder reicht es nicht, etwa die XP-Praktiken aufzulisten und zu definieren, was dabei getan wird. Wieder fehlen *Erfahrungen* mit ihren drei Aspekten: eigene Beobachtung, Gefühl und abgeleitete Schlussfolgerung. Gerade Verfahren wie agile Methoden, die so stark auf die gute Zusammenarbeit von Entwicklern und Kunden im Team bauen, brauchen dort eine breite gemeinsame Erfahrungsbasis, auf die sie sich abstützen können.

Offenbar wurde dieses Defizit auch von anderen empfunden, denn schon seit einiger Zeit gibt es einen interessanten Ansatz, der gezielt die Vermittlung von Erfahrungen zu einem Ausschnitt von agilen Methoden zum Ziel hat: die Extreme Hour [Mer 04] von Peter Merel (wir nennen sie Agile Hour, weil es uns nicht nur um XP geht). Das Prinzip lautet: Es werden zwei Iterationen eines Konstruktionsprojekts durchgespielt. Insgesamt

werden sieben Schritte unterschieden: (1) Erkundung durch Prototypen („Spikes“ genannt), (2) Schreiben von Story Cards, (3) Aufwandsschätzung, Priorisierung, (4) Auswahl und Implementierung. Schritte 2-4 (eine Iteration) werden dann noch einmal ausgeführt. In den meisten Schritten passieren noch einige weitere Aktivitäten (z.B. Refactoring), aber für diese Diskussion ist das ohne Belang. Wichtig ist nun:

- Jeder der sieben Schritte dauert genau 10 Minuten (die Agile „Hour“ also 70 Minuten).
- Das konstruierte Produkt wird nicht programmiert, sondern gezeichnet.
- Die Entwickler arbeiten paarweise (“pair painting”) und haben nur einen Stift pro Paar.

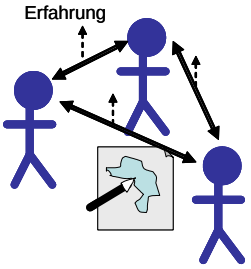

Abb. 2.: Agile Hour: Erfahrung erwächst aus der Interaktion, obwohl nur gemalt wird

Es gibt noch einige weitere Regelungen, die die Agile Hour einem agilen Software-Projekt ähnlicher machen sollen. Was bewirkt eine Agile Hour nun aber in Bezug auf *Erfahrungen*? Wir haben Gruppenübungen zum Thema agile Methoden in Form von Agile Hours durchgeführt. Jede Gruppe umfasste etwa 10 Studierende. Die Übungsbetreuer haben die Agile Hours als Coaches betreut, waren also dabei. Anschließend wurde reflektiert und wir haben mit einem anonymen Fragebogen die Stimmung abgefragt. In den meisten Agile Hours kam es zu Zeitnot und Hektik, mitunter bereitete das Malen in Paaren Probleme, fast immer auch die Integration der Teilergebnisse aller Paare. Die Rückmeldungen sprechen eine klare Sprache: Die Agile Hour vermittelt den Teilnehmern praktische Kenntnisse (so haben viele erst hier das Wechselspiel zwischen Kunden und Entwicklern im *Planungsspiel* richtig durchschaut). Alle fühlen sich *gefühlsmäßig* stark angesprochen, durchaus nicht immer nur mit positiven Gefühlen. Und in der Reflexion haben die Teilnehmer leicht Zusammenhänge zwischen ihrem Verhalten und den Konsequenzen erkannt. Alle Aspekte von Erfahrungen sind also intensiv vermittelt worden. Und das, obwohl gar keine Software entwickelt wurde! Hier ähnelt die Agile Hour dem SESAM-Ansatz: In beiden Fällen ist nur ein Kern authentisch, der Rest hat nicht viel mit einem realen Software-Projekt zu tun. Wir hatten 48 Teilnehmer in den Agile Hours; 31 haben per Fragebogen geantwortet. Alle 31 haben angegeben, die Agile Hour habe ihnen Spaß gemacht. Das reichhaltige mündliche Feedback direkt

nach den Agile Hours half uns, die Intensität der Erfahrungen in so kurzer Zeit als einen Grund dafür zu identifizieren.

4 Gezielte Eingriffe: SWP04

Der dritte vorzustellende Ansatz klingt sicher besonders vertraut. Ähnliche Versuche hat es schon viele gegeben. Im letzen Wintersemester haben 46 Studierende Software-Projekte („SWP04“) in neun Gruppen zu je 5–6 Teilnehmern bei uns durchgeführt. Verschiedene Aufgabenstellungen wurden meist an mehrere Gruppen parallel vergeben. Bei etwas genauerem Hinsehen haben solche universitären Projekte oft nicht viel mit Industrieprojekten zu tun. Zu groß sind die Unterschiede in den Rahmenbedingungen, in der Motivation und in vielen weiteren Aspekten. Wir hielten es für unrealistisch, alle diese Unterschiede überbrücken zu wollen. Wir haben daher entschieden, die meisten Unterschiede zu akzeptieren – aber an einigen wenigen Stellen gezielt zu intervenieren. Die interessantesten waren:

- Strikte ***Rollentrennung*** zwischen Kunde, Entwicklern und der „Firma der Entwickler“. Wir haben am Lehrstuhl die Rollen verteilt und darauf geachtet, sie nicht zu vermischen. Die (fiktive) Entwicklungsfirma war mit Qualitätssicherern bestrebt, die Qualität ausgelieferter (Teil-)Produkte hoch zu halten. Diese Rollenteilung ist an Hochschulen oft unüblich. Unserer Ansicht nach ist sie aber für viele typische Projekteffekte unabdingbar. Und erfreulicherweise bereitet die Rollenteilung auch für Betreuer kaum zusätzlichen Aufwand.
- ***Quality Gates*** zur groben Qualitätssicherung. Das ist eine Variante von eher formalistischen Prüfungen an festgelegten Stellen im zuvor definierten Prozess – eine Art verallgemeinerte Meilensteine für das Multiprojektmanagement. Sie laufen ähnlich wie Review-Sitzungen ab und dienen vor allem dazu, festgelegte Mindeststandards an festgelegten Prozessstellen quer über alle Projekte sicherzustellen. Wir haben durch drei Quality Gates einen Qualitätsmindeststandard in allen abgelieferten Produkten erreicht. Quality Gates sind in der Industrie weit verbreitet. Sie sind eine pragmatische Art und Weise, mit vielen parallel laufenden Projekten umzugehen, von denen man sich nicht alle Details ansehen kann.
- ***Zeitgutscheine***. Die Zeit der Kunden ist knapp und wertvoll. Dazu gibt es an der Hochschule häufig keine Entsprechung. Indem wir jedem Projekt nur sechs mal 15 Minuten für die Kommunikation mit dem Kunden zusprachen, fühlten sich alle Gruppen gefordert, diese Zeit optimal zu nutzen. Letztlich hat nur eine Gruppe überhaupt alle sechs Gutscheine verbraucht. Fast alle anderen haben sogar noch Prototypen mit dem Kunden diskutiert – in weniger als insgesamt 90 Minuten pro Gruppe, während des ganzen Semesters.

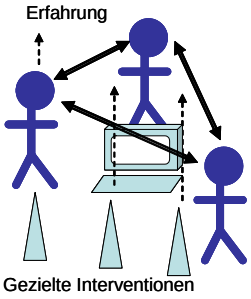

Abb. 3.: SWP04: Erfahrung wird durch gezielte Interventionen provoziert

Wieder haben wir durch Befragung, gezielte Erfahrungserfassung [Sch 00] und durch Fragebögen die Sicht der Studierenden abgefragt. Besonders bei den drei genannten Maßnahmen haben die Studierenden deutlich „Gefühle gezeigt“ und auch geäußert, hier hätten sie unerwartete Erfahrungen gemacht. Aber auch andere Entwicklungstätigkeiten, in die wir nicht eingegriffen haben, wurden als besonders realitätsnah eingestuft. Die Durchführung der Projekte und die dabei verfolgten Zielsetzungen sind in [Lüb 04] detaillierter beschrieben.

5 Erfahrungen per Internet vermitteln: Werkzeuge

Der vierte Ansatz unterscheidet sich stark von den oben beschriebenen. Hier geht es nicht darum, neue Erfahrungen zu machen, sondern bestehende Erfahrungen an andere (Studierende) weiterzugeben. Wir haben die Technik LIDs [Sch 00] verwendet, um mit wenig Aufwand den Studentenprojekten Erfahrungen zu entlocken. Die Erfahrungen sollen nun künftigen Projektgruppen zugute kommen. Dazu werden sie in ein Internet-Werkzeug gesteckt, das in einer Bachelor-Arbeit entwickelt wurde [Lir 04]. Besonders interessant ist, dass Leser die dort stehenden „Erfahrungsberichte“ im Stil von Amazon oder eBay bewerten können. So erhalten Autoren und künftige Leser eine schnelle Einschätzung über den Nutzen eines solchen Berichts, der oft „Erfahrungspaket“ heißt.

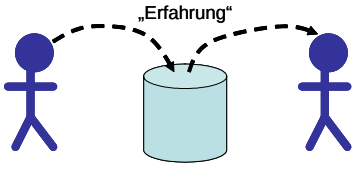

Abb. 4.: Erfahrungsweitergabe mit Werkzeug ist schwieriger, als man denkt

Derlei Werkzeuge werden gerne entwickelt, wo Erfahrungen vermittelt werden sollen. Meiner Überzeugung nach sind zwar Werkzeuge für die Verteilung nötig – aber sie stellen das kleinste Problem dar. Es gelingt nur mit großer Mühe, den Erfahrungscharakter (eigene Beobachtung? Gefühl?) über ein schriftliches Erfahrungspaket zu transportieren – egal wie elegant das Werkzeug dafür ist. Diese Ansicht wird von vielen anderen geteilt, die Erfahrungsvermittlung nicht hauptsächlich als Werkzeugproblem sehen, z.B. [Bar 99]. In [Sch 02] habe ich eine ganze Reihe von Missverständnissen und Hürden beschrieben, die der Wieder-Nutzung von Erfahrungen im Wege stehen. Werkzeuge helfen dabei kaum. Dies deckt sich mit Erkenntnissen bei DaimlerChrysler und hat wiederum Konsequenzen auf die Gestaltung *dennoch* nützlicher Werkzeuge [Sch 01]. Wir sehen Werkzeuge nur als einen eher kleinen Mosaikstein im größeren Zusammenhang der systematischen Erfahrungsnutzung. Sie werden hier hauptsächlich erwähnt, um auf diese Tatsache hinzuweisen.

6 Diskussion und Schlussfolgerungen

In diesem Beitrag habe ich vier unterschiedliche Verfahren vorgestellt, um im Rahmen des Informatik-Studiums gezielt und ausdrücklich Erfahrungen (nicht Wissen) zu vermitteln. Während SESAM, die Agile Hour und das Software-Projekt mit gezielten Interventionen auch dazu beitragen, Erfahrungen aufzubauen, erheben wir mit dem Werkzeug diesen Anspruch nicht: Hier geht es nur um die Verteilung von bestehenden Erfahrungspaketen. Da Erfahrung nach der obigen Definition aber eigene Beobachtung, Gefühle und Hypothesen einschließt, kann das Werkzeug kaum Mehrwert stiften, alles hängt von den Erfahrungspaketen selbst ab. In Tabelle 1 ist gegenübergestellt, ob und wie die vier Ansätze die drei Definitionsbestandteile von Erfahrungen an die Teilnehmer vermitteln.

Tab. 1: Vergleich der vier Verfahren: Wodurch werden die Erfahrungsaspekte umgesetzt?

Erfahrungsaspekt	**SESAM**	**Agile Hour**	**SWP04**	**Werkzeug**
Eigene Beobachtung	Reaktion des simulierten Projekts	Verhalten der anderen Teilnehmer, Zeitknappheit	Besonders bewusst bei gezielten Interventionen	(entfällt)
Gefühle	Je schneller und glatter die Simulation läuft, umso weniger	Wenn Hektik entsteht, schlagen auch Gefühle hoch	Zum Teil sehr stark, da echte Konse-quenzen drohen (z.B. kein Schein)	(aus zweiter Hand) Bewertung ist auch gefühlsbeladen
Schlussfolgerungen	Kaum d. SESAM selbst, erfordert Nachbesprechung	Nachbesprechung wichtig; sprudeln dann	Entstehen besonders um Interventionen	Bildet sich aus Inhalt und Bewertungen

Freilich reichen unsere Rückmeldungen und Fragebögen nicht aus, um die hier präsentierten Schlussfolgerungen statistisch zu belegen; aber dieses Material hilft uns doch immerhin, deutlich über das Niveau persönlicher Vermutungen hinauszukommen.

Der direkte Vergleich zeigt, dass mit dem Werkzeug eine andere Art von Erfahrungsvermittlung gemeint ist, nämlich die technische Speicherung und der Vertrieb. Es zeigt sich auch, dass Nachbesprechungen für die drei anderen Varianten sehr wichtig sind, was zwar nicht überrascht, aber auch nicht selbstverständlich ist [Lew 01]. Besonders eine reine Simulation wie SESAM erfordert unbedingt eine Nachbesprechung, damit sich nicht falsche Schlüsse festsetzen; schon vor zehn Jahren waren viele Teilnehmer schnell bereit, unerwünschtes Projektverhalten der angeblich „fehlerhaften" Simulation zuzuschreiben.

SWP04 als der „üblichste" Ansatz hat uns gezeigt, dass eine kleine Zahl sehr deutlicher Interventionen zu bewusst wahrgenommenen Erfahrungen im vollen Sinn der Definition führen kann. Damit hat man zwar nur wenige Aspekte abgedeckt, diese aber wirksam vermittelt. Wir finden diesen Ansatz sehr vielversprechend, zumal er auch noch den Aufwand in Grenzen hält. Wir werden nun mit anderen Interventionen experimentieren als beim ersten Durchgang.

Das Prinzip der Agile Hour zur Erfahrungsvermittlung scheint zu funktionieren. Gewiss ist auch der Neuigkeitswert für die positive Reaktion mitverantwortlich. Im nächsten Schritt denken wir über ähnliche Anwendungen (nicht nur agile Prozesse) nach, bei denen Erfahrungen konzentriert erzeugt und kompakt vermittelt werden sollen. Die Agile Hour ist geradezu dafür optimiert, in kürzester Zeit Gefühle wachzurufen und echte Erfahrung möglich zu machen. So einen Ansatz kann man freilich nicht ständig einsetzen, weil das Abrufen von Gefühlen auch anstrengend ist und sich nicht beliebig häufig wiederholen lässt.

In Hannover werden wir weiter gezielt nach Techniken forschen, die Erfahrungen erzeugen, sie durch Speicherung und Verteilung „frisch halten" und für die Nutzung bereit machen. Für Anregungen und Gedankenaustausch auf diesem Weg sind wir dankbar.

Literatur

[ASE 04] Abteilung Software Engineering: SESAM-Projektseite: *http://www.iste.uni-stuttgart.de /se/research/sesam/index.html.*

[Bar 99] Bartsch-Spörl, B.: Transfer of Experience – Issues Beyond Tool Building. Workshop on Learning Software Organizations, Kaiserslautern, Germany.

[Bec 00] Beck, K.: Extreme Programming Explained, Addison-Wesley.

[Boe 03] Boehm, B. and R. Turner: Balancing Agility and Discipline – A Guide for the Perplexed, Addson-Wesley.

[Con 00] Conradi, R. and T. Dingsoyr: Software Experience Bases: A Consolidated Evaluation and Status Report. Profes, Oulu, Finland.

[Dei 94] Deininger, M. and K. Schneider: Teaching Software Project Management by Simulation – Experiences with a Comprehensive Model. Conference on Software Engineering Education (CSEE), Austin, Texas.

[Din 00] Dingsoyr, T.: An Evaluation of Research on Experience Factory. Workshop on Learning Software Organizations, Oulu, Finland.

[Dra 95] Drappa, A., M. Deininger, et al.: "Forschungsprojekt SESAM." Informatik Forschung und Entwicklung. 10 (1995)(1): 49-50.

[Joh 99] Johannson, C., P. Hall, et al.: Talk to Paula and Peter – They are Experienced. International Conference on Software Engineering and Knowledge Engineering (SEKE´99). Workshop on Learning Software Organizations, Kaiserslautern, Germany, Springer.

[Lan 99] Landes, D., K. Schneider, et al.: "Organizational Learning and Experience Documentation in Industrial Software Projects." International Journal on Human-Computer Studies (IJHCS) 51(Organizational Memories).

[Lew 01] Lewerentz, C. and H. Rust: Die Rolle der Reflexion in Softwarepraktika. Software Engineering im Unterricht der Hochschulen (SEUH 2001), Zürich, dpunkt.verlag.

[Lir 04] Liro, T.: Konzept und Realisierung eines Werkzeugs zur Erhebung, Verwaltung und Bewertung von Erfahrungen. Fachgebiet Software Engineering. Hannover, Universität Hannover**:** 40.

[Lüb 04] Lübke, D., T. Flohr, et al.: Serious Insights through Fun Software-Projects. EuroSPI 2004: European Software Process Improvement Conference, Trondheim, Norway, Springer.

[Mer 04] Merel, P.: Extreme Hour: *http://c2.com/cgi/wiki?ExtremeHour*.

[Rod 04] Rodriguez, D., M. Satpathy, et al.: Effective software project management education through simulation models. An externally replicated experiment. International Conference on Product Focused Software Process Improvement (PROFES), Kansai Science City, Japan, Bomarius, F.

[Sch 94] Schneider, K.: Ausführbare Modelle der Software-Entwicklung. Struktur und Realisierung eines Simulationssystemes. Zürich, vdf.

[Sch 00] Schneider, K.: LIDs: A Light-Weight Approach to Experience Elicitation and Reuse. Product Focused Software Process Improvement (PROFES 2000), Oulo, Finland, Springer.

[Sch 01] Schneider, K. and T. Schwinn: "Maturing Experience Base Concepts at DaimlerChrysler." Software Process Improvement and Practice 6: 85-96.

[Sch 02] Schneider, K.: "What to Expect from Software Experience Exploitation." Journal of Universal Computer Science (J.UCS). *www.jucs.org,* 8(6): 44-54.

Teamtraining für Software-Ingenieure

Andreas Fleischmann, Katharina Spies
Institut für Informatik, Technische Universität München
Boltzmannstraße 3, 85748 Garching b. München
{fleischa, spiesk}@in.tum.de

Katharina Neumeyer
Institut für Psychologie, Technische Universität Darmstadt
Hochschulstraße 1, 64289 Darmstadt
training@neumeyer-kroeger.de

Zusammenfassung

Die Notwendigkeit, für die Ausbildung in Software Engineering passende Unterrichtsformen zu entwickeln, die es den Studierenden ermöglichen, praktische Erfahrungen zu sammeln, ist eine mittlerweile anerkannte Herausforderung bei der aktuellen Konzeption der Lehre von Software Engineering an der Hochschule. Um den Studierenden solche Erfahrungen mitzugeben, werden an vielen Hochschulen Projekte durchgeführt, in denen studentische Teams weitgehend selbstständig einen Software-Engineering-Prozess durchleben. Aufgrund unserer Erfahrungen bei der Durchführung solcher Projekte an den TU's Darmstadt und München ist festzustellen, dass die größten Herausforderungen für die beteiligten Studierenden dabei weniger in der Bewältigung technischer oder fachlicher Fragestellungen liegen: Diese Problematik ist ihnen aufgrund ihrer Lernerfahrung im Studienverlauf bereits geläufig. Vielmehr stellt der Umgang mit organisatorischen und sozialen Problemen eine viel größere Herausforderung dar und führt daher oft zu für die Studierenden unvorhergesehenen Problemen im Projekt.

In diesem Artikel stellen wir unsere Erfahrungen mit einem vorbereitenden Teamtraining vor, das wir vor diesem Hintergrund als Vorbereitung zu studentischen Software-Projekten durchgeführt haben. Die Zielsetzung dieses dreitägigen Blockseminars besteht darin, die Studierenden als Team zusammenfinden zu lassen und ihnen bereits vor Beginn des Projekts grundlegende Techniken für die Zusammenarbeit im Team sowie eine Einführung in Projektmanagement mitzugeben.

1 Einführung

Die Wichtigkeit der praktischen Durchführung von Software-Projekten in der universitären Ausbildung zukünftiger Software-Ingenieure ist mittlerweile erkannt. Dementsprechend werden solche Projekte vielfach durchgeführt und sind bereits an vielen Universitäten fest im Curriculum verankert [Fle 02], beispielsweise an der TU Darmstadt als einjähriges „Software-Projekt“, an der TU München als einsemestriges „Softwaretechnik-Praktikum“ oder an der Uni Stuttgart gleich zweimal als „Softwarepraktikum“ und als „Studienprojekt“.

Solche Projekte zeichnen sich dadurch aus, dass studentische Teams in einem möglichst praxisnahen Setting über einen größeren Zeitraum hinweg weitgehend selbstständig einen Software-Engineering-Prozess durchlaufen, von der Anforderungsanalyse bis hin zur Abgabe eines Produkts. In München beispielsweise treten in dem Bemühen, eine möglichst praxisnahe Projektumgebung zu schaffen, die Veranstalter wie „echte“ Kunden auf. Hierzu gehören auch die Abgabe unvollständiger Aufgabenbeschreibungen, das Setzen von Deadlines und die Forderung nach zusätzlichen Funktionalitäten im fortgeschrittenen Verlauf des Projekts. Durch die Bündelung solcher Herausforderungen und die Größe der Aufgabe entsteht eine vergleichsweise realitätsnahe Situation (vgl. [Tau 01], [Gna 03]), in der die Studierenden anschaulich die Vielseitigkeit des Berufsbildes eines Software-Ingenieurs erleben: Verhandeln, Diskutieren, Organisieren, Design, Implementieren, Testen, Dokumentieren, Präsentieren. Folglich müssen die Studierenden neben technischen und fachlichen Herausforderungen auch organisatorische und soziale Probleme meistern. Beispiele für diese unterschiedlichen Probleme sind:

- Ein technisches Problem ist beispielsweise: „Wie spielen die Entwicklungsumgebung Eclipse und die Versionsverwaltung CVS zusammen?“
- Ein fachliches Problem ist beispielsweise: „Soll das Programm in der Situation X mit der Aktion A oder B reagieren?“
- Ein organisatorisches Problem ist beispielsweise: „Ist es möglich, die anstehenden Aufgaben im Team sinnvoll aufzuteilen?“
- Ein soziales Problem ist beispielsweise: „Wie gehen wir als Team damit um, dass Person X die ihm zugewiesene Teilaufgabe unbefriedigend gelöst hat?“

Die Durchführung und Betreuung solcher Projekte an den TU’s Darmstadt und München hat gezeigt, dass die größten Herausforderungen für die beteiligten Studierenden dabei in der Bewältigung organisatorischer/sozialer Probleme liegen. Die Ursache hierfür sollte darin zu finden sein, dass einerseits das benötigte Informatik-Fachwissen in den Vorlesungen bereits hinreichend gut vermittelt wurde, und andererseits darin, dass es den Studierenden aufgrund ihrer eigenen Lernerfahrung deutlich leichter fällt, fachliche Defizite zu überwinden.

Aus unserer Einsicht heraus, dass Studierende früherer Projekte im Rückblick die sozialen und organisatorischen Probleme oft als entscheidend empfanden und während des Projekts wenig Zeit und Bereitschaft besteht, auf dieses „unbequeme Thema“ in Ruhe einzugehen, wurde an der TU München ein Teamtraining konzipiert, das die Studierenden vor Beginn des Projekts absolvieren, und das sie auf die Zusammenarbeit

im Projekt-Team vorbereitet. Damit werden die Studierenden bereits vor dem Start des fachlichen Projekts ohne Termindruck auf den Umgang mit den mit Sicherheit auftretenden organisatorischen und sozialen Problemen vorbereitet, und es werden ihnen Wissen und Techniken für den Umgang und die Bewältigung dieser Probleme an die Hand gegeben.

Im vorliegenden Artikel skizzieren wir zunächst kurz den organisatorischen und inhaltlichen Rahmen eines vor kurzem abgeschlossenen Softwaretechnik-Praktikums an der TU München (Kapitel 2). Danach stellen wir Konzeption, Lerninhalte und Ablauf des Teamtrainings vor (Kapitel 3). Abschließend präsentieren wir unsere bisherigen Erfahrungen mit dem Teamtraining, geben einen Ausblick auf die weitere Entwicklung dieses Trainingskonzepts (Kapitel 4) und schließen mit einem kurzen Fazit (Kapitel 5).

2 Das Softwaretechnik-Praktikum „AutoRAID“

Abb. 1.: Das studentische Team von „AutoRAID“

Am Lehrstuhl Broy werden seit einigen Jahren regelmäßig Softwaretechnik-Praktika (STP) angeboten, in denen Studierende der Informatik im Hauptstudium ein Semester lang selbstständig ein Software-Produkt entwickeln. Aufgabe des STP im Sommersemester 2004 war die Entwicklung von „AutoRAID“, einem prototypischen Werkzeug zur modellbasierten Anforderungsanalyse für das Requirements Engineering eingebetteter Systeme [Aut 04]. Mit AutoRAID sollen verschiedene Forschungsergebnisse des Lehrstuhls praktisch demonstriert werden, beispielsweise die Erfassung von Requirements durch direkte Eingabe oder Import aus bestehenden Dokumenten, die Klassifikation der Requirements in mehreren Dimensionen und deren anschauliche Darstellung in einem Werkzeug, automatische Konsistenzanalyse, und das in Beziehung Setzen von Requirements untereinander und zu Design-Elementen (Tracing), inklusive der automatischen Erzeugung von Designelementen.

Das Projekt wurde von wissenschaftlichen Mitarbeitern und externen Firmenvertretern betreut, zum Teil in der Rolle der Kunden, mit denen die Studierenden die Aufgabenstellung verhandelten, zum Teil in der Rolle von Betreuern, an die sich die Studierenden ratsuchend wenden konnten.

Bereits die Ausschreibung der Aufgabenstellung führte zu einer ersten Herausforderung für die Studierenden: Sie mussten sich für die Teilnahme am Praktikum formal bewerben. Bei entsprechender Eignung wurden sie zu einem Auswahlgespräch geladen, das die Grundlage für die Auswahl der Teilnehmer darstellte. Kriterien für die Auswahl von den Studierenden waren dabei neben fachlichen und technischen Kenntnissen, beispielsweise Programmiererfahrung, insbesondere auch die Bereitschaft, ein überdurchschnittlich hohes Arbeitspensum zu bewältigen und in einem selbstorganisierten Team zu arbeiten.

Um den ausgewählten zehn Studierenden das in der Einleitung beschriebene Fundament zur Bewältigung der organisatorischen und sozialen Problematik an die Hand zu geben und sie vor allem für diese Art der Probleme zu sensibilisieren, fand das Teamtraining noch vor Beginn des Praktikums statt, am ersten Semesterwochenende vom 23. bis 25. April 2004. Das Softwaretechnik-Praktikum begann einige Tage später mit einem Kick-off am 27. April 2004; an diesem Kick-off lernten die Studierenden ihre Betreuer und „Kunden“ kennen, bekamen die Aufgabenstellung in Form eines Lastenhefts und einen groben Projektplan, der drei Phasen vorsah: Analyse, Designs und Implementierung.

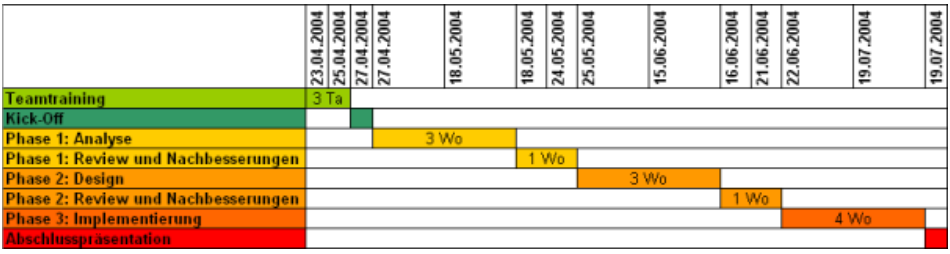

Abb. 2.: Projektplan für das Projekt „AutoRAID“

Abbildung 2 zeigt den Projektplan, der den Studierenden vorgeschlagen wurde. Die Studierenden orientierten sich an diesem Projektplan und schafften es, ihn einzuhalten, allerdings mussten sie nach der Abschlusspräsentation noch drei Wochen länger arbeiten, um die Dokumentation nachzureichen.

Das Team erstellte innerhalb des vorgegebenen Zeitrahmens das fertige Requirements-Engineering-Werkzeug „AutoRAID“ [Aut 04]. Das Werkzeug ist integriert in ein am Lehrstuhl entwickeltes Design- und Simulationswerkzeug „AutoFOCUS“ [Aut 02], das am Lehrstuhl verwendet und beständig weiterentwickelt wird; eine Fallstudie mit AutoRAID in der Industrie ist für Anfang 2005 geplant.

3 Konzeption, Lerninhalte und Ablauf des Teamtrainings

In diesem Abschnitt stellen wir die Konzeption, die Lerninhalte und den Ablauf des Teamtrainings vor.

3.1 Konzeption

Unsere Zielsetzung bei der Konzeption des Teamtrainings bestand vor allem darin, die Bewältigung der tatsächlichen fachlichen Aufgabe (Entwicklung von AutoRAID) mit dem vorgegebenen Projektplan von Beginn an zielgerichtet zu ermöglichen. Die Studierenden sollten mit Beginn der fachlichen Aufgabe bereits als Team agieren und über die nötige Zusatzqualifikation speziell zur gemeinsamen zielgerichteten Kooperation im Team verfügen.

Das von uns in München durchgeführte Teamtraining basiert auf Konzepten, die seit einigen Jahren von der Hochschuldidaktischen Arbeitsstelle der TU Darmstadt angewandt werden:

- Team-Coaching von Software-Projekten am Institut für Informatik der TU Darmstadt [Hen 02].
- Interdisziplinäres Teamtraining an der TU Darmstadt. Einzelpersonen, die sich vor dem Workshop nicht kannten und die nach dem Workshop nicht als Team zusammenarbeiten müssen, bilden während des dreitägigen Workshops zu Trainingszwecken ein „Ad-hoc“-Team [Kom 04].

Für München wurde das Darmstädter interdisziplinäre Teamtraining für „Ad-hoc“-Teams überführt in ein Teamtraining für „echte“ Teams aus Studierenden, die an dem Softwaretechnik-Praktikum teilnehmen werden:

Da es sich nicht um ein „Ad-hoc“-Team handelte, sondern um eine Gruppe von Studierenden, die sich im Anschluss an das Training als Team bewähren musste, musste das Training um spezielle Teambuilding-Aspekte ergänzt werden. Der bereits existierende interdisziplinäre Projektmanagement-Anteil wurde auf den spezifischen Kontext des Praktikums zugeschnitten. Im Gegensatz zu Darmstadt war das Teamtraining nicht optional, sondern als verpflichtender Baustein für alle Teilnehmer am STP vorgeschrieben und wurde mit einem eigenen Seminarschein anerkannt.

Im Jahr 2003 wurde dieses Teamtraining von Katharina Neumeyer und Andreas Fleischmann konzipiert, erstmalig durchgeführt und von Katharina Spies evaluiert. Für AutoRAID wurde das Training dann im Sommersemester 2004 zum zweiten Mal in der hier beschriebenen Form von Andreas Fleischmann durchgeführt.

3.2 Theoretische Grundlage

Unsere Erfahrungen mit Teamarbeit in studentischen Projekten lassen sich mit der „Teamkurve“ (Abbildung 3) vereinfachend zusammenfassen.

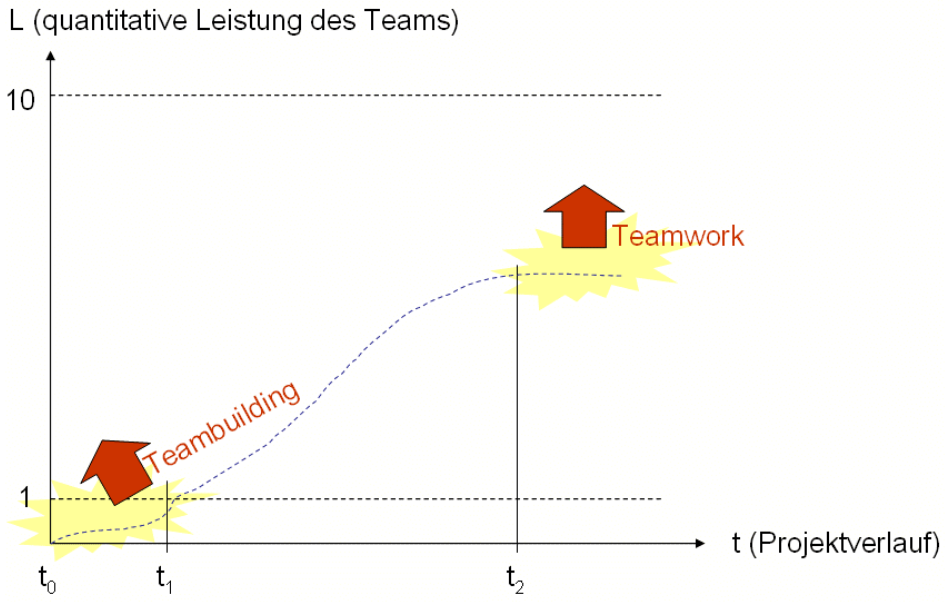

Abb. 3.: Die „Teamkurve“

Während man naiverweise erwarten könnte, dass die zehnfache Personenzahl in einem Team quantitativ zumindest annähernd die zehnfache Leistung erbringt, ist es in der Praxis oft so, dass in einer Anfangsphase (t_0 bis t_1) zehn Personen zusammen weniger leisten als eine Person alleine. In dieser Phase der Teamfindung bilden sich Rollen und Infrastrukturen im Team [Lan 00]. Auch wenn diese Anfangshürde überwunden ist, wird ein Team nie die volle Leistung erreichen (ab t_2), da auch im perfekten Team immer Abstimmung und Koordination nötig ist.

Bei diesen beiden Problempunkten setzen wir im Teamtraining an: Teambuilding soll die unproduktive Anfangsphase der Teamfindung (bis t_1) verkürzen und Frustration vermeiden, und Teamwork-Techniken sollen die Produktivität in den anschließenden Phasen erhöhen (ab t_1, insbesondere ab t_2).

Unserer Erfahrung nach dämpft schon alleine das Wissen um die Teamkurve die (ansonsten zum Teil erhebliche) Anfangsfrustration (Zitat eines Studierenden: „Teamarbeit, das ist, wie wenn sich zehn Leute in einen VW Käfer quetschen, um gemeinsam einzuparken“) erheblich, weil sie nun wissen, dass diese unproduktive Anfangsphase, die sie alle durchlaufen, normal, notwendig und vorübergehend ist.

Auch halten wir es für notwendig, Teamarbeit explizit zu motivieren und gezielt die Vorteile der Teamarbeit herauszustellen, damit sie nicht in den Alltagsproblemen untergehen. Als Argumente für Teamarbeit nennen wir dabei vor allem die letztendlich höhere quantitative Leistung des Teams im Vergleich zu der einer einzelnen Person (in Abbildung 3 ist das ab t_1) und die höhere Qualität, die sich daraus ergibt, dass mehr

Leute mehr Ideen haben, aber auch, dass man sich gegenseitig inspiriert, Fehler in der Arbeit der anderen entdeckt usw.

3.3 Ein Team formen (Teambuilding)

Ein Lernziel des Teamtrainings ist es, aus den Studierenden, die sich noch nicht kennen, ein Team zu formen. Der Schwerpunkt des Workshops liegt dabei nicht im expliziten Teambuilding (z.B. Theorie, vielfältige Kennenlernübungen, Vertrauensübungen usw.), wir vertrauten stattdessen darauf, dass sich die Teammitglieder durch die gemeinsame Arbeit im Workshop (insbesondere im Miniprojekt, Abschnitt 3.6) „automatisch“ besser kennen und einschätzen lernen. Das explizite Teambuilding beschränkte sich daher auf folgende Elemente:

- Zu Beginn des Workshops machten wir eine ausführliche Kennenlernrunde, in der jeder Studierende sich den anderen anhand eines „Steckbriefes“ vorstellte (siehe Abbildung 4).

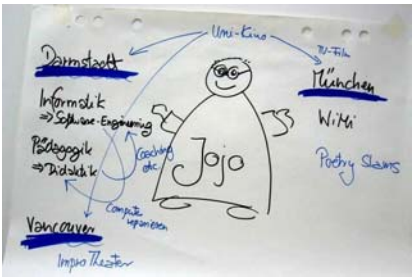

Abb. 4.: Ein typischer „Steckbrief“

- In Diskussionsrunden wurden die Studenten für die Unterschiedlichkeiten innerhalb des Teams und das damit verbundene Problempotenzial sensibilisiert Dazu diskutieren wir beispielsweise Fragen wie: „Euer Projekt steht unter großem Zeitdruck. Was ist dir wichtiger?“ a) „Die Deadline wird auf jeden Fall eingehalten, ggf. werden Abstriche bei der Qualität zugelassen.“ b) „Das Abliefern eines Qualitätsprodukts hat Vorrang, ggf. muss die Deadline überschritten werden.“
- Bei ganztägigen Blockveranstaltungen empfiehlt es sich, gerade nach der Mittagspause und am Nachmittag hin und wieder auflockernde Übungen zu machen; wir haben dabei Übungen eingesetzt, bei denen die Kooperation im Team im Vordergrund stand (z.B. Reise nach Solidarien [Sen 04], Deckendrehen [Sen 04], Stuhlskulptur [Henn 02]).

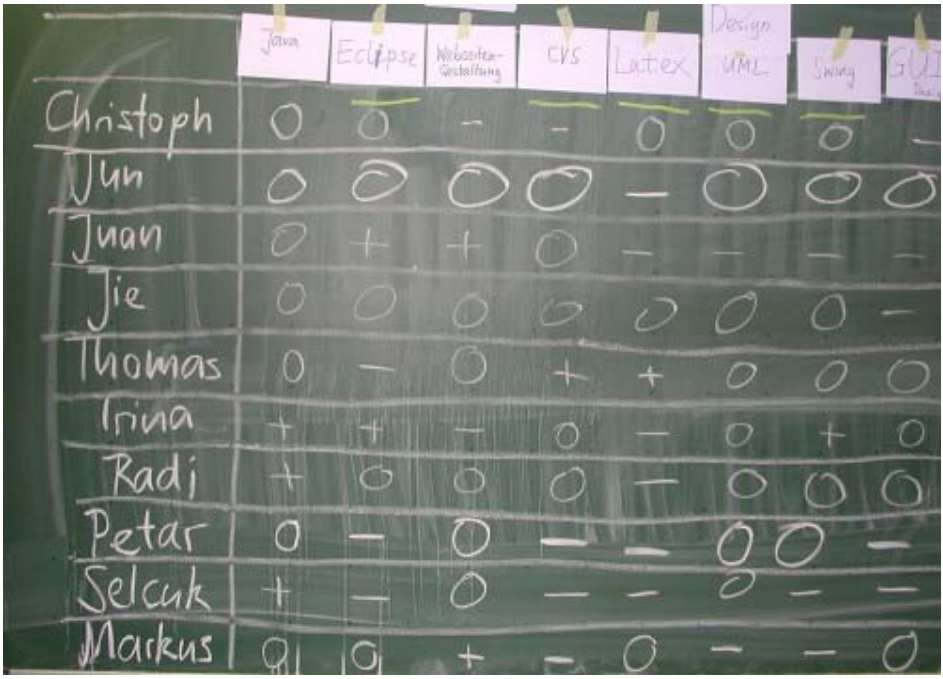

Abb. 5.: Zwischenergebnis auf dem Weg zum Teamprofil

Im Rahmen des Miniprojekts (siehe Abschnitt 3.6) lernten die Studierenden verschiedene Elemente einer Teaminfrastruktur kennen (z.B. Kommunikation über Mailinglisten, Jour fixe) und entschieden sich gemeinsam, welche sie einsetzen wollen; sie verschafften sich einen Übersicht über ihre Stundenpläne, um herauszufinden, wann sie in der Woche Zeit haben, um gemeinsam zu arbeiten.

Das Team diskutierte auch, welche Rollen man innerhalb des Teams vergeben will (Projektleiter, Verantwortliche für verschiedene Tools, Chefentwickler); sie verschafften sich eine Übersicht über die fachlichen Fähigkeiten und Interessen der einzelnen Teammitglieder, u.a. um einschätzen zu können, wer für welche Rolle am besten geeignet wäre, und einigten sich darauf, wer welche Rolle übernehmen würde.

3.4 Als Team zusammenarbeiten (Teamwork)

Im Vordergrund des Workshops stand die Vermittlung von Arbeitstechniken, die das gemeinsame Arbeiten unterstützen: zum einen als ganzes Team effizient und kooperativ zusammenzuarbeiten, zum anderen eine effektive Arbeitsteilung.

Dazu wurden den Studierenden verschiedene Techniken kurz vorgestellt, die sie dann in einer Übung selbst erprobten; anschließend wurden die Vor- und Nachteile der Technik reflektiert. Im Miniprojekt haben die Studierenden dann selbstständig Techniken ihrer Wahl eingesetzt und bekamen zur Wahl und Durchführung einer Technik Feedback von den Trainern.

Sitzungen strukturieren

Hier wurden folgende Grundlagen vermittelt:

- Für die Struktur und Durchführung eines Meetings sollte eine Person verantwortlich sein; diese Person bereitet das Meeting vor (Raum reservieren, Erinnerungsmail verschicken, Tagesordnungspunkte sammeln) und moderiert das Meeting.
- Zu Beginn des Meetings sollten gemeinsam die Tagesordnungspunkte gesammelt werden, die besprochen werden müssen; diese werden dann entsprechend ihrer Wichtigkeit und Dringlichkeit in eine Reihenfolge gebracht und der Reihe nach abgearbeitet.
- Verschiedene Arten der Ergebnissicherung; zum Beispiel Sofort-Protokoll am Laptop, das mit einem Beamer projiziert wird.

Arbeitstechniken

Hier wurden verschiedene Arbeitstechniken zum gemeinsamen Sammeln, Ordnen/Strukturieren und Bewerten/Entscheiden vorgestellt:

- Brainstorming und Kartentechniken (wie zum Beispiel in Abbildung 6 dargestellt).
- Regeln für das effektive und kooperative Diskutieren, zum Beispiel Herstellen einer gemeinsamen Ausgangsbasis, Zieldefinition und Strukturierung von Diskussionen, Ergebnissicherung. Abbildung 6 zeigt Tipps zum Diskutieren, die die Studierenden selbst formuliert haben, betreffend das Einzelverhalten („ich“), das Gruppenverhalten („Gruppe“) und die gesamte Organisation der Diskussion.

- Verschiedene Methoden, um gemeinsam Entscheidungen zu treffen, zum Beispiel Abstimmungen, Entscheidungsmatrizen oder Konsensdiskussion.

Uns war in diesem Zusammenhang besonders wichtig, nur solche Medien einzusetzen, die die Studierenden auch während des Praktikums einsetzen können; daher haben wir zum Beispiel auf Metaplanwände und Moderationskarten größtenteils verzichtet und stattdessen Wände, Beamer, Laptop und Flipcharts eingesetzt.

Arbeitsteilung

Etliche Aufgaben müssen nicht von allen Teammitgliedern gemeinsam bearbeitet werden, sondern können auf Teilgruppen verteilt werden. Dies erhöht die Effizienz des Teams erheblich – wenn einige Regeln beachtet werden, beispielsweise:

- Für jede Aufgabe sollte es eine verantwortliche Person und eine Deadline geben.
- Zu Beginn der Aufteilung sollte sichergestellt werden, dass die einzelnen Teilgruppen genau wissen, was sie bearbeiten sollen und welche Form die Ergebnisse haben sollen (z.B. Stichpunktliste, kurze Präsentation).
- Wird ein Dokument aufgeteilt, bietet es sich an, dass vorher eine Teilgruppe eine Vorlage erstellt (aus der die Formatierungen, Formulierungsstil, Umfang ersichtlich sind); diese Vorlage sollte gemeinsam von allen Teammitgliedern diskutiert und verabschiedet werden.
- Wird Sourcecode aufgeteilt, gibt es Richtlinien für das gemeinsame Nutzen eines CVS Repositories (zum Beispiel: regelmäßiges Einchecken; Update, bevor man eine Datei bearbeitet; nur Einchecken von compilierbaren Code).

3.5 Projektmanagement

In der Regel haben nicht alle Teilnehmer an dem Praktikum bereits eine Software-Engineering-Vorlesung gehört und daher keine rechte Vorstellung, wie man systematisch an ein größeres Programmierprojekt herangeht. Aus diesem Grund haben wir den Lernblock „Projektmanagement für Software-Entwicklung“ dem Teamtraining hinzugefügt.

Als Einstieg diskutierten wir mit den Studierenden anhand der beiden exemplarischen Vorgehensmodelle des Rational Unified Process (RUP) und des eXtreme Programmings (XP) die grundlegend gemeinsamen Elemente von solchen Vorgehensmodellen.

Im Anschluss an die Diskussion bekamen die Studierenden von uns ein begründetes verbindliches Vorgehensmodell vorgegeben, das angesichts des kurzen Zeitrahmens des Projekts relativ simpel ist: Es handelt sich im Wesentlichen um ein aufgebohrtes Wasserfall-Modell, das aus drei Phasen besteht: Analyse, Design und Implementierung.

- In der Analyse wird hauptsächlich das Benutzerhandbuch erstellt. Gleichzeitig sollen die Studierenden aber auch schon einen Oberflächenprotoyp und ein logisches Datenmodell entwerfen.
- Im Design wird schwerpunktmäßig die Systemarchitektur entworfen. Gleichzeitig sollen die Studierenden aber anhand eines funktionalen Prototyps wichtige fachliche und technische Aspekte ausprobieren.
- In der Implementierung werden das ausführbare Programm und die Dokumentation erstellt.

Zusätzlich zu diesem Vorgehensmodell, das ein Software-Projekt in einzelne definierte Phasen segmentiert, schlugen wir ein allgemeines Vorgehen zum Problemlösen auf „Mikro-Ebene“ vor:

- Zunächst Problemanalyse und Zieldefinition (Ergebnisorientierung: Es wird festgelegt, was „hinten rauskommen“ soll).
- Als Zweites wird die Meilensteinplanung (Zeitorientierung: Es wird eine Deadline für die Aufgabe bestimmt und die Zeit bis zur Deadline in Einzelschritte aufgeteilt; die Einzelschritte sind definiert durch die Teilergebnisse bzw. den Reifegrad eines Ergebnisses, das am Ende des Schrittes vorliegen soll).
- Danach je nach Situation (zum Beispiel wenn es um Strategien, gemeinsame Ideenfindung, Diskussionen oder Beschlüsse geht) gemeinsames Arbeiten im Team:
  - „Weite Phasen“: Sammeln von Ideen, zum Beispiel in einem Brainstorming.
  - „Enge Phasen“: Bewerten von Ideen, Aussortieren und Entscheiden.
- Oder paralleles Arbeiten in Kleingruppen (Arbeitsteilung).

Die vermittelten Arbeitstechniken und Problemlöseschritte haben die Studierenden im anschließenden Miniprojekt selbstständig ausgewählt und angewendet.

3.6 Miniprojekt

Etwa die Hälfte des Teamtrainings bestand aus einem „Miniprojekt“: In dieser Phase haben die Studierenden selbstständig gearbeitet, wurden dabei von den Trainern beobachtet und etwa alle 30 Minuten unterbrochen, um Feedback und Tipps zu geben. Ziele des Miniprojekts waren:

- Die Studierenden sollten die gelernten Techniken (siehe Abschnitt 3.4, Teamwork) selbstständig einsetzen und dadurch ihr Verständnis vertiefen; die Trainer sehen, wie die Studierenden die gelernten Techniken umsetzen, und haben die Gelegenheit, durch situationsbezogene Tipps zu korrigieren und zu ergänzen.
- Das Miniprojekt sollte den Übergang markieren vom “geschützten Raum” des Teamtrainings zum Praktikum. Es gab erste „echte“ Aufgaben, die von Relevanz für das Praktikum waren, z.B. Rollenverteilung (Entscheidung des Teams, wen sie zum Projektleiter bestimmen).

- Die Studierenden sollten sich durch die gemeinsame selbstständige Arbeit an den ersten „echten“ Aufgaben des Praktikums besser kennen und einschätzen lernen (siehe Abschnitt 3.3, Teambuilding).

Die Aufgabenstellung für das Miniprojekt lautete (in geraffter Form): *„Werdet euch klar über Stärken/Fähigkeiten/Wissen der einzelnen Teammitglieder! Bestimmt, wer von euch Projektleiter und wer Webmaster wird; überlegt euch, ob ihr noch weitere feste Rollen im Projekt festlegen wollt, und wer sie einnehmen soll!*

Ihr werdet mit Tools wie Java, Eclipse, CVS, ArgoUML/UML-Sequenz-Diagrammen, Latex und AutoFocus arbeiten müssen. Überlegt euch, wer sich bereits damit auskennt und wie ihr am besten erreichen könnt, dass sich alle damit auskennen!

Verschafft euch einen Überblick, wer wann in der Woche Zeit hat und was geeignete Termine für regelmäßige Treffen sind!

Ihr werdet beim Kick-off ein etwa 50seitiges Pflichtenheft mit der Aufgabenbeschreibung überreicht bekommen; ihr werdet einige Zeit benötigen, um es zu lesen, und dann etliche Fragen haben, die ihr mit den Kunden klären müsst. Erstellt einen ersten vorläufigen Zeitplan für die Woche nach dem Kick-off!”

Um diese Aufgaben zu bearbeiten, mussten die Studierenden selbst einen Zeitplan definieren; klären, wie sie die Aufgaben lösen wollen; Ergebnissicherung betreiben usw. Während die Studierenden selbstständig arbeiteten, wurden sie von den Trainern beobachtet und etwa alle 30 Minuten unterbrochen, um gemeinsam über den Arbeitsstil und die erreichten Ergebnisse zu reflektieren und Tipps zu geben.

Abb. 7: Schnappschuss von Studierenden während des Miniprojekts

3.7 Ablauf des Teamtrainings

Das Teamtraining fand als dreitägiges Blockseminar am ersten Semesterwochenende statt; Beginn war jeweils um 9 Uhr, Ende jeweils um 17 Uhr.

Der erste Tag begann mit einem ersten Kennenlernen der Teilnehmer; es wurde ein wenig Theorie (die „Teamkurve“) vermittelt. Thematischer Schwerpunkt waren die Strukturierung von Meetings, Arbeitstechniken für Teams und Umgang mit Kritik. Am Vormittag des zweiten Tags standen effektives und kooperatives Diskutieren sowie Projektmanagement im Vordergrund. Am Nachmittag begann das Miniprojekt. Am dritten Tag wurde das Miniprojekt fortgeführt und beendet.

Die Teilnehmer bekamen Hand-outs mit Checklisten zu den wichtigsten Themen sowie ein Fotoprotokoll des Teamtrainings, in dem alle Tafelanschriften etc. abfotografiert waren.

4 Beobachtungen und Perspektiven

Das Teamtraining wurde von den Studierenden sehr positiv aufgenommen – sowohl in der Evaluation direkt nach dem Training im April als auch rückblickend in der Evaluation am Ende des gesamten Projekts im August.

Zu Beginn des Projekts konnten wir beobachten, dass das studentische Team selbstbewusst und mit optimistischer Stimmung ins Kick-off ging. Während in den Jahren zuvor ohne Teamtraining die Studierenden anfangs schüchtern und desorganisiert waren, war das diesjährige Team sofort in der Lage (und fühlte sich auch entsprechend sicher vorbereitet), mit der fachlichen Arbeit mit Beginn des STP zu beginnen. Die Folge war, dass dadurch mindestens ein bis zwei Wochen „einspart“ wurden, die dem Team somit zusätzlich für die Bewältigung der fachlichen Aufgabe zur Verfügung standen – insofern war das Teambuilding erfolgreich.

In Bezug auf die Umsetzung der Arbeitstechniken zeigte die Beobachtung des Teams während des Projekts, dass viele gelernte Techniken nicht voll eingesetzt wurden. Dafür sehen wir vor allem drei Gründe:

Struktur

Die Studierenden hatten bei vielen Meetings das Empfinden, die gelernte Systematik (Tagesordnung, Sitzungsleitung, Protokoll) nicht einsetzen zu müssen, da es sich ja „nur“ um ein „normales“ bzw. „einfaches“ Meeting handele. Unsere Beobachtung ergab allerdings, dass das „Einsparen“ einiger Minuten Sitzungsorganisation am Beginn des Meetings die Besprechungen insgesamt deutlich ineffizienter gemacht hat.

Hier wollen wir beim nächsten Teamtraining noch stärker die Vorteile einer strukturierten Teamsitzung deutlich machen und versuchen, bereits im Teamtraining eine Routine zu entwickeln, in der Hoffnung, dass diese dann von den Studierenden quasi automatisch in die eigentliche Projektarbeit übernommen wird.

Führung

Der studentische Projektleiter hatte Schwierigkeiten, die mit seiner Rolle verbundene Führungsaufgabe wahrzunehmen, da er dies als unzumutbare Bevormundung seiner Mitstudierenden empfand. Sein „Laisser-faire“-Stil konnte allerdings das Bedürfnis der Studierenden nach Führung und Fokussierung nicht ausreichend befriedigen.

Hier wollen wir beim nächsten Teamtraining in der gesamten Gruppe transparenter machen, was genau die Führungsaufgaben eines (studentischen) Projektleiters sind, und dass sich Basisdemokratie und Führung nicht notwendigerweise widersprechen. Zudem überlegen wir, beim nächsten Praktikum der Gruppe die Wahl zu lassen, ob sie eine Projektleitung möchten.

Reflektion

Der Zeitdruck im Projekt war von Anfang an sehr hoch, so dass sich das Team sehr schnell kaum mehr des Lernumfelds bewusst war; vielmehr stand nur noch der erfolgreiche Abschluss des Projekts im Vordergrund. Zeit für Reflektion über Vorgehen und Techniken des Software Engineering blieb kaum.

Zurzeit überlegen wir daher, ob wir im nächsten Projekt versuchen sollen, solche Reflektionsphasen zu erzwingen, auch wenn wir nicht sicher sind, wie sich das auf den „Fluss“ des Projekts auswirken wird. Alternativ ist angedacht, jeden Studierenden am Ende des Projekts einen individuellen Projektbericht schreiben zu lassen, in dem er informell den Verlauf des Projekts im Rückblick beschreibt. Eine weitere Idee wäre, Absolventen eines solchen Projekts im darauf folgenden Jahr einen Hiwi-Job als Betreuer eines nächsten Projekts anzubieten. In der Distanz als Betreuer und im nochmaligen Erleben eines Projekts liegt ein großes Lernpotenzial.

5 Fazit

Zusammenfassend stellen wir fest, dass, auch wenn noch viele Detailverbesserungen denkbar sind, ein Teamtraining ein Software-Engineering-Projekt in der Lehre signifikant unterstützen kann: Die Studenten können durch das Teambuilding viel schneller in die eigentliche fachliche Projektarbeit einsteigen, sie arbeiten im Team deutlich effizienter und strukturierter, sie können souveräner mit sozialen und organisatorischen Herausforderungen umgehen.

Ein Teamtraining unterstützt folglich den Lernerfolg eines solchen Projekts zum einen dadurch, dass während des Projekts weniger die sozialen Probleme dominieren, sondern stattdessen mehr Raum für das Lernen und Ausprobieren „klassischer“ fachlicher Software-Engineering-Inhalte ist.

Zum anderen wird der Lernerfolg durch eine explizite Behandlung von sozialen und organisatorischen Themen unterstützt, die im Praxisalltag zwar wichtig sind, aber als „unbequem“ empfunden und gerne verdrängt werden. Allerdings plädieren wir dafür, neben den fachlichen und technischen Inhalten verstärkt auch „Soft Skills“ wie Teamwork und Projektmanagement als *gleichberechtigt wichtige* Lernziele innerhalb des Software Engineering wahrzunehmen – oder, wie Prof. Ludewig etwas plakativer

zum Thema Soft Skills sagte: „Es gibt keinen Gott der Informatik, der das Gebot verkündet hätte: Du sollst nur Beweisbares lehren!“ [Lud 02].

Wir empfehlen allen Lehrstühlen, die solche Praktika oder Projekte durchführen, im Rahmen der Vermittlung für Software-Ingenieure wichtiger Soft Skills auch solch ein vorbereitendes Teamtraining in Betracht zu ziehen. Wie das Beispiel der TU Darmstadt zeigt, können bei der Konzeption eines solchen Trainings Kooperationspartner von Psychologie-Lehrstühlen oder Hochschuldidaktischen Stellen der Hochschule unterstützend hinzugezogen werden; damit kann auch der Aufwand bei der Schulung der Trainer und bei der Konzeption eines Teamtrainings eingeschränkt werden. Und auch wir selbst stehen gerne bei der Initiierung von Teamtrainings beratend zur Seite.

Literatur

[Aut 02] Homepage des Design- und Simulationswerkzeuges AutoFocus. München, 2004. *http://autofocus.in.tum.de*

[Aut 04] Homepage des AutoRAID Praktikums. München, 2004. *http://wwwbroy.in.tum.de/~autoraid*

[Den 98] M. Deneke, U. Schröder, M. Brunner: Constructionist Learning in Software Engineering Projects. SEES 1998, Software Engineering Education Symposium 18 - 20 November 1998. Poznan, Polen, 1998.

[Fle 02] A. Fleischmann: Entwurf eines berufsbegleitenden Aufbaustudiengangs für Software-Engineering. Diplomarbeit am Fachbereich Informatik der Technischen Universität Darmstadt; Betreuer: Prof. Henhapl. Darmstadt, 2002.

[Gna 03] M. Gnatz, L. Kof, F. Prilmeier, T. Seifert: A Practical Approach of Teaching Software Engineering. In P. Knoke, A. Moreno, M. Ryan (Hrsg.): 16th Conference on Software Engineering Education and Training, pages 120-128. IEEE Computer Society, March 2003.

[Hen 02] W. Henhapl (Hrsg.): Informationsseite des Instituts für Praktische Informatik an der TU Darmstadt zum Software-Engineering-Projekt. Darmstadt, 2002. *http://www.pi.informatik.tu-darmstadt.de/se2002/*

[Henn 02] M. Henninger, H. Mandl: Zuhören, verstehen, miteinander reden. Hans Huber Verlag, 2003.

[Kom 04] Hochschuldidaktische Arbeitsstelle der Technischen Universität München: Teamtrainings nach dem KOMPASS Konzept. Darmstadt, 2004. *http://www.tu-darmstadt.de/hda/tutorengruppe/kompass.html*

[Lan 00] B. Langemaack, M. Braune-Krickau: Wie die Gruppe laufen lernt. Anregungen zum Planen und Leiten von Gruppen. Beltz Verlag, Weinheim 2000.

[Lud 02] J. Ludewig: Thesen zur Softwaretechnik in den Universität. Vortrag am 30. April 2002 in der TU Darmstadt auf dem Symposium „Softwaretechnik Aus- und Weiterbildung“. Darmstadt, 2002. *http://www.informatik.uni-stuttgart.de/ifi/se/publications/download/VortragDA.pdf*

[Sen 04] T. Senninger (Hrsg.): Kooperationsspiele, eine Sammlung. 2004. *http://www.abenteuerprojekt.de/Spiele/kooperation.php*

[Tau 01] D. Taubner: Software-Entwicklung im industriellen Maßstab. In: J. Desel (Hrsg.:) "Das ist Informatik", Springer-Verlag 2001. Seite 85-98, *http://www.sdm.de/dt/tec/pub/art/art/seiim/seiim.pdf*

Softwaretechnik live – im Praktikum zur Projekterfahrung

Peter Göhner, Friedemann Bitsch, Hisham Mubarak
Universität Stuttgart, Institut für Automatisierungs- und Softwaretechnik
Pfaffenwaldring 47, 70550 Stuttgart
{goehner, bitsch, mubarak}@ias.uni-stuttgart.de

Zusammenfassung

Hochschulabgängern von Ingenieurstudiengängen fehlt in der Regel Projekterfahrung sowie Erfahrungen mit Teamarbeit. Um Studierende auf diese beiden wichtigen Aspekte der industriellen Praxis vorzubereiten, bietet das Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart das Fachpraktikum Softwaretechnik an, dessen Konzept in diesem Beitrag vorgestellt wird. Das Praktikum wird wie ein industrielles Projekt durchgeführt. Die Studierenden bilden Teams, die wie kleine Firmen agieren und einen Auftrag zur Softwareentwicklung termingerecht bearbeiten müssen. Die Studierenden sammeln Projekterfahrungen in international zusammengesetzten Teams, da das Praktikum sowohl von Studierenden aus Diplomstudiengängen als auch von Studierenden aus einem internationalen Masterstudiengang besucht wird.

1 Einleitung

In den vergangenen Jahren hat die industrielle Softwareentwicklung immer mehr an Bedeutung für die Ingenieurwissenschaften gewonnen. Sie macht einen nicht zu vernachlässigenden Teil der beruflichen Tätigkeit der Ingenieure aus. Es ergibt sich daher die Notwendigkeit, diese Tatsachen auch in der Lehre zu berücksichtigen und Studierende in Ingenieurstudiengängen umfassend auf dem Gebiet der Softwaretechnik zu unterrichten. Ein wesentliches Problem hierbei ist der mangelnde Bezug zur Praxis. Im regulären Lehrbetrieb der Hochschulen werden den Studierenden die fachlichen Grundlagen in Form von Vorlesungen und Übungsveranstaltungen vermittelt. Zwangsläufig sind die gelehrten Inhalte sehr theoretischer Natur und erscheinen nicht nur unnötig komplex, sondern auch fern der von vielen Studierenden auf die Programmierung reduzierten Softwareentwicklung. Eine Ursache hierfür ist die Tatsache, dass Studierende der Ingenieurwissenschaften im Verlauf des Studiums keine oder wenig praktische Er-

fahrung mit der systematischen und industriellen Softwareentwicklung sammeln können. Die Vermittlung der Lehrinhalte scheitert daher zumeist an der Akzeptanz der Notwendigkeit, Software systematisch zu entwickeln.

Gleichzeitig erwartet die Industrie Hochschulabsolventen, die nicht nur die fachlichen Grundlagen, also die so genannten „Hard Skills“, beherrschen, sondern auch über gewisse „Soft Skills“, wie Teamfähigkeit, Konfliktfähigkeit und Fähigkeit zur Projektarbeit, verfügen. Im heutigen industriellen Umfeld, wo Software fast ausschließlich im Team entsteht, wird der Mangel an oben genannten Fähigkeiten häufig beklagt [Voos03]. Leider ist festzustellen, dass die Hochschulausbildung in Ingenieurstudiengängen kaum Elemente aufweist, in denen diese Fähigkeiten gezielt vermittelt und gefördert werden. Im Gegenteil: Die Hochschulausbildung hat diese Herausforderungen bisher vernachlässigt und sich auf die fachliche Ausbildung des einzelnen Studierenden konzentriert. Studierende haben also keine Möglichkeit, derartige für die Berufsausübung nicht unerheblichen Fähigkeiten im Rahmen ihres Studiums zu erlernen.

Vor diesem Hintergrund sah sich das Institut für Automatisierungs- und Softwaretechnik (IAS) der Universität Stuttgart in der Verantwortung, eine universitäre Lehrveranstaltung zu schaffen, die sowohl den Praxisbezug der fachlichen Ausbildung herstellt, als auch Aspekte wie die Team- und Projektarbeit gebührend berücksichtigt und vermittelt. Entstanden ist hierbei das Fachpraktikum *Softwaretechnik*, das zur Ergänzung und Vertiefung der Lehrinhalte der Vorlesung zur Ausbildung von Ingenieuren im Bereich der Softwaretechnik dient. Gleichzeitig trägt es zur Vorbereitung der Studierenden auf das Berufsleben bei.

Dieser Beitrag beschreibt das Konzept des Fachpraktikums *Softwaretechnik* und berichtet über die Erfahrungen, die während der vergangenen 9 Jahre am IAS gewonnen wurden.

2 Praktikumsdurchführung in Projektform

Die Lehrveranstaltungsform Fachpraktikum wird für Studierende des Hauptstudiums der Diplomstudiengänge *Elektro- und Informationstechnik* sowie *Automatisierungstechnik in der Produktion* und des internationalen Masterstudiengangs *Information Technology (INFOTECH)* angeboten. Fachpraktika verfolgen das Ziel, die in Vorlesungen erworbenen Fachkenntnisse der Teilnehmer zu vertiefen und anzuwenden. Üblicherweise wird in Fachpraktika eine Reihe an Versuchen von Studierenden absolviert, die an vorbestimmten Terminen durchgeführt werden.

Im Gegensatz hierzu steht das Fachpraktikum *Softwaretechnik*. Es wird in Form eines industriellen Softwareprojekts durchgeführt. Dabei wird das Projekt nicht von jedem Studierenden einzeln, sondern in einem von drei studentischen Teams mit jeweils sechs bis acht Teammitgliedern durchgeführt. Ein Team stellt dabei eine kleine Firma dar, die als Auftragnehmer eines Kunden tätig ist, und für diesen ein Softwaresystem termingerecht zu realisieren hat. Jedes Team wird von einem wissenschaftlichen Mitar-

beiter betreut. Der Teambetreuer nimmt dabei nicht nur die Rolle eines Tutors ein, er ist auch gleichzeitig als Berater des Teams anzusehen und dadurch Teil des Teams. Die Tätigkeiten des Teams sind auf den Auftraggeber, also den Kunden, ausgerichtet. Die Rolle des Kunden wird zusammen mit dem Professor von einem vierten wissenschaftlichen Mitarbeiter ausgeübt. Dieser vierte Mitarbeiter übernimmt auch gleichzeitig die Gesamtorganisation des Fachpraktikums.

2.1 Organisatorischer Rahmen

Das Fachpraktikum wird während der Vorlesungszeit im Sommersemester durchgeführt. In den ersten Wochen des Praktikums müssen die Studierenden die Anforderungen an das zu erstellende Softwaresystem analysieren und definieren. Den Studierenden wird hierzu keine exakte schriftliche Aufgabenbeschreibung vorgelegt. Der Kunde äußert den Wunsch nach einer Software, die einen mobilen Fahrroboter so schnell wie möglich durch einen unbekannten Parcours an ein vordefiniertes Ziel steuert. In Gesprächen und Interviews mit dem Kunden müssen die Studierenden die Anforderungen ermitteln. Dabei lernen sie, mit den oft vagen Vorstellungen ihres Kunden umzugehen und diese in einem iterativen Prozess zu konkretisieren. Sie lernen auch, den Kunden als Ressource anzusehen, die nicht permanent zur Verfügung steht und nicht immer „mal eben Zeit für eine kurze Frage“ hat. Die Terminplanung und -einhaltung wird somit von Beginn an vermittelt. Wie auch im industriellen Alltag können sich die Kundenwünsche und damit die Anforderungen an das zu entwickelnde System ändern. Die Ergebnisse der Anforderungsanalyse werden in einem Pflichtenheft festgehalten, das vom Kunden akzeptiert werden muss. Neben dem Pflichtenheft wird von den Studierenden auch die Abgabe eines realistischen Preisangebots für das zu entwickelnde System verlangt. Die Studierenden werden dadurch angehalten, auch die wirtschaftlichen Aspekte zu berücksichtigen, und das Rollenspiel „Firma/Kunde“ wird belebt.

Begleitet wird das Fachpraktikum durch regelmäßig stattfindende Seminare, in denen wichtige Hintergrundinformationen und theoretische Grundlagen vermittelt werden. Die Seminare werden von den wissenschaftlichen Mitarbeitern des Instituts abgehalten und haben jeweils einen konkreten Aspekt der Softwaretechnik zum Thema, der in unmittelbarem Zusammenhang mit anstehenden Aufgaben steht. Aus organisatorischer Sicht stellen die Seminare die Rahmenveranstaltungen des Fachpraktikums dar, die auch zur regelmäßigen Steuerung und Überwachung des Praktikumsverlaufs dienen. Den Studierenden wird im Rahmen der Seminare die Möglichkeit gegeben, Fragen zu stellen und unklare Punkte zu diskutieren. Im Anschluss an ein Seminar werden den Studierenden Teilaufgaben gestellt. Zweck dieser Teilaufgaben ist es, den Studierenden einen groben zeitlichen Rahmen zur Durchführung wesentlicher Arbeitsschritte sowie eine Unterstützung in der Aufteilung von Aufgaben im Team zu geben.

Der Höhepunkt des Fachpraktikums ist ein Roboterwettrennen (siehe Abb. 1), in dem die Softwaresysteme der Studierenden in Wettbewerbsform gegeneinander antreten. Die dadurch entstehende Konkurrenz trägt entscheidend zur Motivation der Studierenden bei und bereitet die Studierenden gleichzeitig auf industrielle Arbeitsbedingun-

gen vor. Vor der Rennveranstaltung müssen die Studierenden ihr „Produkt“ dem Kunden präsentieren und versuchen, ihn von der Qualität oder der Pfiffigkeit ihrer Lösung zu überzeugen.

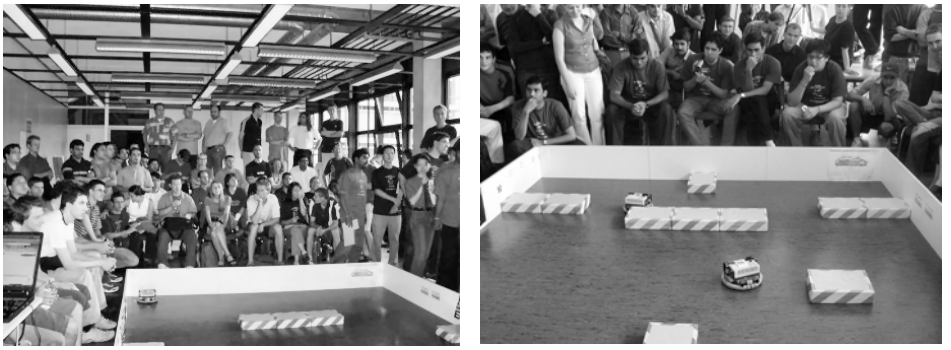

Abb. 1: Beispiel-Parcours und Besucheransturm beim Roboterwettrennen

2.2 Vorgehensmodell zur Steuerung und Kontrolle des Projektverlaufs

Ein wichtiges Mittel zur Unterstützung der zeitlichen Planung und Durchführung des Softwareprojekts ist die Vorgabe eines Vorgehensmodells, das die Studierenden in der Praktikumsdurchführung anleitet und die Durchführung der Softwareentwicklung im Fachpraktikum systematisiert. Dieses Vorgehensmodell wurde auf der Basis des V-Modells [BrDr95], des Entwicklungsstandards für IT-Systeme des Bundes, entwickelt. Ein Ziel bei der Entwicklung war die Einfachheit des Modells, sodass es für die Studierenden im Rahmen des Fachpraktikums überschaubar und leicht erlernbar ist. Während das V-Modell ein generisches Vorgehensmodell darstellt, das weder eine spezielle Organisationsstruktur noch einen bestimmten zeitlichen Ablauf vorschreibt, werden im Vorgehensmodell des Fachpraktikums alle für die Projektbearbeitung notwendigen generellen Aktivitäten und Produkte und deren Abhängigkeiten definiert (Tailoring). Dabei wurden spezielle Anpassungen für die Projektbearbeitung in kleinen Teams vorgenommen. Zur Festlegung des zeitlichen Ablaufs der Projektbearbeitung wurde das V-Modell mit einem einfachen Phasenmodell kombiniert. Das Ergebnis sind die folgenden Projektphasen:

- Anforderungserfassung
- Anforderungsanalyse
- Softwareentwurf
- Softwareimplementierung und -integration
- Softwaretest und Überarbeitung

Der Einsatz des Vorgehensmodells bietet zahlreiche Vorteile sowohl für die Studierenden als auch für die Betreuer. Durch das Vorgehensmodell werden die Studierenden frühzeitig dazu angehalten, sich Gedanken über die anstehenden Aufgaben zu machen, diese zu formulieren und eine detaillierte Planung zu erstellen. Dadurch wird das Projekt überschaubar, und die Studierenden können den zeitlichen Ablauf abschätzen. Unnötige Leerlaufzeiten aufgrund von ungeplantem Vorgehen können weitgehend minimiert oder ganz ausgeschlossen werden. Die Studierenden werden durch das Vorgehensmodell zu einer fortwährenden strukturierten Projektdurchführung und -dokumentation angehalten. Auch für die Betreuer sind die Arbeiten der Teams einfacher zu kontrollieren. Es kann festgestellt werden, welche Aufgaben verzögert sind und welche Auswirkungen die Verzögerung haben kann.

Wie bereits genannt, ist die Projektdokumentation Teil des Vorgehensmodells und eines der Lernziele des Fachpraktikums. Zur Unterstützung einer systematischen Dokumentation werden den Studierenden Dokumentvorlagen zur Verfügung gestellt. Der Einsatz der Vorlagen ist für die Studierenden verbindlich, wobei lediglich die Dokumentstruktur und Hinweise zum Inhalt der Kapitel vorgegeben werden. Auf Basis der erstellten Dokumente werden zu bestimmten Zeitpunkten des Arbeitsverlaufs („Meilensteine“) Reviews durchführt. Dabei erfolgt eine Diskussion der zu diesem Zeitpunkt vorliegenden Ergebnisse zwischen Studierenden und Betreuern.

3 Fahrrobotersteuerung durch unbekannten Hindernisparcours

Die Aufgabenstellung des Fachpraktikums ist die Entwicklung einer Steuerungssoftware, die einen Fahrroboter von einem vordefinierten Startpunkt durch einen unbekannten Hindernisparcours in einen definierten Zielbereich steuern soll. Für die Manövrierung des Roboters durch den Parcours müssen die Studierenden geeignete Wegfindungsalgorithmen bereitstellen und als Steuerungssoftware umsetzen. Welche Strategie der Roboter dabei anwendet, bleibt jedem Team frei überlassen, wodurch vielfältige Lösungen für die Aufgabe entstehen. Der Zielbereich, den die Roboter ansteuern sollen, befindet sich in einem Korridor, der durch die Ziellinie vom restlichen Parcours abgetrennt ist (vgl. Abb. 2). Überfährt ein Roboter die Ziellinie, ist das Ziel im Sinne des Wettbewerbs erreicht.

Die eingesetzten Roboter wurden am Lehrstuhl für Realzeit-Computersysteme der TU München (Prof. Färber) entwickelt [Spat93, Land93]. Die äußeren Abmessungen des Roboters sind durch seine Stoßstange gegeben (siehe Abb. 2). Die Stoßstange ist beweglich gelagert und betätigt beim Auffahren auf ein Hindernis einen oder mehrere taktile Sensoren, durch die der Roboter eine Kollision mit einem Hindernis und die Position der Kollision erkennen kann.

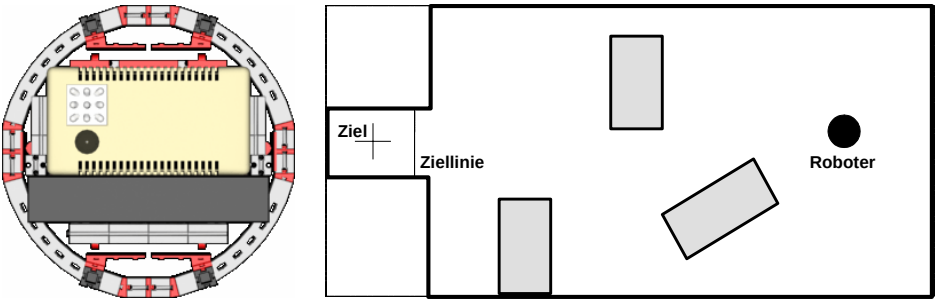

Abb. 2: Sicht auf Roboter von oben (links) und schematische Darstellung eines Parcours (rechts)

Den Robotern werden Fahrbefehle über Infrarot gesendet, und sie informieren über ihren aktuellen Roboterstatus (z. B. fahrend, stehend oder in Kollision). Die Aufgabe der Steuerungssoftware ist nun, mithilfe dieser Informationen den Roboter schnellstmöglich durch einen Hindernisparcours zu lenken und diesen zu kartografieren, um die Hindernispositionen bei einem späteren Lauf berücksichtigen zu können. Die erfassten Parcoursdaten, die Roboterpositionen sowie der geplante Weg zum Ziel müssen grafisch auf dem Bildschirm ausgegeben werden.

Zur Steuerung der Fahrroboter steht den Studierenden eine Softwareumgebung [Günt99] zur Verfügung, welche eine Kommunikationsbibliothek für den koordinierten Zugriff auf die Roboter enthält. Zum Kennenlernen des technischen Prozesses können die Roboter mithilfe einer Demo-Steuerung per Tastaturbefehle gesteuert werden. Ein Simulator erlaubt das Testen der Software ohne Einsatz der realen Roboter. Die Roboter des Simulators verhalten sich im Gegensatz zu den realen Robotern weitgehend ideal. Dadurch lernen die Studierenden mögliche Unterschiede von Simulationen und realen technischen Prozessen sowie dadurch entstehende Probleme kennen.

4 Aspekte der Projektarbeit als Lernziele

Die Lernziele des Fachpraktikums gliedern sich in folgende Bereiche:

- Teamarbeit, Gebrauch von Englisch zur Kommunikation in der Teamarbeit, Zusammenarbeit in multikulturellen Teams
- Projektmanagement
- Methodische Softwareentwicklung
- Qualitätssicherung
- Konfigurationsmanagement
- Zeitmanagement

4.1 Teamarbeit

Das Hauptlernziel des Praktikums ist das Erlernen und Erleben von Teamarbeit vor dem Einstieg in das Berufsleben. Für die Teilnehmer am Fachpraktikum ist Gruppenarbeit vielfach ein neues Thema. Die Studierenden müssen die Zusammenarbeit im Team selbstständig organisieren. Sie sollen lernen, mit Problemen umzugehen, die sich bei der Teamarbeit ergeben, und ein effizientes Zusammenarbeiten einüben. Problematisch kann beispielsweise die Kommunikation im Team sein. Zum einen haben die einzelnen Gruppenmitglieder einen unterschiedlichen Kenntnisstand. Zum anderen kann es persönliche Zuneigungen und Abneigungen geben. Auch muss die Frage nach der im Team verwendeten Sprache geklärt werden. Nicht immer sprechen alle Teammitglieder eine gemeinsame Muttersprache.

Den Studierenden wird vermittelt, dass für eine gute Teamarbeit das Arbeitsklima entscheidend ist, welches wiederum durch die Gruppenmitglieder geprägt wird. Eine gewisse Kritikfestigkeit wird von jedem Gruppenmitglied gefordert ebenso wie Sensibilität beim Kritiküben.

Eine weitere Schwierigkeit, mit der die Studierenden konfrontiert werden, ist bei wichtigen Fragestellungen in der Projektorganisation oder Entwicklung gemeinsame Entscheidungen zu treffen. Dabei müssen unterschiedliche Ansichten zunächst toleriert werden. In der Diskussion müssen die Teammitglieder dann eine Lösung finden, die für das Team annehmbar ist. Viele der Studierenden bewerten die dabei gewonnenen Erfahrungen als äußerst wertvoll, da die Auseinandersetzungen im Team zu einer sehr intensiven Befassung mit Gruppenarbeit und der Aufgabenstellung führt.

Die Studierenden sollen lernen, selbstständig zu handeln und Ideen sowie Vorschläge in ihr Team einzubringen. Bevor die Betreuer um Hilfestellung gebeten werden, versuchen die Teams, Fragen und Probleme alleine zu klären und zu bewältigen.

Die Teams definieren zu Beginn des Fachpraktikums die Rollen mit den zugehörigen Aufgabenbereichen. Als Hilfestellung wird den Studierenden folgende Rollenverteilung empfohlen, vgl. [LaGö99, S. 13]:

1. Ein Studierender übernimmt die Aufgabe des **Projektmanagers**. Er ist verantwortlich für die Planung der Ressourcen (Studierende, Zeit, Rechner, Systemumgebung, Teambetreuer), für die Koordination und für die Kosten- und Zeiterfassung.

2. Vier bis fünf Studierende übernehmen die Aufgabe der **Entwickler**. Diese Untergruppe führt hauptsächlich die Anforderungserfassung, die Systemanalyse, den Entwurf, die Implementierung und die Überarbeitung des Softwaresystems durch.

3. Zwei Studierende übernehmen die Aufgabe der **Qualitätssicherung**. Neben der Planung und Durchführung von Qualitätssicherungsmaßnahmen testet diese Untergruppe das Softwaresystem.

4. Ein Studierender übernimmt als Nebenaufgabe die des **Schnittstellenverwalters**. Er hat die Aufgabe für konsistente Schnittstellen zwischen verschiedenen Softwaremodulen während der gesamten Projektlaufzeit zu sorgen. Die Ergebnisse der Teammitglieder müssen zusammenpassen. Aus diesem Grund ist eine sorgfältige Schnittstellendefinition zwischen den zu entwickelnden Modulen von großer Bedeutung.
5. Ein Studierender ist neben seiner Hauptaufgabe für das **Konfigurationsmanagement** zuständig. Der Konfigurationsmanager ist für die Archivierung und die Versionsverwaltung der Dokumentation sowie der Produkte der einzelnen Entwicklungsphasen verantwortlich.

Je nach ausgeübter Rolle machen die Studierenden sehr unterschiedliche Erfahrungen. Da das Fachpraktikum auch von Studierenden des englischsprachigen Masterstudiengangs *INFOTECH* besucht wird, stehen sowohl die Studierenden als auch die Betreuer vor der Herausforderung, die Kommunikation in Englisch durchzuführen. Lehren und Lernen erfolgen also in einer Fremdsprache. Insbesondere die Dokumente, die in den verschiedenen Projektentwicklungsphasen angefertigt werden, müssen in englischer Sprache erstellt werden. Der Gebrauch von Englisch gehört im heutigen Berufsleben zum Alltag. Im Fachpraktikum Softwaretechnik werden die Studierenden durch Besprechungen, Diskussionen und Dokumentation in englischer Sprache auf die industrielle Praxis vorbereitet.

In einer Zeit fortschreitender Globalisierung ist die Fähigkeit der Zusammenarbeit in multikulturellen Teams von zunehmend wichtigerer Bedeutung und sollte in der Ausbildung nicht ignoriert werden. Die Zusammensetzung der Teams aus Studierenden der drei genannten Fachrichtungen macht nicht nur den Gebrauch von Englisch erforderlich, sondern schult die Studierenden in der Zusammenarbeit mit Menschen aus anderen Kulturen. Am INFOTECH-Studiengang nehmen Studierende aus der ganzen Welt mit unterschiedlichstem kulturellem Hintergrund teil. Durch die enge Zusammenarbeit der Studierenden in den Teams sind die Studierenden gefordert, sensibel für die Arbeitskulturen der Teammitglieder zu sein und einen gemeinsamen Weg für die Zusammenarbeit zu finden.

4.2 Projektmanagement

Die Studierenden werden im Fachpraktikum im Bereich Projektmanagement geschult. Bestandteile des Projektmanagements sind die Projektplanung, die Projektkontrolle, die Ergreifung von Maßnahmen für die Einhaltung des Projektplans sowie Kostenberechnung und Teamorganisation, vgl. [LaGö99, S. 17 f.]. Zu den Lernzielen in diesem Aufgabenbereich gehört auch das Üben der ansprechenden Präsentation von Arbeitsergebnissen. Die Projektplanung umfasst die Teilaufgaben Organisations-, Aufwands- und Terminplanung sowie die Ressourcenplanung. Die Projektplanung wird anhand eines *Projektplans* dokumentiert und projektbegleitend verfolgt.

Zudem gehört es auch zu den Aufgaben des Projektmanagements, Probleme in der Teamarbeit zu erkennen und diesen entgegenzuwirken. Dies schließt auch die Motivation der Teammitglieder mit ein. Am Ende des Projekts verfasst der Projektmanager einen Projektabschlussbericht, der eine Zusammenfassung des gesamten Projektverlaufs enthält und die erzielten Ergebnisse erläutert.

4.3 Methodische Softwareentwicklung

Bei der Durchführung des Projekts werden die Studierenden in methodischer Softwareentwicklung geschult. Dabei werden Techniken und Methoden der Softwaretechnik angewandt, welche die Studierenden in den Vorlesungen *Einführung in die Informatik III* [Göhn04a] und *Softwaretechnik I* [Göhn04b] kennen gelernt haben. Die Softwareentwicklung erfolgt objektorientiert, wobei für Analyse (OOA) und Entwurf (OOD) die UML-Notation verwendet wird. Die objektorientierte Implementierung (OOP) erfolgt wahlweise in den Programmiersprachen C++ oder Java. Die Entscheidung über die eingesetzte Sprache wird den Teams überlassen.

Das Softwaresystem muss nicht komplett neu entwickelt werden. Wie in den meisten realen Softwareprojekten in der Industrie werden zum Teil vorgefertigte Softwarekomponenten wiederverwendet. Diese Softwarekomponenten sind zum Teil leicht fehlerbehaftet und nicht optimal dokumentiert. Die Studierenden lernen mit derartigen Schwierigkeiten, die nicht selten in realen Softwareentwicklungsprojekten auftreten, umzugehen. Gleichzeitig wird ihnen damit der Nutzen einer guten Projektdokumentation vermittelt.

4.4 Qualitätssicherung

Um die Erfüllung der vorgegebenen Anforderungen zu gewährleisten, werden Qualitätssicherungsmaßnahmen angewandt. Softwarequalität wird durch den Einsatz konstruktiver Qualitätssicherungsmaßnahmen erreicht (wie z. B. die Gliederung des Entwicklungsprozesses durch ein Vorgehensmodell sowie die Unterstützung des Entwicklungsprozesses durch Methoden und Werkzeuge). Diese Maßnahmen werden durch analytische Qualitätssicherungsmaßnahmen (wie z. B. Reviews sowie Komponenten- und Systemprüfungen) ergänzt, vgl. [LaGö99, S. 34 f.]. Beide Arten von Maßnahmen sind Bestandteil der Ausbildung im Fachpraktikum Softwaretechnik.

Für die Komponenten- und Systemprüfung werden aus den Anforderungen an das zu entwickelnde System allgemeine Prüfanforderungen abgeleitet. Danach werden Prüfmethoden und -kriterien festgelegt, mit denen die Prüfanforderungen zu erfüllen sind. Außerdem sind für die einzelnen Komponenten und für das Gesamtsystem Prüffälle zu definieren. Die Ergebnisse und Auswertungen der einzelnen Tests sind im Prüfprotokoll zu dokumentieren.

4.5 Konfigurationsmanagement

Im Bereich Konfigurationsmanagement lernen die Studierenden, ein Produkt oder Zwischenprodukt (z. B. Dokument, Quellcode-Datei usw.) bezüglich seiner funktionellen wie auch äußeren Merkmale eindeutig zu identifizieren. Diese Identifikation dient der systematischen Kontrolle von Änderungen und der Sicherstellung der Integrität. Das Konfigurationsmanagement überwacht die Konfigurationen während der gesamten Entwicklung, sodass die Zusammenhänge und Unterschiede zwischen früheren Konfigurationen und den aktuellen Konfigurationen jederzeit erkennbar sind. Dadurch sind Änderungen nachvollziehbar und überprüfbar [LaGö99, S. 16 f.]. Zudem muss ein Konzept bereitgestellt werden, durch das die Bearbeitung von Zwischenprodukten in der Teamarbeit koordiniert wird.

4.6 Zeitmanagement

Die für das Fachpraktikum zur Verfügung stehende Zeit ist insbesondere dadurch begrenzt, dass die Studierenden das Praktikum in der Vorlesungszeit des Semesters durchführen, in der sie reguläre Vorlesungen besuchen.

Die Studierenden verwenden ca. 8-14 Stunden pro Woche für das Praktikum. Zur Vermeidung von Stress und chaotischer Teamarbeit erhalten die Studierenden zu Beginn des Fachpraktikums eine Einführung in das Thema Zeitmanagement angelehnt an [Hump97]. Dabei lernen sie im Hinblick auf ihre spätere Tätigkeit in der Industrie ihre Zeit selbstständig einzuteilen, realistisch abzuschätzen, die in der Realität benötigte Zeit zu erfassen und aufgrund dessen ihre Zeiteinteilung und -abschätzung zu bewerten und kontinuierlich zu verbessern.

5 Motivation der Studierenden

Bei einer derartig arbeitsintensiven und zeitaufwändigen Lehrveranstaltung stellt sich die Frage, wie die Studierenden über die gesamte Praktikumsdauer motiviert werden können, um ein Einbrechen der Motivation und der Lern- bzw. Leistungsbereitschaft zu vermeiden. Dies ist sicherlich keine leichte Aufgabe und kann nicht durch einzelne, sondern nur durch die Kombination abgestimmter Maßnahmen bewältigt werden. Dabei sind soziale und technische Aspekte zu berücksichtigen. Die Erfahrung zeigt, dass diejenigen Teams am erfolgreichsten waren, in denen ein ausgeprägter Teamgeist aufgrund der Motivation aller Teammitglieder entwickelt wurde. Folgende Mittel werden am IAS erfolgreich zur Motivation eingesetzt: Meilensteine, Rollenspiel, Teamarbeit, Konkurrenz, eine herausfordernde Aufgabenstellung, soziale Veranstaltungen und die Ergebnispräsentation in Form eines Roboterwettrennens.

5.1 Meilensteine

Zu Beginn des Praktikums werden Meilensteine definiert, zu denen bestimmte Zwischenprodukte erstellt und vorgelegt werden müssen. Durch diese Zwischenziele wird nicht nur die Projektkontrolle unterstützt, sondern die Studierenden können auch Teilerfolge in ihrer Arbeit erleben, die eine Basis für die Weiterarbeit darstellen und eine motivierende Wirkung besitzen. Je härter ein Team für die Erreichung eines Meilensteins gearbeitet hat, desto größer ist die Motivation, das Projekt fortzuführen. Gleichzeitig wird dabei der Teamgeist gestärkt, da Meilensteine meist nur durch erfolgreiche Teamarbeit erreicht werden können.

In der neunten Praktikumswoche findet eine Prototyppräsentation statt. Die Studierenden präsentieren dabei den gegenwärtigen Entwicklungsstand ihrer Steuerung. Dem Kunden wird somit die Möglichkeit gegeben, den Fortschritt der Teams zu ermitteln, und eventuelle Abweichungen zu den Anforderungen zu erkennen. Für die Teams stellt diese Prototyppräsentation eine wichtige Rückmeldung über den eigenen Stand im direkten Vergleich mit der „Konkurrenz“ dar. Dies hat schon bei einigen Teams zu ungeahnten Motivationsschüben geführt.

5.2 Rollenspiel und Teamarbeit

Das Rollenspiel kommt beim Fachpraktikum in unterschiedlicher Weise zum Einsatz. Zum einen tritt der Praktikumsorganisator den Studierenden als Auftraggeber gegenüber. Zum anderen sehen sich die Studierenden von Anfang an als Auftragnehmer und lernen, als Team aufzutreten. Diese Sicht wird dadurch unterstützt, dass die Studierenden innerhalb ihres Teams unterschiedliche Verantwortungsbereiche wahrnehmen und vertreten. Um die Teamidentität weiter herauszubilden, wählt jedes Team einen Namen und ein Logo. Dadurch entsteht in den Teams ein „Wir“-Gefühl und damit eine Verantwortung des Einzelnen gegenüber seinem Team.

5.3 Konkurrenz

Ein Mittel, das die Motivation der Studierenden während der gesamten Praktikumsdauer nachhaltig fördert, ist die Wettbewerbssituation, in der sich die unterschiedlichen Teams von Beginn an befinden. Erfahrungsgemäß macht diese Situation den besonderen Reiz des Fachpraktikums aus. Die Studierenden werden dazu angehalten, ihre Mitbewerber nicht als erbitterte Konkurrenten anzusehen. Die Gruppen können und sollen sich in technischen und organisatorischen Fragen helfen. Dies führt zu einer regelmäßigen Kommunikation unter den Teams und damit zu einem ständigen Abschätzen des Entwicklungsstands der Mitbewerber. Die eigenen Ergebnisse und der Projektfortschritt stehen immer in Relation zu denen der anderen Teams. Ein weiterer Vorteil der Wettbewerbssituation ist, dass die entwickelten Softwaresysteme ein hohes Niveau erreichen und damit zu spannenderen Roboterrennen führen, die wiederum jüngere Studierende ansprechen.

5.4 Angemessene, herausfordernde und interessante Aufgabenstellung

Die Wahl der Aufgabenstellung ist von großer Bedeutung, insbesondere beim Softwaretechnikunterricht für Ingenieure. Die Aufgabenstellung muss zum einen interessant und anspruchsvoll genug sein, um die Teams über die Praktikumsdauer zu beschäftigen und die Lernziele zu vermitteln. Zum anderen darf sie nicht zu schwer sein, da sonst Resignation unter den Studierenden hervorgerufen wird. Auch sollte die Aufgabenstellung einen klaren Bezug zum späteren Berufsbild des Ingenieurs haben und nicht Softwareentwicklung um deren selbst vermitteln.

Die Realisierung einer Softwaresteuerung für mobile Roboter und die damit verbundene Beschäftigung mit der Hardware stellt für die meisten Studierenden eine spannende und anspruchsvolle technische Aufgabe dar. Sie sehen sich vor der Herausforderung, nicht nur die Software des Systems, sondern auch Eigenheiten der Hardware zu beherrschen. Am IAS wird immer wieder die Erfahrung gemacht, dass der Schwierigkeitsgrad von den Studierenden meist unterschätzt wird, sodass die technische Herausforderung aus Sicht der Studierenden im Verlauf des Praktikums anwächst.

5.5 Soziale Veranstaltungen

Soziale Veranstaltungen, wie beispielsweise ein Geschäftsessen, sind Teil des Berufslebens. Im Rahmen des Fachpraktikums finden daher zwei Veranstaltungen statt, welche die sozialen Kontakte zwischen den Studierenden, aber auch zu den Betreuern und dem Organisator fördern. Zunächst werden nach Ablauf der ersten Hälfte des Fachpraktikums die Studierenden zu einem informellen Gespräch und Erfahrungsaustausch in eine Gaststätte eingeladen. Bei diesem Gespräch in lockerer Runde ist es möglich, Probleme anzusprechen und den bisherigen Verlauf zu diskutieren. Insbesondere für Studierende aus fremden Kulturkreisen stellt dieses „etwas andere“ Zwischengespräch einen Gewinn dar.

Die zweite Veranstaltung findet zum Abschluss des Fachpraktikums statt. Auch hier sollen die Studierenden ihre zum Teil sehr intensiven Erfahrungen bei der Bearbeitung des Fachpraktikums schildern. In einer offenen Diskussion können Kritik und Lob in Anwesenheit des Professors und der Betreuer ausgesprochen werden. Fehler werden gemeinsam analysiert und ausgewertet. Die Veranstaltung wird mit einem gemeinsamen Abendessen abgeschlossen.

Beide Veranstaltungen haben sowohl bei Studierenden als auch bei Betreuern einen hohen Stellenwert und tragen dazu bei, dass die beim Fachpraktikum gewonnenen Erfahrungen im Gedächtnis bleiben.

5.6 Ergebnispräsentation

Die Ergebnispräsentation nimmt im Fachpraktikum eine besondere Rolle ein. Viele der teilnehmenden Studierenden kennen das Fachpraktikum aufgrund des Roboterwettrennens. Das Besondere an dieser Veranstaltung ist, dass nicht nur die Teams im Mittelpunkt stehen, sondern auch das Publikum. Die Attraktivität der Roboterwettrennen wird

für die Zuschauer noch durch ein Wettbüro erhöht. Die Zuschauer können, nachdem sich die Teams präsentiert haben, Tipps für den Ausgang der Rennen abgegeben und kleine Sachpreise gewinnen. Für Zuschauer aus aller Welt findet eine Live-Übertragung im WWW statt, die auch live kommentiert wird. Insbesondere für Bekannte und Angehörige ausländischer Studierender ist dies eine gute Möglichkeit, die Rennen zu verfolgen. Die Studierenden wollen sich natürlich vor den Zuschauern gut präsentieren und sind besonders motiviert, ein gutes und zuverlässiges Softwareprodukt zu entwickeln. Das Rennen wird mit einer Siegerehrung durch den Professor abgeschlossen.

6 Wertvolle Erfahrungen

Die Studierenden beurteilen die gewonnenen Erfahrungen in der Regel sehr positiv, wie es die Ergebnisse der regelmäßig durchgeführten Umfragen zeigen oder Zeichen der Anerkennung der Art, wie es in Abb. 3 zu sehen ist.

Abb. 3: Der Nutzen des Fachpraktikums wird von den Studierenden sehr positiv bewertet

Sowohl Studierende als auch die wissenschaftlichen Mitarbeiter gewinnen durch das Fachpraktikum in vielfältiger Weise Erfahrungen für das Berufsleben. Für beide stellt das Arbeiten in bzw. das Betreuen von multikulturellen Teams eine Herausforderung dar. Während des Praktikums bringen die kulturell bedingten unterschiedlichen Arbeitsweisen sowie Kommunikationsprobleme aufgrund von Sprachschwierigkeiten viele Probleme und Schwierigkeiten mit sich, die im Nachhinein betrachtet jedoch sowohl

von den Betreuern wie auch von den Studierenden als wertvolle Erfahrungen gewertet werden.

Die wissenschaftlichen Mitarbeiter werden durch die besondere Konstellation des Fachpraktikums darin geschult, Personalverantwortung zu übernehmen. Sie tragen die Verantwortung für die funktionierende Zusammenarbeit, das zielorientierte Arbeiten und die kompetente Betreuung in allen Teams. Sie müssen Probleme in der Teamarbeit und Schwachstellen in der Projektbearbeitung erkennen und Verantwortung dafür tragen, dass eine rechtzeitige Auseinandersetzung mit Problemen stattfindet. Sie machen Erfahrungen damit, wie unterschiedliche Studierende auf verschiedene Weise motiviert werden müssen.

Viele Studierende konnten in dem Praktikum Erfahrungen aus Fehlern sammeln, die wesentlich größere Auswirkungen gehabt hätten, wenn ihnen diese erst später im Berufsleben passiert wären. Studierende haben die Erfahrung gemacht, dass die Realisierung von Softwareeigenschaften, die der Kunde nicht gefordert hat, unnötig viel Zeit und Arbeit kosten kann, wenn der Kunde diese Eigenschaften nicht toleriert. Viele Studierende haben zudem auch die Erfahrung gewonnen, dass Softwareänderungen, die kurzfristig vor der Produktvorstellung durchgeführt werden, sodass ein ausreichender Test der Gesamt-Software nicht mehr möglich ist, meist zu einer Verschlechterung statt zu einer Verbesserung der Software führen.

In den letzten Jahren ist der Trend zu beobachten, dass die Softwareprodukte im Fachpraktikum immer besser werden. Ein entscheidender Faktor dafür ist wohl, dass in den letzten Jahren die Ingenieursausbildung in Informatik-Grundkenntnissen im Elektrotechnik- und Informationstechnik-Studium wesentlich verbessert wurde. So lernen die Studierenden an der Universität Stuttgart bereits im Vordiplomstudium objektorientierte Softwareentwicklung sowie entsprechende Modellierungsnotationen (UML) und Programmiersprachen (Java) sowie die Gestaltung von Benutzungsoberflächen kennen.

7 Fazit

Das Fachpraktikum Softwaretechnik hat sich als fester Bestandteil im Lehrangebot des IAS etabliert. Der Aufwand zur Durchführung des Fachpraktikums ist sowohl für Studierende wie auch wissenschaftliche Mitarbeiter hoch. Es ist jedoch ein Aufwand, der sich für alle Beteiligten lohnt. Sie gewinnen durch das Praktikum sehr gute Erfahrungen hinsichtlich Projekt- und Teamarbeit. Sie sammeln Erfahrungen, wie durch die Entwicklung eines Teamgeists das Arbeiten Freude bereitet und die Qualität der Arbeit gefördert wird. Vielfach wurde die Erfahrung gemacht, wie sehr gute Ideen und Lösungsansätze in Diskussionen und durch die gegenseitige Ergänzung in der Teamarbeit entstehen. Viele Studierende haben es nach der Durchführung des Praktikums nicht nur leichter gehabt, Studien- und Diplomarbeiten systematisch zu bearbeiten, insbesondere hat ihnen das Praktikum den Einstieg in den Beruf vereinfacht. Aus den gewonnenen Erfahrungen wäre ein vermehrter Einsatz von Projektpraktika mit höherem Stellenwert im Studium sinnvoll.

Die Konzepte des Fachpraktikums wurden in den Fakultäten Elektrotechnik und Informatik der Universidade Federal do Amazonas, Brasilien, sehr positiv bewertet, sodass die Konzepte und die Technik zurzeit dort eingeführt werden. In Zukunft wird sich der Ingenieursnachwuchs der Universität Stuttgart auch mit brasilianischen Studierenden messen können.

Literatur

[BrDr95] A.-P. Bröhl, W. Dröschel (Hrsg.): Das V-Modell. Der Standard in der Softwareentwicklung mit Praxisleitfaden, 2. Auflage, R. Oldenbourg Verlag, München, Wien, 2005.

[Göhn04a] P. Göhner: Skript zur Vorlesung Einführung in die Informatik III, *http://www.ias.uni-stuttgart.de/info3*, IAS, Universität Stuttgart, 2004.

[Göhn04b] P. Göhner: Skript zur Vorlesung Softwaretechnik I, *http://www.ias.uni-stuttgart.de/st1*, IAS, Universität Stuttgart, 2004.

[Günt99] H. Günther: Studienarbeit, Portierung der zur Steuerung von Fahrrobotern entwickelten Software-Umgebung auf Windows NT, Institut für Automatisierungs- und Softwaretechnik, Universität Stuttgart, 1999.

[Hump97] W.S. Humphrey: Introduction to the Personal Software Process, Addison Wesley Longman Inc., Reading, 1997.

[Land93] M. Landfarth: Diplomarbeit, Entwurf und Aufbau eines ferngesteuerten Roboters, München, Lehrstuhl für Prozessrechner, Technische Universität München, 1993.

[LaGö99] R. Lauber, P. Göhner: Prozessautomatisierung 2. 1. Auflage, Springer-Verlag, Berlin, Heidelberg, New York, 1999.

[Spat93] O. Spatz: Diplomarbeit, Funktionserweiterung des Roboters für das Software-Engineering-Praktikum, München, Lehrstuhl für Prozessrechner, Technische Universität München, 1993.

[Voos03] J. Voosen: Schlechte Noten für die Praxis, Hochschulanzeiger Nr. 68, Frankfurt, 2003.

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.

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.

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)

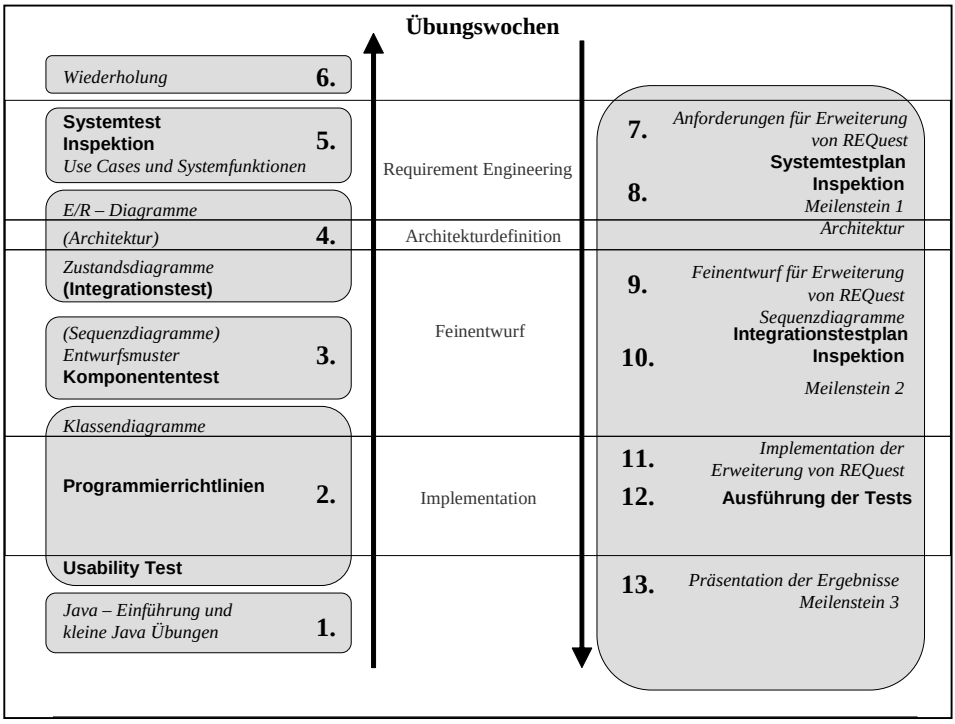

Abb. 1.: Abfolge der Themen

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

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-

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

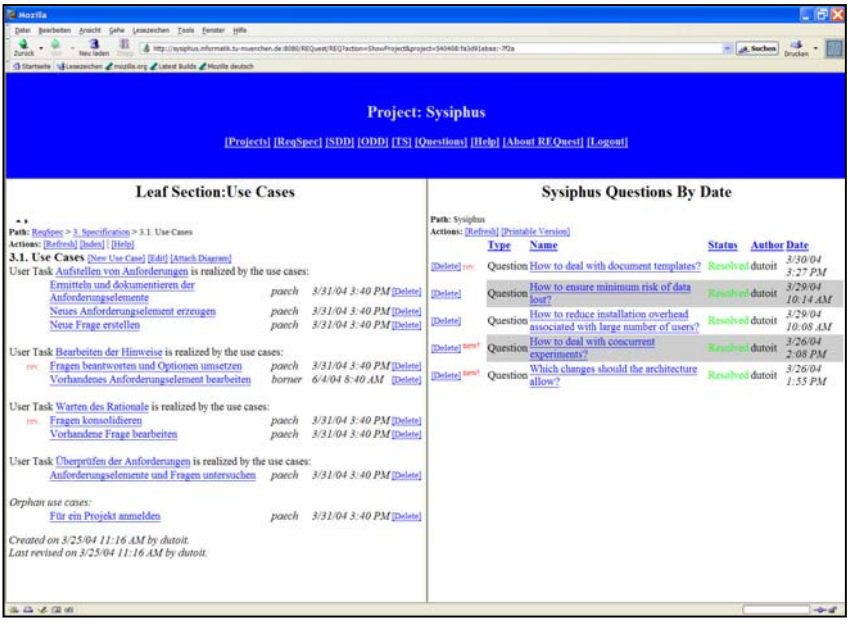

Abb. 2: Screenshot des bearbeiteten Systems

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*

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

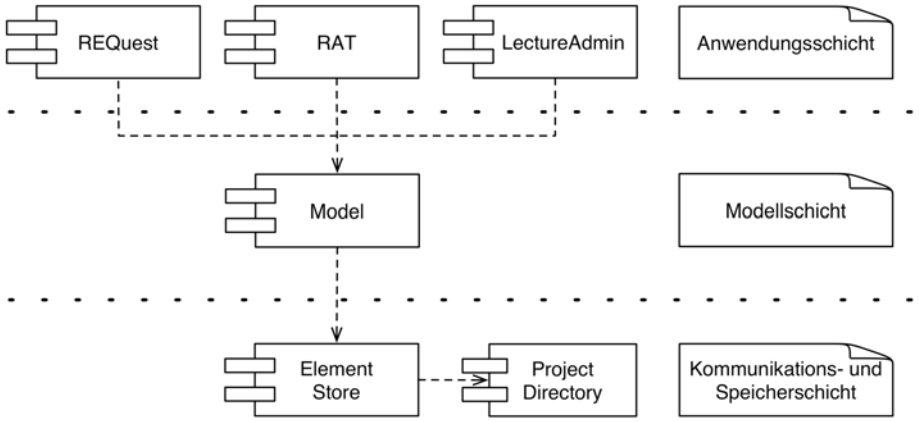

Abb. 3: Schichtenarchitektur von Sysiphus (UML-Komponentendiagramm)

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-

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

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-

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.

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-

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].

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.

Ein Softwaretechnik-Praktikum als Sommerkurs

Christian Lindig, Andreas Zeller
Lehrstuhl für Softwaretechnik, Universität des Saarlandes,
Postfach 15 11 50
66041 Saarbrücken
{lindig,zeller}@cs.uni-sb.de

Zusammenfassung

Ein semesterbegleitendes Softwaretechnik-Praktikum verleitet die Teilnehmer dazu, vor lauter Begeisterung und Gruppendruck andere Veranstaltungen zu vernachlässigen. An der Universität des Saarlandes wird das Praktikum daher in der vorlesungsfreien Zeit absolviert, und zwar als sechswöchiger Vollzeitkurs mit begleitender Vorlesung. Diese Form wirkt studienverkürzend und vereinfacht die Teamarbeit durch Ganztagspräsenz. Risikomindernde Maßnahmen wie ein einheitliches, vorgegebenes Pflichtenheft, spielerische Elemente und automatisch testbare Erfolgskriterien sorgen für hohe Motivation bei reibungslosem Ablauf.

1 Stand der Dinge

Die Universität des Saarlandes hat 2001 im Rahmen der Umstellung auf ein Bachelor/Master-System ein neues Softwaretechnik-Praktikum eingeführt, das mit 14 von 30 ECTS-Punkten etwa die Hälfte des 3. Semesters einnahm. Das Ziel des Praktikums ist das systematische Programmieren im Großen im Team. Dazu durchlief im Praktikum jedes Team aus fünf Studenten fünf Phasen:

- Anforderungsanalyse, abgeschlossen mit Pflichtenheft,
- Grobentwurf, abgeschlossen mit Objektmodell und Sequenzdiagrammen,
- Feinentwurf, abgeschlossen mit Prosa-Beschreibung und Unit-Tests,
- Implementierung, abgeschlossen mit fertigem Produkt,
- Test, abgeschlossen mit Testbericht.

Die Gliederung und die Aufgabenstellung waren betont realistisch. Dazu gehörte, dass die Aufgabe von lehrstuhlfremden *Kunden* gestellt wurde – dies waren Mitarbeiter der Informatik, die im Rahmen von Forschungsprojekten geeignete Programmieraufgaben zu vergeben hatten, von der inhaltlichen Betreuung des Praktikums aber entbunden wa-

ren. Im Laufe des Semesters gab es Meilensteine, zu denen die entsprechenden Phasendokumente vorliegen mussten und bewertet wurden. Die Softwaretechnik-Vorlesung fand zeitgleich statt; sie war im Inhalt auf den Fortschritt im Praktikum abgestimmt.

Obwohl das Praktikum zu den beliebtesten Veranstaltungen des Grundstudiums gehörte, gab es eine Reihe von Problemen:

- **Zeitaufwand.** Unter Druck von Gruppe und Kunden neigten Studierende dazu, ihre Arbeit besonders gut zu machen und darüber andere Vorlesungen zu vernachlässigen [DHZS99]. Dies wirkte tendenziell studienverlängernd.
- **Gruppen aus schlechten Studenten**. Die Leistung von Informatikern schwankt bekanntlich in großem Maße [DHZS99]. Wir haben festgestellt, dass gute Studenten gerne mit anderen guten Studenten eine Gruppe bilden, was die Leistungsunterschiede der Gruppen potenziert – und schlechtere sowie unkommunikative Studenten alleine zurücklässt.
- **Gruppen aus guten Studenten.** Gruppen nur aus guten Studenten handeln sich Probleme ein, wenn sie glauben, dank ihrer Programmierkompetenz Entwurf und Gruppenkommunikation vernachlässigen zu können – was sich oft als Fehleinschätzung herausstellt.
- **Schlechte Abstimmung von Vorlesung und Praktikum.** Die das Praktikum begleitende Vorlesung ist in den allgemeinen Vorlesungsplan des Semesters eingebunden. Dieser favorisiert eine *lineare* Vermittlung von Stoff, die aber nicht dem Bedarf in einem Praktikum entspricht: Themen wie Entwurf und Gruppenarbeit müssen vermehrt am Anfang des Praktikums vermittelt werden, während in späteren Phasen nur noch eine praktische und individuelle Betreuung nötig ist.
- **Mangelnde Vergleichbarkeit**. Die Aufgabenstellungen der Kunden unterschieden sich in Schwierigkeit, Zeitaufwand und didaktischer Eignung beträchtlich [DHZS99]. Als Folge war die von den Teilnehmern geforderte Leistung kaum objektiv zu vergleichen oder zu bewerten.
- **Schwierige Betreuung**. Die Aufgaben der Kunden erforderten tendenziell bereichsspezifische Architekturen und Algorithmen (Netzwerke, Computergrafik, Numerik, Information Retrieval), die die betreuenden Mitarbeiter und Studenten sich erst spontan aneignen mussten.
- **Schlechte Skalierbarkeit**. Die Anzahl der Teilnehmer von Praktikum zu Praktikum schwankte stark und war oft erst kurzfristig bekannt. Eine entsprechende Anzahl von Kunden und damit Aufgabenstellungen zu finden, war damit ein wiederkehrendes Problem. Eine große Anzahl verschiedener Aufgabenstellungen verschärft die Probleme der Betreuung und Vergleichbarkeit.

Aus diesen Erfahrungen heraus haben wir beschlossen, das Praktikum rundzuerneuern und komplett neu zu organisieren[1].

[1] *http://www.st.cs.uni-sb.de/edu/sopra/2004/*

2 Sechs Wochen Vollzeit

Unsere erste (und wesentliche) Entscheidung war, *das Praktikum vom sonstigen Vorlesungsbetrieb zu trennen* und in der vorlesungsfreien Zeit („Semesterferien") zu veranstalten. Hierfür gibt es eine Reihe von Gründen:

- In der vorlesungsfreien Zeit gibt es keine Konflikte mit anderen Vorlesungen oder sonstigen Verpflichtungen.
- Studierende benötigen die vorlesungsfreie Zeit nicht zur Prüfungsvorbereitung, da im Bachelor-Studiengang alle Prüfungen studienbegleitend stattfinden.
- Studierende können in der Vorlesungszeit anstelle des weggefallenen Praktikums weitere Veranstaltungen besuchen, was die Studiendauer verkürzt.

Die vorlesungsfreie Zeit im Sommer umfasst etwa drei Monate. Wir gönnen den Studierenden sechs Wochen für Urlaub und andere Aktivitäten, womit sechs Wochen für das Praktikum zur Verfügung stehen. Dies bedingt, dass das Praktikum in Vollzeit absolviert werden muss – für Studierende wie Betreuer. Vollzeit hat für die Studierenden viel Gutes, da sie den ganzen Tag vor Ort sind und sich so die Teams ganz auf die Aufgabe konzentrieren können. Auch Betreuer und Dozenten kommen zu kaum etwas anderem, was allerdings in der folgenden Vorlesungszeit ausgeglichen wird: Im kommenden Wintersemester bietet unser Lehrstuhl keine Veranstaltungen an und kann sich für sechs Monate der Forschung widmen.

Ein Praktikum in den Semesterferien konkurriert nicht mit anderen Veranstaltungen, ermöglicht volles Engagement der Studierenden und verkürzt die Studienzeit.

3 Risiken vermeiden

Ein von 13 auf 6 Wochen reduziertes Software-Praktikum muss sich auf den wesentlichen Stoff konzentrieren und lässt allen Beteiligten weniger Raum für Fehler. Anders als im semesterbegleitenden Praktikum können Gruppen kaum Fristverlängerungen gewährt werden. Eine *homogene Struktur* minimiert diese Risiken:

- **Zufällig zusammengestellte Gruppen.** Im Gegensatz zu früheren Praktika, wo die Studierenden selbst Gruppen bildeten, haben wir die Gruppen zufällig zusammengestellt. Dies sorgt für eine gleichmäßigere Verteilung der Leistungen und größere Diversität in den Gruppen. Da sich die Gruppen anfangs nicht einschätzen können, werden sorgfältiger Entwurf und gute Teamarbeit als willkommene Mittel zur Risikominimierung empfunden.
- **Wegfall der Anforderungsanalyse.** In einem 6-Wochen-Praktikum fehlt die Zeit, sich in fremde Anwendungsgebiete einzuarbeiten. Daher haben wir auf Kunden und die Analyse ihrer Anforderungen im Pflichtteil verzichtet.

- **Eine Aufgabenstellung.** Alle Gruppen bearbeiten eine identische Aufgabenstellung anhand eines Pflichtenheftes, für das zuvor eine Referenzimplementierung erstellt wurde.
- **Einheitliche, testbare Erfolgskriterien.** Es muss ein faires (und weitgehend automatisierbares) Verfahren geben, um den Erfolg festzustellen. Dies bestimmt maßgebend die Wahl der Aufgabenstellung.
- **Inhalt statt Oberflächen.** Gut gestaltete grafische Oberflächen (GUIs) können mehr als die Hälfte der Implementierungsarbeit ausmachen, sind aber gleichzeitig schwer zu testen und im Hinblick auf Ergonomie nur aufwändig zu bewerten. Für ein 6-Wochen-Praktikum können GUIs daher nur optionale Teile sein.

Ein einheitlicher, detailliert geplanter Praktikumsablauf minimiert Risiken.

4 Ablauf des Praktikums

Um die in Abschnitt 3 aufgeführte Struktur umzusetzen, haben wir das Praktikum in drei Phasen aufgeteilt, dargestellt in Abbildung 1:

- Zu Beginn erhalten die Studierenden ein einheitliches, fertig ausgearbeitetes *Pflichtenheft* mit vollständig automatisch testbaren Anforderungen.
- In den ersten zwei Wochen soll das System entworfen werden. Bereits nach einer Woche muss das *UML-Objektmodell* (Klassen und ihre Beziehungen) stehen. Meilenstein nach einer weiteren Woche ist der vollständige Entwurf, der neben einem überarbeiteten Objektmodell eine Reihe von *Standard-Szenarien* mit Sequenzdiagrammen und Unit-Tests beschreibt. Alle Entwürfe werden in Kolloquien mit den Gruppen durchgesehen.
- Nach abgeschlossenem Entwurf wird ein *überarbeitetes Pflichtenheft* ausgegeben, in dem sich einige Details geändert haben. Die Studierenden wissen zu Beginn, dass sich Details ändern können (aber nicht, welche Details das sind), und streben angesichts dieses Risikos einen möglichst flexiblen Entwurf an.
- In den folgenden zwei Wochen wird das System implementiert. Meilenstein ist hier das Bestehen eines vorgegebenen *automatischen Tests*, der die gesamte Funktionalität des Systems abdeckt. Zum Einreichen und Testen der Programme setzen wir Praktomat [Zel00] ein; Plagiaten spüren wir mit JPlag [PMP00] nach.
- In den letzten zwei Wochen wird das System eingesetzt. Ohne Kunden muss es für die Studierenden eine weitere Motivation geben, das System auf Herz und Nieren zu prüfen. Meilenstein ist daher die Qualifikation für ein *Turnier*, in dem die besten Systeme gekürt werden.

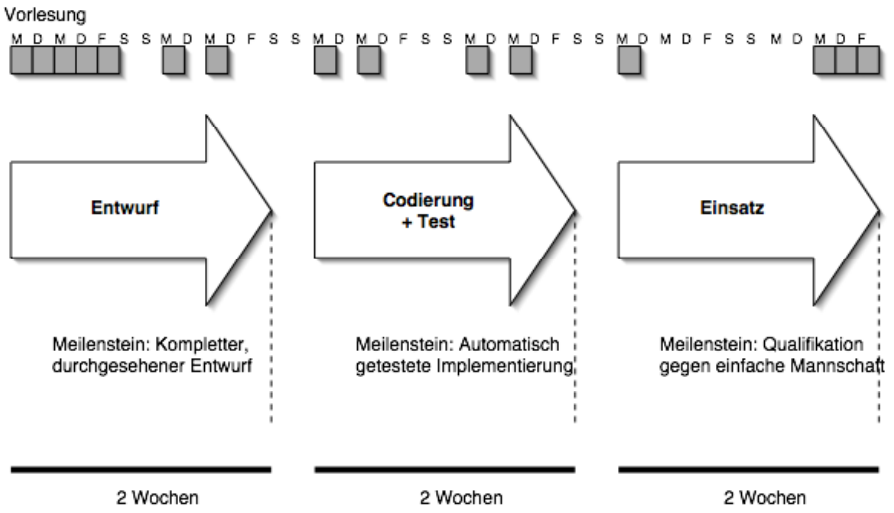

Abb. 1.: Schematischer Ablauf des Praktikums

Die Forderung nach einem objektorientierten Entwurf innerhalb von zwei Wochen bedingt eine große Menge an Stoff, der zu Beginn behandelt werden muss. Konkret haben die Studierenden in der ersten Woche jeden Morgen zwei Vorlesungsstunden absolviert – mit Themen wie Teamarbeit, objektorientierte Modellierung, Entwurfsmuster, Architekturmuster und Dokumentation. Ab der zweiten Woche fand die begleitende Vorlesung zweimal wöchentlich statt, mit dann implementierungsorientierten Themen wie Unit-Tests, Refactoring, Qualitätssicherung oder Fehlersuche. Zum Ende des Praktikums war keine Vorlesung mehr nötig. Man bemerke, dass solch unregelmäßige Vorlesungstermine in der Vorlesungszeit nur schwer durchzuführen sind; so ist es kaum möglich, den größten Hörsaal für alle Vormittage der ersten Semesterwoche zu requirieren.

Insgesamt haben wir die Vorlesung auf den Fortschritt im Praktikum abgestimmt – einerseits zeitlich, aber auch *inhaltlich:* Wir vermitteln das Material, was die Studierenden für das Praktikum brauchen, und sonst nichts. Anders gesagt: Die Vorlesung richtet sich nach dem Praktikum, und nicht umgekehrt. Anforderungsanalyse, Prozessmodelle, Spezifikation, Zertifizierung nach ISO 9000, Management von Softwareprojekten – all dies sind wichtige Themen der Softwaretechnik, aber in unseren Augen erst dann gut vermittelbar, wenn die Teilnehmer die Probleme des Programmierens im Großen selbst erlebt haben. Daher behandeln und üben wir diese Stoffe in einer Stammvorlesung „Softwaretechnik" ab dem 4. Semester.

In einem kurzen Praktikum muss man genau das unterrichten, was die Studierenden zum Erfüllen der Aufgabe benötigen.

5 Eine spielerische Aufgabenstellung

In jedem Softwaretechnik-Praktikum muss man eine Aufgabe finden, die nur im Team bewältigt werden kann. Bei uns lösen alle Gruppen die gleiche Aufgabe: In einer zweidimensionalen Welt konkurrieren zwei Schwärme von *Bugs* um Futter. Es gilt einen Schwarm zu entwickeln, der mehr Futter in sein Nest trägt als ein vorgegebener Schwarm.

Alle Bugs eines Schwarms werden durch denselben endlichen Automaten kontrolliert, der in einem sehr einfachen Maschinencode spezifiziert wird (Beispiel in Abbildung 2) – Bugs können sich bewegen, ihre Umwelt erfühlen und einfache Markierungen hinterlassen, etwa um Wege zu kennzeichnen.

Dieser Code wird von einem *Simulator* eingelesen, der die Automaten zweier Bug-Schwärme in einer gegebenen Welt ausführt und so bestimmt, welcher Schwarm erfolgreicher ist (Abbildung 3).

Zustand	Code	Kommentar
0	`sense ahead 1 3 food`	Wenn Futter voraus, weiter bei 1; sonst bei 3
1	`move 2 0`	Gehe auf Futter, weiter bei 2; sonst bei 0
2	`pickup 8 0`	Nimm Futter auf und weiter bei 8
3	`flip 3 4 5`	Suchen: Münzwurf; weiter bei 4 oder 5
4	`turn left 0`	Drehe nach links; weiter bei 0
5	`flip 2 6 7`	Münzwurf; weiter bei 6 oder 7
6	`turn right 0`	Drehe nach rechts; weiter bei 0
7	`move 0 3`	Nach vorne gehen; weiter bei 0
8	`sense ahead 9 11 home`	Rückweg: Wenn Nest voraus, weiter bei 9
9	`move 10 8`	Gehe auf Nest...
10	`drop 0`	... und lege Futter ab; Neubeginn bei 0
11	`flip 3 12 13`	Wieder mit Münzwurf suchen (wie 3-7)
12	`turn left 8`	
13	`flip 2 14 15`	
14	`turn right 8`	
15	`move 8 11`	

Abb. 2.: Ein (sehr einfacher) Bug sucht erst Futter und dann sein Nest

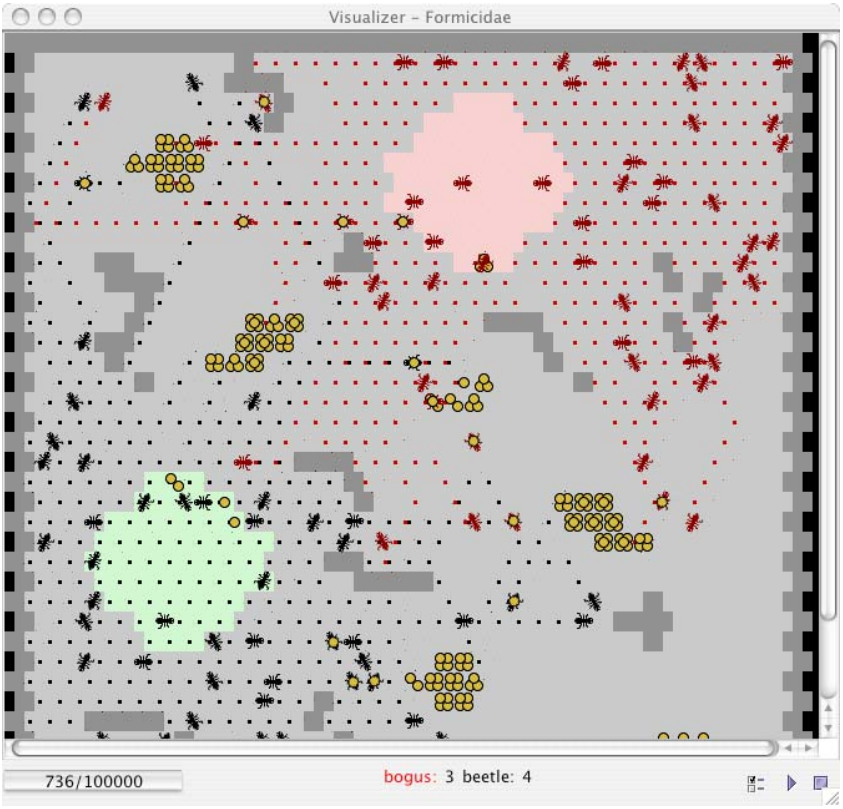

Abb. 3.: Simulator mit schwarzen und roten Bugs, Futter und zwei Nestern

Die Studierenden müssen einen solchen Simulator und einen erfolgreichen Schwarm entwickeln. Die Aufgabe vereinigt Aspekte, die unterschiedliche Kenntnisse und Talente in einer Gruppe ansprechen:

- Der *Simulator* ist eine klassische Aufgabe für einen objektorientierten Entwurf [GR83, DM91]. Er lässt sich präzise spezifizieren und validieren, was ihn für ein Praktikum gut geeignet macht.
- Die Entwicklung der *Strategie* und ihre Kodierung als endlicher Automat hingegen bieten Raum für Originalität. Die endlichen Automaten für das Verhalten werden in einer maschinennahen Sprache formuliert.
- In einem *Turnier* am Ende des Praktikums stehen die Gruppen im sportlichen Wettstreit um die besten Schwarm-Strategien. Dies stachelt Studenten zu Spitzenleistungen an und verhindert den allzu freien Austausch von Ideen und Code [H95].
- Um eine gute Strategie zu entwickeln, benötigt man *Werkzeuge* – zunächst einen korrekten und leistungsfähigen Simulator (was Testbarkeit und Qualitätssicherung motiviert). Darüber hinaus lassen sich intelligente Bugs leichter

in einer selbst entworfenen Hoch- oder Assemblersprache formulieren, die von den absoluten Adressen und dem expliziten Kontrollfluss der Maschinensprache abstrahieren. Kenntnisse der theoretischen Informatik können zur Minimierung von Automaten verwandt werden. Als Folge hängt der Turniererfolg wesentlich von der Qualität der selbst entwickelten Werkzeuge ab.

Trotz ihrer weiterführenden Merkmale ist die unmittelbar zu lösende Aufgabe überschaubar. Während ein ausdifferenzierter Entwurf etwa 30 Klassen enthält, kann ein monolithischer Entwurf mit 10 Klassen auskommen; Abbildung 4 vergleicht die Größe verschiedener Implementierungen. Eine moderate Komplexität garantiert allen Gruppen ein Erfolgserlebnis, selbst wenn die softwaretechnischen Ziele nur ausreichend erreicht werden.

Die Aufgabenstellung spezifiziert die Syntax und Semantik der Sprache für endliche Automaten und das Dateiformat der Umwelt. Die Spezifikation ist eine Mischung aus informaler Beschreibung, Grammatiken zur Definition der Syntax und Programmfragmenten in der funktionalen Sprache ML zur Spezifikation der operationalen Semantik des Simulators.

Implementierung	Code-Zeilen	Module	Methoden
Referenz. ML	1635	10	141
Referenz, C++	2223	32	134
22 Lösungen, C++, Durchschnitt	2432	24	
22 Lösungen, C++, Minimum	1146	12	
22 Lösungen, C++, Maximum	4629	45	

Abb. 4.: Größe von Implementierungen im Vergleich

In der überarbeiteten Aufgabenstellung, die die Studenten nach erfolgreichem Entwurf erhielten, wurde die Geometrie des Spielfeldes von vier- auf sechseckige Zellen geändert, ein Zustand im Automaten verboten und verschiedene Wertebereiche verändert.

Die Korrektheit jedes Simulators wird durch einen Vergleich mit einer Referenzimplementierung geprüft. Jeder Simulator erzeugt Log-Dateien, die den vollständigen Zustand der simulierten Welt nach jedem Simulationszyklus enthalten. Die Log-Dateien der Referenzimplementierung stehen den Studierenden während der Entwicklung zur Fehlersuche zur Verfügung. Eine lange Simulation von zahlreichen Bugs in einer kleinen Welt kann auch unwahrscheinliche Situationen provozieren, die so zu einer guten Abdeckung führen und mögliche Fehler aufdecken.

Eine solch vielseitige Aufgabenstellung ist zu schade, um sie nur für ein Praktikum einzusetzen. Tatsächlich ist die Aufgabe eine isomorphe Kopie der Aufgabe des ICFP Programming Contest [ICFP04]. Dort hatten internationale Gruppen 72 Stunden Zeit, einen möglichst leistungsfähigen Schwarm zu entwickeln, der gegen die Schwärme anderer Gruppen antreten soll. Während der Schwerpunkt des Praktikums auf der Implementierung des Simulators liegt, lag er im Wettbewerb auf der Entwicklung einer

Hochsprache, um leistungsfähige Strategien ausdrücken zu können. Die Tatsache, dass es im Contest 230 Gruppen schafften, innerhalb von 72 Stunden einen Simulator und einen Schwarm zu implementieren, gab uns die Sicherheit, eine trotz ihrer Vielseitigkeit nicht zu große Aufgabenstellung gefunden zu haben.

Ein Wettbewerb am Ende des Praktikums steigert das Teamgefühl und stachelt die Studierenden zu Spitzenleistungen an.

6 Betreuung

Am Praktikum nahmen 117 Studierende teil, die zufällig in 22 Gruppen zu 5 Studierenden aufgeteilt wurden [M04]. Jeweils drei dieser Gruppen waren einem studentischen Tutor zugeordnet, der das Software-Praktikum bereits erfolgreich absolviert hatte und Hilfestellung geben konnte. Aufgabe eines Tutors war es, die Erfolgschancen der Gruppe zu maximieren, dabei aber selbst nicht mitzuarbeiten – etwa wie ein Trainer im Sport. Entsprechend dieser Rolle waren Tutoren auch nicht mit der Bewertung von Leistungen befasst.

Der typische Tagesablauf eines Studierenden sah in der zwei Wochen dauernden Entwurfsphase so aus:

09:00-11:00 Uhr: Vorlesung
11:00-12:30 Uhr: Erstes Treffen mit Tutor (alle drei Gruppen gemeinsam)
13:00-17:30 Uhr: Arbeit in der Gruppe
17:30-18:00 Uhr: Zweites Treffen mit Tutor (jede Gruppe einzeln)

In den Vormittags-Treffen wurden die Ziele für den Tag festgesetzt, meist aufgrund des gerade behandelten Stoffs der Vorlesung. In den Nachmittags-Treffen wurden die Ergebnisse besprochen, und zwar für jede Gruppe einzeln. Die lange Pause zwischen den Treffen bewirkt, dass alle Gruppenmitglieder präsent sind und sich tatsächlich voll und ganz der Gruppenarbeit widmen können.

Da alle Gruppen vor den gleichen Entwurfsproblemen standen, stellten sich schnell Entwurfsmuster heraus, die zunächst von einzelnen Tutoren an andere von ihnen betreute Gruppen weitergegeben wurden, aber auch später beim Treffen der Tutoren diskutiert wurden und so ihren Weg in die Vorlesung und andere Gruppen fanden. Tatsächlich wiesen alle Entwürfe ähnliche Grundkonzepte auf.

In der Implementierungsphase trafen sich die Gruppen nur noch einmal täglich mit ihrem Tutor. Zu diesem Zeitpunkt war das Gruppengefühl schon so weit gefestigt, dass weitergehende disziplinarische Regeln nicht mehr notwendig waren. Hier gaben die Tutoren Hilfestellungen zum Programmieren und zur Fehlersuche; auch hier wurden auftretende Probleme weitergemeldet, so dass sie schnell behoben werden konnten.

In den zwei Wochen der Einsatzphase waren unsere Tutoren im Wesentlichen arbeitslos, da die Gruppen an ihren Strategien arbeiteten und die Tutoren ihnen nicht mehr folgen konnten. Stattdessen wurden die Tutoren zur Durchführung des Turniers eingesetzt.

Ständige Präsenz vereinfacht die Teamarbeit und verringert Risiken.

7 Prüfung

Ein Software-Praktikum ist eine Prüfungsleistung und soll deswegen so objektiv und fair wie möglich geprüft werden. Die Prüfung teilt sich in Teilprüfungen, die zum Abschluss jeder Phase stattfanden.

- In der Entwurfsphase beurteilte ein Prüfer den Entwurf in zwei Kolloquien mit Gruppe und Tutor. Hierbei ging der Prüfer den Entwurf mit der Gruppe durch, prüfte, ob die gesamte Gruppe mit dem Entwurf vertraut war, und gab eine Reihe von Auflagen mit auf den Weg, um den Entwurf zu verbessern. Die Prüfer sind die Dozenten und wissenschaftlichen Mitarbeiter des Lehrstuhls für Softwaretechnik; sie bieten die höchste Qualifikation für die Diskussion von Entwurfs- und Konstruktionsproblemen.
- In der Implementierungsphase musste der Simulator einen Systemtest bestehen. Einreichung und Test wurden durch Praktomat [Zel00] automatisiert und basierte auf der Referenzimplementierung, so dass hier keine manuelle Prüfung stattfand.
- In der Einsatzphase musste der selbst programmierte Schwarm ein Spiel gegen eine gegebene (schwache) Mannschaft bestehen. Dies wurde durch einen Lauf des offiziellen Simulators der Referenzimplementierung entschieden.
- Am Ende des Praktikums fand ein Turnier statt, in dem die Schwärme gegeneinander antraten. Die Teilnahme war freiwillig; es gab Buchpreise für die Gewinner sowie Sonderpreise für den schönsten Simulator und das beste Programmierwerkzeug.
- Das Praktikum wurde formal durch ein Kolloquium abgeschlossen, in dem die Gruppen ihrem Prüfer noch einmal Details des Simulators und der Strategie vorstellten; Hauptzweck dieses Kolloquiums war sicherzustellen, dass alle Gruppenmitglieder sich ausreichend beteiligt hatten, und den Studierenden Raum für Lob und Kritik zu bieten.

Auch wenn wir einen großen Teil der Arbeit automatisieren konnten, bleibt die Organisation eines solchen Praktikums nicht trivial. Neben den Tutoren waren Dozent und drei wissenschaftliche Mitarbeiter über zehn Wochen Vollzeit eingespannt – und das bei weitgehend vorgegebener Aufgabenstellung. Die sechswöchige Durchführung benötigt eine fast ebenso lange Vorbereitungsphase, in der die Aufgabenstellung, geschrieben,

eine oder zwei Referenzimplementierungen erstellt und die technische organisatorische Infrastruktur (Praktomat, Tutorials, Installation von Software) vorbereitet wird.

Tutoren sollten keinen Einfluss auf die Benotung haben. So können sie allein mit ihrer Erfahrung Ratschläge geben, die von den Gruppen besser angenommen werden.

8 Evaluation

Trotz der Umstellung lief das Praktikum problemlos. Wir freuen uns, dass wir die Risiken eines 6-Wochen-Praktikums erfolgreich reduzieren konnten – und das bei hoher Erfolgsquote:

- Alle Gruppen haben fristgerecht den Meilenstein der Entwurfsphase erreicht und einen tragfähigen objektorientierten Entwurf abgeliefert.
- Alle Gruppen haben fristgerecht den Meilenstein der Implementierungsphase erreicht und eine Implementierung abgeliefert, die alle Tests bestanden hat.
- Alle Gruppen hatten bereits fünf Tage vor Praktikumsende den Meilenstein der Einsatzphase erreicht und somit das Praktikum erfolgreich abgeschlossen.

„Alle Gruppen“ bedeutet nicht „alle Studenten“: Einige wenige Teilnehmer wurden in den ersten zwei Wochen wegen Nichtbeteiligung oder mangelnder Kenntnisse ausgeschlossen.

Die studentische Evaluation dokumentiert die Wirkung der Verbesserungen – hier im Vergleich mit dem (semesterbegleitenden) Praktikum des Vorjahres:

- Der durchschnittliche Arbeitsaufwand ist kaum verändert – statt semesterbegleitend 22 Stunden/Woche hatten wir nun 40 Stunden/Woche Vollzeit.
- Die Arbeitsbelastung war für 40% „gerade richtig“ (zuvor 0%), für 60% „zu hoch“ (zuvor 55%) und für 0% „viel zu hoch“ (zuvor 45%) – insgesamt also eine leichte Überforderung, die genau so erwünscht ist.
- Die gute Stimmung drückte sich auch in der Bewertung der Dozenten und Materialien aus, die (obwohl weitgehend identisch) in sämtlichen Fragen zur Lehreffektivität mit 0,5–0,7 Notenstufen besser bewertet wurden.
- Die größte Steigerung (1 Notenstufe) gab es in der Bewertung der Tutoren, was für unser Betreuungskonzept spricht.
- 80% wünschen sich eine zufällige oder auf Fähigkeiten basierte Zusammensetzung der Gruppen, wie bei uns eingesetzt.
- Alle Teilnehmer (mit einer Ausnahme) meinen, das Praktikum solle in der vorlesungsfreien Zeit stattfinden.

Ein Softwarepraktikum in den Semesterferien ist bei gleichem Arbeitsaufwand deutlich weniger belastend.

9 Vergleich

Der Unterschied zwischen einem Praktikum in einer semesterbegleitenden Form mit diversen Kunden und Aufgaben und einem sechswöchigen Kurs ohne Kunden und einheitlicher Aufgabe ist gewaltig. Deswegen ist es nicht immer einfach, die Faktoren für den Erfolg der neuen Form zu benennen. Einen interessanten Vergleichspunkt zwischen diesen Extremen bietet das Softwarepraktikum der TU Kaiserslautern [M04].

Das Praktikum in Kaiserslautern ist semesterbegleitend, hat aber ansonsten viele Merkmale unseres neuen Praktikums: eine einheitliche Aufgabe (Ampelsteuerung), eine Modellierungsphase mit UML, die frühe Erstellung von Testfällen, Gruppen von 6 Studierenden und ein abschließender Wettbewerb. Metzger berichtet in [M04] von einem insgesamt erfolgreichen Verlauf, gegenüber einem früheren Praktikum einem engeren und besseren Notenspektrum und einer positiven Evaluierung. Dies deckt sich mit unserer Erfahrung, dass eine einheitliche Aufgabe die Vergleichbarkeit von Leistungen fördert und der risikoärmere Verlauf zu größeren Erfolgen der Teilnehmer führt.

Wegen der Ähnlichkeit zu unserem Praktikum sind die von Metzger beobachteten Probleme interessant: 45% der Teilnehmer klagten über einen sehr hohen Arbeitsaufwand; außerdem sahen sich die Organisatoren gezwungen, Präsenzzeiten einzuführen, da Studierende dazu neigten, zu Hause zu arbeiten und dort nicht zu betreuen waren. Das Problem der Arbeitsbelastung (durch Konflikt mit anderen Verpflichtungen) deckt sich mit unseren früheren Erfahrungen und war kein Problem in unserem Vollzeit-Praktikum. Gleichzeitig bestand in unserem Praktikum eine Anwesenheitspflicht, die die Betreuung erleichterte und von den Teilnehmern positiv aufgenommen wurde. Dies legt nahe, dass der Wechsel zu einem Vollzeitpraktikum tatsächlich einige spezifische Probleme eines semesterbegleitenden Praktikums löst.

10 Fazit

Mit dem Softwaretechnik-Praktikum als 6-Wochen-Sommerkurs können wir nicht nur die vorlesungsfreie Zeit besser ausnutzen, sondern es können sich auch alle Beteiligten voll und ganz dem Praktikum widmen – sei es in der Teamarbeit, in der Betreuung oder in der Vorlesung. Enthusiasmus geht nicht zu Lasten anderer Lehrveranstaltungen; in den Semesterferien ist Abstimmung von Vorlesung und Praktikum kein Problem; eine einheitliche Aufgabenstellung für alle ist fair, reduziert Risiken und vereinfacht das Finden genereller Lösungen in der Betreuung. Schließlich hat der Wettbewerb die Motivation der Studierenden noch einmal erheblich gesteigert und so Freude an der Teamarbeit und am Programmieren im Großen vermittelt. Insofern haben wir zahlreiche Probleme früherer Praktika erfolgreich gelöst, was die Evaluation bestätigt.

Offensichtlicher Mangel eines Blockkurses ist, dass sich die Vorlesungsinhalte dem Praktikum unterordnen müssen. Erfahrungen aus früheren Veranstaltungen zeigen, dass Studierende im Praktikum genau an solchen Inhalten interessiert sind, die ihnen einen

unmittelbaren Nutzen bringen. Weiterführende Themen der Softwaretechnik sollten in eigenen, spezifischen Übungen vermittelt werden. Ohnehin greift das Verständnis für Programmieren im Großen erst, wenn entsprechende Programmier- und Teamerfahrungen vorhanden sind – und diese werden im Praktikum gemacht. Was uns allerdings schmerzt, ist der Wegfall der Anforderungsanalyse, die wir wieder ins Praktikum einfließen lassen möchten. Dies kann so geschehen, dass ein Wahlpflichtteil wie „Erstellen einer Simulator-GUI“ oder „Model Checking von Schwarm-Automaten“ von einem „Kunden“ grob vorgegeben wird und über einige Iterationen in Abstimmung mit dem „Kunden“ verfeinert wird – ebenfalls innerhalb von sechs Wochen, parallel zur Hauptaufgabe.

In [BBSZ04] lesen wir, dass „Erfahrungen im Programmieren und in der Projektarbeit sich am besten intensiv, d.h. ganztägig vermitteln lassen“. Dem können wir nur zustimmen. Wenn es weiter heißt „Die Zwänge, die sich aus den festgelegten Veranstaltungsformen und -rastern, der Raumvergabe und der Verknüpfung von Hauptstudium und Nebenfach ergeben, verhindern länger laufende Projektveranstaltungen“, so schlagen wir als pragmatische, nunmehr erprobte Lösung vor, Projekte und Praktika in der vorlesungsfreien Zeit stattfinden zu lassen.

Vorlesungsfreie Zeit ist Praktikumszeit!

Literatur

[BBSZ04] Petra Becker-Pechau, Wolf-Gideon Bleek, Axel Schmolitzky und Heinz Züllighoven: Integration agiler Prozesse in die Softwaretechnik-Ausbildung im Informatik-Grundstudium. Proc. SEUH 8, dpunkt.verlag, Heidelberg, 2004.

[DHZS99] Birgit Demuth, Heinrich Hußmann, et al.: Erfahrungen mit einem frameworkbasierten Softwarepraktikum. Proc. SEUH 6, Teubner, 1999.

[DM91] Jocelyn R. Drolet and Colin L Moodie: Object Oriented Simulation with Smalltalk-80: A Case Study. In Nelson, Kelton Clark (eds.): Proceedings of the 1991 Winter Simulation Conference, 1991.

[GR83] Adele Goldberg und David Robson: Smalltalk-80, The Language and ist Implementation, Part Three. Addison Wesely, 1983

[H95] Eva Hornecker: Präsentationstechniken und Teamtraining für das Software-Praktikum. Proc. SEUH ’95, Teubner, 1995, S. 69-81.

[ICFP04] ICFP 2004 Programming Contest. *http://www.cis.upenn.edu/proj/plclub/contest/index.php*

[M04] Andreas Metzger: Konzeption und Analyse eines Softwarepraktikums im Grundstudium. Proc. SEUH 8, dpunkt.verlag, Heidelberg, 2004, S. 41–48.

[PMP00] Lutz Prechelt, Guido Malpohl, Michael Philippsen: JPlag: Finding plagiarisms among a set of programs. Interner Bericht 2000-1, Universität Karlsruhe, März 2000.

[Zel00] Andreas Zeller: Making Students Read and Review Code. Proc. 5th ACM SIGCSE/SIGCUE Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE ‘2000), Helsinki, Finnland, Juli 2000, S. 89-92.

Eine Plattform für die Softwaretechnik-Fernlehre

Philipp Bouillon, Jens Krinke, Stephan Lukosch
FernUniversität in Hagen, 58084 Hagen
{philipp.bouillon,jens.krinke,stephan.lukosch}@fernuni-hagen.de

Zusammenfassung

*Softwaretechnik-Lehre ist ohne entsprechende Praktika und Projekte nicht sinnvoll. An der FernUniversität in Hagen ist die Softwaretechnik-Lehre auch immer **entfernte** Lehre und die Anforderungen an die Softwaretechnik-Praktika sind somit höher. Wir beschreiben eine integrierte Entwicklungsumgebung, die auf Eclipse basiert und alle Werkzeuge, die für die entfernte Softwaretechnik-Lehre nötig sind, unter einer einzigen, in sich geschlossenen Oberfläche vereint.*

1 Einführung

Die Lehre der Softwaretechnik an Universitäten ist kompliziert: Die Studierenden müssen nicht nur mindestens eine Programmiersprache lernen und die Softwaretechnik sowohl theoretisch als auch praktisch verinnerlichen, sondern sie müssen zusätzlich auch noch lernen, sich im Team zurechtzufinden. Sie müssen ihre Kommunikationsfähigkeiten verbessern und ein Verständnis für die Probleme bekommen, die sich während der Entwicklung eines großen, realistischen Projekts ergeben. Um das Verständnis für die Probleme der Softwareentwicklung zu schärfen, muss vom Dozenten eine große Projektaufgabe vorgegeben werden, die sich von den „Spielprogrammen“ aus Anfängervorlesungen allein schon durch ihre schiere Komplexität abhebt. Um die Aufgabe als ganze bewältigen zu können, werden die Studierenden in Teams eingeteilt, und es werden ihnen verschiedene Werkzeuge an die Hand gegeben, die es ihnen ermöglichen, Teile des Gesamtsystems zu meistern.

An der FernUniversität in Hagen werden sämtliche Softwaretechnik-Kurse als Fernkurse angeboten (sei es über das Internet oder auf dem Postweg); bei den Praktika in der Softwaretechnik kommt es aufgrund der Entfernung zu und zwischen den Studierenden aber zu besonderen Problemen, die gelöst werden müssen. Zurzeit müssen die Studierenden, die an einem Praktikum teilnehmen möchten, die FernUniversität besuchen und sich dann vor Ort zu einer Gruppe zusammenfinden. Die ersten Phasen des Praktikums (grobes Pflichtenheft und Grobentwurf) werden vor Ort in Hagen durchgeführt. Danach reisen die Studierenden zu ihren jeweiligen Heimatorten zurück, von

wo aus sie dann online über ein System namens CURE (Collaborative Universal Remote Education Environment) [Haa04a,Haa04b] weiter an dem Design und der Implementierung des Projekts arbeiten. CURE ist die kooperative Lernplattform der FernUniversität.

Um die Zusammenarbeit einer Gruppe während eines Praktikums zu ermöglichen, muss diese in den Bereichen Kommunikation, Koordination und Kollaboration unterstützt werden. Diese Bereiche werden häufig zur Klassifizierung von Groupware herangezogen [Teu95]. Bei der Kommunikation steht der Informationsaustausch zwischen den Gruppenmitgliedern im Vordergrund. Die Koordination dient der Abstimmung bei gemeinsamen Aufgaben. Die Kooperation schließlich fordert zusätzlich die Verfolgung gemeinsamer Ziele. Aus der oben beschriebenen Situation heraus lassen sich in den Praktika an der FernUniversität in all diesen Bereichen Probleme ausmachen:

Kommunikation. Studierende der FernUniversität benutzen in der Regel elektronische Hilfsmittel, um miteinander zu kommunizieren, also z.B. E-Mail, Instant Messaging, Chat oder auch das Telefon. Da sie aber oftmals unterschiedliche Arbeitszeiten (und Arbeitsweisen) haben, ist der überwiegende Anteil der Kommunikation *asynchron*. Eine direkte Rückmeldung auf eine evtl. dringende Frage ist somit meistens nicht möglich.

Koordination. Aus der räumlichen Distanz der Studierenden ergibt sich automatisch ein hoher Koordinationsbedarf, um die einzelnen Tätigkeiten untereinander abzustimmen und eine Zusammenarbeit zu ermöglichen. Abstimmungsprozesse werden oft durch Kommunikation oder durch gemeinsame Artefakte (z.B. gemeinsamer Projektplan) unterstützt.

Kollaboration. Da die Studierenden sich nicht in einem Computerraum treffen können, um miteinander zu arbeiten, müssen sie stattdessen wieder auf elektronische Hilfsmittel zurückgreifen. Meist werden die Studierenden dann auch wieder zu unterschiedlichen Zeiten an dem Projekt arbeiten, weshalb es wichtig ist, dass jeder Einzelne weiß, was sich seit dem letzten Zugriff am Projekt verändert hat. Außerdem muss klar ersichtlich sein, was als Nächstes zu erledigen ist.

Wir werden im Folgenden *Kooperation* als Oberbegriff für diese drei Begriffe benutzen.

Um die genannten Probleme in den Griff zu bekommen, haben wir eine einheitliche Entwicklungsplattform auf der Grundlage von Eclipse entwickelt. Diese beinhaltet sämtliche Werkzeuge, die für die entfernte Entwicklung von Softwareprojekten nötig sind. Dazu wurden bestehende Werkzeuge in die Eclipse-Plattform integriert und einige weitere Programmteile hinzugefügt, die dafür sorgen, dass das Konglomerat verschiedenster Werkzeuge reibungslos zusammenarbeiten kann. Von unserer Plattform können auch Präsenzuniversitäten profitieren, da die meisten der hier genannten Probleme der Softwaretechnik-Lehre auch dort auftreten.

Der Rest dieser Diskussion ist wie folgt aufgebaut: Abschnitt 2 stellt einen Überblick über die aktuelle Situation in der Softwaretechnik-Lehre dar. Es wird dabei beson-

ders auf die Probleme eingegangen, die wir mit unserer Plattform lösen. In Abschnitt 3 werden die Komponenten unserer Plattform genauestens beschrieben. Abschnitt 4 gibt einen Ausblick über die Dinge, die noch zu erledigen sind, während der Artikel schließlich mit Abschnitt 5 zusammengefasst und abgeschlossen wird.

2 Softwaretechnik-Lehre heute

Studierende der Informatik erlernen heutzutage bereits im Grundstudium das Schreiben eines Programms in mindestens einer Programmiersprache. Normalerweise wird in diesen Vorlesungen Wert darauf gelegt, dass die Studierenden das Konzept der Programmiersprache verstehen und somit selbstständig in der Lage sind, große Programme zu entwickeln. Um das Verständnis der *Softwareentwicklung im Großen* noch zu erhöhen, gibt es theoretische Softwaretechnik-Vorlesungen (die häufig erst im Hauptstudium stattfinden), in denen die Studierenden erstmals direkt mit der Komplexität eines großen Softwaresystems in Berührung kommen.

Die rein theoretischen Betrachtungen der Softwaretechnik reichen allerdings nicht aus, um den Studierenden ein Gefühl von den Kommunikations-, Koordinations- und Kollaborationsproblemen zu geben, die während eines Softwareprojekts auftreten können. Studierende neigen dazu, sich auf die technischen Probleme zu konzentrieren und zu glauben, dass die sozialen Aspekte erst gar nicht auftreten. Daher werden an vielen Universitäten und insbesondere auch an der FernUniversität in Hagen begleitend zu den Softwaretechnik-Vorlesungen Praktika angeboten, in denen eine Gruppe von Studierenden ein größeres Softwaresystem entwickeln muss. Um solche Praktika zu erleichtern, wird eine Reihe von Softwarewerkzeugen verwendet. Bezogen auf Eclipse gibt es bislang noch nicht viele Plug-ins, die Kollaborationsprozesse unterstützen. Es gibt zwar einige Plug-ins, die Instant-Messaging-Protokolle in Eclipse integrieren, wie z.B. das IM-Plug-in[1] oder PepeMax[2], aber bei Kollaboration geht es um weit mehr als nur um Instant Messaging. Andere Plug-ins wie Jazz [Che03] versuchen, Eclipse um Kollaborationsaspekte zu erweitern, aber Jazz konzentriert sich dabei auf synchrone Kommunikationsprozesse und auf Teams, die eng beieinander arbeiten. Solche Plug-ins sind für die Situation an der FernUniversität also nicht geeignet.

Andererseits unterstützt CURE [Haa04a,Haa04b] als kooperative Lernplattform der FernUniversität bereits die synchrone und asynchrone Kooperation. CURE unterstützt gemeinsames Lernen in verteilten Teams mittels eines Standard-Web-Browsers und des Internets. CURE basiert auf der Kombination der Raum-Metapher, die häufig zur Strukturierung der Kollaboration eingesetzt wird [Gre02, Pfi98], mit Ideen aus Web-basierten Arbeitsbereichen (Wiki) [Leu01] und Kommunikations- und Koordinationswerkzeugen. Zu den Werkzeugen zählen ein E-Mail-System, Dokumentenverwaltung und ein Kalender mit Terminfindungsfunktionen. Studierende

[1] *http://eimp.sourceforge.net/d/*
[2] *http://pepemax.jabberstudio.org/*

der FernUniversität sind bereits an dieses System gewöhnt, so dass sich eine Einbindung in unsere neue Plattform für Softwaretechnik-Projekte anbietet.

Jedes Softwaretechnik-Projekt, das im Rahmen der Fernlehre durchgeführt wird, muss als *verteilte Softwareentwicklung* (engl. DSE: Distributed Software Engineering) betrachtet werden. DSE wird bereits von Projektmanagement oder Groupware-Lösungen unterstützt, dabei reicht die angebotene Bandbreite von Kommunikations-, Koordinations- und Kollaborationsunterstützung bis hin zu vollständigen Softwaresystemen wie SourceForge[3] oder GForge[4], die bei Open-Source-Projekten zum Einsatz kommen. In der Tat stellen die beiden letztgenannten Systeme eine gute Infrastruktur zur Erstellung von großen Softwaresystemen zur Verfügung, weshalb einige Universitäten GForge benutzen. Wir haben uns jedoch dazu entschlossen, eine kommerzielle Variante zu verwenden: Intlands CodeBeamer[5] ist leichter zu bedienen und bietet gleichzeitig mehr Funktionalität.

Eine weitere große Anforderung an die neue Plattform ist die Möglichkeit für die Betreuer des Projekts, einzelne Studierende bewerten und benoten zu können. Außerdem müssen die Betreuer die Gruppe bei Problemen beraten können und ihnen Ratschläge erteilen, bevor es dafür zu spät ist. Zu diesem Zweck ist es wichtig, dass ein Betreuer die Kommunikation der Gruppe nachvollziehen kann: Sowohl die informellen projektbezogenen Gespräche als auch die eingeforderten Dokumente, die während des Projekts entstehen, müssen analysiert werden können. Außerdem muss natürlich auch das Endprodukt, also die entstandene Software jeder Gruppe, ausprobiert und bewertet werden können. Einige Werkzeuge zur Analyse von Software existieren bereits, die vom Betreuer benutzt werden können, um eine solche Analyse durchzuführen. Das JRefleX-Projekt bietet sogar Eclipse Plug-ins an, die zur Analyse der Kollaboration im Team und zur Evolutionsanalyse des Programms herangezogen werden können.

Um eine Plattform zu entwickeln, die die Kooperation der einzelnen Teammitglieder verbessert, mussten neue Plug-ins entwickelt werden und bestehende so erweitert werden, dass sie besser mit anderen zusammenarbeiten können. Welche Plug-ins das im Einzelnen sind, wird im nächsten Abschnitt beschrieben.

3 Softwaretechnik-Lehre morgen

Zu Beginn der Entwicklung einer neuen Plattform stellt sich die Frage, welche Werkzeuge in die IDE integriert werden müssen, um das Teamwork unter den Studierenden zu verbessern, obwohl sie so weit voneinander entfernt sind. Die Frage ist recht einfach, die Antwort hingegen nicht. Studierende (und auch professionelle Softwareentwickler) haben alle ihre unterschiedlichen Arbeitsweisen und Methoden, um ein Softwareprojekt voranzutreiben: Einige arbeiten lieber nachts, andere sind echte Frühaufsteher und erledigen einen Teil der Arbeit vor dem Frühstück. Manche Programmierer tendieren dazu,

[3] *http://www.sourceforge.org/*
[4] *http://www.gforge.org/*
[5] *http://www.intland.com/*

alles zunächst genau zu planen und dann zu implementieren, andere hingegen programmieren zuerst und testen danach. Wie also kann eine IDE dabei helfen, die jeweils bevorzugte Arbeitsweise eines Entwicklers zu unterstützen und ihn nicht dazu zu zwingen, eine vorgeschriebene Methode wählen zu müssen? Die wichtigen Punkte, auf die es dabei ankommt, sind *Kommunikation*, *Koordination* und *Kollaboration*. Wie oben schon angedeutet wurde, gibt es an der FernUniversität zwei wichtige Systeme für Softwareprojekte: CURE und CodeBeamer. Beide unterstützen Kooperation.

3.1 Kooperation mit CURE

CURE [Haa04a, Haa04b] ist eine kooperative Lernplattform zur Unterstützung kooperativer Lernsituationen über das Web. Benutzer können unabhängig von ihrem Standort gemeinsam auf Informationen zugreifen. Verteilte Teams können in CURE ihr gemeinsames Lernen selbst organisieren, indem sie in gemeinsamen Arbeitsbereichen Lernmaterial kooperativ bearbeiten und synchron bzw. asynchron miteinander kommunizieren. Eine Gruppe ist dabei definiert als die Menge der Nutzer eines gemeinsamen Arbeitsbereichs. Gemeinsame Arbeitsbereiche und deren Zugangsberechtigungen werden in CURE mit Hilfe der Raum-Metapher modelliert. Der Zugriff auf einen Arbeitsbereich wird durch die Schlüssel-Metapher gesteuert. Um Mitglied einer Gruppe auf einem Arbeitsbereich (d.h. Raum) zu werden, muss man einen passenden Schlüssel für den Raum besitzen. Durch die Kombination von Räumen und Schlüsseln, die an Schlüssel geknüpften Rechte und die auf Schlüssel definierten Operationen werden verschiedenste Formen der Gruppenbildung unterstützt.

So kann jeder Benutzer Räume für bestimmte Gruppen und Zwecke anlegen, und die Besitzer eines Raums können die Zugriffsrechte einschränken. Ein Raum besteht aus mindestens einer Seite, die von jedem Benutzer mit den entsprechenden Rechten mit Hilfe einer einfachen Wiki-Syntax editiert werden kann. Es ist auch möglich, TeX-Kommandos in diese Seiten einzubinden, um beispielsweise komplexe mathematische Formeln zu erzeugen. Neben normalen Seiten kann ein Raum auch Datenseiten enthalten, auf denen ein Benutzer beliebige Dateien ablegen kann. Über eine Anbindung von CURE an NetMeeting[6] oder DyCE [Tie01], ein komponentenbasiertes Groupware-Framework, ist auch eine synchrone Kooperation unter den Benutzern möglich. Außerdem kann ein Raum einen eigenen Chat und eine eigene Mailbox besitzen, deren Daten jeweils persistent gespeichert werden: Keine der Informationen, die an einen Raum gesendet werden, gehen verloren. Sowohl die Diskussionsbeiträge in den Mail-Foren des Raums als auch die Einträge im Chat bleiben erhalten.

Bisherige Erfahrungen mit der CURE-Umgebung haben gezeigt, dass die Studierenden überwiegend die asynchronen Kommunikationsmechanismen von CURE benutzen. Dies hat zwei Gründe: Erstens kommt synchrone Kommunikation an der FernUniversität selten vor (wie oben bereits erläutert wurde), und zweitens bevorzugen die Studierenden ihre eigenen, gewohnten Instant-Messaging-Programme. Die bekannten Instant-Messaging-Werkzeuge lassen sich noch zusammen mit der IDE der Studieren-

[6] *http://www.microsoft.com/windows/netmeeting/*

den anzeigen, wohingegen CURE in einem eigenen Browserfenster läuft. Um also zu den synchronen Kommunikationswerkzeugen von CURE zu gelangen, muss die IDE der Studierenden mit CURE überlagert werden: Es findet ein Kontextwechsel statt, der den Arbeitsfluss unterbricht. Dieser notwendige Wechsel führte dazu, dass die Studierenden aufgehört haben, CURE für ihre Chats zu benutzen, was wiederum dazu führte, dass die Betreuer des Praktikums keine Möglichkeit mehr hatten, auf diese Chats einzugehen. Um diesen Nachteil zu umgehen, haben wir ein CURE Plug-in für Eclipse entwickelt, das aus mehreren Sichten besteht. Ein Screenshot dieses Plug-ins ist in Abbildung 1 zu sehen. Die Hauptsicht zeigt die Seite eines Raums, während die Baumansicht auf der rechten Seite alle momentan zur Verfügung stehenden Räume anzeigt. Ein Softwareprojekt wird in einem eigenen Raum abgelegt, während die einzelnen Seiten Informationen über die Teilprojekte, Meilensteine und verschiedene andere Themen enthalten.

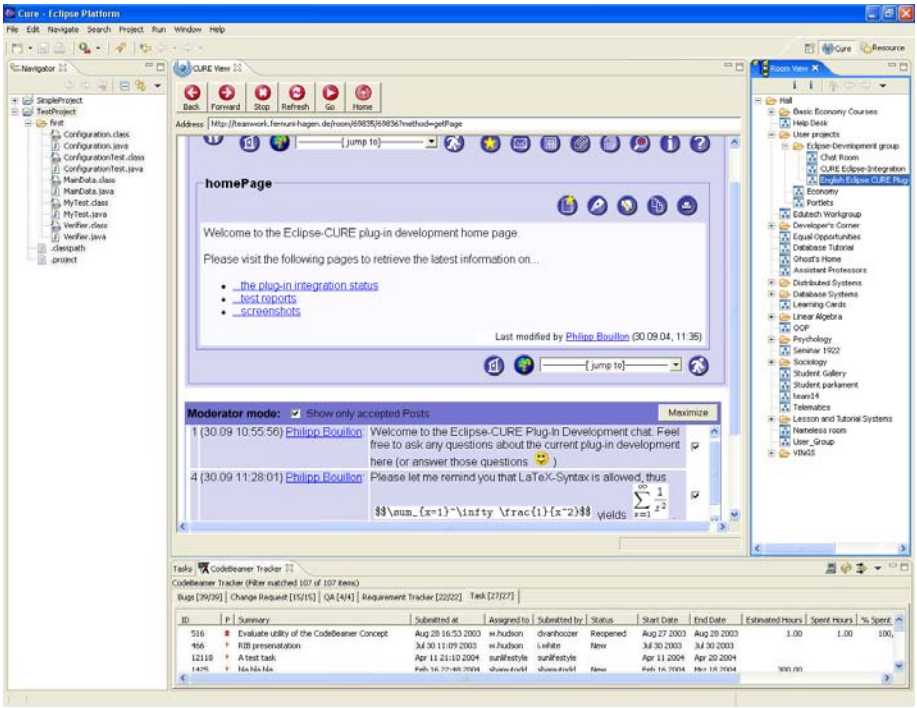

Abb. 1.: ECLIPSE mit CURE und CodeBeamer-Sicht

3.2 Projektmanagement

Softwareprojekte in der Fernlehre benötigen, genau wie andere Softwareprojekte auch, ein strenges Projektmanagement. Die Erfahrung an der FernUniversität zeigt, dass Teams, die CURE zum Projektmanagement eingesetzt haben, bessere Ergebnisse erzielt

haben als Teams, die keine besonderen Werkzeuge zur Verfügung hatten. Allerdings bietet CURE nicht die Funktionalität von existierenden Projektmanagement-Werkzeugen. Daher setzen wir zum Projektmanagement CodeBeamer ein. Dabei handelt es sich um eine serverbasierte Lösung, die eine Reihe von leicht verständlichen Kooperationshilfen anbietet. CodeBeamer unterstützt Projektmanagement durch verschiedene *Tracker*, die zum Verwalten von Anforderungen, Aufgaben, Bugs usw. genutzt werden können. Die einzelnen Trackerdaten können in Diagrammen (z.B. Gantt-Diagramme) visualisiert werden. Durch diese Tracker können sowohl die Studierenden als auch die Betreuer leicht sehen, wie das Projekt voranschreitet. Dieser Aspekt wird auch noch dadurch verstärkt, dass CodeBeamer anzeigt, welche Dinge zuletzt erledigt worden sind, bzw. welche Aufgaben als Nächstes zu bewältigen sind. Es gibt bereits ein Eclipse Plug-in von CodeBeamer, das die Tracker-Ansicht des Produkts in eine eigene Eclipse-Sicht integriert.[7] Der CodeBeamer-Dienst wird von einem Server an der FernUniversität angeboten und kann somit von jedem Teilnehmer des Softwaretechnik-Praktikums in Anspruch genommen werden. Neben dem Plug-in muss auf Studierendenseite keine weitere Software installiert werden. Abbildung 1 zeigt einige Tracker von CodeBeamer in der unteren Sicht.

3.3 Werkzeuge für die Phasen eines Projekts

Neben den zwei oben genannten „großen“ Komponenten unserer Plattform, haben wir einige bereits existierende Plug-ins für Eclipse auf ihre Verwendbarkeit in Softwareprojekten (der Fernlehre) hin untersucht. Für die einzelnen Phasen des Projekts werden unterschiedliche Werkzeuge angeboten, die wir im Folgenden vorstellen.

Phase 1: Erstellen des Projekts

Wenn ein Projekt erstellt wird, muss der Betreuer in der Lage sein, eine Projektbeschreibung zu erstellen und die Meilensteine anzugeben, zu denen bestimmte Teile des Projekts fertig gestellt sein müssen. Eine allgemeine Beschreibung des Projekts kann in einem Raum für das Projekt in CURE hinterlegt werden. Für eine detaillierte Planung können die Task-Tracker von CodeBeamer verwendet werden, wobei jeder Task einen Abschlusstermin haben kann. Es wäre wünschenswert, wenn diese Abschlusstermine automatisch in einen Teamkalender eingetragen werden könnten, der von jedem einzelnen Teammitglied zu jeder Zeit eingesehen werden kann. Außerdem könnte jeder im Team diesen Teamkalender noch um eigene teaminterne Termine erweitern. Leider haben wir kein Plug-in gefunden, das einen solchen Kalender implementieren würde. Die optimale Lösung bliebe an dieser Stelle wohl eine Integration von Eclipse mit MS Outlook oder IBM Lotus Notes, da dies die Anwendungen sind, mit denen die meisten Leute ihre Termine synchronisieren. Aktuelle Open-Source-Projektmanagement-Werkzeuge bieten ihre eigenen Kalender an, die jedoch nicht mit anderen Kalendern

[7] Ursprünglich hatten wir vor, GForge als eine kostenlose Alternative zu benutzen, aber GForge ist nicht leicht in Eclipse integrierbar.

synchronisiert werden können. Zurzeit gibt es sogar zwei verschiedene Kalender in der Universitätsplattform: einen in der Verwaltungs- und E-Learning-Plattform und einen in CURE; beide sind voneinander unabhängig und können nicht miteinander synchronisiert werden.

Phase 2: Anforderungsanalyse

Zu Beginn eines jeden Projekts beschäftigen sich die Studierenden fast ausschließlich damit, Unklarheiten zu diskutieren und Anforderungen niederzuschreiben. Um für eine lebhafte Diskussion zwischen den Mitgliedern eines Teams zu sorgen, muss sowohl synchrone als auch asynchrone Kommunikation möglich sein. Weiterhin muss es in der gleichen Umgebung möglich sein, das Anforderungsdokument zu erstellen, ohne ständig zwischen den Anwendungen zur Diskussion und zum Niederschreiben hin- und herwechseln zu müssen. Somit ist die CURE-Architektur mit den Wiki-Seiten, dem Chat und der Mail-Funktion an dieser Stelle ideal. Durch die Versionsverwaltung in CURE können auch immer ältere Versionen der einmal erstellten Dokumente abgerufen werden.

Das Team der Studierenden muss aus der Anforderungsanalyse heraus auch einzelne Tasks erstellen und diese in ihren Trackern eintragen. Die Studierenden müssen ihr Projekt also planen sowie für jede Anforderung und jede Aufgabe einen Tracker-Eintrag erstellen. Somit wird die Nachvollziehbarkeit für alle Teammitglieder und den Betreuer erhöht, da CodeBeamer alle zuletzt bearbeiteten Tasks anzeigen kann. Neben CodeBeamer gibt es auch noch andere Plug-ins, die sich z.B. zum Ziel gesetzt haben, Bugzilla in Eclipse zu integrieren.

Phase 3: Entwurf

Es gibt bereits eine Reihe von UML-Entwurfswerkzeugen, aber keines der verfügbaren Open-Source-Werkzeuge erlaubt das gemeinsame Modellieren über das Internet. In dieser Hinsicht wäre zumindest eine grafische Diff-Sicht sinnvoll, die dem Benutzer direkt anzeigt, welche Veränderungen seit dem letzten Zugriff vorgenommen wurden (zum Beispiel indem veränderte, hinzugefügte oder entfernte Elemente grafisch hervorgehoben werden). Des Weiteren sind die meisten UML-Werkzeuge nur schwer in ein Softwaretechnik-Praktikum zu integrieren, da sie entweder zu schwer zu bedienen sind oder nicht genug Funktionalität besitzen. Für unsere Zwecke am besten geeignet sind Together[8] und MagicDraw[9]. Andere Werkzeuge sind entweder (noch) nicht mit Eclipse integrierbar oder haben Stabilitätsprobleme.

Ein vielversprechendes Projekt namens GroupUML [Bou04] wird derzeit an der TU München entwickelt. Dabei handelt es sich um ein Werkzeug, mit dem mehrere Benutzer gleichzeitig halb-formale UML-Diagramme erstellen können. Zum einen ist es möglich, komplette UML-Diagramme zu zeichnen, auf der anderen Seite können aber auch

[8] *http://www.borland.com/together/eclipse/*
[9] *http://www.nomagic.com/*

Kommentare, Freihand-Zeichnungen und beliebige andere Figuren in die Dokumente integriert werden. Kurz gesagt handelt es sich also um eine Kombination aus Shared Whiteboard und UML-Diagrammeditor. Das Besondere an dem Werkzeug ist, dass alle Teammitglieder gleichzeitig sehen können, was gerade vor sich geht. Gleichzeitig können sie auch, wenn sie erst später dazukommen, nachvollziehen, was sich bereits getan hat, da Chat-Meldungen ebenfalls von dem Programm aufgezeichnet werden. Die Chat-Meldungen können gegebenenfalls sogar mit bestimmten Elementen des aktuellen Arbeitsbereichs verbunden werden. Wir werden dieses Projekt noch genauer evaluieren und dann entscheiden, ob sich eine Integration in unsere Plattform realisieren lässt.

Phase 4: Implementierung und Modultests

Eclipse ist schon hervorragend ausgestattet, um den Programmierer bei der Entwicklung von Java-Programmen zu unterstützen. Andere Sprachen werden zwar noch nicht so gut unterstützt, aber das wird sich in Zukunft ändern. Zum Testen wird JUnit schon mit Eclipse ausgeliefert. Unsere Studierenden müssen JUnit außerdem benutzen, um ihre Module zu spezifizieren. Unabhängig von der Programmiersprache gibt es bereits eine Team-Schnittstelle in Eclipse, die eine Anbindung an ein CVS-System erlaubt. An der FernUniversität gibt es einen CVS (oder SubVersion)-Server, auf den die Studierenden dann mit Hilfe der Team-Schnittstelle zugreifen können.

Zusätzlich wird an der FernUniversität CodeBeamer dazu eingesetzt, die Studierenden während der Implementierungsphase zu unterstützen. Mit Hilfe der Tracker, die in CodeBeamer zur Verfügung stehen, können sich die Studierenden einen Überblick über die aktuellen Aufgaben verschaffen und auch gegenseitig die Bugs aus den Programmteilen entfernen. Mit Hilfe der automatischen Testfunktion von CodeBeamer ist es möglich, das gesamte Projekt über Nacht testen zu lassen. Am nächsten Morgen werden dann die Tasks automatisch aktualisiert, so dass jedes Teammitglied einen messbaren Fortschritt des Projekts erkennen kann. Des Weiteren gibt es noch eine grafische Visualisierung der Testergebnisse, so dass sich jeder Studierende schnell einen Überblick über Probleme verschaffen kann.

Zur Verbesserung der Implementierungsphase sollen noch externe Werkzeuge wie JML (eine Spezifikationssprache für Java) zur Verfügung gestellt werden. Wir werden auch noch die Effekte von Code-Checking Plug-ins wie CheckStyle oder PMD sowie von Debug-Hilfen wie DeltaDebugging [Bou03] oder anderen Werkzeugen evaluieren. Gefundene Bugs werden natürlich im Bug Tracker von CodeBeamer verwaltet.

Phase 5: Abgabe und Benotung

Wenn die Studierenden am Ende ihres Praktikums ihre Software zusammen mit den geforderten Dokumenten abgegeben haben, ist es die Aufgabe der Betreuer, die Studierenden zu beurteilen und zu benoten. Es gibt verschiedene Kriterien zur Leistungsbewertung: (1) Kooperationsfähigkeiten, (2) Qualität des Designs und der Implementierung, (3) Grad der Übereinstimmung der fertigen Software mit den Designdokumenten und (4) benötigte Zeit. Um diese Kriterien aber vernünftig und gerecht bewerten zu

können, muss die Nachvollziehbarkeit für die Betreuer unbedingt gewährleistet sein. In klassischen Softwaretechnik-Projekten, bei denen einer Gruppe eine Aufgabe gestellt wird und die am Ende ein fertiges Produkt liefert, ist es nahezu unmöglich, die Studierenden am Ende individuell zu bewerten. Mit der hier vorgestellten Plattform wird das anders: Der Betreuer ist in der Lage, die benötigten Daten über die Kommunikationsbeteiligung aus der persistenten Kommunikation in CURE zu extrahieren (Chat-Protokolle, Mail-Archive, alle Versionen der Wiki-Seiten). Zusätzlich hat der Betreuer Einsicht in alle Tracker von CodeBeamer und in das CVS-Archiv. (Beispielsweise bietet CodeBeamer grafische Visualisierungen der Vorgänge im CVS-Archiv.) Außerdem kann die Architektur des fertigen Systems mit Hilfe von Reverse-Engineering-Werkzeugen mit dem ursprünglichen Entwurf verglichen werden. Ein Plug-in, das speziell für diesen Zweck entworfen wurde, ist JRefleX [Won03]. JRefleX kann sowohl eine Kollaborationsanalyse (wie hat das Team zusammengearbeitet) als auch eine Evolutionsanalyse (wie hat sich das System über die Zeit hin entwickelt) durchführen. Wir werden JRefleX noch evaluieren und es dann in unsere Plattform integrieren.

4 Erfahrungen und Ausblick

Bei der Analyse der verschiedenen Plug-ins haben wir festgestellt, dass die meisten Plug-ins gute Lösungen für eine einzelne, spezifische Problemstellung bieten. In unserer Situation reicht es jedoch nicht aus, dass eine Unterstützung in Eclipse existiert. Stattdessen brauchen wir eine Lösung, in der die einzelnen Plug-ins enger miteinander zusammenarbeiten. So gibt es zwar einige Instant Messaging Plug-ins für Eclipse, aber sie sind untereinander nicht austauschbar, da sie für diesen Zweck auf einem Framework wie Koi[10] basieren müssten.

Ähnliches gilt für Kollaborations- und Projektmanagement-Werkzeuge: Jede Lösung implementiert ihre eigenen Diskussionsforen, Kalender, Dokumentmanager usw. Aber es ist unmöglich, diese mit anderen Foren oder Kalendern zu integrieren. Das führt zu einer Situation, in der unsere Softwaretechnik-Plattform mindestens vier (!) Diskussionsforen anbietet: (1) klassische Newsgroups, (2) ein Diskussionsforum in der allgemeinen Lehrplattform, (3) die Mail- und Chat-Lösung in CURE und (4) die Diskussionsforen innerhalb von CodeBeamer. Das verwirrt natürlich und muss deshalb von *einer* allgemeineren, integrierten Lösung abgelöst werden.

CURE wurde bereits einmal alleine und wird gerade ein weiteres Mal in Verbindung mit CodeBeamer für die Unterstützung von Studierenden in Softwaretechnik-Praktika eingesetzt. Im ersten Softwaretechnik-Praktikum arbeiteten 6 Gruppen mit 5 bis 7 Studierenden an der Realisierung eines synchronen kooperativen Spiels. Die Gruppen nutzten CURE hauptsächlich zur Kommunikation und zur Dokumentation von wichtigen Entwurfsentscheidungen. Alle Gruppen berichteten, dass CURE die Gruppenkommunikation erleichtert hat. Alle Gruppen gaben zu, einzelne Arbeitspunkte und Zuständigkeiten nicht intensiv genug in CURE dokumentiert zu haben. Letzteres führte

[10] *http://www.eclipse.org/koi/*

dazu, dass die Aufgabenverteilung in den einzelnen Gruppen nicht für alle Mitglieder klar war und es so zu Missverständnissen bei der Koordination gekommen ist. Bei der Begutachtung der einzelnen Gruppenergebnisse war jedoch auffällig, dass die Gruppen mit dem höchsten Kommunikationsaufkommen in CURE die besten Ergebnisse abgeliefert hatten.

Im zweiten Softwaretechnik-Praktikum arbeiten zurzeit 3 Gruppen mit jeweils 7 Studierenden an Erweiterungen der CURE-Plattform, die synchrones kollaboratives Lernen ermöglichen sollen. Aufgrund der Erfahrung des ersten Praktikums wurde in der ersten Präsenzphase des Praktikums CodeBeamer vorgestellt und die Wichtigkeit einer klaren Aufgabenverteilung herausgestellt. Die Gruppen konnten sich frei entscheiden, ob sie zu ihrer Koordination CodeBeamer einsetzen wollten. Eine der Gruppen hat sich für die Nutzung von CodeBeamer entschieden, während die anderen Gruppen ihre Zuständigkeiten nur in CURE dokumentieren wollten.

Die Gruppe, die CodeBeamer verwendet, stellte bereits in der Präsenzphase erste Aufgaben in den Task Tracker von CodeBeamer ein und machte im Vergleich zu den anderen Gruppen in der Anfangsphase des Praktikums die schnellsten Fortschritte. Ob dies an der transparenteren Aufgabenverteilung lag, muss durch eine endgültige Evaluation geklärt werden. Nach etwa zwei Monaten stellte diese Gruppe jedoch die Nutzung von CodeBeamer ein. Die Koordination erfolgte von diesem Zeitpunkt an nur noch über einen wöchentlich stattfindenden Team-Chat. Die Gruppenmitglieder begründeten diese Entscheidung damit, dass CodeBeamer zu Beginn des Praktikums zur klareren Aufgabenverteilung beigetragen hat. Später jedoch seien die einzelnen Aufgaben und Zuständigkeitsbereiche klar gewesen, so dass zur Koordination der wöchentliche Team-Chat und die Benachrichtigungen durch CVS über den Gruppenfortschritt gereicht hätten.

5 Zusammenfassung

Wir haben gezeigt, wie wir Eclipse als die Basisumgebung für Softwaretechnik-Projekte in der Fernlehre verwenden möchten. Zur Gesamtplattform gehören neben Eclipse die beiden großen Komponenten CURE, das auch zurzeit schon die kooperative Lernplatform der FernUniversität ist, und das Projektmanagementsystem CodeBeamer. CURE wurde bereits in Eclipse integriert, wodurch der Benutzer nun innerhalb von Eclipse auf die Wiki-, Mail-, Chat- und Kalenderfunktionen von CURE zugreifen kann, ohne dazu den Browser öffnen zu müssen. CodeBeamer wird mit einem Eclipse Plug-in ausgeliefert, das die Tracker in eine Sicht in Eclipse integriert.

Erste Erfahrungen haben gezeigt, dass CURE und CodeBeamer Studierende bei der verteilten Softwareentwicklung unterstützen und deren Kollaboration erleichtert. Zusammen mit weiteren Plug-ins, die entsprechend erweitert oder angepasst werden müssen, wird diese Plattform die Durchführung von Softwaretechnik-Projekten (nicht nur in der Fernlehre) in allen Phasen erleichtern: Studierende werden in der Lage sein, einfacher und effizienter zu kollaborieren, sie werden ihr Projekt leichter verwalten können und ein besseres Endprodukt abgeben können. Da nahezu sämtliche Vorgänge

persistent gehalten werden (Kommunikation über Chat und Mail, Tracker, Versionskontrolle), wird es für die Betreuer leichter sein, einzelne Studierende separat zu bewerten.

Die vorgestellte integrierte Entwicklungsumgebung mit CURE und CodeBeamer wird zurzeit zum ersten Mal eingesetzt. Über erste Erfahrungen mit dem Einsatz der Entwicklungsumgebung haben wir berichtet. Eine abschließende Evaluation der Kollaboration in den Gruppen wird zeigen, ob Studierende weitere Hilfen und Werkzeuge benötigen und ob sie die bereitgestellten Hilfsmittel akzeptieren.

Literatur

[Bou03] P. Bouillon, M. Burger, A. Zeller. Automated debugging in eclipse. In: Proceedings of the 2003 OOPSLA Workshop on Eclipse TechnologyeXchange, S. 1-5, 2003.

[Bou04] N. Boulila, A. H. Dutoit, B. Brügge. Scoop: A framework for supporting synchronous collaborative object-oriented software design process. In: Proceedings of the 2004 ASE Workshop on Cooperative Support for Distributed Software Engineering Processes, S. 39-53, 2004.

[Che03] L.-T. Cheng, S. Hupfer, S. Ross, J. Patterson. Jazzing up eclipse with collaborative tools. In: Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange}, S. 45-49, 2003.

[Gre02] S. Greenberg, M. Roseman. Using a room metaphor to ease transitions in groupware. In: Beyond Knowledge Management: Sharing Expertise, Cambridge, MA, 2002. MIT Press.

[Haa04a] J. M. Haake, T. Schümmer, M. Bourimi, B. Landgraf, A. Haake. CURE – Eine Umgebung für selbstorganisiertes Gruppenlernen. i-com Zeitschrift für interaktive und kooperative Medien, September 2004.

[Haa04b] J. M. Haake, T. Schümmer, M. Bourimi, B. Landgraf. Supporting flexible collaborative distance learning in the CURE platform. In: Proceedings of the Hawaii International Conference On System Sciences (HICSS-37), 2004.

[Leu01] B. Leuf, W. Cunningham. The WIKI way. Addison-Wesley, Boston, MA, USA, 2001.

[Pfi98] H. Pfister, C. Schuckmann, J. Beck-Wilson, M. Wessner. The metaphor of virtual rooms in the cooperative learning environment CLear. In: *Cooperative Buildings*, LNCS 1370, S. 107-113. Springer-Verlag Berlin Heidelberg, 1998.

[Teu95] S. Teufel, C. Sauter, T. Mühlherr, K. Bauknecht. Computerunterstützung für die Gruppenarbeit. Addison-Wesley, 1995.

[Tie01] D. A. Tietze. A Framework for Developing Component-based Co-operative Applications. Dissertation, Technische Universität Darmstadt, 2001.

[Won03] K. Wong, W. Blanchet, Y. Liu, C. Schofield, E. Stroulia, Z. Xing. JRefleX: Towards supporting small student software teams. In: Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange, S. 50-54, 2003.

Eine Werkstatt zum Vermitteln objektorientierter Entwurfs- und Sprachkonzepte mit Teachlets

Axel Schmolitzky
Arbeitsbereich Softwaretechnik, Fachbereich Informatik, Universität Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg
schmolitzky@informatik.uni-hamburg.de

Zusammenfassung

Teachlets sind eine neuartige Lehrform, die ursprünglich für die Vermittlung von Entwurfsmustern entwickelt wurde. Ausgehend von lauffähigem Code wird eine Aufgabe gestellt, die gemeinsam und interaktiv von allen Teilnehmern innerhalb einer Unterrichtseinheit gelöst wird, ein Moderator bedient dabei auf Zuruf den Rechner mit Entwicklungsumgebung und Beamer. Teachlets können bereits für sich genommen als innovative Lehrform eingesetzt werden, sie können aber auch als Entwurfsgegenstand in seminarartigen Veranstaltungen dienen. In der hier beschriebenen Lehrveranstaltung zu fortgeschrittenen Konzepten objektorientierter Programmierung haben die Teilnehmer Teachlets in einer Teachlet-Werkstatt selbst entwickelt und eingesetzt.

1 Einführung

Entwurfsmuster (siehe unter anderem [GHJV95]) sind eine interessante Herausforderung in der Informatik-Ausbildung. Der Raum der Möglichkeiten zu ihrer Vermittlung kann aufgespannt werden von einer reinen Vorlesung ohne Übungen bis hin zu lose betreuten Aufgabenstellungen, die den Einsatz von Entwurfsmustern nahe legen. Da Entwurfsmuster eng mit praktischen Entwurfserfahrungen verknüpft sind, birgt eine reine Vorlesung die Gefahr, dass lediglich Schlagworte hängen bleiben. Immerhin kann in einer Vorlesung eine breite Abdeckung der wichtigsten Entwurfsmuster gewährleistet werden. Eine lose Betreuung von Aufgaben mit hohem Praxisanteil hingegen (also ein Praktikum) hat den Vorteil der individuellen praktischen Erfahrung, aber auch Nachteile: Das individuelle Feedback zu einzelnen Entwurfsentscheidungen kommt meist zu kurz, und nicht alle Teilnehmer kommen möglicherweise mit allen relevanten Entwurfsmustern in Kontakt.

Teachlets wählen einen Mittelweg zwischen Frontalvermittlung und individueller Auseinandersetzung, indem in einer speziell vorbereiteten Unterrichtseinheit alle Teil-

nehmer gemeinsam an einer Aufgabenstellung arbeiten. Der Ansatz trägt unter anderem der nachdrücklichen Aussage von Vlissides in [CHV00] Rechnung, dass Entwurfsmuster immer im Zusammenhang mit den Problemen, die sie lösen, betrachtet werden sollten. Das Konzept der Teachlets wurde ursprünglich zur Vermittlung von Entwurfsmustern entwickelt, scheint aber Potenzial für eine breitere Anwendung zu haben. In diesem Artikel werden primär Teachlets zur Vermittlung von Entwurfsmustern diskutiert, weil hier die meiste Erfahrung gesammelt wurde. Im Ausblick werden weitere mögliche Anwendungsfelder angesprochen.

Das Konzept wurde in einem Projektseminar im Sommersemester 2004 an der Universität Hamburg erstmalig vorgestellt und von den Teilnehmern sehr positiv aufgenommen. Der Veranstalter führte zu Beginn einige Teachlets selbst durch, um das Konzept zu verdeutlichen. Anschließend hat im Laufe der Veranstaltung jeder Teilnehmer mindestens ein Teachlet selbst entwickelt und an der Gruppe ausprobiert.

Dieser Artikel ist in erste Linie ein Erfahrungsbericht. Im folgenden Abschnitt wird zunächst das Grundkonzept eines Teachlets vorgestellt. Abschnitt 2 stellt ein Beispiel dar und Abschnitt 3 fasst die Erfahrungen zusammen, die bisher bei der Durchführung von Teachlets gesammelt wurden. In Abschnitt 4 wird das Konzept einer seminarartigen *Teachlet-Werkstatt* dargestellt, in der die Teilnehmer selbst Teachlets entwerfen und auf diese Weise intensiv mit dem zu vermittelnden Stoff umgehen müssen. Abschnitt 5 gibt einen Ausblick auf weitere Anwendungsfelder sowie auf Aktivitäten, die den Ansatz weiter erkunden können.

2 Teachlets

Die folgende Definition bildet die programmatische Grundlage des Teachlet-Konzepts:

„Ein Teachlet ist eine interaktive Lehreinheit, in der ein lauffähiges Stück Software um eine klar definierte Funktionalität erweitert werden soll, um ein Entwurfsmuster oder ein Programmiersprachkonzept zu veranschaulichen. Ein Moderator motiviert mit Hilfe eines Rechners und eines Beamers das Ausgangssystem sowie die vorzunehmende Erweiterung und lässt sich dann von den Teilnehmern anleiten, die dazu notwendigen Änderungen am Quelltext vorzunehmen.“

Die Grundidee eines Teachlets ist somit, dass durch den praktischen Umgang bei den Teilnehmern ein größerer Lerneffekt als mit rein theoretischen Diskussionen erzielt werden soll. Eine Betonung liegt hier auf dem Wort *interaktiv*, denn der Moderator eines Teachlets demonstriert nicht einfach nur eine praktische Realisierung, sondern motiviert die Teilnehmer zur aktiven Gestaltung einer Lösung.

2.1 Statische Struktur eines Teachlets

Die folgende vier Bestandteile sind unabdingbare Voraussetzung für ein erfolgreiches Teachlet und sollten explizit bei der Vorbereitung berücksichtigt werden:

- Ein *Ausgangssystem*: eine möglichst kleine, lauffähige Anwendung, die im Quelltext vorliegt.
- Ein *Lernziel*: Kenntnis eines Entwurfsmusters oder eines Programmiersprachenkonzepts, das praktisch kennen gelernt und verstanden wurde.
- Eine oder mehrere *Aufgaben*: Sie implizieren eine Erweiterung/Änderung am Ausgangssystem; beim Lösen der Aufgaben soll das Lernziel durch gemeinsame Programmierung erreicht werden.
- Ein *Foliensatz*: zur Darstellung von Ausgangssystem, Lernziel und Aufgabe, zur Erläuterung grundlegender Terminologie, aber auch zur Diskussion von Lösungsalternativen und Verallgemeinerungen; ergänzend sind Hand-outs möglich.

Das Ausgangssystem sollte klein genug sein, um von den Teilnehmern in 15 bis 20 Minuten erkundet werden zu können. Dabei sollte deutlich zwischen (nach außen sichtbarer) Funktionalität und (innerer) Architektur unterschieden werden. Die Funktionalität sollte klar beschrieben bzw. leicht erkundbar sein. Als sehr motivierend haben sich Ausgangssysteme mit einer grafischen Oberfläche erwiesen.

2.2 Dynamik eines Teachlets

Die Qualität eines Teachlets zeigt sich erst bei der Durchführung in einer Unterrichtseinheit. Abhängig von den beteiligten Personen können, ausgehend von derselben statischen Teachlet-Struktur, unterschiedliche Verläufe auftreten. Ein Einsatz eines Teachlets in einer Lehr- oder besser Lerneinheit wird im Folgenden als *Teachlet-Einheit* bezeichnet. Eine Teachlet-Einheit verhält sich zur statischen Beschreibung aus dem vorigen Abschnitt wie ein Exemplar zu seiner Klasse in der Objektorientierung. Eine typische Teachlet-Einheit dauert 90 Minuten.

Jede Teachlet-Einheit hat eine Anzahl an *Teilnehmern* und einen *Moderator*, der das Teachlet mit Hilfe eines Rechners (dem *Teachlet-Rechner*), an den ein Beamer angeschlossen ist, durchführt. Auf dem Teachlet-Rechner sind eine Präsentationssoftware und mindestens eine Entwicklungsumgebung installiert. Der Moderator erklärt anhand der Entwicklungsumgebung das Ausgangssystem, eventuell unterstützt durch Folien. Er erklärt außerdem das Lernziel und erläutert die Aufgabenstellung. Anschließend fungiert er überwiegend passiv und setzt die Anweisungen um, die ihm von den Teilnehmern der Teachlet-Einheit zur Lösung der Aufgabe gegeben werden. Er kann jederzeit moderierend eingreifen, wenn die Vorschläge zu sehr vom gewünschten Lernziel wegführen; er sollte jedoch auf keinen Fall die Lösung „herunterprogrammieren“. Üblicherweise hat ein Moderator eine „Choreographie“ im Kopf, nach der das Teachlet durchgeführt wird. Er sollte auch immer eine fertige Lösung in der Hinterhand haben, die er im Notfall präsentieren kann.

2.3 Eine typische Choreographie

Üblicherweise wird das Ausgangssystem zu Anfang in eine kleine Story eingebettet, insbesondere dann, wenn aus didaktischen Gründen starke Vereinfachungen gegenüber einem realistischen System vorgenommen wurden. Dann startet der Moderator die Anwendung, führt selbstständig eine möglichst kurze Demo durch und ermuntert dann die Teilnehmer dazu, ihn per Zuruf zu weiterer oder wiederholender Erkundung der Funktionalität aufzufordern. Auf diese Weise lernen die Teilnehmer bereits die Funktionalität interaktiv kennen.

Erst wenn der Leistungsumfang klar geworden ist, sollte die Architektur untersucht werden. Insbesondere hier tritt der Moderator in den Hintergrund („Für welche Klasse soll ich zuerst den Quelltext zeigen?“) und folgt den Anweisungen der Teilnehmer („Starte bitte noch einmal die Anwendung, das möchte ich ausgeführt sehen“ etc.).

Nachdem das Ausgangssystem aus Außen- und Innensicht allen Teilnehmern klar geworden ist, stellt der Moderator die (erste) Aufgabe. Idealerweise ist diese auf mindestens einer eigenen Folie formuliert. Meist besteht die Aufgabe in einer Erweiterung der Funktionalität („ein *Undo* soll möglich sein“), aber das muss nicht der Fall sein. Gerade Entwurfsmuster werden häufig dann eingesetzt, wenn die innere Architektur nicht ausreichend wartbar ist. Eine Aufgabe kann deshalb auch lauten, die Anzahl der Klassen zu minimieren oder die Austauschbarkeit eines Quelltextabschnitts zukünftig einfacher zu gestalten.

Für die sehr interaktive Implementierungsphase sollten mindestens 30 Minuten eingeplant werden. Der Moderator muss dabei immer im Auge behalten, dass genügend Zeit für eine abschließende Zusammenfassung bleibt. In dieser Zusammenfassung sollte, unterstützt durch geeignete Folien, das gerade Getane noch einmal ausführlich reflektiert werden. Außerdem sollten hier alternative Lösungen und weitere Einsatzmöglichkeiten des zu lernenden Konzepts dargestellt werden. Im Abschluss wird eine Teachlet-Einheit somit meist wieder zu einer klassischen Präsentation.

Idealerweise sollten allen Teilnehmern nach einer Teachlet-Einheit die Quelltexte der gemeinsam erarbeiteten Lösung zur Verfügung gestellt werden, damit sie bei Bedarf die Möglichkeit haben, sich noch einmal individuell mit dem Gelernten auseinander zu setzen.

3 Ein Beispiel

Um die theoretische Darstellung des letzten Abschnitt zu verdeutlichen, wird in diesem Abschnitt ein Beispiel beschrieben. Es handelt sich um ein Teachlet für die Vermittlung des *Befehlsmusters*. Es wurde in der in Abschnitt 5 beschriebenen Musterwerkstatt entwickelt und bereits einmal außerhalb dieser Veranstaltung eingesetzt.

3.1 Die Exposition

Die Teachlet-Einheit beginnt mit der Vorführung des Ausgangssystems. Abbildung 1 zeigt seine grafische Benutzungsoberfläche. Es handelt sich um ein sehr einfaches System zum Üben der vier Grundrechenarten, das in dieser Form beispielsweise für Kinder im Grundschulalter geeignet sein könnte.

Die Oberfläche besteht aus einem Fenster, das in der oberen Zeile fünf Knöpfe enthält: die Symbole der vier Grundrechenarten sowie das Gleichheitszeichen. In der zweiten Zeile folgt die Anzeige eines so genannten Ausgangswertes (zu Beginn 1) sowie eines zufällig gewählten Operanden. Es folgen zwei anfangs funktionslose Knöpfe: Der erste ist mit einem Undo-Pfeil unterlegt, während der zweite die Anzeige eines Protokolls ermöglichen soll, das in der noch leeren Listbox rechts unten darzustellen ist.

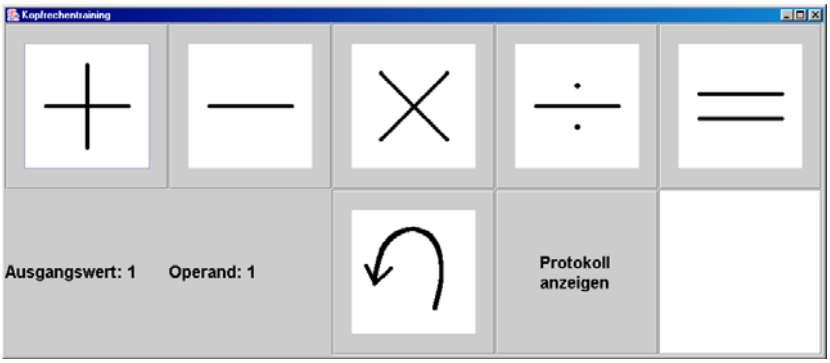

Abb. 1.: Die grafische Oberfläche des Ausgangssystems

Das eigentliche Kopfrechentraining wird nun wie folgt durchgeführt: Der Anwender wählt per Knopfdruck einen der vier Operatoren aus, sodass dieser im Hintergrund auf die „Ausgangswert“ genannte Zahl als ersten Operanden und den mit „Operand“ bezeichneten Wert als zweiten Operanden angewandt wird. Dieses Ergebnis stellt im nächsten Rechenschritt den ersten Operanden dar, während unter „Operand“ ein neuer zufällig gewählter Wert von 1 bis 9 erscheint, welcher wiederum den zweiten Operanden der folgenden Operation repräsentiert. Der ursprüngliche Ausgangswert wird mittlerweile ausgegraut, um zu verdeutlichen, dass er nicht mehr direkt in die folgenden Operationen involviert ist. Nach Klick auf das Gleichheitszeichen wird das aktuelle Resultat schließlich angezeigt, und zwar wieder schwarz unter „Ausgangswert“. Bei der nächsten Folge von Rechenoperationen stellt er folglich auch den neuen Ausgangswert dar.

Das Programm wird gestartet und der Moderator führt die Funktionalität einmal eigenständig vor. Anschließend ermuntert er die Teilnehmer, ihn bei der Benutzung anzuleiten, um mit ausreichenden Wiederholungen allen Teilnehmern die Funktionalität klar zu machen. Dieser einleitende Abschnitt sollte ca. 5 bis höchstens 10 Minuten dauern. Auf einer Folie wird die Funktionalität noch einmal zusammenfassend dargestellt.

Im nächsten Schritt erfolgt die Code-Inspektion. Das System besteht aus drei Klassen, die ausführlich und interaktiv betrachtet werden. Alle Teilnehmer sollten die interne Struktur der Anwendung verstanden haben. Auch dieser Abschnitt sollte nicht länger als 10 Minuten dauern.

3.2 Aufgabe, Lösungsvorschlag, Implementierung

Nachdem alle die Ist-Situation kennen, kann die Aufgabe gestellt werden. Es soll das noch fehlende Protokoll implementiert werden, das die geklickten Operatoren, jeweils mit ihrem Operanden, auf Knopfdruck (*Protokoll anzeigen*) darstellt. Außerdem sollen geklickte Rechenoperationen auch wieder zurückgenommen werden können. Diese Aufgabenstellung wird auf einer eigenen Folie dargestellt.

Nachdem die Aufgabe vermittelt wurde, schlägt der Moderator den Einsatz des Befehlsmusters zur Lösung vor. Die Aufrufe an Objekte der Klasse `Zwischenergebnis` mit ihren Berechnungsoperationen sollen quasi „objektifiziert“ werden. Diese Befehlsobjekte können dann gespeichert werden, in lesbarer Form angezeigt und auch gelöscht werden. Der Moderator zeigt eine Folie mit dem allgemeinen Klassendiagramm des Befehlsmusters und bildet dieses dann gemeinsam mit den Teilnehmern auf einen konkreten Entwurf an der Tafel ab. Für diesen Abschnitt sollen 10 bis 15 Minuten eingeplant werden.

Wenn sich alle Teilnehmer auf die Klassennamen im Entwurf geeinigt haben, wird der Entwurf gemeinsam implementiert. Je nach Programmierkultur kann inkrementell iterativ oder in einem großen Entwicklungsschritt vorgegangen werden, und je nach vorhandener Zeit kann auch noch ein Makrobefehl implementiert werden. Für diesen Abschnitt sollten mindestens 30 Minuten eingeplant werden.

3.3 Zusammenfassung, Abschluss

Der Entwurf und die Implementierung werden auf vorbereiteten Folien reflektiert. Auf einer Folie sind die Teilnehmer des generischen Musters den Teilnehmern der konkreten Realisierung gegenübergestellt (bei der die Namen durchaus von den in der Gruppe gewählten abweichen können). Eine weitere Folie zeigt das spezifische Klassendiagramm der Lösung, bei der auch sehr wahrscheinlich die Namen abweichen. Schließlich werden die Vorteile des Befehlsmusters und weitere Anwendungsmöglichkeiten auf Folien zusammengefasst. Dies alles sollte nicht länger als 10 Minuten dauern.

4 Bisherige Erfahrungen mit Teachlets

Bei den bisherigen Erfahrungen mit Teachlets konnten unterschiedliche Muster bei der Choreographie beobachtet werden.

Einige Teachlets wurden direkt mit dem Ausgangssystem begonnen, ohne einführende Folien zum Lernziel (keine „trockene“ Darstellung eines Entwurfsmusters zu Anfang). Die Aufgabe wurde bearbeitet und anschließend wurde „enthüllt“, dass da

gerade etwas benutzt wurde, das üblicherweise als XY (beispielsweise als Adapter) bezeichnet wird. Dies war meist dann sinnvoll, wenn das Entwurfsmuster selbst nicht sehr kompliziert ist und primär der Terminologiebildung dient.

Andere Teachlets wurden mit einer Beschreibung des Lernziels eingeleitet, weil das Entwurfsmuster zu anspruchsvoll ist, um von den Teilnehmern innerhalb der Lerneinheit selbstständig abgeleitet zu werden. Dies war bei einem Teachlet zum Interpretermuster so und würde vermutlich auch etwa beim Besuchermuster so sein. Hier wird die grundlegende Struktur des Musters zuerst vorgestellt und anschließend das Muster auf eine konkrete Problemstellung angewendet.

Abweichend von der Darstellung im letzten Abschnitt kann es auch sinnvoll sein, die Aufgabe zu stellen, bevor die Architektur des Ausgangssystems untersucht wurde. Wenn die Aufgabe sich ausschließlich auf die Funktionalität bezieht, kann es für die Teilnehmer sogar sinnvoller sein, sich problemorientiert in den Quelltext einzulesen.

4.1 Hinweise für Moderatoren

Die Moderation eines Teachlets ist anspruchsvoll; anspruchsvoller als beispielsweise ein reiner Folienvortrag. Grundsätzlich gelten die üblichen Hinweise zu Folienvorträgen auch für Teachlets, insbesondere für den Folienanteil.

Ein Teachlet wird häufig im Sitzen moderiert. Für den Folienanteil ist das möglicherweise ungewohnt, es sollte dem Moderator vorher bewusst sein. Der Moderator sollte auch den Teachlet-Rechner kennen, um Tastenkombinationen (wie Strg-v) flüssig benutzen zu können.

Aus den bisherigen Erfahrungen mit Teachlets konnten die folgenden Hinweise für Teachlet-Moderatoren destilliert werden:

- *Mache die Aufgabenstellung so deutlich wie möglich.* Wenn die Aufgabe nicht gut von allen Teilnehmern verstanden ist, kommt ein Teachlet oft ins Stocken. Idealerweise ist die Aufgabe auf eigenen Folien formuliert. Es sollte explizit nachgefragt werden, ob alle Teilnehmer die Aufgabenstellung verstanden haben; sich nicht äußernde Teilnehmer sollten explizit zum Nachfragen ermuntert werden, wenn noch nicht alles klar ist.
- *Halte die Stille aus.* Wenn bei einem Teachlet die Teilnehmer ruhig sind, dann oft deshalb, weil sie sich in den Quelltext einlesen bzw. eindenken. Das braucht Zeit, die auch gelassen werden sollte.
- *Behalte den Überblick und die Kontrolle.* Häufig gibt es ausführliche Diskussionen unter den Teilnehmern. Diese sind erwünscht, sollten aber nicht zu sehr ausufern. Notfalls müssen schlechte Beiträge knapp behandelt und langwierige Diskussionen beendet werden. Die Choreographie sollte nicht aus den Augen verloren werden. Als guter Tipp hat sich erwiesen: Die Rückkehr zum Ablauf sollte nicht mit der Vorwegnahme des nächsten Schrittes erzwungen werden, sondern mit einer Zusammenfassung des bisher Geschehenen und der Frage, wie es nun weitergehen soll.

- *Denke laut*. Es sollten alle Dinge erläutert werden, die in der Entwicklungsumgebung passieren. Alle eingesetzten Werkzeuge sollten bei ihrer Benutzung kommentiert werden („So, jetzt übersetze ich das mal“, „An dieser Stelle setze ich ein Refactoring von Eclipse ein“ etc.). Insbesondere eher ungewöhnliche oder seltene Tätigkeiten müssen erläutert werden.
- *Beherrsche den Lösungsraum*. Für eine spezifische Aufgabenstellung gibt es oft mehrere Lösungen. Der Moderator sollte gut auf verschiedene Lösungsvorschläge vorbereitet sein. Er sollte sich gute Argumente gegen Lösungen zurechtlegen, die vom Lernziel ablenken würden, aber andererseits auch offen sein für gute Vorschläge, die ihm selbst vorher nicht eingefallen sind.
- *Setze gezielt Wiederholungen ein*. Wiederholungen ermöglichen Teilnehmern, die kurz abgeschaltet haben, wieder „auf die Spur“ zu kommen. Zusammenfassungen sollten immer wieder eingestreut werden, in denen der Moderator das bisherige Geschehen knapp wiederholt. Sie können oft mit dem Satz eingeleitet werden: „Was haben wir bisher gemacht?“
- *Benutze Klassendiagramme zur Visualisierung*. Wenn ein Standardmuster aus [GHJV95] vorgestellt wird, dann sollte auf den Folien zumindest das generische Klassendiagramm des Musters aufgeführt sein. Für das konkrete System ist außerdem oft ein spezifisches Klassendiagramm wünschenswert. Je nach Teachlet ist entweder ein Diagramm des Ausgangssystems oder ein Diagramm des fertigen Systems sinnvoll.
- *Beherrsche die notwendigen Grundlagen deines Teachlets*. Wenn sich etwa für die Umsetzung des Entwurfsmusters *Iterator* die inneren Klassen von Java anbieten, dann sollte der Moderator erstens gut über innere Klassen Bescheid wissen und zweitens Folien in der Hinterhand haben, mit deren Hilfe er die notwendigen Grundlagen bei Bedarf erläutern kann.

5 Eine Teachlet-Werkstatt: Lernen durch Lehren

„I hear, and I forget.
I see, and I remember.
I do, and I understand.“ (nach Konfuzius)

Dieses Sprichwort spiegelt die insbesondere in der Informatik gültige Erkenntnis wider, dass ein dauerhaftes Verständnis vor allem durch eigenes Handeln im Problembereich gefördert wird (siehe etwa [Ell98]). Als Motivation für die eingangs erwähnte Veranstaltung zu fortgeschrittenen Konzepten objektorientierter Programmierung wurde das Sprichwort durch folgenden Satz ergänzt:

„I teach, and I really understand.“

Die Grundidee der Veranstaltung beruht somit auf der Annahme, dass eine Person, die anderen Personen ein Konzept in einer Lehrveranstaltung vermitteln muss, noch stärker zur Reflexion gezwungen ist als eine ausschließlich handelnde Person.

Die Definition des Begriffs *Teachlet-Werkstatt* in Abgrenzung zum Teachlet-Begriff lautet:

„Eine Teachlet-Werkstatt ist eine seminarartige Semesterveranstaltung, in der die Teilnehmer neue Teachlets ausarbeiten. Jeder Teilnehmer muss sich dazu mit mindestens einem Lernziel befassen und für dieses Lernziel ein Teachlet entwerfen. Die neuen Teachlets werden an den Teilnehmern der Werkstatt selbst ausprobiert und anschließend von den Teilnehmern analysiert und bewertet.“

Eine Teachlet-Werkstatt ist also deutlich gegenüber einer fiktiven Lehrveranstaltung beispielsweise zu Entwurfsmustern abgegrenzt, in der der Dozent ausschließlich mit bewährten Teachlets arbeitet. Der bereits hohe interaktive Anteil einer Teachlet-Veranstaltung wird somit in einer Teachlet-Werkstatt weiter gesteigert.

5.1 Erstmalige Durchführung

Die in Abschnitt 2 dargestellte Definition für Teachlets wurde den Teilnehmern in der Einführungsveranstaltung vorgestellt. An den folgenden beiden Terminen führte der Veranstalter zwei Teachlets selbst durch, um das Konzept zu verdeutlichen. Zu den restlichen Terminen führten die Teilnehmer nur noch Teachlets durch, die sie selbst ausgearbeitet hatten. Bei der Vorbereitung wurden sie ähnlich intensiv betreut, wie es in klassischen Seminaren oft der Fall ist: Mindestens zwei Wochen vor dem Termin musste dem Veranstalter ein Grobkonzept vorgestellt werden, spätestens eine Woche vorher mussten alle statischen Teile des Teachlets präsentiert werden.

Alle Moderationen wurden allein durchgeführt. Nach jeder Teachlet-Einheit gab es intensives Feedback auf den unterschiedlichen Ebenen: zum Lernziel selbst, zum Teachlet sowie zur Moderation. Sowohl die Teilnehmer selbst als auch der Veranstalter haben jeweils Feedback gegeben.

5.2 Beobachtungen

Der Versuch, eine Unterrichtseinheit zu einem Entwurfsmuster als Teachlet zu gestalten, führt tatsächlich zu einer vertieften Beschäftigung mit dem Lernziel. Alle studentischen Teachlet-Moderatoren haben ihren Stoff gut beherrscht.

Nicht jede Person ist unmittelbar zum Teachlet-Moderator geeignet. Voraussetzung ist unter anderem ein flüssiger Vortragsstil bereits bei Folienvorträgen. Diese Voraussetzung wurde nicht von allen Teilnehmern gleich gut erfüllt. Darüber hinaus werden aber noch weitere Soft Skills benötigt: die ständige Reflexion darüber, welcher Einwand wie nützlich für das Vorankommen ist, sowie wiederholtes Zusammenfassen des bisher Geschehenen – denn der Moderator muss jederzeit den Ablauf im Blick haben. Die Fähigkeit, starke Präsenz in einer Diskussion zu zeigen – denn der Moderator muss jederzeit den Ablauf unter Kontrolle haben. Dies sind Fähigkeiten, die jedem Software-

techniker zugute kommen. Bei der Moderation eines Teachlets wird schnell deutlich, in welchen dieser Bereiche noch Defizite existieren und an welchen Stellen somit noch gearbeitet werden sollte.

Auf der Meta-Ebene wurden etliche Diskussionen über den Programmierstil, aber auch über die jeweils verwendete Entwicklungsumgebung (im Folgenden kurz IDE) geführt. Zum Einsatz kamen *Eclipse* [Eclipse], *IntelliJ IDEA* [Jetbrains] und *BlueJ* [BlueJ]. Insbesondere über die professionell ausgerichteten Umgebungen Eclipse und IDEA gab es am Rande immer wieder kurze Diskussionen. Nach Eindruck des Veranstalters waren diese Diskussionen ausgesprochen fruchtbar, da sie unterstrichen, wie wichtig das Werkzeug IDE für den Softwaretechniker ist. Es entstand eine Atmosphäre des Austauschs über Unterstützungsmöglichkeiten durch die IDE (insbesondere für Refactorings), die in einer Veranstaltung über objektorientierten Entwurf durchaus angemessen war.

6 Bewertung und Ausblick

6.1 Das Teachlet-Konzept

Bewertung

Das Teachlet-Konzept selbst hat sich für die Vermittlung von Entwurfsmustern in Gruppen von bis zu 16 Teilnehmern bewährt. Das Feedback der Teilnehmer hat deutlich gemacht, dass ein *problembasierter Lehransatz* ([Ell98]) gerade zum Aneignen von Entwurfsmustern ausgesprochen motivierend ist. Die Vorteile einer engen Verzahnung von theoretischem Vortrag und praktischer Arbeit in der Programmierausbildung sind kürzlich auch von Kölling und Barnes in [KöB04] beschrieben worden. Das Teachlet-Konzept löst die traditionelle Trennung von Vorlesung und Übung noch weiter auf und stellt den Umgang mit Software-Artefakten in den Mittelpunkt einer hochinteraktiven Lehr- bzw. Lerneinheit. Teachlets ermöglichen somit die Gestaltung sehr effektiver Veranstaltungen.

Kölling und Barnes sprechen in [KöB04] außerdem einen interessanten Aspekt an, der auch bei Teachlets eine starke Bedeutung hat: Der Teachlet-Moderator kann als Rollenmodell dienen, das gute Praktiken der Softwareentwicklung vorlebt.

Als Nachteil von Teachlets ist zu bemerken, dass sie ausschließlich in echten Lehrsituationen evaluiert werden können. Auf diese Weise „reifen“ sie nur sehr langsam.

Ausblick

Zukünftig gilt es zu untersuchen, inwieweit die zu vermittelnden Inhalte und die Teilnehmerzahl eines Teachlets variiert werden können:

- Es wurde bisher nur wenig Erfahrung bei der Vermittlung von Programmiersprachkonzepten gesammelt, obwohl dies ursprünglich für die Teachlet-

Werkstatt vorgesehen war. Aufgrund der Wahlmöglichkeiten der Teilnehmer gab es keine Teachlets zu beispielsweise multipler (Implementierungs-) Vererbung, Generizität oder Multi-Methoden. Hier müssen weitere Erfahrungen gesammelt werden. Auch das Vermitteln von Algorithmen könnte durch den Einsatz von Teachlets interessanter gestaltet werden, insbesondere weil die Einbettung eines Algorithmus in einen Problemkontext vermutlich stark motivierend für die Studierenden ist.

- Es sollte untersucht werden, inwieweit der Ansatz skaliert, d.h. mit welcher Anzahl von Teilnehmern das Konzept maximal umsetzbar ist. Wenn die Gruppe sehr groß wird, kann sich nicht mehr jeder Teilnehmer einbringen. Trotzdem könnte der Einsatz von Teachlets auch bei größeren Gruppen fruchtbar sein, weil die passiven Teilnehmer an der interaktiven Problemlösung, die von den aktiveren Teilnehmern vorangetrieben wird, vermutlich besser teilhaben können als an einem reinen Frontalvortrag. Hier müssen noch weitere Versuche unternommen werden.

Teachlets, die sich als zielführend erwiesen haben und einen ausreichenden Reifegrad erreicht haben, sollten anderen Lehrenden zur Verfügung gestellt werden. Denkbar ist eine koordinierte Sammlung von Teachlets zu unterschiedlichsten Lernzielen, die über einen dedizierten Webauftritt zugänglich gemacht wird, ebenso wie eine Einbindung in beispielsweise MuSofT [Alf03], einem deutschen Portal für multimediales Lehrmaterial in Softwaretechnik-Veranstaltungen.

Da Teachlets bisher ausschließlich von der Person durchgeführt wurden, die sie ausgearbeitet hat, muss außerdem untersucht werden, wie Teachlets geeignet dokumentiert werden können. Ziel sollte sein, dass eine ausreichend kompetente Person ein ihr unbekanntes Teachlet mit minimalem Vorbereitungsaufwand durchführen kann.

6.2 Die Teachlet-Werkstatt

Bewertung

In einer eigenen Veranstaltung wurde das Erstellen von Teachlets in den Mittelpunkt gestellt. Die relativ starke Formalisierung des Teachlet-Konzepts hat dabei den Studierenden die Arbeit erleichtert.

Die Erstellung eines guten Teachlets stellt eine hohe Motivation für Studierende dar. Sie erfordert Kreativität für die grundlegende Idee und handwerkliches Können für die Entwicklung des Ausgangssystems. Die Durchführung des Teachlets erfordert außerdem ein souveränes Umgehen mit einer IDE samt Folien und Beamer vor einer Gruppe, dabei werden diverse Soft Skills trainiert. Und schließlich besteht die Aussicht, dass das Ergebnis einem größeren Kreis zugänglich gemacht werden könnte. Entsprechend hoch war die Motivation der Teilnehmer an der Teachlet-Werkstatt.

Ausblick

Eine Wiederholung der Veranstaltung ist für das Sommersemester 2005 geplant. Einige der Teachlets der ersten Veranstaltung werden dabei in überarbeiteter Form erneut zum Einsatz kommen.

Das Bedienen eines Rechners mit Folien und IDE erfordert viel Aufmerksamkeit, die Moderation der fachlichen Diskussion in einer größeren Gruppe ebenfalls. Es wäre deshalb denkbar, die Moderation im Paar durchführen zu lassen. Für den breiten Einsatz von Teachlets in Lehrveranstaltungen ist dies vermutlich nicht tragfähig, für eine Lehr-Werkstatt hingegen schon. In einer ersten Runde könnten die Studierenden neue Teachlets in Paaren ausarbeiten und moderieren und erst bei ihrem zweiten Teachlet allein moderieren.

7 Danksagungen

Ich danke allen Teilnehmern der ersten Teachlet-Werkstatt für ihre engagierte Mitarbeit, die das Konzept weiter bestätigt hat. Meinen Kolleginnen und Kollegen im Arbeitsbereich Softwaretechnik an der Universität Hamburg danke ich für einige fruchtbare Diskussionen über Teachlets und gute Anregungen für Abwandlungen. Den Anstoß zur Teachlet-Idee verdanke ich Heinz Züllighoven, der mir für einen unerwartet ungefüllten Seminartermin vor zwei Jahren vorschlug, eine kleine, interaktive Musterwerkstatt durchzuführen.

Literatur

[Alf03] Alfert, K., et al.: "MuSofT: Multimedia in der Softwaretechnik", Proc. Software Engineering im Unterricht der Hochschulen (SEUH), Berlin, S. 70-80, 2003.

[BlueJ] BlueJ – The Interactive Java Environment, *http://www.bluej.org*.

[CHV00] Chambers, C., Harrison, B., Vlissides, J.: "A Debate on Language and Tool Support for Design Patterns", Proc. 27th ACM SIGPLAN-SIGACT symposium on principles of programming languages, Boston, MA, S. 277-289, 2000.

[Eclipse] Eclipse – An Open Platform for Tool Integration, *http://www.eclipse.org*.

[Ell98] Ellis, A., et al.: "Resources, Tools, and Techniques for Problem Based Learning in Computing", Proc. ITiCSE '98, Dublin, Ireland, S. 46-50, 1998.

[GHJV95] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995.

[Jetbrains] IntelliJ IDEA – An Intelligent Java IDE, *http://www.jetbrains.com/idea*.

[KöB04] Kölling, M., Barnes, D. J.: "Enhancing Apprentice-Based Learning of Java", Proc. SIGCSE 36, Norfolk, Virginia, S. 286-290, 2004.

Using Personal Software Process exercises to teach process measurement

A. Inkeri Verkamo, Asko Saura
Department of Computer Science, P.O. Box 68, FIN-00014 University of Helsinki
verkamo@cs.helsinki.fi, as@iki.fi

Abstract

Software process measurement is an essential skill to be taught to future software engineers. It is not adopted through classroom teaching, unless the students are required to try out the suggested methods, in order to experience both the advantages and the pitfalls of process measurement. The Personal Software Process (PSP) is a well known tool for this purpose. In our university we have used PSP as a part of our graduate course for teaching process measurement. Our experience and the feedback from our students suggest some changes both in the exercises and in the process itself.

1 Introduction

It is generally admitted that the basis of successful software process improvement is measurement. Any changes in the software process must be preceded by measurements of the present process, to give a basis for evaluating the effects of the change. On the other hand, measurements of the software process are not necessarily reliable or comparable unless the process itself is well defined and rigorously followed.

Students specializing in software engineering need practice in measurement. Even though many of them have some experience in real world software development, they have seldom observed process measurement in the industry, and it is even rarer that such measurements would be carried out in a systematic way. To point out both the usefulness of measurement and the problems in its rigorous implementation, the curriculum of future software engineers should include some practical exercise in process measurement.

One possible tool for teaching process measurement is the Personal Software Process (PSP) [Hum95] advocated by CMU-SEI. By doing the set of programming exercises and the corresponding process measurements the students are expected to detect and address the most important problems in their personal process, at the same time learning a systematic way of process improvement. PSP is taught in courses

offered by CMU-SEI but the course textbook [Hum95] describes the method in adequate detail that it can also be applied independently.

In the University of Helsinki, we use PSP as a part of our graduate course for teaching process measurement. Our experience and the feedback from our students suggest some changes both in the exercises and in the process itself. In this report we describe the experience we have gained in using PSP, based on two master's theses that analyzed the measurements and the self-evaluations of the students [Ros03, Sau04].

2 Personal Software Process exercises

The material provided in [Hum95] is organized as ten programming exercises that are performed using the steps of PSP. In each exercise, the student collects measurements of his/her own process, such as lines of code, defects detected, and time spent in each phase. The measurements are analyzed and used for predicting corresponding values in the subsequent exercises. The number of measurements and predictions and the analysis increases with the levels of PSP in seven steps ranging from PSP0 to PSP3. In this way the student is expected to gain more understanding of his/her own process, to detect those phases of the process that are most time consuming or failure prone, and hence to be able to improve his/her personal process where it is most needed. During the course the student also produces five reports that define standards to guide his/her work and analyze the process. The exercises, process levels, and reports in PSP are shown in Table 1.

Table 1: Exercises, process levels, and reports in PSP [Hum95]

Exercise	Process	Report
1. Standard deviation	PSP0	
2. LOC counter	PSP0.1	R1 LOC counting standard R2 Coding standard
3. LOC and object counter	PSP0.1	R3 Defect analysis report
4. Linear regression	PSP1	
5. Numerical integration	PSP1.1	
6. Prediction interval	PSP1.1	R4 Midterm report
7. Correlation	PSP2	
8. Sorting	PSP2.1	
9. Chi-squared test	PSP2.1	
10. Multiple regression	PSP3	R5 Final report

The programs developed in the exercises are small statistical routines that can be used for analyzing the measurement data. The students may use any programming language that they are familiar with. They are advised to develop the software using their usual development process, extended with the measurements required in each step of PSP, so that the measurements and predictions apply to their personal way of developing software. However, since the measured phases are those that are typical to the traditional waterfall model, combining the measurements with a strongly iterative process like XP might require some redefining of phases and even adoption of more suitable measures (e.g., number of iterations). – For a more detailed description of PSP, see [Hum95].

Attempts to introduce PSP as an off-the-shelf process have been mixed successes. Industry practitioners report that they see value in the ideas presented in PSP – planning based on collected data, error prevention, etc. – but when they have the chance, they either do not start using PSP in the first place or quickly cease to use major parts of the process [Fer97, Mor00]. If the data collection and analysis overhead and the context switches from development to process analysis hamper adoption it would be useful to integrate data collection with the development tools [Joh02]. It has even been suggested that existing team processes do not produce the kind of development tasks that are easily structured as PSP projects or measured by PSP measures [Mor00].

3 PSP exercises in the University of Helsinki

In the University of Helsinki, students majoring in computer science can select software engineering as their field of specialization. As part of their specialization studies they can measure their own software process along the lines of PSP. This is offered as an alternative way of accomplishing the course *Software Processes and Quality* [Sof04], where one of the main goals is to introduce the concepts of process quality and measurement. The course gives a short introduction to PSP and TSP (Team Software Process) [Hum99] as examples of process models that rely heavily on measurement. The duration of the course is only eight weeks, and accordingly a slightly modified set of PSP exercises is used, consisting of seven programming exercises and four reports, leaving out the original exercises 7, 8, and 9 and report R4 (see Table 1).

During the years 1999–2003, a total of 164 students took the course, and 75 of them accomplished it with the PSP exercises. In the first year of the course, only a few students finished their PSP exercises, but we have gradually increased the amount of tutoring, and recently about half of the students take the PSP exercises, instead of accomplishing the course in the more traditional way of classroom exercises and exam. It can be assumed that the PSP students are typically practically rather than theoretically oriented and experienced in programming.

4 Experience

In our study, we were particularly interested in observing the effect that PSP might have on the quality of the students' software process. As a basis for our research hypotheses, let us first give a brief overview of some previously reported results.

Various sources report a declining defect density on their PSP courses [Hum96, Hay97, Abr02]. This means that students learn to produce fewer defects in the first place, which is a major PSP objective. During the first few exercises, as the students proceed from PSP0 to PSP1, the defect density declines sharply, by over 25% [Hay97]. Later, when moving from PSP1 to PSP2, the decrease is only slight. In addition to lowering the number of defects, PSP aims to address their impact by finding defects early on. Initially, over 90% of all defects are found and removed during the late phases: compile and test [Hay97]. After code and design reviews are introduced in PSP2, up to half of all defects are found in the early process phases [Hay97].

Several studies suggest improvement either in the estimation accuracy of program size or effort [Hum96, Hay97, Kam00, Pre01], but contradicting observations have also been reported [Abr02]. The PSP course mandates frequent process changes (seven out of ten new programs are written with an altered process), which means that the previously collected data quickly loses its value as a basis for predicting future performance.

PSP adds a considerable load of new tasks to each programming assignment. These include keeping records, conducting reviews, and making predictions and postmortem analyses. While these new tasks initially lower productivity, they are expected to pay off in continuously increasing productivity once the programmer has learned to make use of the tools [Hum95, p. 19]. Students have reported frustration on the amount of bookkeeping [Pre01, Abr02]. During the PSP course, students are just learning to use the process, which means we should expect their productivity to decrease while they are adding new process elements to their work. Surprisingly, many sources report a basically unchanging average productivity over PSP courses [Hay97, Pre01, Abr02].

We formulated the following hypotheses of the expected results of our PSP course:

H1 *Over the PSP course, defect density decreases.*

H2 *The most important defect removal phase changes during the course: In the beginning, most defects are removed in compile. Later on, testing and reviews will account for removing the most defects.*

H3 *Over the course, both size and effort prediction accuracy will increase.*

H4 *Despite the process changes, productivity will stay roughly the same.*

We analyzed the coursework reports of 36 students from our three first PSP courses (1999–2001). The data collected by students was of poor quality, as has also been noted elsewhere [Joh98]. Before analyzing the data, we manually corrected many obvious errors in the reports.

As expected in hypothesis H1, defect density decreased over the course. We calculated the density in defects per KLoC (Figure 1). Median defect density decreased constantly. Strong decrease in defect density occurred at exercises 4 and 10. Exercise 4 was the first exercise to use PSP1 and exercise 10 was the first and only exercise to use PSP3. However, as PSP1 only adds the size estimation tool PROBE [Hum95] to the process, it is unlikely that the decrease of defect density in exercise 4 was caused by this process change. We suspect that the very sharp drop on exercise 10 may result from it being a very big exercise and the last one, so that the students were just tired of the whole thing. Another explanation is that they used the reviews adopted in exercise 6 to avoid some defect types.

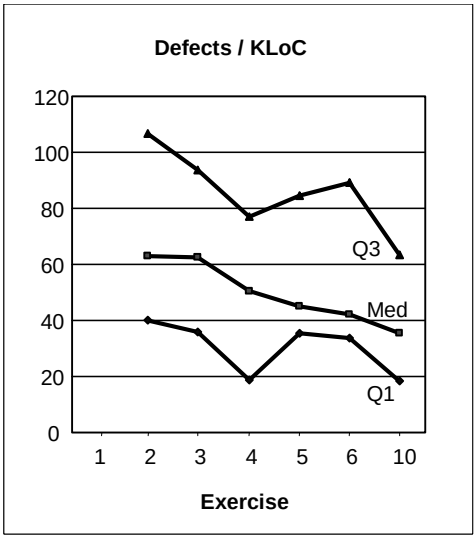

Figure 1: Defect density by exercise, in defects / KloC.

Surprisingly, hypothesis H2 proved to be false. Contrary to the previous reports, the compile phase accounted for about half of all removed defects during the entire course (Figure 2). Introducing reviews in exercise 6 sharply decreased the amount of compile defects, but the compile phase still remained on top. At the same time, defects found in testing were noticeably less. Perhaps our students would require more coaching on review methods.

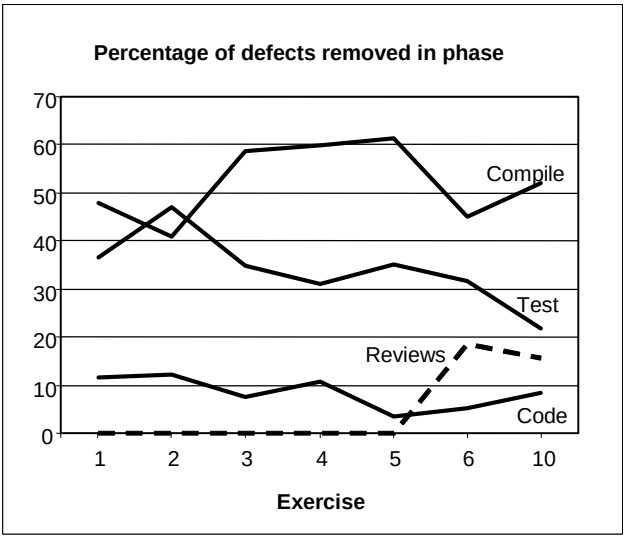

Figure 2: Percentage of defects removed by phase.

Both effort and size estimation accuracy increased clearly during the course (Figure 3), especially after the size estimation tool was introduced in exercise 4. After exercise 4, only exercise 6 shows major underestimation of effort, possibly because it introduces the time-consuming reviews to the process. Many students heavily underestimated the size of exercise 10 which was considerably larger than all previous exercises.

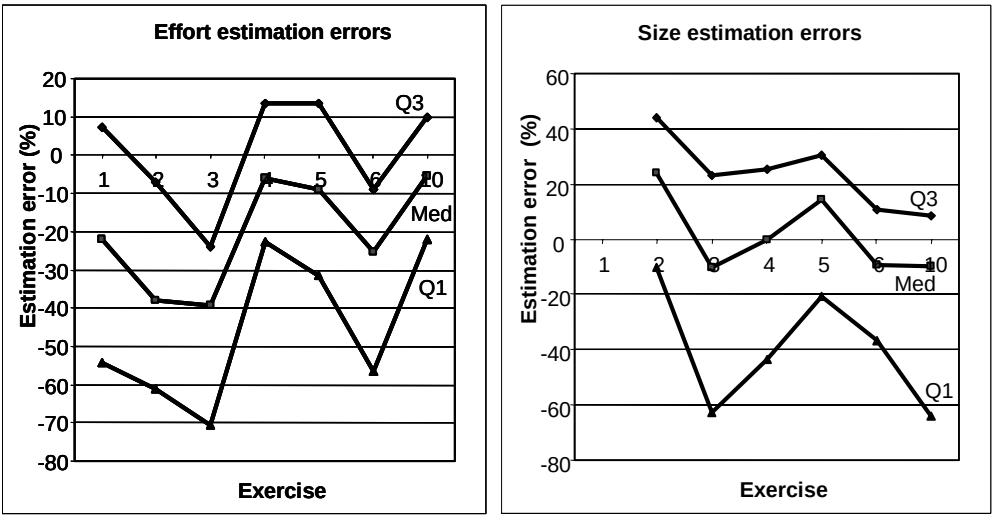

Figure 3: Estimation errors by exercise.

Hypothesis H4 expected productivity to stay roughly constant for the duration of the course, but this proved false (Figure 4). There is a noticeable drop in average productivity when PSP1 is introduced (exercises 4 and 5). This may be due to the time-consuming size prediction and paperwork. The average productivity stays low, except for the last exercise, where, as we previously noted, students may not spend too much time searching for the last few defects. Moreover, the additional PSP1 process elements seem to slow down the most productive students more than the others (Q3 in Figure 4).

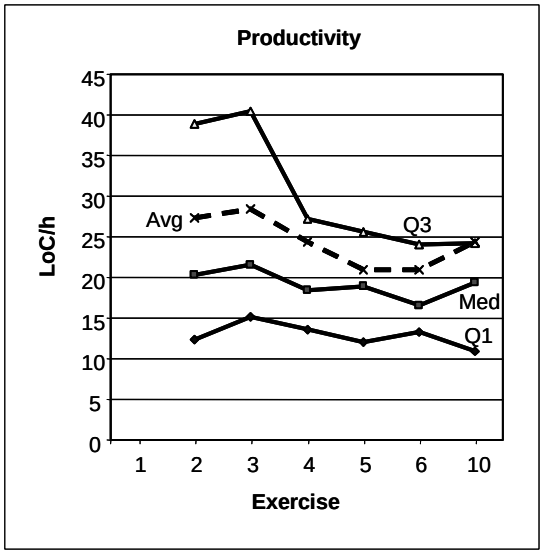

Figure 4: Productivity by exercise, in LoC per hour.

In their final reports, the students evaluated the PSP exercises and their own learning during the course. They considered the series of exercises a good introduction to a measurement based process, but to be really valuable, more exercise is needed: many students were disappointed with their own prediction skills and their slow (or nonexistent) improvement. Nevertheless they felt that by guiding them to plan their entire process, instead of just starting to code from some part of the task, the method could potentially help them in learning to make better estimates. Reviewing one's own work was considered difficult and even unmotivating, but still several of the students mentioned reviews as one of the tools that they might consider using in the future.

5 Future plans

Based on our experience we consider PSP a useful small scale tool for teaching process measurement. The basic measurements (time, size, and number of defects) are easily

comprehensible to the students. The series of exercises is long enough to allow some learning, and also to show some basic difficulties in starting a measurement program.

The main drawback of the PSP exercises is the continuously changing process. In our course, seven exercises are performed using six different processes (in the original course, ten exercises using seven processes), which means that in almost every exercise the student is required to learn some new measures. In consequence, the measurement data collected during the exercises represent the student's process in a learning phase rather than in actual use. When predictions of future performance are based on this data the result may be haphazard. As we observed, some students grasp the new concepts quickly and have stable results, while others may have one or several exercises that are too much influenced by transient difficulties to be useful at all. The small number of measurements emphasizes this problem, since there is little room for discarding even obvious outliers from the small collection of historical data.

To be useful for collecting historical data the process should be the same from one measurement to the other. Hence we plan to change our set of exercises to use only two processes. In the first 4 exercises the students use their own basic process with added measurements for size, time, and defects. The measurements on these exercises form the historical data that is analyzed to define a basis for process improvement. The analysis should reveal the most time consuming and/or most error prone phases of the process, and the student can then choose one of several strategies to improve his/her process. The last 3 exercises are then performed using the modified process, and again the collected measurement data can be used to analyze the effect of the change.

A second source of dissatisfaction was the content of the programming exercises. The software produced in the exercises consists of statistical software, which is very different from the kind of programs that the students are used to working on. Moreover, statistical software packages exist in abundance, so that writing yet another piece of software for this purpose is not motivating. The advantage of using (and testing) your own software in doing the statistical calculations needed for measurement analysis is outweighed by the difficulty of understanding the concepts under a strained schedule. We plan to change the topic of the exercises into something more familiar to the students. As before, the exercises should form a continuum that allows and encourages reuse of previous work, at the same time producing software that the students might have a chance of using even later in their studies or work.

Finally, a strong point of criticism that we feel necessary to address is the lack of measurement and analysis tools. The original collection of paper sheets where students write down their measurements by hand has already been replaced by web pages, both in our course and elsewhere [Joh98, Mor00, Bor02]. Likewise the students are provided Excel sheets with calculations of the various quality measures. A more controversial topic is to what extent the analysis should be automated. Certainly when this kind of analysis is used routinely in the industry, the more it can be automated the better [Mor00]. On the other hand, at the learning phase it is crucial to understand where the numbers come from: we have numerous examples of students who will accept any value computed by the Excel sheet, no matter how unreasonable it may be.

Nevertheless we hope to improve the tool assistance offered for PSP, by providing simple tools for collecting the measurements.

6 Conclusion

We feel that in spite of certain weaknesses, PSP forms a useful basis for teaching the art and practice of process measurement to future software engineers. While the students may find detailed and rigorous bookkeeping and analysis tedious, they are also rewarded with an understanding of how effective even simple means of prediction are in software development. With the improvements suggested above we expect the PSP exercises to be even more useful in improving the students' awareness of the methods of process improvement.

References

[Abr02] P. Abrahamsson, K. Kautz: Personal Software Process: classroom experiences from Finland. In: J. Kontio, R. Conradi (ed.), Software Quality – ECSQ 2002, number 2349 in Lecture Notes in Computer Science, pages 175-185. Springer. June 2002

[Bor02] J. Börstler, D. Carrington, G. W. Hislop, S. Lisack, K. Olson, L. Williams: Teaching PSP: challenges and lessons learned. IEEE Software, 19(5):42-48. 2002

[Fer97] P. Ferguson, W. S. Humphrey, S. Khajenoori, S. Macke, A. Matvya: Results of applying the Personal Software Process. Computer, 30(5):24-31. 1997

[Hay97] W. Hayes, J. W. Over: The Personal Software Process (PSP): an empirical study of the impact of PSP on individual engineers. Technical Report CMU/SEI-97-TR-001. Carnegie Mellon University, Software Engineering Institute. 1997

[Hum95] W. S. Humphrey: A discipline for software engineering. Addison-Wesley. 1995

[Hum96] W. S. Humphrey: Using a defined and measured Personal Software Process. IEEE Software, 13(3):77-88. 1996

[Hum99] W. S. Humphrey: Introduction to the Team Software Process. Addison-Wesley. 1999

[Joh98] P. M. Johnson, A. M. Disney: The Personal Software Process: a cautionary case study. IEEE Software, 15(6):85-88. 1998

[Joh02] P. M. Johnson, H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen, W. E. J. Doane: Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined. Proc. 2003 International Conference on Software Engineering. Portland, Oregon. 2003

[Kam00] J. Kamatar, W. Hayes: An experience report on the Personal Software Process. IEEE Software, 17(6):85-89. 2000

[Mor00] M. Morisio: Applying the PSP in industry. IEEE Software, 17(6):90-95. 2000

[Pre01] L. Prechelt, B. Unger: An experiment measuring the effects of Personal Software Process (PSP) training. IEEE Transactions on Software Engineering, 27(5):465-472. 2001

[Ros03] R. Rossi: Adoption of the Personal Software Process (in Finnish). Technical Report C-2003-68. Department of Computer Science, University of Helsinki. 2003

[Sau04] A. Saura: Software fault density and workload estimation in the PSP (in Finnish). Technical Report C-2004-28. Department of Computer Science, University of Helsinki. 2004

[Sof04] Department of Computer Science, University of Helsinki: Software Processes and Quality, course home page. *http://www.cs.helsinki.fi/u/verkamo/swpr/swprs2004_eng.html*

Erfahrungen mit XP

Doris Schmedding, Ingrid Beckmann
LS Software-Technologie, Fachbereich Informatik, Universität Dortmund, 44221 Dortmund
{Doris.Schmedding, Ingrid.Beckmann}@udo.edu

Zusammenfassung

In einem Software-Praktikum hat ein Studierendenteam versuchsweise unter XP gearbeitet, während drei Gruppen einer phasenorientierten, modellbasierten Vorgehensweise folgten. Wir stellen Ergebnisse unsere Beobachtungen und Befragungen vor und ziehen ein Fazit für unsere weitere Arbeit.

1 Einleitung

Das eigene Ausprobieren aktueller Trends in der Softwaretechnik bietet Lehrenden die Chance, nicht nur ganz neue Erfahrungen mit den neuen Techniken zu gewinnen, sondern auch die bisher praktizierten Vorgehensweisen zu verbessern, indem Konzepte des neuen Vorgehens mit den bisher praktizierten und bewährten Verfahren verknüpft werden. Die gründliche Auseinandersetzung mit dem extrem anderen kann viele nützliche Erkenntnisse liefern.

E**x**treme Programming (XP) [Beck00] ist in den neunziger Jahren aus der Praxis als Gegensatz zur dokumentationslastigen Vorgehensweise bei der Softwareentwicklung entstanden, wie sie typischerweise heute an den Universitäten gelehrt und gelernt wird. Während die iterative und inkrementelle Vorgehensweise XP weitgehend auf Dokumentation verzichtet, soll in der konventionellen Vorgehensweise das Verständnis des Entwicklerteams von der Aufgabe und ihrer Lösung durch ein gemeinsam erstelltes, ausführlich dokumentiertes Modell erreicht werden, das heute meist in UML [BRJ99] notiert wird. Bei XP dient der Code selbst als Dokumentation.

Dem UML-basierten Vorgehen liegt die Vorstellung zugrunde, dass, wenn man das Problem nur gut genug analysiert und die Lösung genau genug spezifiziert, die Umsetzung des Modells in ein Programm ganz einfach wird und sich fast von allein ergibt. Es werden mächtige Werkzeuge eingesetzt, die aus Modellen Code-Rahmen erzeugen, in denen nur wenige Zeilen Code ergänzt werden müssen. Bei XP dagegen ist der Stellenwert der Programmierung als Entwicklungstätigkeit und die Qualität des erzeugten Codes besonders hoch. Gute Teamarbeit und die Integration des Kunden in den Entwicklungsprozess werden als Grundlage für erfolgreiche Projekte angesehen.

Um die Vor- und Nachteile alternativer Vorgehensweisen bei der Softwareentwicklung klarer erkennen zu können, wurde in studentischen Projekten im Rahmen des Software-Praktikums (SoPra) an der Universität Dortmund und auf der Informatica Feminale (IF), der Bremer Sommerschule für Informatikerinnen (*www.informatica-feminale.de*), mit XP experimentiert. Auf der IF wurde mit 10 Teilnehmerinnen ein einwöchiges XP-Projekt durchgeführt. Das Software-Praktikum ist eine Pflichtveranstaltung im Informatik-Grundstudium, in dem die Studierenden im Team von 6 bis 8 Studierenden entweder ein Semester oder sechs Wochen lang in einer Blockveranstaltung Softwareentwicklungs-Projekte durchführen. Insbesondere hat uns als Lehrende die Frage interessiert, wie auf sinnvolle Weise die XP-Konzepte einführt werden können, wie Programmieranfänger damit zurechtkommen und welche Techniken von XP besonders nutzbringend sind.

Im Anschluss werden zunächst ähnliche Projekte zur Einführung von XP vorgestellt, unser Versuchsrahmen beschrieben und unser Einstieg in XP erläutert. Dann wird genauer auf die für die Lehre wichtigen Merkmale von XP eingegangen, der Unterschied zur UML-basierten Vorgehensweise herausgestellt und unsere Ergebnisse in den Experimenten mit XP vorgestellt. Aus den positiven und negativen Erfahrungen werden Konsequenzen für die weitere Ausbildung im Software-Praktikum gezogen.

2 Die Einführung in XP

Newkirk und Martin beschreiben in [NeMa01] sehr anschaulich ein reales Pilotprojekt zur Einführung von XP in einer Softwarefirma. Keine der beteiligten Personen, alle erfahrene Softwareentwickler, hatte zuvor in einem echten XP-Projekt gearbeitet. Die lebhaften Schilderungen der positiven Erfahrungen und Fehler haben unsere Neugier geweckt, XP selbst auszuprobieren.

Auch an den Universitäten wird mit XP experimentiert. Williams und Upchurch [WiUp01] diskutieren, welchen positiven Einfluss die Konzepte von XP in der Softwaretechnik-Ausbildung haben könnten. Der tatsächliche Einfluss auf den Wissensstand der Studierenden lässt sich schlecht messen, aber ebenso wie wir konnten sie feststellen, dass die im XP-Kurs erzeugten Programme eine hohe Qualität besitzen.

Lippert et al. [LRWZ01] beschreiben ebenfalls den Einsatz von XP in einem Grundstudiums-Praktikum. Zwei Praktikumsgruppen, die beide nach der XP-Vorgehensweise arbeiteten, werden miteinander verglichen. Die Projektteams unterscheiden sich in erster Linie dadurch, wie viel Einfluss die Betreuer auf den Ablauf des Projekts genommen haben. Entsprechend mehr oder weniger chaotisch lief der Entwicklungsprozess ab.

In [MSK04] berichten die Autoren, welche Konflikte bei der Simulation betrieblicher Abläufe in einer Lehrveranstaltung auftreten können. Als ein Ergebnis der beschriebenen XP-Praktika an der Universität Bonn ist ein Rollenkonzept für die Lehre von XP an der Universität entstanden.

Angeregt durch die Beispiele wollten wir eigene Erfahrungen mit XP sammeln. Die überwiegend positiven Erfahrungen auf der Informatica Feminale haben uns ermutigt,

direkt im Anschluss XP auch im Software-Praktikum auszuprobieren. Das Praktikum fand als sechswöchige ganztägige Blockveranstaltung statt. Eine von vier Arbeitsgruppen, bestehend aus 6 Studierenden, führte ihr zweites Projekt nach der XP-Vorgehensweise durch, während die anderen nach einer UML-basierten Vorgehensweise [Schm01] arbeiteten. Das erste Projekt führten alle Gruppen nach der UML-basierten Vorgehensweise durch, die in der dem Praktikum vorangehenden Softwaretechnik-Vorlesung gelehrt wird. Dass nur eins der vier Praktikumsteams XP ausprobieren durfte, hängt mit dem großen Betreuungsaufwand in einem XP-Projekt zusammen. Ein XP-Coach, in diesem Fall die Gruppenbetreuerin, muss immer anwesend bzw. sofort erreichbar sein. Wissen, auch über den Entwicklungsprozess, wird in direkter Kommunikation vom Coach weitergegeben. Außerdem wollten wir die XP-Gruppe bei der Arbeit beobachten.

Zur Einführung der XP-Konzepte wurde in den beiden Lehrveranstaltungen jeweils zu Beginn ein XP-Spiel durchgeführt, um den iterativen Ablauf eines XP-Projekts kennen zu lernen und die Begriffe und die Rollen einzuüben, wie es z.B. in [AuMi01] beschrieben ist. Dieser Einstieg hat sich sehr bewährt und kann nur weiterempfohlen werden. Auf der Informatica Feminale hatten die Teilnehmerinnen sich außerdem durch Literaturstudium und durch Kurzvorträge in die Konzepte von XP eingearbeitet. Im Software-Praktikum haben wir nach dem Prinzip „learning by doing“ gearbeitet. Refactoring wurde vermittelt, indem nach der ersten Iteration mit Hilfe eines Beamers gemeinsam der Code verbessert wurde. Dabei wurde das Tool RefactorIt eingesetzt. Über Test-First hatten wir einen kurzen Vortrag vorbereitet, danach haben wir es gemeinsam am Beamer ausprobiert. Auch dabei haben wir mit Tool-Unterstützung gearbeitet. JUnit und RefactorIt lassen sich in die eingesetzte Entwicklungsumgebung TogetherJ integrieren.

3 Erfahrungen mit XP im Gegensatz zum UML-basierten Vorgehen

Wir stellen einige unserer Erfahrungen vor, die wir bei Beobachtungen und bei Befragungen der TeilnehmerInnen mit Hilfe von Fragebögen gewonnen haben. Dabei wurden wir von der Abteilung Organisationspsychologie der Universität Dortmund unterstützt. Besonders interessierte uns in unseren Experimenten der Einfluss bestimmter Konzepte, die teilweise im Gegensatz zu der bisher praktizierten UML-basierten Vorgehensweise [Schm01] stehen, auf den Lernprozess. In der Industrie werden auch viele andere Vorgehensmodelle, z.B. das V-Modell, erfolgreich eingesetzt, die mit dem hier beschriebenen UML-basierten Vorgehen die zentrale Stellung des Modells und die Wichtigkeit der erzeugten Dokumente gemeinsam haben.

3.1 Paararbeit

Bei der Paararbeit wird immer zu zweit an einem Rechner gearbeitet, eine tippt, ein anderer kontrolliert, fragt nach und liefert Ideen. Bereits während der Implementierung findet ein erstes Code-Review statt. So sollen Fehler vermieden und durch bessere Lesbarkeit und Verständlichkeit soll die Qualität des Codes gesteigert werden. In einer Lehrveranstaltung fördert Paararbeit mit ständig wechselnden Partnern das Lernen voneinander.

Als Konsequenz davon, dass durch so genannte Stories beschriebene Funktionalität realisiert wird, wobei viele Klassen des Systems verändert werden müssen und alle EntwicklerInnen abwechselnd an allen Programmteilen arbeiten, gehört der gesamte Code im XP-Projekt allen EntwicklerInnen, die ihn nicht nur lesen, sondern auch ändern dürfen. XP wird wegen dieses Konzepts und wegen der besonderen Bedeutung der direkten Kommunikation in erster Linie für kleinere Projekte mit etwa 10 EntwicklerInnen empfohlen [Beck00].

Die Paararbeit hat sowohl auf der Informatica Feminale als auch im Software-Praktikum wie auch bei Lippert [LRWZ01] gut geklappt, obwohl einige männliche Studierende anfangs Vorbehalte äußerten. Auf der Informatica Feminale konnten wir beobachten, dass ein Team, das in Paararbeit arbeitet, problemlos neue Mitglieder integrieren kann. Unser XP-Projektteam setzte sich täglich etwas anders zusammen. Das Ausscheiden eines Mitglieds konnte leicht verkraftet werden, da es keine ausgeprägten Spezialisten gab.

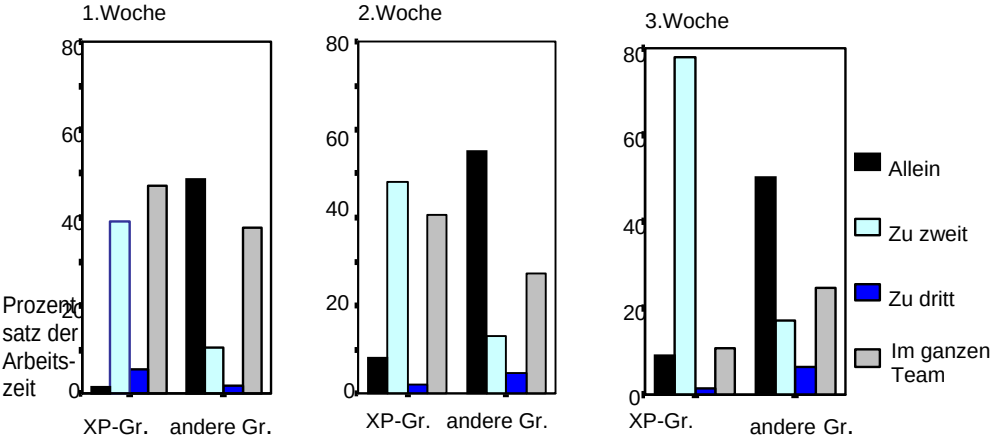

Abb. 1.: Paararbeit versus Einzelarbeit

Abbildung 1 zeigt als ein Ergebnis der Sopra-Befragung, dass die XP-Gruppe überwiegend in Paararbeit gearbeitet hat, während die anderen Gruppen überwiegend allein gearbeitet haben. Der große Anteil der Arbeit im gesamten Team insbesondere am Anfang des Projekts in der ersten und auch noch in der zweiten Woche ist bei der XP-Gruppe auf die Diskussion der Aufgabenstellung, die Planung der Iterationen und

die angeleitete Einarbeitung in die XP-Konzepte zurückzuführen. Diese Diskussionen über die Aufgabenstellung und die Projektsitzungen zur Organisation der Arbeit finden sich auch in den anderen Gruppen, die etwa ein Viertel ihrer Arbeitszeit im ganzen Team gearbeitet haben. Auch die UML-Gruppen haben mehr als 10 Prozent ihrer Zeit zu zweit gearbeitet.

3.2 Kommunikation und Teamklima

Die XP-Vorgehensweise ist darauf ausgerichtet, dass durch eine angenehme Arbeitsatmosphäre die gute Zusammenarbeit im Team und damit auch die Qualität des erstellten Produkts gefördert werden. Die Kommunikation unter den EntwicklerInnen wird als besonders wichtig für den Erfolg eines Projekts angesehen. In einer Lehrveranstaltung besteht der Reiz einer Vorgehensweise, die sich die Verbesserung des Teamklimas zum Ziel setzt, darin, durch ein gutes Teamklima auch ein gutes Lernklima zu erreichen.

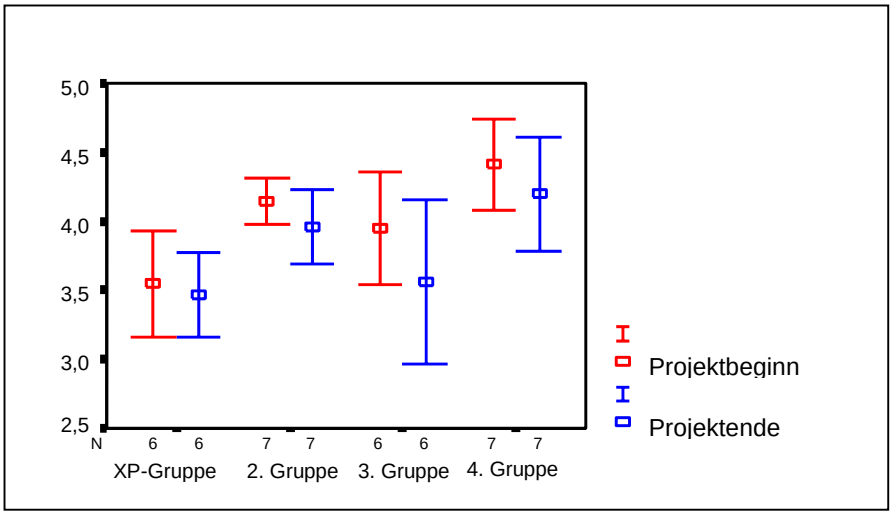

Abb. 2.: Teamklima in den Gruppen (links bei Projektbeginn, rechts bei Projektende)

In unserem Sopra-Experiment wurden zur Messung des Teamklimas Auszüge aus einem standardisierter Fragebogen verwendet, der erprobt ist und auch in Betrieben eingesetzt wird. Abbildung 2 zeigt, dass alle Gruppen ihr Teamklima zu Beginn des Projekts (Zeitpunkt 1) mit gut (>3,5) bewerten. Im Vergleich zu anderen Arbeitsteams in unserer Branche ist das ein guter Wert. Ausgerechnet in der XP-Gruppe (1. Gruppe) war das Arbeitsklima am schlechtesten, was wir vor der Befragung nicht ahnten. In dieser Gruppe waren die Vorkenntnisse besonders heterogen, was das kooperative Arbeiten belastete. In allen Gruppen zeigt sich am Ende des Projekts ein verschlechtertes Klima. Mit einer Verringerung des Mittelwerts um 0,08 fällt die Verschlechterung in der XP-Gruppe am geringsten aus. Die Standardabweichung

nimmt bei allen Gruppen außer der XP-Gruppe zu, hier verringert sich der Wert. Die Teammitglieder beantworteten die Fragen homogener als zu Beginn. Da unsere Datenbasis so klein ist, lässt sich aus dieser Beobachtung aber keine allgemeine Erkenntnis ableiten.

3.3 Kleine Iterationen und ständige Integration versus Modellierung und Dokumentation

Anstelle von aufwändiger Dokumentation in Form einer möglichst formalen Beschreibung wird durch ständig wechselnde Partner bei der Arbeit erreicht, dass das Wissen über die Inhalte und den Fortschritt des Projekts innerhalb des Entwicklerteams verbreitet wird. Die Betonung der Kommunikation gegenüber der Dokumentation steht eher im Gegensatz zu den bisherigen Lehrzielen im Grundstudium, wo viel Wert auf formal saubere Beschreibungen von Informatik-Inhalten gelegt wird.

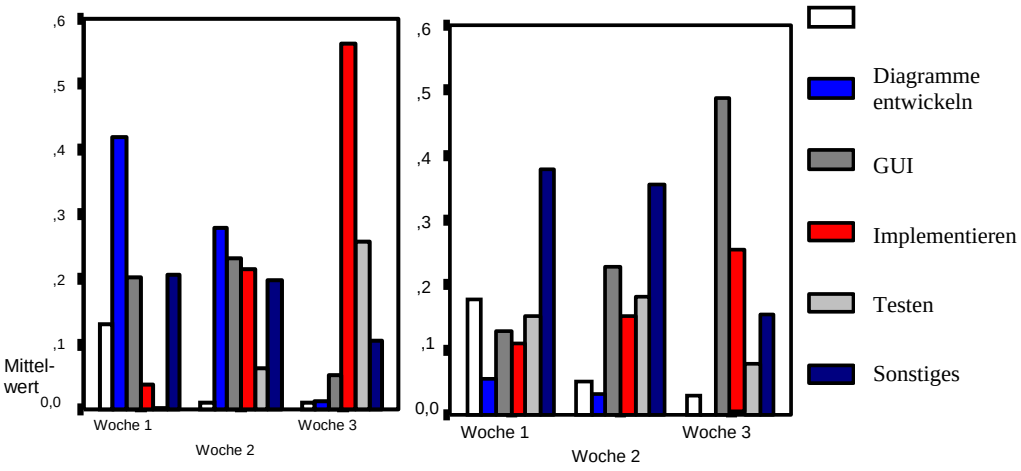

Abb. 3.: Tätigkeiten der Gruppen, links UML-Gruppen, rechts XP-Gruppe

Im iterativen XP-Prozess wird durch eine konsequente Beschränkung auf das Wesentliche ein komplexes Problem einfacher lösbar, und es entsteht sehr schnell eine erste funktionierende Version des Programms. Durch ständige Integration neuer Funktionalität wird das Programm in kleinen Iterationen schrittweise ausgebaut. Bei einer UML-basierten Vorgehensweise muss in den frühen Phasen viel Zeit für die Modellierung aufgewendet werden. Es besteht dabei die Gefahr, dass trotz hervorragender Modellierung nach Ablauf der Projektzeit kein lauffähiges Programm vorliegt. Bei der XP-Vorgehensweise entsteht bereits in der ersten Iteration eine vorzeigbare Version des Programms. Das führt frühzeitig zu Erfolgserlebnissen und trägt zur Motivation bei.

Abbildung 3 zeigt deutlich, wie unterschiedlich die verschiedenen Entwicklungstätigkeiten zu Buche schlagen. Während die UML-Gruppen (links), die einem wasserfallartigen Prozess [Schm01] folgen, zunächst Diagramme erstellen und

dann implementieren sowie testen, implementiert die XP-Gruppe von Anfang an. Die drei Wochen Projektzeit wurden in drei Iterationen aufgeteilt. Am Ende der ersten Woche existierte eine erste, allerdings winzig kleine Version des Programms, während die anderen nur Diagramme vorzeigen konnten. Die Entscheidung, welche Stories sich sinnvoll in einer ersten Iteration realisieren lassen, konnten die Studierenden in der XP-Gruppe allerdings nicht ohne Hilfe treffen. Sie hatte zu wenig Erfahrung, die Konsequenzen einer Story-Auswahl zu überblicken und die benötigte Zeit dafür einzuschätzen.

Die UML-Gruppen haben besonders zu Projektbeginn viel Zeit für die Modellierung verwendet. In den ersten beiden Wochen des Projekts wurden 42 bzw. 29 Prozent der Arbeitszeit für das Entwickeln der Diagramme aufgewendet. In der letzten Woche haben sich die Gruppen fast ausschließlich mit Implementieren und Testen beschäftigt. Die XP-Gruppe dagegen hat von Anfang an implementiert und getestet. Der Aufwand für die Tests war zumindest in den ersten beiden Wochen, als das XP-Vorgehen noch diszipliniert verfolgt wurde, höher als für die Implementierung.

Außerdem ist die Gefahr, den Überblick über das entstehende Programmsystem zu verlieren, bei XP ohne umfassendes Design relativ groß. UML-Diagramme können durchaus helfen, die Struktur des Systems und Abläufe zu verdeutlichen. Während wir auf der Informatica Feminale auf Wunsch der Teilnehmerinnen völlig auf UML-Diagramme verzichtet haben, haben wir im Software-Praktikum bewusst UML-Diagramme (Klassen- und Sequenzdiagramme) eingesetzt, wenn wir gemerkt haben, dass Abläufe und Zusammenhänge nicht allen klar waren. Wie weit man bei der Softwareentwicklung auf die Modellierung verzichten will und kann, hängt im Wesentlichen von der Erfahrung der EntwicklerInnen und von der Komplexität des Problems ab. AnfängerInnen hilft ein wenig Modellierung beim Verständnis der Aufgabe und ihrer Lösung.

3.4 Codequalität

Es leuchtet ein, dass der Qualität des Codes eine besondere Bedeutung bei XP zukommt. Der Code gehört allen und muss jederzeit für alle EntwicklerInnen verständlich sein. Er ist bereits während der Programmentwicklung einem ständigen Wartungsprozess unterworfen. Fester Bestandteil der XP-Vorgehensweise ist das „Refactoring“, die Verbesserung des existierenden Codes. Es reicht nicht aus, dass das Programm läuft und der erzeugte Code die Spezifikation erfüllt, vielmehr soll der erzeugte Code auch von hoher Qualität sein. Martin Fowler beschreibt in seinem Buch [Fowl00] verschiedene Kategorien von „schlecht riechendem“ Code und zeigt Wege, wie die Mängel beseitigt werden können. Die Programmierausbildung an den Universitäten konzentriert sich im Wesentlichen darauf, Sprachkonstrukte und ihre Anwendung vorzustellen und einzuüben. Der erzeugte Code wird hauptsächlich dahingehend beurteilt, dass das Programm die funktionalen Anforderungen erfüllt. Die Qualität des Codes bleibt weitgehend unberücksichtigt.

Unter XP wurde besserer Code erzeugt. Ein Vergleich des Programmcodes der XP-Gruppe mit dem Programmcode der anderen Gruppen ergab, dass die XP-Gruppe

kürzeren, leichter lesbaren und verständlicheren Code geschrieben hat als die anderen Gruppen [Bckm04]. Paararbeit und Refactoring scheinen die Lesbarkeit des Codes positiv zu beeinflussen. Wir konnten durch Messungen belegen, dass die XP-Gruppe z.B. sehr viel weniger duplizierten Code als die anderen Gruppen erzeugt hat [Bckm04]. Duplizierter Code ist besonders problematisch, weil er schlecht wartbar und bei Änderungen besonders fehleranfällig ist. Eine ganze Reihe der von Fowler [Fowl01] vorgeschlagenen Refactorings beschäftigen sich mit der Beseitigung von dupliziertem Code. Der massive Einsatz von Werkzeugen wie z.B. GUI-Buildern führt zu Code, der sehr „geschwätzig“ und schwerfällig ist. Er enthält sehr viel Redundanz, unnötige Kommentare und ist wenig elegant. Ein typisches Problem ist, dass die generierten Namen nicht durch sprechende Bezeichner ersetzt werden. Anstatt mit Vererbung zu arbeiten, wird bei ähnlichen Klassen der Code kopiert und abgewandelt. Das ist in der Wartungsphase äußerst problematisch, wenn alle Stellen gesucht werden müssen, an denen Änderungen vorgenommen werden müssen.

3.5 Testen

Die Bedeutung des Testens in der Softwareentwicklung wird bei XP besonders betont. Bevor eine Komponente implementiert wird, wird nach dem Test-First-Prinzip der Test dafür geschrieben. Die Spezifikation einer Komponente erfolgt also durch die Definition des Tests, den sie erfüllen muss. Es wird nur genau so viel implementiert, dass der Test durchläuft, und keine Zeile mehr. Kommen neue Anforderungen hinzu, wird zunächst das Testprogramm erweitert, das dann natürlich nicht mehr durchläuft. Danach die Klasse ergänzt, bis der Test wieder durchläuft. So soll vermieden werden, dass mehr implementiert wird, als unbedingt notwendig ist.

Im traditionellen Prozess ist Testen nach Abschluss der Implementierung bei Studierenden besonders unbeliebt, JUnit dagegen motiviert durch schnelle Erfolgserlebnisse. Aus Abbildung 3 lässt sich ablesen, dass die XP-Gruppe tatsächlich viel getestet hat, anfangs mehr, als sie implementiert hat, während sich die UML-Gruppen erst ab der zweiten Woche mit der Implementierung und dann hauptsächlich in der dritten Woche mit dem Testen beschäftigt haben.

Nach einer gewissen Umgewöhnungszeit waren auch die PraktikumsteilnehmerInnen in der Lage, nach dem Test-First-Prinzip zu arbeiten. Wir konnten beobachten, dass Testen sogar Spaß machen kann, wenn der „grüne Balken“ erscheint. Von der XP-Gruppe wurde im Software-Praktikum mehr Zeit für das Testen als für das Implementieren aufgewendet. Für Test-First von GUI-Klassen ist uns allerdings keine Lösung eingefallen.

3.6 Kundenorientierung

Starke Kundenorientierung durch direkte Integration des Kunden als aktives Projektmitglied stellt ein weiteres Merkmal von XP dar. Durch die iterative Vorgehensweise kann leicht auf sich ständig ändernde Anforderungen des Kunden reagiert werden. Die Bedeutung des Kunden im Entwicklungsprozess lässt sich in

studentischen XP-Projekten nur mit hohem Aufwand simulieren [MSK04]. Als Kunde ist ein Lehrender nach XP auch aktiv in den Entwicklungsprozess eingebunden, er formuliert Abnahmetests und bestimmt bei der Iterationsplanung mit. Ein Kunde, der vom Veranstalter simuliert wird, drängt auf die Fertigstellung des möglichst umfangreichen Produkts. Dieser Rollenkonflikt zwischen Lehrendem, der auf die korrekte Einhaltung der Abläufe und Konzepte achten muss, und dem simulierten aktiven Kunden wird auch von Mügge, Speicher und Kniesel [MSK04] bestätigt.

Bei der phasenorientierten Vorgehensweise geht man davon aus, dass zu einem bestimmten Zeitpunkt alle Anforderungen des Kunden erhoben und dokumentiert sind. Dann kann mit der eigentlichen Entwicklung begonnen werden. Das lässt sich von Lehrenden in studentischen Projekten zwar leicht realisieren, indem von ihnen entsprechende Dokumente vorbereitet werden, ist aber eine unrealistische Sicht der Wirklichkeit, denn reale Kunden ändern ständig während der Entwicklung ihre Anforderungen. Agile Vorgehensweisen wie XP sind hier den wenig flexiblen, auf Dokumenten basierenden Vorgehensweisen überlegen.

Aber Projekte mit realen Kunden lassen mit vielen studentischen TeilnehmerInnen nur unter unvertretbar hohem Aufwand durchführen.

3.7 Spezialistentum versus Wissenstransfer

Bisher konnten wir in den Studenten-Projekten ein ausgeprägtes Spezialistentum beobachten. Aus der Sicht der Gruppe ist es äußerst effizient, Spezialisten z.B. für das grafische User Interface auszubilden und den erfahrensten Programmierer möglichst ungestört implementieren zu lassen, während andere das Benutzungshandbuch schreiben. Aus der Sicht von Lehrenden ist hierbei problematisch, dass sowohl die Arbeit als auch der Wissenserwerb ungleich verteilt sind. Das Ziel einer Lehrveranstaltung im Grundstudium muss aber sein, dass alle TeilnehmerInnen alle Inhalte zumindest ansatzweise lernen. Bei XP ist die Gefahr des extremen Spezialistentums durch die Paararbeit bei ständig wechselnden Partnern und Aufgaben nicht groß.

XP hat sich durchaus als sehr effizient erwiesen. Das XP-Team hat weniger Stunden als die übrigen Gruppen gearbeitet, aber bei kürzerem Code ein im Funktionsumfang vergleichbares Produkt entwickelt, obwohl konsequent zu zweit an einem Rechner gearbeitet wurde. In Paararbeit wird sehr konzentriert gearbeitet. Beide Partner müssen ständig bei der Sache sein. XP verlangt eine Vereinbarung von festen Arbeitszeiten, damit Paararbeit im Team funktionieren kann.

Zurzeit führen wir eine intensive Studie zur Paararbeit durch. Dabei beobachten wir die gesamte Arbeit der Teams, die sich unterschiedlich organisiert haben. Die Kommunikation untereinander und die Weitergabe von Wissen sind in den Paararbeitsteams sehr viel umfangreicher als bei Nicht-Paararbeitsteams.

4 Fazit

XP ist für programmierunerfahrene EntwicklerInnen schwierig und verlangt viel Disziplin. Es fällt AnfängerInnen schwer zu entscheiden, welche Stories sich sinnvoll zu einem funktionalen Kern für eine Iteration zusammenfassen lassen. Das Schätzen der für die Realisierung einer Story benötigen Zeit ist für unerfahrene EntwicklerInnen schwierig. Ein leichtgewichtiger Prozess bietet außerdem die Gefahr, ins Chaos abzugleiten, wenn nicht diszipliniert gearbeitet wird. Ähnliche Beobachtungen schildern auch Lippert et al. [LRWZ01]. Wir konnten beispielsweise beobachten, dass bei Zeitdruck auf Test-First verzichtet wurde, was fatale Folgen hatte. VeranstalterInnen sollten dann im Interesse der Lernenden Einfluss nehmen und auf die Einhaltung des Prozesses dringen. Ein geplant vorliegender, phasenorientierter Prozess [Schm01] bietet Anfängern, die ja Softwareentwicklung gerade erst lernen sollen, Halt und Übersicht.

Ein XP-Projekt ist betreuungsintensiv. VeranstalterInnen von XP-Projekten müssen mehrere Rollen einnehmen, die nur schwer vereinbar sind. Als Lehrende müssen sie insbesondere auf die saubere Umsetzung der Konzepte achten, auch wenn das Projektziel gefährdet ist. Daneben müssen die Lehrenden im XP-Projekt auch die Rolle der aktiven KundInnen übernehmen, die immer anwesend oder zumindest schnell erreichbar sein sollen, falls es keine echten Kunden gibt.

Literatur

[AuMi01] K. Auer, R. Miller: Extreme Programming Applied – Playing to Win. Addison Wesley, 2001.

[Beck00] K. Beck: Extreme Programming explained: Embrace Chance. Addison Wesley, 2000.

[Bckm04] I. Beckmann: Erkennen von Code-Mängeln zum Refaktorisieren. Diplomarbeit am Fachbereich Informatik der Universität Dortmund, 2004.

[BRJ99] G. Booch, J. Rumbaugh, I. Jacobson: The Unified Modeling Language – User Guide. Addison Wesley, 1999.

[Fowl00] M. Fowler: Refactoring, oder: Wie Sie das Design vorhandener Software verbessern. Addison-Wesley, 2000.

[KSS04] C. Kopka, D. Schmedding, J. Schröder: Der Unified Process im Grundstudium – Didaktische Konzeption, Einsatz von Lernmodulen und Erfahrungen. In: DELFI 2004 – Die 2. Deutsche e-Learning Fachtagung Informatik, Paderborn, LNI, 2004.

[LRWZ01] M. Lippert, S. Roock, H. Wolf, H. Züllighofen: XP lehren und lernen. in: H. Lichter, M. Glinz (Editors): Software Engineering im Unterricht der Hochschulen, SEUH 7, Zürich 2001. dpunkt.verlag.

[MSK04] H. Mügge, D. Speicher, G. Kniesel: Extreme Programming in der Informatik-Lehre – Ein Erfahrungsbericht. In: P. Dadam, M. Reichert (Hrsg.): Informatik 2004 – Informatik verbindet, Proceedings der 34. GI-Jahrestagung, Band 2, Ulm, LNI, 2004.

[NeMa01] J. Newkirk, R. C. Martin: Extreme Programming in Practice. Addison Wesley, 2001.

[Schm01] D. Schmedding: Ein Prozessmodell für das Software-Praktikum. In: H. Lichter, M. Glinz (Hsg.): SEUH 2001. Software Engineering im Unterricht der Hochschulen. S. 87-97. Zürich. 2001. dpunkt.verlag.

[WiUp01] L. Williams, R. Upchurch: Extreme Programming for Software Engineering Education? In: 31st ASEE/IEEE Frontiers in Education Conference, Reno, NV, 2001.

Softwaretechnologie-Praktikum im Grundstudium: universitäres oder reales Projekt?

Birgit Demuth[+], *Lothar Schmitz*[++], *Barbara Wittek*[+]
+ Fakultät Informatik, Institut für Software- und Multimediatechnik, Technische Universität Dresden
01069 Dresden
{Birgit.Demuth, Barbara.Wittek}@inf.tu-dresden.de
++ Fakultät Informatik, Institut für Softwaretechnologie, Universität der Bundeswehr München
85577 Neubiberg
lothar@informatik.unibw-muenchen.de

Zusammenfassung

Ein gemeinsam entwickeltes Softwaretechnologie-Praktikum ist an der Fakultät Informatik der TU Dresden und an der Fakultät Informatik der UniBw München obligatorischer Bestandteil im Grundstudium. Das Praktikum wird an beiden Hochschulen in gleicher Weise als „internes Praktikum" im universitären Rahmen durchgeführt. Seit einigen Jahren erhält an der TU Dresden ein Teil der Studenten die Möglichkeit, unter vergleichbaren Rahmenbedingungen ein reales Projekt als „externes Praktikum" in regionalen Firmen oder wissenschaftlichen Institutionen durchzuführen. Über die Erfahrungen, die mit dem Praktikum generell, speziell aber mit den beiden alternativen Ausbildungskonzepten gesammelt wurden, wird im Folgenden berichtet.

1 Einleitung

In vergangenen SEUH-Workshops wurden häufig die Konzeption sowie Erfahrungen bei der Durchführung von Software-Praktika vorgestellt; in [SEU 99] beispielsweise war dem Thema Software-Praktika eine eigene Sitzung gewidmet. Die geschilderten Erfahrungen beziehen sich auf sehr unterschiedliche Studiensituationen und Rahmenbedingungen: Praktika im Grundstudium oder im Hauptstudium, Praktika mit sehr vielen oder relativ wenigen Teilnehmern, Praktika, die „intern" innerhalb der Hochschule oder „extern" in Zusammenarbeit mit Softwarefirmen durchgeführt wurden.

Das Softwaretechnologie-Praktikum an der TU Dresden [Stp 04] ist für die Studiengänge Informatik, Medieninformatik und Informationssystemtechnik eine Pflichtveranstaltung von zwei Semesterwochenstunden (SWS) im Grundstudium (auch im Ergän-

zungsstudiengang Softwaretechnik obligatorischer Bestandteil) und baut auf der Vorlesung/Übung „Softwaretechnologie I“ auf. Analog ist die Situation in den Studiengängen Informatik und Wirtschaftsinformatik der UniBW München. Die aus dem Framework *SalesPoint* und einer umfangreichen Dokumentation bestehende Praktikumsinfrastruktur wurde von beiden Universitäten gemeinsam entwickelt, gepflegt und genutzt. Darüber haben wir in [Dem 99] bereits ausführlich berichtet.

Die Durchführung des Praktikums in einer frühen Phase des Studiums soll eine umfassende Ausbildung aller Studenten auf dem Gebiet der Softwaretechnologie gewährleisten, birgt aber auch Probleme in sich. Insbesondere die jährliche Teilnehmerzahl von ca. 500 Studenten an der TU Dresden (ca. 90 an der UniBw München) erfordert einen außerordentlich hohen organisatorischen und personellen Aufwand. Zeitweilige Engpässe bei den Hilfskraftmitteln vor etwa fünf Jahren gaben dann den Anstoß, an der TU Dresden die externe Form der Praktikumsdurchführung als mögliche Alternative zu erproben. Seitdem nehmen dort je nach Anzahl der verfügbaren Praktikumsplätze jährlich 15-20 % der Studenten am externen Praktikum teil. Die Studenten müssen sich explizit dafür bewerben und werden aufgrund ihrer Vorleistungen in der Grundlagenlehrveranstaltung Softwaretechnologie ausgewählt.

Für uns ergab sich die seltene Gelegenheit, die beiden unterschiedlichen Formen der Praktikumsdurchführung bei identischer Studiensituation über einige Jahre nebeneinander zu beobachten und zu vergleichen.

Im Folgenden beschreiben wir Ziele und Rahmenbedingungen des Softwarepraktikums innerhalb eines Massenstudiums (Abschnitt 2) anhand der internen Praktikumsform, die davon abweichenden Merkmale externer Projekte (Abschnitt 3) sowie Beobachtungen und Erfahrungen, die wir im Laufe der Jahre bei der Durchführung dieses Praktikums mit mehreren tausend Studenten gewinnen konnten (Abschnitt 4). Abschnitt 5 fasst aus unserer Sicht wichtige und praxisrelevante Lehren für unser Ausbildungskonzept zusammen.

2 Ziele und Rahmenbedingungen

Das Praktikum hat die Vermittlung objektorientierter Softwareentwicklungstechniken im Rahmen eines Softwareprojekts sowie das Erlernen arbeitsteiliger und teamorientierter Arbeitsweisen zum Ziel. Darüber hinaus wird die **Wiederverwendung** von Software vorgeschrieben, um die Arbeit mit Softwarebausteinen und deren Anpassung bewusst einzuüben [Dem 99]. Im internen Praktikum besteht daher die Aufgabe in der Erstellung einer Verkaufsanwendung auf Basis des Anwendungsframeworks *SalesPoint.* Es werden Teams von meist fünf Studenten gebildet, die eine grobe Beschreibung einer komplexen Aufgabenstellung als Basis für ihre Arbeit erhalten. Zunächst ist eine Anforderungsermittlung durchzuführen. Dann erfolgt in mehreren Schritten die Modellierung des zukünftigen Produkts mit Hilfe verschiedener UML-basierter Beschreibungstechniken. Das Ziel der Modellierung ist ein auf die wiederzuverwendenden

Softwarebausteine abgestimmter Entwurf, der dann mittels der Programmiersprache Java umgesetzt werden muss.

Zur **Teambildung** können die Studenten vorab Wünsche äußern, die nach Möglichkeit erfüllt werden. Für die Arbeitsteilung innerhalb der Teams wird das Chefprogrammiererprinzip empfohlen. Ein Student übernimmt die Rolle des Chefprogrammierers, der Entscheidungen trifft und als Ansprechpartner für Kunden und Betreuer fungiert. Die weiteren Rollen Assistent, Sekretär und Programmierer werden unter den restlichen Teammitgliedern aufgeteilt.

Für jedes Team im internen Praktikum wird ein studentischer **Tutor** engagiert, der den Softwareentwicklungsprozess kontrolliert, die Rolle des Kunden übernimmt und den Gruppen bei auftretenden technischen und kommunikativen Problemen hilfreich zur Seite steht. Die Tutoren haben dabei eine hohe Verantwortung, da Entscheidungen über Korrektheit von Dokumenten, Einhaltung von Zeitvorgaben und Beteiligung aller Teammitglieder am Projekt zu treffen sind. Die Teams müssen wöchentlich einmal an Pflichtkonsultationen mit ihrem Betreuer teilnehmen. Hierbei werden Ergebnisse besprochen, Probleme geklärt, die Einhaltung des Zeitplans überprüft etc.

Das interne Praktikum ist in **Phasen** eingeteilt, für die ein strenger Zeitplan vorgegeben wird. Nach zweiwöchiger Einarbeitungsphase, in der sich die Studenten die Grundlagen des wiederzuverwendenden Anwendungsframeworks aneignen sollen, entscheidet ein Zwischentest über die weitere Teilnahme am Praktikum. Die Bearbeitung der Aufgabe erfolgt danach in den Phasen Analyse, Entwurf, Implementation/Test und Wartung. Nach Abschluss dieser Phasen stellen die Teams ihre Ergebnisse in einer Endpräsentation vor. Für die Bearbeitung des Projekts stehen den Studierenden insgesamt 12 Wochen zur Verfügung. Außerdem ist strikt vorgegeben, welche Ergebnisse in jeder Phase entstehen sollen, wie diese anzufertigen sind. Dieses Vorgehen bietet zwar wenig Raum für eigene Experimente, hilft aber den meisten Teams, das Praktikum erfolgreich zu absolvieren. Sie besitzen bis zu diesem Zeitpunkt oft keine Erfahrung mit der Durchführung von Softwareprojekten und benötigen daher ausführliche Anleitung.

Ein großer Vorteil des internen Praktikums ist in der umfangreichen **Dokumentation** und den zahlreichen Hilfsangeboten seitens des Lehrstuhls zu sehen. Für die Arbeit mit dem zur Wiederverwendung im universitären Praktikumskonzept vorgeschriebenen Anwendungsframework *SalesPoint* steht vielfältige Unterstützung bereit: ein vollständig implementiertes einfaches Beispiel („Videoautomat“), zwei „Kochbücher“ zu Implementierungsfragen („Hooks“ und „HowTos“) sowie das ständig verfügbare Expertenwissen der *SalesPoint*-Entwickler über ein Praktikumsforum. Darüber hinaus ist das Framework selbst gut dokumentiert (Technischer Überblick, API-Spezifikation).

Neben der eigentlichen Anwendung entstehen im Verlauf des Praktikums weitere Artefakte, die von den Gruppen ausführlich zu dokumentieren sind. Dazu ist von jedem Team eine **Webseite** zu entwerfen, auf der alle Ergebnisse des Praktikums zum Zeitpunkt ihrer Entstehung zu veröffentlichen sind. Diese Maßnahme dient hauptsächlich der Kontrolle von Qualität und termingerechter Anfertigung der Dokumente, hält die

Studierenden durch die Außenwirkung aber auch zu vollständiger und gewissenhafter Arbeit an.

Für alle Studierenden des Praktikums ist die Verwendung des **Versionsverwaltungssystems** CVS vorgeschrieben. Dies vermittelt den Nutzen eines solchen Systems in der arbeitsteiligen Softwareentwicklung. Auf der anderen Seite stellt das CVS ein Kontrollinstrument dar, das den Betreuern der Teams vor allem während der Implementierung erlaubt, den Projektstand zu überwachen und die Beteiligung aller Studenten am Projekt zu überprüfen. Sowohl für das interne als auch für das externe Praktikum wird die entsprechende Infrastruktur zentral bereitgestellt.

Als positiv für den Verlauf des internen Praktikums kann auch die **Homogenität und Vielzahl** der Projekte angesehen werden. Da alle Teams den gleichen zeitlichen Restriktionen und organisatorischen Rahmenbedingungen unterliegen und das *SalesPoint*-Praktikum auf jahrelange Erfahrung zurückgreifen kann, entstand eine ausführliche Praktikumsdokumentation, und es gibt sehr viele Beispielprojekte im Web. Außerdem stoßen viele Teams während ihrer Arbeit auf die gleichen Probleme, sodass gegenseitige Hilfe (z.B. im Praktikumsforum) möglich ist. Und nicht zuletzt waren die studentischen Betreuer der Gruppen mit vielen anstehenden Schwierigkeiten auch während ihres eigenen Praktikums konfrontiert und können so kompetente Hilfe anbieten.

3 Das externe Praktikum

Bezüglich der Rahmenbedingungen und des Anforderungsniveaus unterscheiden sich internes und externes Praktikum nicht wesentlich, eher bei Aufgabenstellung, Arbeitsbedingungen und Kontrollmechanismen. Während im internen Praktikum die Aufgabe in der Erstellung einer Verkaufsanwendung auf Basis des am Lehrstuhl entwickelten Anwendungsframeworks *SalesPoint* besteht, hängt die **Aufgabenstellung** im externen Praktikum von den Anforderungen der jeweiligen Firma/Institution ab. Das Spektrum reicht dabei von hauseigenen Werkzeugen über Prototypen zur Ermittlung von Produktanforderungen und zur Nutzerakquisition bis zu realen (Web-)Anwendungen. Es wird jedoch darauf geachtet, dass der Umfang in etwa dem der universitären Projekte entspricht. Das setzt eine enge Zusammenarbeit zwischen dem Lehrstuhl und den am Praktikum beteiligten Firmen voraus: Damit die Rechte und Pflichten aller Beteiligten gewahrt werden können, wird bereits in der Vorbereitungsphase ein Kooperationsvertrag geschlossen. Auf der Basis dieser Vereinbarung können während der Bearbeitungszeit auftretende Probleme sowie berechtigte Interessen der Firmen (etwa Geheimhaltung) angemessen berücksichtigt werden. Selbstverständlich hat dabei der Lernerfolg Priorität gegenüber Anforderungen an den Funktionsumfang des Produkts. Nicht zu unterschätzen ist der hohe Aufwand bei der Vorbereitung externer Projekte. Das betrifft die Erstellung der Kooperationsverträge sowie die Einarbeitung in jeweils neue Themenstellungen und in die konkreten organisatorischen und technischen Rahmenbedingungen bei der jeweiligen Firma.

Für das externe Praktikum wird ein ähnlicher Ablauf angestrebt wie für das interne, jedoch wird hier von den Studierenden eine größere Eigenverantwortung erwartet. Insbesondere können sie in Absprache mit ihren **Betreuern** den im internen Praktikum vorgeschriebenen Softwareentwicklungsprozess an ihre Situation anpassen. Die Firmen des externen Praktikums betreuen ihre Teams autonom. Wenn es jedoch gewünscht wird, stellt der Lehrstuhl ebenfalls einen studentischen Betreuer zur Verfügung, der die softwaretechnische Seite des Praktikums überwacht, während die Firma die Kundenrolle übernimmt. Diese Praxis hat sich als günstig erwiesen, da so die Einheitlichkeit der Anforderungen im internen und externen Praktikum besser gewährleistet ist; künftig wollen wir daher regelmäßig so vorgehen.

Im externen Praktikum verwenden die Teams verschiedenste **Technologien**, Frameworks und Softwarekomponenten wieder, zum Teil Open-Source-Software, zum Teil firmeninterne Entwicklungen, zum Teil kommerzielle Produkte. Die Dokumentation dazu ist oft unzureichend und Expertenwissen gar nicht oder nur über Umwege erhältlich. Die Studenten müssen stärker auf eigene Recherchen und Experimente setzen, was oft einen erhöhten Arbeitsaufwand bedeutet. Allerdings ist dadurch der persönliche Gewinn auch höher, da sich die Studenten im externen Praktikum oft selbstständig mit sehr aktuellen Technologien beschäftigen müssen, was ihnen im weiteren Verlauf ihres Studiums bzw. im späteren Berufsleben von Nutzen sein kann.

Im externen Praktikum ist die Forderung nach Veröffentlichung von Modellierungsergebnissen und Implementationsdetails meist nicht durchzusetzen, da hier die Geheimhaltung von Firmeninterna Vorrang hat. Das führt häufig dazu, dass die Studenten die **Dokumentation** generell vernachlässigen, was sich am Ende des Projekts auch in einer mangelhaften Anwender- und Entwicklerdokumentation widerspiegelt.

4 Beobachtungen und Erfahrungen

Die Planmäßigkeit des Softwareentwicklungsprozesses hat im Praktikum einen hohen Stellenwert. Während jedoch im internen Praktikum durch die lückenlose Betreuung die Einhaltung eines sauberen **Entwicklungsprozesses** durchgesetzt wird, neigen Studenten im externen Praktikum oft zu einer missverstandenen Form des Extreme Programming. Die Ursache dafür liegt darin, dass die Betreuung in den Firmen eher *ergebnisorientiert* verläuft, im universitären Rahmen dagegen eher *prozessorientiert.* Andererseits stellt die größere Freiheit im externen Praktikum auch eine Chance für einen größeren Lernerfolg dar. Im internen Praktikum erhalten die Studenten strenge Vorgaben, zu welchem Zeitpunkt welches **UML-Diagramm** anzufertigen ist. Dabei erschließt sich für viele Studierende oft nicht der Zweck des jeweiligen Modellierungsschrittes für den weiteren Verlauf des Projekts. Das „Malen von Diagrammen“ wird nur als Belastung empfunden. Im externen Praktikum bekommen die Studenten zwar auch Hinweise auf die Verwendung von UML-Modellen, sie müssen die Entscheidung über deren sinnvollen Einsatz größtenteils jedoch selber treffen. Dabei erfolgt im positiven Fall eine stär-

kere Auseinandersetzung mit den Zielen und Möglichkeiten der UML-Modellierung, und der Lernerfolg ist höher.

Beim Vergleich der **Ergebnisse** im externen und internen Praktikum lassen sich kaum qualitative oder quantitative Unterschiede feststellen. Der Arbeitsaufwand ist für die Studierenden sowohl im internen als auch im externen Praktikum wesentlich höher als die vorgesehene SWS. Da die Firmen die im Praktikum entstandenen Anwendungen häufig tatsächlich verwenden oder als Basis für weitere Entwicklungen nutzen, besteht für die Studenten eine hohe Motivation zur Erstellung eines qualitativ hochwertigen Produkts. Aber auch innerhalb des internen Praktikums entstehen immer wieder interessante und ausgereifte Softwarelösungen. Die Motivation zur Anfertigung guter Ergebnisse gründet sich dabei hauptsächlich auf das Erlebnis, systematisch und arbeitsteilig ein eigenes Produkt entstehen zu lassen und im Vergleich mit anderen Teams gut abzuschneiden. Die qualitative Bandbreite der universitären Projekte ist durch die Vielzahl der Produkte allerdings sehr groß. Quantitativ sind die Applikationen aus externem und internem Praktikum vom Funktionsumfang her ähnlich und auch bezüglich ausgewählter Metriken ergeben sich Gemeinsamkeiten. Interessant ist dabei, dass die durchschnittlichen Werte für Lines of Code (LOC) und Anzahl der Klassen in ähnlichen Größenordnungen liegen, externe Gruppen aber durchschnittlich einen geringeren Zeitaufwand für ihre Arbeit angeben. Das kann sowohl auf ein effektiveres Vorgehen als auch auf bessere Voraussetzungen, z.B. bezüglich Programmiererfahrungen, hinweisen.

Bezüglich **technischer Voraussetzungen** kann über keine erheblichen Unterschiede zwischen externem und internem Praktikum berichtet werden. Intern ist es den Studenten freigestellt, ob sie im Fakultätsrechenzentrum oder mit eigener Technik arbeiten. Die Verwendung von Softwareentwicklungswerkzeugen wird zwar empfohlen und durch Bereitstellung von Lizenzen und Tutorials unterstützt, aber nicht vorgeschrieben. Das externe Praktikum weicht nur dann von diesen Gegebenheiten ab, wenn die jeweilige Firma den Teams eigene Arbeitsplätze zur Verfügung stellt oder die Verwendung bestimmter Werkzeuge explizit vorschreibt. Die Integration der studentischen Projekte in die firmeninterne Infrastruktur kann dabei sowohl von Vorteil als auch von Nachteil sein. Einerseits entfallen Entscheidungen zur Entwicklungsumgebung und das technische Know-how der Firma kann genutzt werden, andererseits entstehen aber Restriktionen, mit denen sich die Studenten zusätzlich zur eigentlichen Projektarbeit auseinander setzen müssen.

Studenten, die sich vor dem Praktikum bereits kannten, können sich schneller aufeinander einstimmen, besser gemeinsame Termine finden und früher mit der eigentlichen Arbeit beginnen. Dafür werden bestehende Probleme aus falsch verstandener Freundschaft oft nicht angesprochen und führen mitunter später zum Zerwürfnis in der Gruppe. Umgekehrt benötigen Teams, deren Mitglieder sich zunächst nicht kennen, zu Beginn einen höheren **Kommunikations- und Koordinationsaufwand**, gehen aber meist offener mit Problemen um. Beide Konstellationen können daher sowohl zum Erfolg des Projekts als auch zu dessen Scheitern führen. Im externen Praktikum ist die

Ausrichtung auf das gemeinsame Ziel jedoch häufig stärker ausgeprägt, sodass hier eher Lösungen für Teamprobleme gesucht werden und seltener Teams oder einzelne Mitglieder aus dem Praktikum ausscheiden. Die Auswahl besonders leistungsstarker Studenten für das externe Praktikum kann für die Teamarbeit sowohl Vor- als auch Nachteile haben. Im internen Praktikum sind die Gruppen oft heterogen bezüglich der Fähigkeiten ihrer einzelnen Mitglieder. Teamprobleme resultieren oft aus unterschiedlicher Arbeitsleistung. Im externen Praktikum sind alle Teammitglieder auf einem etwa gleich hohen Niveau und werden damit eher gleich stark am Projekt beteiligt. Eventuelle Teamprobleme sind hier oft darauf zurückzuführen, dass starke Individualisten aufeinander treffen. Diese Studenten entsprechen oft dem Klischee des „Hackers“, der zudem keine Teamerfahrung mitbringt. Bei der Betreuung sind beide Phänomene zu berücksichtigen und entsprechend zu behandeln.

Es hat sich gezeigt, dass es notwendig ist, den Tutoren konkrete Entscheidungshilfen und „Druckmittel“ zur **Durchsetzung der Praktikumsanforderungen** zur Verfügung zu stellen, und ihr Einsatz häufig auch erforderlich ist. Speziell im internen Praktikum führt die Arbeit an einem komplexen, zeitaufwändigen Projekt ohne realen Kunden bei vielen Studenten zu geringem Engagement. Der Lernerfolg und der Nutzen für zukünftige Projekte sind für sie in dieser frühen Phase des Studiums oft schwer abzuschätzen. Daher neigen die Studenten dazu, die Anforderungen nicht ernst zu nehmen sowie Fehler bzw. Zeitüberschreitungen zu bagatellisieren. Hier ist es entscheidend, Meilensteine klar zu definieren und deren Nichteinhaltung mit Konsequenzen zu verbinden. Ebenso muss das Einfordern definierter Dokumente und Ergebnisse zu konkreten Terminen durch die Tutoren konsequent durchgesetzt werden. Problematisch ist in diesem Zusammenhang die Einschätzung von Zwischenergebnissen. Häufig können abgegebene Dokumente oder Codefragmente nicht explizit als richtig oder falsch eingestuft werden. So sind z.B. bereits erste richtige Ansätze als Erfolg zu werten, während Fehler aufgrund mangelnder Arbeitsmoral nicht toleriert werden können. Daraus resultieren gewisse Unsicherheiten sowohl bei den Tutoren als auch bei den Studierenden, die durch strenge Richtlinien und intensive Anleitung der Tutoren minimiert werden müssen. Dazu finden wöchentliche Tutorenbesprechungen statt.

Bei den Studenten im externen Praktikum ist der Einsatz von Druckmitteln seltener nötig. Durch die Arbeit in einer Firma an einer realen Aufgabe ist die Motivation erheblich höher. Hier liegen die Probleme dann oft in anderen Bereichen. So können Diskrepanzen zwischen den Anforderungen der Firma und den Fähigkeiten der Studenten entstehen, die im Grundstudium mit vielen Technologien noch nicht vertraut sind. Auch dadurch kann es zu einer Demotivation kommen, die jedoch andere Ursachen hat und mit anderen Mitteln als im internen Praktikum zu beheben ist. Als hilfreich hat sich das Durchführen einer **Zwischenpräsentation** erwiesen, bei der die Studierenden den aktuellen Arbeitsstand vorstellen und selbst einen Überblick über bisher Erreichtes erhalten. Dabei können dann auch seitens des Lehrstuhls die Ansprüche der Firmen mit den Leistungsgrenzen der Studenten abgeglichen werden. Im internen Praktikum sind solche

Zwischenpräsentationen aufgrund der Vielzahl der Gruppen nicht möglich, sodass Probleme oft tatsächlich erst gegen Projektende sichtbar werden.

5 Was haben wir gelernt?

Neben den oben beschriebenen konkreten Beobachtungen und Erfahrungen aus unserem Praktikum ergibt sich für uns eine Reihe von Lehren, die sich vermutlich auch gewinnbringend auf ähnliche Veranstaltungen andernorts übertragen ließen:

Die **Betreuung durch studentische Tutoren** hat sich als sehr effektiv erwiesen. Dies setzt eine sorgfältige Auswahl der Tutoren sowie deren intensive Betreuung durch den Lehrstuhl voraus. Des Weiteren ist der wöchentliche Gedanken- und Erfahrungsaustausch der Tutoren untereinander ein wirksames Instrument bei der qualitativen Absicherung in der Ausbildung. Eine das Praktikum im Sommersemester 2004 begleitende psychologische Studie [iko 04] konnte die Vorbildwirkung des studentischen Tutors klar herausarbeiten. Tutoren, die motivieren, intellektuell anregen und Stärke ausstrahlen, können allerdings nur dann positiven Einfluss auf das Team und dessen Projektergebnisse nehmen, wenn sich die Teammitglieder auch engagieren.

Ein weiterer wesentlicher Faktor für den Erfolg des Praktikums ist die **Motivation der Studenten**. Hier ist das externe Praktikum klar im Vorteil, da die Identifikation mit einem realen Projekt leichter fällt als mit einem Übungsbeispiel und das Ziel der Arbeit deutlicher wird. Daher ist es notwendig, im internen Praktikum den persönlichen Wissenszuwachs stärker hervorzuheben und den Studenten häufiger positive Rückmeldungen zu geben. Zu Beginn des Praktikums ist die Demonstration eines vorbildlichen Projekts aus dem Vorjahr nützlich. Dem Bericht von Studenten wird erfahrungsgemäß eher Glauben geschenkt als dem von Lehrbeauftragten. Das Stiften von Preisen für die besten Praktikumsteams hat sich als weiteres Mittel zur Motivationssteigerung erwiesen.

Reale Projekte können nur mit leistungsfähigen und motivierten Studenten erfolgreich durchgeführt werden. Die Trennung des externen Praktikums vom universitären Umfeld birgt Risiken in sich. Die Firmen kennen den aktuellen Ausbildungsstand eines Studenten im Grundstudium unzureichend und stellen ggf. gemessen an den Fähigkeiten der Studenten inadäquate Anforderungen an den Verlauf der Arbeit und an das Ergebnis. Außerdem wird in den Firmen meist zu wenig Wert auf die systematische Softwareentwicklung gelegt, sondern der Fokus zu stark auf das Endprodukt gerichtet. Das widerspricht den eigentlichen Zielen des Praktikums, softwaretechnische Kompetenzen zu fördern. Für beide Probleme stellt der Einsatz von studentischen Betreuern auch für externe Praktikumsgruppen eine geeignete Lösung dar. Durch den intensiven Kontakt der Betreuer mit den Teams, dem Lehrstuhl und den Firmenvertretern kann sowohl das Arbeitspensum immer wieder abgestimmt als auch das Einhalten eines sauberen Entwicklungsprozesses kontrolliert werden. Für die Teilnehmer am externen Praktikum erweist sich die Möglichkeit, Erfahrungen auf dem Gebiet der Softwareentwicklung an einem realen Projekt unter kontrollierten Bedingungen zu sammeln, als

besonders positiv. Da die Studenten hier stärker selbstverantwortlich bezüglich Zeiteinteilung, Einarbeitung in neue Technologien und Arbeitsteilung handeln und sich zudem einem realen Kunden gegenüber behaupten müssen, wird das Bewusstsein für die dabei entstehenden Probleme geschärft.

Die Teams im **internen Praktikum** benötigen aufgrund ihrer meist leistungsmäßig heterogenen Zusammensetzung ständige Kontrolle und erhöhte Aufmerksamkeit, um die gleichmäßige Beteiligung aller Studenten am Praktikum zu gewährleisten. Hier muss von erhöhten Betreuungsanforderungen ausgegangen werden, denen durch das Anbieten verschiedener Anlaufstellen bei Problemen (Forum, Tutor, Lehrstuhl) und umfangreicher organisatorischer und technischer Hilfestellungen Rechnung zu tragen ist. Durch die ständige Kontrolle der Teams und die intensive Anleitung im internen Praktikum lassen sich positive Auswirkungen auf den Lernerfolg erzielen. Die Studenten können hier die Softwareentwicklung an einem Übungsbeispiel „gefahrlos“ ausprobieren und finden dabei jederzeit Unterstützung in allen Bereichen, ohne dass z.B. Fehlentscheidungen negative Konsequenzen für das gesamte Projekt bedeuten. Wenn diese Chance von den Studenten erkannt wird, kann der Lerneffekt sogar höher ausfallen als im externen Praktikum, wo durch den höheren Erfolgsdruck schnell gewohnte Vorgehensweisen bei der Programmierung adaptiert und dabei die Ziele eines sauberen Softwareentwurfs nicht verinnerlicht werden. Eine Anforderung an die Studierenden aus dem letzten Praktikumszyklus, die sich auch nach studentischen Aussagen sehr bewährt hat, ist, dass jedes Teammitglied in der Entwurfsphase einen eigenen kleinen Prototypen schreiben muss.

Eine weitere wichtige Erfahrung bei der Durchführung des Praktikums sowohl für universitäre als auch für reale Projekte ist die **Durchsetzung eines festen Endtermins.** Im internen Praktikum kommt noch die konsequente terminliche Einhaltung von Meilensteinen dazu. Die teilweise geringe Motivation bzw. fehlende softwaretechnische, programmiertechnische und auch soziale Kompetenz der Studenten in den universitären Projekten führt häufig zu einer legeren oder sogar ablehnenden Haltung gegenüber dem Praktikum. Dem ist durch den drohenden Ausschluss aus dem Praktikum von einzelnen Teammitgliedern oder eines gesamten Teams entgegenzuwirken. Damit wird auch die Position der Betreuer gegenüber den Gruppen gestärkt und so die Durchsetzung softwaretechnischer Anforderungen gewährleistet. Im externen Praktikum ist die Ausübung von Druck auf die Studenten nicht oder kaum nötig, da hier qualitativ hochwertige Ergebnisse aus eigenem Antrieb entstehen und Termin- und Teamprobleme meist selbstständig erkannt und gelöst werden.

Im Hinblick auf die Teilung des Angebots in **interne und externe Projekte** fassen wir zusammen:

- Unter direkter Anleitung und strikter Kontrolle durchgeführte Projekte eignen sich für unerfahrene Entwickler, d.h. für die Mehrzahl der Studenten im Grundstudium.

- Reale, stärker selbstverantwortliche Projekte im betrieblichen Umfeld stellen für die Gruppe der Studenten mit besserem softwaretechnischen Hintergrund eine interessante Herausforderung dar.
- Externe Projekte sind für den Lehrstuhl in Vorbereitung und Durchführung aufwändiger als interne Projekte.
- Die vorgesehene Semesterwochenstundenzahl wird künftig von zwei auf vier SWS erhöht, um dem Arbeitsaufwand und der Stellung des internen als auch des externen Praktikums besser gerecht zu werden.

Beide Konzepte lassen sich im Grundstudium erfolgreich durchführen, sofern geeignete Randbedingungen hergestellt werden. Mit dem Angebot beider Alternativen können wir auf die unterschiedlichen Interessen und Begabungen verschiedener Praktikumsteilnehmer eingehen.

Literatur

[Dem 99] B. Demuth, H. Hußmann, S. Zschaler, L. Schmitz: Erfahrungen mit einem frameworkbasierten Softwarepraktikum. In: [SEU 99].

[iko 04] TU Braunschweig, Institut für Psychologie: Initiative für Kommunikation in der Softwarebranche (ikoso). *http://www.ikoso.de/*

[SEU 99] SEUH'99, Teubner, 1999.

[Stp 04] TU Dresden, Fakultät Informatik: Softwaretechnologie-Praktikum *http://www-st.inf.tu-dresden.de/Lehre/praktikum.htm*

Projekte der Lehre mit hochschulexternen Kunden

Tilmann Hampp, Stefan Opferkuch, Rainer Schmidberger
Abteilung Software Engineering, Institut für Softwaretechnologie, Universität Stuttgart
Universitätsstr. 38, 70569 Stuttgart
{hampp, opferkuch, schmidberger}@infomatik.uni-stuttgart.de

Zusammenfassung

Praktische Projektarbeit ist Teil vieler Informatik- und Softwaretechnik-Curricula. Die Aufgabenstellungen, die diesen Projekten zugrunde liegen, entstammen oft dem Themenkreis des betreuenden Instituts oder sind speziell von den Betreuern erdacht und für keinen wirklichen Verwendungszweck vorgesehen. Teilweise ist die Aufgabenstellung auch von den Studenten selbst festgelegt. In allen drei Fällen wird die Kundenrolle nicht realitätsnah im Projekt vertreten. Dieser Artikel beschreibt am Beispiel zweier durchgeführter Praktika, wie externe Partner eine Kundenrolle in einem studentischen Projekt übernehmen können und welche Schwierigkeiten und welcher Nutzen hierdurch entstehen.

1 Einführung

Immer häufiger gehört praktische Projektarbeit zum Curriculum des Informatik- oder Softwaretechnik-Studiums. Die Projekte werden typischerweise über einen längeren Zeitraum (ein bis zwei Semester) mit Beteiligung mehrerer Studenten (typischerweise drei bis zwölf) durchgeführt. Die geleistete Entwicklungsarbeit entspricht kleineren bis mittleren Projekten, wie sie in der Wirtschaft durchgeführt werden. Selbst wenn berücksichtigt wird, dass Studenten noch nicht die Produktivität eines in der Wirtschaft angestellten Entwicklers haben und zudem nicht Vollzeit am Projekt arbeiten, steht eine Gesamt-Entwicklungsleistung von mehreren Entwicklermonaten zur Verfügung.

Die Aufgabenstellungen der studentischen Projekte stammen meist aus dem Tätigkeitsfeld des betreuenden Instituts, sind „künstlich“ vom Betreuer erdacht oder von den Studenten selbst gewählt. „Künstlich“ erdachte Aufgabenstellungen haben den Vorteil, dass der Gesamtaufwand durch Ändern der Aufgabenstellung leicht gesteuert werden kann. Man kann die Aufgabe Jahr für Jahr leicht modifizieren, was zu geringerem Betreuungsaufwand führt. Auch Musterlösungen können leichter erstellt werden. Der Nachteil der „künstlich“ erdachten Aufgabenstellung ist, dass eine Verwendung der

Ergebnisse nach Ende des Praktikums praktisch ausgeschlossen ist. Damit wird geleistete Arbeit verschwendet und die Motivation der Studenten gemindert. Von Studenten selbst gewählte Aufgabenstellungen führen u. U. zu einer Verwendungsmöglichkeit der Ergebnisse, die Studenten erhalten aber den fatalen Eindruck, dass sie die Anforderungen an das Produkt selbst festlegen können. Sowohl Projekte mit „künstlich“ erdachten Aufgabenstellungen als auch von den Studenten selbst gewählte Aufgabenstellungen haben den gravierenden Nachteil, dass sie das spätere Tätigkeitsprofil eines Softwareentwicklers nur unzureichend widerspiegeln, vor allem bezüglich der Analyse und Spezifikation.

Die Aufgabenstellung an einem externen Bedarf zu orientieren und die Kundenrolle des Projekts mit einem externen Partner zu besetzen, hat zwei offensichtliche Vorteile: Erstens ist es für die Motivation der Studierenden fördernd, wenn das Ergebnis eingesetzt wird (oder der Einsatz zumindest nicht grundsätzlich ausgeschlossen ist). Zweitens kann die Analysephase weit realistischer als bei institutsinternen oder erdachten Aufgabenstellungen durchgeführt werden, da ein „echter“ Kunde als Interviewpartner für die Studierenden zur Verfügung steht.

Im Folgenden werden zwei studentische Projekte beschrieben, die am Institut für Softwaretechnologie der Universität Stuttgart im Studiengang Softwaretechnik durchgeführt wurden und deren Aufgabenstellung durch externe Partner vorgegeben war. In beiden Projekten wurde die Kundenrolle auch durch diese externen Partner besetzt.

2 Softwarepraktikum im Grundstudium

2.1 Rahmenbedingungen

Das Softwarepraktikum muss von allen Studenten der Softwaretechnik im Grundstudium absolviert werden. Das Praktikum wird in Gruppen mit je drei Studenten durchgeführt, der Umfang beträgt je Student 6 Semesterwochenstunden. Pro Gruppe steht damit eine Arbeitsleistung von 720 Entwicklerstunden zur Verfügung.

2.2 Vorbereitungen

Vor Beginn des Praktikums musste ein Partner gefunden werden, der Bedarf für ein Softwaresystem hat, das innerhalb der Rahmenbedingungen des Softwarepraktikums realisiert werden kann. Dieser Bedarf sollte einerseits so groß sein, dass für den Partner die Teilnahme am Praktikum interessant war, andererseits sollte diese Software natürlich in keiner Weise unternehmenskritisch sein, da keinerlei Gewährleistung für die Qualität der Ergebnisse übernommen werden konnte. Zudem sollte keine spezielle Infrastruktur wie z.B. spezielle Hardware oder eine spezielle Datenbank erforderlich sein. Die Implementierung sollte in Java oder in Ada95 erfolgen, weil die Studenten zuvor diese beiden Programmiersprachen gelernt hatten. Bedingt durch die Laufzeit eines Praktikums und den Vorlauf, den man bei der Partnersuche eingeplant hatte, konnten

keine kurzfristig verfügbaren Ergebnisse in Aussicht gestellt werden. Die Zeitspanne zwischen Partnersuche und Ende des Praktikums betrug etwa neun Monate. Nach vielen Gesprächen mit möglichen Partnern, die alle sehr aufgeschlossen waren, konnten die Weiterbildungsabteilung einer großen deutschen Bank sowie eine Weiterbildungs-Akademie als geeignete Partner mit einer passenden Aufgabenstellung gefunden werden.

Beide Partner sind Weiterbildungseinrichtungen, die die Zufriedenheit der Seminarteilnehmer über Beurteilungsbögen erfragen. Die Teilnehmer füllen in beiden Häusern nach Seminarende diese (Papier-)Bögen aus, der Dozent sammelt sie anschließend ein. Verdichtungen der Abgaben oder statistische Trends können nur mit hohem manuellem Aufwand erstellt werden. Aus dieser Problemstellung heraus entstand nun die Idee für ein System, das online die Erfassung des Fragebogens für die Seminarteilnehmer ermöglicht. Die ausgefüllten Fragebögen werden in einer Datenbank gespeichert und können dort für verschiedene Abfragen genutzt werden. Systeme dieser Art waren am Markt praktisch nicht verfügbar oder hatten deutliche Mängel.

Nachdem die Idee grob fixiert war, wurden die wesentlichen Anforderungen an ein solches System – vergleichbar einem Lastenheft – gemeinsam mit den Kunden erhoben. Es wurde eine Grob-Architektur entworfen und eine Aufwandsschätzung vorgenommen. In dieser Phase waren noch keine Studenten beteiligt. Es zeigte sich schnell, dass der Aufwand zur Realisierung deutlich über dem vorgesehenen Rahmen des Softwarepraktikums lag. Aus diesem Grunde wurde das Gesamtsystem in drei Komponenten aufgeteilt: eine Komponente zur Erfassung der Fragebogen-Schablone, eine Web-Server-Komponente zur Anzeige und Erfassung der Fragebögen und eine Auswertungs-Komponente. Damit hatten nicht alle Gruppen des Softwarepraktikums die gleiche Aufgabenstellung, der Umfang war aber in etwa gleich.

2.3 Durchführung

Das Praktikum begann im Februar 2003 und endete im August 2003. Es nahmen etwa 60 Studenten teil. In der ersten Veranstaltung stellten sich die Kunden vor, und das Lastenheft wurde präsentiert. Nach Zusammenstellung der Gruppen (aus je drei Studenten) und Zuordnung der Gruppen zu einer der drei Komponenten wurden sechs Teams gebildet, die aus je drei Gruppen für die drei Einzelkomponenten bestanden. So erstellte jedes Team ein Gesamtsystem. Wichtig hierbei war, dass alle Teams die Komponenten an genau festgelegten Schnittstellen verbinden mussten. Der Ausfall einer Gruppe konnte so durch eine andere Gruppe ersetzt werden.

Die Schnittstelle zwischen den Komponenten bildeten die Klassen des fachlichen Domänenmodells, das über eine Persistenzschicht auf eine relationale Datenbank abgebildet wurde. Die Persistenzschicht wurde von den Betreuern gestellt und war nicht Teil der studentischen Arbeit. Das fachliche Domänenmodell wurde in einem Workshop gemeinsam mit den Studenten erarbeitet und anschließend den Gruppen als UML-Modell und Java-Code zur Verfügung gestellt. Aus den Domänenklassen wurde auch das relationale Modell abgeleitet, das damit ebenfalls allen Gruppen vorgegeben wurde.

Nach der Kundenpräsentation entwickelten nun die Studenten einen Fragenkatalog für die zwei Wochen später stattfindende Kundenbefragung. Die Kundenbefragung war derart organisiert, dass ein Team (also drei Gruppen) etwa eine Stunde Befragungszeit hatte. Es entwickelten sich sehr lebhafte Diskussionen. Die Betreuer waren anwesend, griffen in die Diskussion aber nur ein, um eine gleiche Aufgabenstellung für die insgesamt sechs Teams sicherzustellen.

Die auf die Kundenbefragung folgenden Projektabschnitte Spezifikation, Reviews der Spezifikation, Entwurf, Implementierung und Test waren von der Tatsache, dass die Aufgabenstellung von einem externen Kunden stammte, nicht signifikant betroffen. Lediglich nach der Qualitätssicherung der Spezifikation durch Reviews mussten Unklarheiten bei der Aufgabenstellung erneut mit dem Kunden geklärt werden.

Die Abnahme der Resultate fand als Präsentation wieder mit Kundenbeteiligung statt.

2.4 Qualität der Abgaben

Nahezu alle Gruppen erzielten vorzeigbare Ergebnisse. Die Abschlusspräsentationen wurden überwiegend professionell vorgetragen, und die gezeigten Ergebnisse hatten – soweit es aus der Präsentation erkennbar war – Produktqualität. Bei etwa zwei Drittel der Abgaben wurde die Bedienbarkeit von den Kunden besonders gelobt.

Die besten zwei Abgaben je Komponente wurden schließlich von den Betreuern ausgewählt und auf weitere Produktqualität wie Wartbarkeit, speziell Änderbarkeit und Lesbarkeit, genauer untersucht. Diese recht zeitintensive Untersuchung ergab, dass nahezu alle Abgaben im Bereich der Wartbarkeit Schwächen hatten. Die Lesbarkeit war z. B. durch ungeschickte Bezeichner für Klassen und Methoden schlecht. Auch der Entwurf, speziell der Zusammenhalt innerhalb der Klassen, hatte oftmals Mängel, sodass es mühsam war, Anpassungen vorzunehmen oder Teile wiederzuverwenden. Den Studenten fehlte in diesen Bereichen offensichtlich noch die Erfahrung.

Dennoch konnte ein Gesamtsystem integriert werden, das bei den Kunden zu Präsentationszwecken eingesetzt wurde. Produktqualität im Sinne einer unmittelbaren Vermarktung hatten die Abgaben allerdings nicht.

2.5 Einschätzung des Praktikums durch die Studenten

Im Anschluss an das Softwarepraktikum wurde eine empirische Untersuchung über die Einschätzung des Praktikums durch die Studenten durchgeführt. Dabei wurden die Studenten in einem Fragebogen nach ihrer Einschätzung verschiedener Aspekte des Softwarepraktikums befragt. Der Fragebogen wurde anonym beantwortet, sodass für die Betreuer des Praktikums und den Auswerter der Bögen kein Rückschluss auf die jeweiligen Studenten möglich war. Der Fragebogen umfasste 14 teils offene, teilweise geschlossene Fragen. Der Rücklauf betrug 50 beantwortete Bögen bei 57 Teilnehmern am Softwarepraktikum. Die darin enthaltenen Antworten können also als repräsentativ gelten.

Der überwiegende Teil der Studenten gibt an, dass ihnen das Softwarepraktikum insgesamt Spaß gemacht hat. Diese Aussage machten 43 der 50 Studierenden.

Des Weiteren bewerteten die Studenten ihre Motivation bei der Durchführung des Praktikums im Vergleich zu anderen Lehrveranstaltungen des Studiums als höher. Dieser Meinung waren 35 Studenten, 9 bewerteten ihre Motivation als gleich hoch, 6 Studenten als niedriger. Fast alle Studenten, nämlich 49 von 50, hatten das Gefühl, dass im Softwarepraktikum ein Lerneffekt eingetreten ist.

Als besonders positiver Aspekt des Softwarepraktikums wurde von den Studenten hervorgehoben, dass im Praktikum eine eigenständige Teamarbeit unter realen, praxisnahen Projektbedingungen durchgeführt werden konnte. Negativ wurden die nach Meinung der Studenten veraltete Rechner- und Softwareausstattung und Terminkonflikte mit anderen Lehrveranstaltungen eingeschätzt.

Ebenfalls negativ wurden die Aspekte verspätete Absprachen über Schnittstellen, ungenaue Anforderungen des Kunden an die Software und Unstimmigkeiten innerhalb der Gruppen und zwischen den Teams bewertet. Dabei ist jedoch festzustellen, dass dies Schwierigkeiten sind, unter denen viele Softwareprojekte in der Industrie ebenfalls leiden und somit die Studierenden in gewisser Weise auf die Probleme, die sie in ihrem späteren Berufsleben unter Umständen erwarten, vorbereitet werden.

Der Umfang des Praktikums war nach Einschätzung von 32 Studierenden genau richtig, 16 Studierende empfanden den Aufwand als leicht zu hoch. Dies zeigt, dass der Umfang der Aufgabe für das Softwarepraktikum richtig gewählt worden war und zukünftige Aufgaben nicht umfangreicher gestaltet werden sollten. Schließlich wurde die spezielle Konstellation dieses Softwarepraktikums mit der Beteiligung eines externen Kunden und dem geplanten Einsatz der entwickelten Software von der Mehrheit der Studierenden, 37 von 50, als positiv bewertet.

2.6 Abschließende Bewertung

Der Nutzen für den externen Partner steht in einem guten Verhältnis zu seinem Aufwand, der für Vorbesprechung, Kundengespräche und Abnahme etwa 3 bis 4 Arbeitstage betrug. Das Projektresultat konnte zwar nicht unmittelbar als Produkt eingesetzt werden, aber es stand für interne Diskussionen als Prototyp bereit.

Der Nutzen für die Studenten lässt sich aus den Resultaten der Untersuchung ablesen. Der subjektive Eindruck der Betreuer war, dass der Lernerfolg der Studenten über dem der Vorjahre lag. Zudem war ein von den externen Partnern finanziertes Fest ein gelungener Abschluss.

Für die Betreuer bedeutet ein Praktikum mit externen Kunden einen erkennbar höheren Arbeitsaufwand. Die einzelnen Aufwände wurden zwar nicht genau erfasst, wir schätzen aber, dass der Gesamtaufwand um etwa ein Drittel höher liegt als bei Projekten ohne externen Partner. Dies liegt an der Partner- und Aufgabensuche im Vorfeld, aber auch an den Aufwänden zur Koordination im laufenden Projekt.

Als besonders wichtig hat sich die einheitliche Fixierung der Schnittstellen zwischen den Komponenten herausgestellt. Hier ist auch eine gewisse Erfahrung im Be-

reich Klassenmodellierung bei den Betreuern erforderlich. Wie zu erwarten war, hatten sich einige Gruppen aufgelöst (durch Schwund der Mitglieder) oder neu formiert. Durch die festen Schnittstellen hat sich dies aber auf die übrigen Gruppen nicht weiter ausgewirkt. So konnte jede Gruppe ihre Abgabe mit den jeweils anderen Komponenten integrieren.

3 Studienprojekt im Hauptstudium

Im Hauptstudium des Studiengangs Softwaretechnik sind zwei Studienprojekte vorgesehen, in denen von Studenten jeweils ein vollständiges Softwareprojekt durchgeführt wird. Das Studienprojekt A ist das erste Projekt und wird von den Informatik-Instituten angeboten, das Studienprojekt B wird im Anwendungsfach absolviert. Wir haben im Wintersemester 2003/04 ein Studienprojekt A mit externen Kunden angeboten und durchgeführt.

3.1 Rahmenbedingungen

Die Rahmenbedingungen für das Studienprojekt A werden durch die Prüfungsordnung festgelegt. Die Teilnehmerzahl ist auf 7 bis 12 Teilnehmer begrenzt, der Umfang für die praktische Arbeit ist auf 10 Semesterwochenstunden je Teilnehmer festgelegt, das sind 400 Entwicklerstunden. Die vorgegebene Dauer beträgt 9 Monate. Im praktischen Teil sollen die Teilnehmer ein vollständiges Softwareprojekt durchführen. Dazu gehören die Phasen Konzeption (Analyse, Spezifikation), Realisierung (Entwurf, Feinentwurf, Codierung, Modul-Test) und Montage (Integration, Gesamttest, Abnahme und Übergabe). Des Weiteren muss Qualitätssicherung und Projektmanagement geleistet werden. Neben der praktischen Arbeit gehören zum Studienprojekt begleitende Vorlesungen und ein Seminar.

In der Regel wird ein Studienprojekt von einem Institut angeboten. Dabei übernimmt ein Mitarbeiter die Kundenrolle, zwei Mitarbeiter betreuen die Teilnehmer. Die Anforderungen an Projekt und Produkt werden vom Institut festgelegt. Weil ein Mitarbeiter die Rolle des Kunden nicht vollständig wahrnehmen kann, haben wir hochschulexterne und damit reale Kunden in das Studienprojekt eingebunden.

3.2 Vorbereitung des Projekts

Ein Semester vor Beginn des Studienprojekts wurde das Projektziel innerhalb des Instituts geklärt und festgelegt: Im Studienprojekt sollte ein System zur Verwaltung und Darstellung von Anforderungen und weiteren Informationen erstellt werden. Als Architektur war eine Basissoftware für die Verwaltung von Projektinformationen gefordert, dazu sollte ein Plug-in für die Verwaltung von Anforderungen und Beziehungen zwischen Anforderungen erstellt werden. Weitere Plug-ins für die Verwaltung von Handbucheinträgen oder Testfällen, integriert mit den Anforderungen, sollten möglich sein.

Dieses System sollte als Web-Anwendung in Java realisiert werden und auf einer Datenbank basieren.

Wir konnten zwei externe Kunden gewinnen: Ein Kunde war ein Schulungsunternehmen, das ein System zur Verwaltung der Schulungen einsetzt. Für dieses System sollten die Anforderungen und das Handbuch mit dem Produkt nachdokumentiert werden. Dieser Kunde war auf das Werkzeug angewiesen: Der Mitarbeiter, der das Verwaltungssystem sehr genau kennt, verlässt das Unternehmen. Als zweiter Kunde stellte sich ein Softwareunternehmen zur Verfügung.

Alle Studienprojekte eines Semesters starten zu Semesterbeginn. Die Studenten melden sich für ihr Wunschprojekt an; wollen zu viele Studenten an einem Projekt teilnehmen, werden die Teilnehmer für das Projekt ausgewählt. Alle Studenten können Alternativprojekte angeben, auf jeden Fall aber an einem Studienprojekt teilnehmen. Für das Studienprojekt wurden nach der Anmeldung 10 Teilnehmer ausgewählt.

3.3 Projektorganisation

Das Studienprojekt war als Entwicklungsprojekt organisiert, also als Projekt für die Herstellung eines Produkts für den Softwaremarkt. Die externen Kunden übernahmen die Rolle von Pilotkunden. Ein Mitarbeiter fungierte als Marketing und interner Kunde. Diese Organisation hat mehrere Gründe:

- Der interne Kunde kann die Anforderungen der beiden Pilotkunden bündeln und die Konsistenz so weit sichern, dass ein Produkt machbar ist. Da die Kunden aus sehr unterschiedlichen Bereichen kamen, haben wir mit widersprüchlichen Anforderungen gerechnet.
- Das Ziel war, ein erweiterbares Produkt zu erstellen, das durch Plug-ins für unterschiedliche Zwecke angepasst werden kann. Die Anforderungen auf dieses Ziel hin zuzuschneiden, war eine Aufgabe des internen Kunden.
- Externe Kunden sind nicht ständig verfügbar. Um die Kommunikationswege kurz zu halten und die externen Kunden nicht zu sehr zu beanspruchen, diente der interne Kunde als Ansprechpartner für die Studenten.
- Als Risiko haben wir gesehen, dass ein externer Kunde abspringt. Durch den internen Kunden hängt das Studienprojekt jedoch nicht existenziell vom externen Kunden ab.

Zusätzlich standen den Teilnehmern zwei Mitarbeiter als Betreuer zur Seite. Die Teilnehmer stellten den Projektleiter, zusätzlich geforderte Rollen waren ein Configuration-Management- und Qualitätssicherungs-Beauftragter und ein Chefarchitekt.

3.4 Anforderungen an den Ablauf des Projekts

Das Projekts gliederte sich in die folgenden von den Mitarbeitern vorgegebenen Phasen:

Im Vorprojekt arbeiteten die Teilnehmer in drei Gruppen. Jede Gruppe führte Anforderungsanalyse, Entwurfsüberlegungen, Kostenschätzung und Planung durch und

erstellte ein Angebot. Zur Analyse fanden Gespräche mit dem internen und den externen Kunden statt. Abschluss der Vorprojektphase war die Angebotspräsentation, in der jede Gruppe den Kunden ihr Angebot vorstellte. Die Kunden einigten sich auf eine Gruppe, die den Zuschlag erhielt.

Gefordert worden war die Projektabwicklung in zwei Iterationen. Abschluss der ersten Iteration bildete ein lauffähiges Produkt mit eingeschränkter Funktionalität. Dieses Produkt wurde von den Kunden evaluiert, sodass Anforderungsänderungen in der zweiten Iteration realisiert werden konnten. Nach der zweiten Iteration wurde das vollständige Produkt ausgeliefert.

Dieses iterative Vorgehen haben wir gewählt, weil einer der beiden Kunden auf das Produkt angewiesen war. Dieser Kunde konnte das Produkt früh einsetzen; außerdem konnten noch Änderungen und Korrekturen in der zweiten Iteration durchgeführt werden. Als zweiten Vorteil sahen wir, dass die externen Kunden während des Projekts eingebunden werden und den Fortschritt verfolgen können.

Die Pilotkunden sind durch diesen Ablauf und die Organisation an mehreren Stellen in das Projekt eingebunden: Zur Analyse der Anforderungen finden Kundengespräche statt, die externen Kunden wählen ein Angebot in der Angebotspräsentation aus, bewerten einen Prototyp und nehmen so die Spezifikation ab. Dann nehmen sie die erste Iteration ab und stehen zur Analyse von Anforderungsänderungen für die zweite Iteration und zur Abnahme des Endprodukts zur Verfügung.

Begonnen hat das Studienprojekt Ende Oktober 2003; das Vorprojekt endete im Dezember 2003. Das Ende der ersten Iteration war auf Mai 2004 festgelegt, Ende August 2004 wurde das Projekt abgeschlossen.

3.5 Durchführung des Projekts

Nach Abschluss des Vorprojekts wird das Projekt von den Teilnehmern, insbesondere vom Projektleiter, geplant. Dieser hatte die folgenden externen Meilensteine geplant:

M0: Projektplan
M1: Spezifikation des Gesamtsystems
M2: Entwurf für die erste Iteration
M3: Abnahme der ersten Iteration
M4: Entwurf für die zweite Iteration
M5: Abnahme des Gesamtsystems
M6: Projektende

Die externen Pilotkunden waren nicht bei allen externen Meilensteinen beteiligt, Projektplan und Entwurf wurden ausschließlich vom internen Kunden und den Betreuern abgenommen.

Im Projekt wurden alle Anforderungen in der Spezifikationsphase der ersten Iteration erfasst und dokumentiert. Die Abnahme der Spezifikation erfolgte durch den internen Kunden, dann wurde den externen Kunden ein Prototyp gezeigt.

Für die erste Iteration wurden die wesentlichen Anforderungen identifiziert, damit ein einsatzfähiges Produkt entstand. Für diese Anforderungen wurde ein Entwurf er-

stellt, der ausreichend erweiterbar für alle Anforderungen sein sollte. Implementiert wurden für die erste Iteration nur diese wesentlichen Anforderungen. Am Ende der ersten Iteration wurde das lauffähige Produkt den Kunden präsentiert und für den Einsatz zur Verfügung gestellt. In Analyserunden wurden weitere Anforderungen erhoben; kleinere Korrekturen waren noch notwendig. Mit dem internen Kunden wurden die neu erhobenen Anforderungen und die noch nicht realisierten Anforderungen priorisiert. Anschließend wurde der Entwurf angepasst, erweitert und dann implementiert. Das Handbuch wurde während der zweiten Iteration erstellt.

Die Qualitätssicherung erfolgte durch interne Reviews aller Dokumente einschließlich Codeaudits, Reviews mit dem internen Kunden und den Betreuern und durch Test des Systems. Codeaudits wurden eingesetzt, weil sich Schwierigkeiten beim Unit-Test herausstellten. Ein umfangreicher Systemtest wurde vor dem Ende jeder Iteration durchgeführt. Die Abnahme erfolgte jeweils zuerst durch den internen Kunden und die Betreuer, dann durch die Pilotkunden.

3.6 Ergebnisse des Projekts

Das Projekt wurde erfolgreich durchgeführt. Alle Meilensteine wurden zum geplanten Termin erreicht. Der Aufwand lag um rund 20% über den vorgesehenen 4000 Entwicklerstunden, aber im Rahmen des durch Kostenschätzung ermittelten und geplanten Aufwands.

Bereits nach der ersten Iteration zeigte sich eine hohe Brauchbarkeit des Produkts, diese Einschätzung bestätigte sich bei Projektende. Beide externen Kunden setzen das Produkt derzeit ein. Das System wurde bereits erweitert, die Wartbarkeit des Produkts wurde dabei vom Wartungsentwickler als sehr gut eingeschätzt.

3.7 Einschätzung des Projekts durch die Studenten

Im Anschluss an das Studienprojekt wurde ebenfalls eine empirische Untersuchung über die Einschätzung des Projekts durch die Studenten durchgeführt. Es konnten 9 von 10 am Projekt beteiligte Studenten befragt werden.

Alle befragten Studenten machten die Aussage, dass ihnen das Studienprojekt insgesamt Spaß gemacht hat. Entscheidend hierfür war die positive Grundstimmung im Projektteam, der reale Einsatz der Software außerhalb der Hochschule, der hohe Stellenwert der Software für den Kunden, der Einsatz interessanter neuer Technologien und die Möglichkeit zur Erweiterung der eigenen Kenntnisse.

Die Beteiligung eines hochschulexternen Kunden beurteilten alle Studenten als positiv. Hierbei war vor allem von Bedeutung, dass eine sehr hohe Motivation bestand, das Projekt erfolgreich durchzuführen. Dies wurde von den Studenten mit realistischen Arbeitsbedingungen und einer erhöhten Erwartungshaltung an die Funktionalität des Produkts begründet. Als ebenfalls motivationssteigernd wurde die Möglichkeit genannt, Vorgehensweisen in der Softwareentwicklung außerhalb der Universität kennen zu lernen. Von den Studenten wurde lediglich negativ beurteilt, dass aus ihrer Sicht nach

Beendigung des Projekts zu wenig Feedback durch die hochschulexternen Kunden gegeben wurde.

Die Frage, ob sie ein ähnliches Projekt nochmals durchführen würden, bejahten 8 von 9 Studenten. Dabei war vor allem ausschlaggebend, dass diese Art von Projekt eine sehr gute Vorbereitung auf reale Projekte in der Industrie bietet und die höheren Anforderungen an die Projektdurchführung die Motivation steigern.

3.8 Abschließende Bewertung

Die Betreuung des Studienprojekts ist mit externen Kunden aufwändiger als ohne externe Kunden: Der Aufwand für die Rolle des Kunden fällt für die Mitarbeiter nicht weg, sondern bleibt bei vermutlich gleichem Aufwand als interner Kunde erhalten. Zusätzlich steigt der Aufwand für die Vorbereitung deutlich: Externe Kunden müssen für das Studienprojekt gewonnen und die Aufgabenstellung muss abgesprochen und angepasst werden. Während des Studienprojekts müssen Termine koordiniert werden.

Für die Industriepartner, die als externe Kunden beteiligt sind, fällt ebenfalls Aufwand an. Bei fast allen Treffen waren zwei oder mehr Mitarbeiter jedes externen Kunden anwesend, für die Treffen ist damit pro Kunde ein Aufwand von rund 8 Arbeitstagen angefallen (ohne Anreise). Der Nutzen ist dafür sehr hoch: Der Kunde bekommt ein einsatzfähiges Produkt, das seine Anforderungen realisiert.

Für die Studenten, die am Studienprojekt teilnehmen, bedeuten externe Kunden eine größere Herausforderung. Der Kontakt zu realen Kunden stellt sie vor eine neue Situation.

Schwierigkeiten zeigten sich darum besonders in der Angebotsphase: Den Studenten war zwar klar, dass sie den Kunden für ihr Angebot gewinnen müssen, aber nicht, wie sie ihr Angebot so gestalten, dass sich der Kunde darin wiederfindet. Erschwert wurde diese Aufgabe durch die unterschiedlichen Erwartungen der beiden externen Kunden. Schwierigkeiten beim Kundenkontakt zeigten sich auch bei der Analyse von Anforderungsänderungen nach der ersten Iteration: Die Studenten hatten eine abwehrende Haltung gegenüber den Änderungen, eine konstruktive Diskussion wurde abgeblockt, indem auf den Mehraufwand verwiesen wurde.

Deutlich hat sich in der Analyse gezeigt, dass ein externer Kunde nicht die Sprache der Entwickler spricht. Dadurch tauchten nach der Analyserunde weitere Fragen auf, die in einem zweiten Treffen geklärt wurden.

Während die Erfahrung bei der Analyse gewollt ist, sehen wir in den Schwierigkeiten der Angebotspräsentation ein Defizit der Lehre. Nachgereicht wurde daher ein Überblick über den typischen Ablauf der Angebotsphase in der Industrie und Hinweise zur erfolgreichen Angebotspräsentation. Diese Inhalte werden wir in die begleitenden Vorlesungen für kommende Studienprojekte einfließen lassen, um die Studenten besser auf die Kundenkontakte vorzubereiten. Außerdem wurden alle Projektergebnisse, die die externen Kunden bekommen, vorher vom internen Kunden und den Betreuern geprüft.

Das iterative Vorgehen schätzen wir grundsätzlich positiv ein: Für die Kunden hat es den Vorteil, dass noch Änderungen in das Endprodukt übernommen werden können, dass sie relativ früh wieder in das Projekt eingebunden werden und dass sie eine lauffähige Version sehen. Aber auch für die Studenten ist eine lauffähige Version, die den Kunden vorgestellt werden kann, motivierend. Der Zeitplan für das Projekt wird durch die zwei Iterationen allerdings knapp, ein Vorgehen mit mehr als zwei Iterationen halten wir daher unter diesen Rahmenbedingungen für nicht durchführbar.

Ein externer Kunde wirkt auf die Studenten stark motivierend. Sie erfahren, dass ihre Arbeit gebraucht wird und gute Ergebnisse geschätzt werden. Dazu gehört auch, dass das Produkt tatsächlich eingesetzt und gewartet wird.

4 Zusammenfassung

Zusammenfassend ergibt sich für die externen Partner ebenso wie für die Studenten eine „win-win"-Situation: Für wenig Aufwand und Kosten erhält der externe Partner ein maßgeschneidertes Produkt. Die Studenten erleben echte Kundengespräche und sind durch die Perspektive, dass ihre Ergebnisse verwendet werden, angespornt. Wichtig ist auch der „Spaßfaktor", den die Studenten überwiegend in den Befragungen angegeben hatten.

Die Betreuer haben zwar einen etwas höheren Aufwand, dafür entstehen aber Kontakte in die Industrie, die vielfältig weitergenutzt werden können. Als notwendig erachten wir ausreichend Vorlaufzeit, um die externen Kunden zu gewinnen und die Aufgabenstellung zu klären.

Die Erfahrung mit den Resultaten aus Grundstudiums- und aus Hauptstudiumsprojekten sind unterschiedlich: Die Ergebnisse aus den Grundstudiumsprojekten können als Prototyp oder für einen Testbetrieb durchaus dienen, die Codequalität wird aber für eine längerfristige Investition nicht tragfähig sein. Ergebnisse aus Hauptstudiumsprojekten entsprechen unserer Einschätzung nach qualitativ dem gängigen Industriestandard oder liegen sogar darüber.

Wir werden daher zukünftig Projekte der Lehre nach Möglichkeit mit externen Kunden durchführen.

Literatur

[Lud 99] J. Ludewig: Softwaretechnik in Stuttgart – ein konstruktiver Informatik-Studiengang. Informatik-Spektrum 22(1), 1999, 57-62.

[Lud 01] J. Ludewig (Hrsg.): Praktische Lehrveranstaltungen im Studiengang Softwaretechnik: Programmierkurs, Software-Praktikum, Studienprojekte, Fachstudie. Mit Beiträgen von Stefan Krauß, Jochen Ludewig, Patricia Mandl-Striegnitz, Ralf Melchisedech, Ralf Reißing. Bericht der Fakultät Informatik, Universität Stuttgart, 3. Auflage, 2001.

AMEISE – Didaktische Vielfalt für SESAM

Susanne Jäger[1)], Elke Hochmüller[2)], Roland Mittermeir[1)], Andreas Bollin[1)], Daniel Wakounig[1)]

[1)] Institut f. Informatik-Systeme, Universität Klagenfurt
Universitätsstr. 65-67, A-9020 Klagenfurt
{susi,roland,andi,daniel}@isys.uni-klu.ac.at

[2)] Fachhochschule Technikum Kärnten
Primoschgasse 8, A-9020 Klagenfurt
E.Hochmueller@fh-kaernten.at

1 Motivation

Der Stuttgarter SESAM-Simulator[1] [Dra00] hilft Studierenden, Projektmanagementfähigkeiten ohne Gefährdung eines realen Projekts zu erwerben. Allerdings ist die Feedbackgenerierung aufwändig. AMEISE[2] beschleunigt dies und erlaubt didaktische Variationen der Projektsimulationen [Mit03].

2 AMEISE-spezifische Erweiterungen

AMEISE erlaubt SESAM in größeren Lehrveranstaltungen einzusetzen. SESAMs Auswertungsengpass wurde durch einen Mechanismus zur Online-Auswertung überwunden. Die Benutzerschnittstelle wurde so gestaltet, dass durch Auswahl aus entsprechenden Menüs SESAM-Befehle entstehen. Die SESAM-Grundidee blieb aber erhalten.

Zwei Hilfsmittelkomponenten, Ratgeber (Consultant) und Schutzengel (friendly peer), reduzieren den Betreuungsaufwand des Lehrveranstaltungsleiters während der Simulationsläufe. Sie erlauben den Simulator auch ohne Vor-Ort-Betreuung zu verwenden. Der Ratgeber kann vom studentischen Projektmanager konsultiert werden. Der Schutzengel ist einem erfahrenen Kollegen gleichzusetzen, der das studentische Handeln beobachtet und bei drohender Krise mit Tipps zur Seite steht.

Mit Hilfe des externen Software-Hauses (Partialspiel) ist auch die Auslagerung von Aufgabenstellungen konkreter Projektphasen an Dritte möglich. Dadurch kann die Simulation auf einzelne Entwicklungsphasen fokussiert werden, ohne dabei die Gesamtperspektive zu verlieren. Mittels Rollback kann man ein Projekt auf spezifische Entscheidungspunkte zurücksetzen. So können Effekte von Entscheidungen erprobt und analysiert werden, ohne das gesamte Projekt nochmals von Anfang an simulieren zu müssen. Eine weitere Überlegung war, den Studierenden Vergleichsmöglichkeiten anzubieten. Mittels Gruppenvergleich kann das eigene Projekt relativ zu den Projekten anderer beurteilt werden.

---

[1] SESAM – Software Engineering Simulated by Animated Models*; http://www.iste.uni-stuttgart.de/se/research/sesam* (Stand November 2004).

[2] AMEISE – A Media Education Initiative for Software Engineering; gefördert von bm:bwk unter NML-1/77, 2001. *http://ameise.uni-klu.ac.at* (Stand November 2004).

3 Erfahrungen mit dem Einsatz von AMEISE

Eine Schulung besteht aus fünf Teilen (Einführung in den Simulator, erste Simulation, Feedback-Sitzung, zweite Simulation, Feedback-Sitzung). Der Lernerfolg zwischen den AMEISE-Simulationen deckt sich mit dem in [Man01] berichteten. Allerdings haben die AMEISE-Features doch zu neuen Erfahrungen geführt [Jae04]. So hat die neue Benutzungsschnittstelle den durchschnittlichen Zeitaufwand für eine Simulation von 4,5 auf 3,5 Stunden reduziert. Da zwischen der dritten und vierten Stunde deutliche Ermüdungserscheinungen auftreten, ist diese Verbesserung beachtlich.

Durchwegs positives Feedback erhielten wir für die Hilfsmittelkomponenten Ratgeber und Schutzengel. Sie reduzieren die von SESAM geschaffene nicht-fachliche Komplexität und verringern den Betreuungsaufwand während eines Spiels deutlich.

Mit Hilfe der Online-Auswertung können Studierende bereits am Ende der Simulation eine Eigenanalyse ihres Projekts vornehmen und ggf. reagieren. Die mehrtägigen Wartezeiten auf Auswertungen entfallen nun. Allerdings ist es dennoch sinnvoll, nach dem ersten Simulationslauf eine Feedback-Sitzung anzubieten. Dadurch können Studierende Effekte und deren Ursachen hinterfragen und Strategien gemeinsam mit der Lehrveranstaltungsleitung bewerten.

Die Vergleichskomponente unterstützt Lehrveranstaltungsleiter und Studierende. Sie ermöglicht, Projekte einander gegenüberzustellen und Lernfortschritte aufzuzeigen. Ebenso können Studierende Vergleiche mit Alternativlösungen anstellen.

4 Resümee

Studierende wie Praktiker sind sich weitestgehend einig, dass sie aus dem Experimentieren mit AMEISE, das sie in die Rolle eines virtuellen Projektleiters gebracht hat, wesentliche Erfahrungen gewonnen haben. Die Simulationsergebnisse werden weitgehend als realistisch eingestuft. Die Hilfsmittelkomponenten können, je nach didaktischer Zielsetzung, optional zugeschaltet werden. Sie sind einfach erweiterbar.

Auch die Systementwicklung selbst hat wesentliche Ausbildungseffekte. Sie bietet Studierenden die Möglichkeit, an einem großen Legacy-System Probleme zu erfahren, die in üblichen Semesterprojekten nicht auftreten können.

References

[Dra00] A. Drappa, J. Ludewig: Simulation in Software Engineering Training. In: Proc. 23rd Internat. Conf. on Software Engineering, IEEE-CS, 2001, p. 199 - 208.

[Jae04] S. Jäger, E. Hochmüller, R. Mittermeir, A. Bollin, D. Wakounig: Systemunterstützung der Software-Projektmanagement-Ausbildung. TR-ISYS 01/11/04, Univ. Klagenfurt, 2004.

[Man01] P. Mandl-Striegnitz: Qualifizierte Software-Projektmanager durch simulationsbasierte Ausbildung. In: Lichter; Glinz (Hrsg.): Software Engineering im Unterricht der Hochschulen/SEUH, Zürich, 2001.

[Mit03] R. Mittermeir, E. Hochmüller, A. Bollin, S. Jäger, M. Nusser: AMEISE: Concepts, the Environment and Initial Experiences. In: Auer (ed): Proc. Internatl. Workshop ICL, Villach, 2003.

Das Software-Engineering-Praktikum (SEP) – Konzept und Erfahrungen

Peter Kaiser, Sven Klaus
Fachbereich Informatik, Fachhochschule Mannheim
Windeckstr. 110, 68163 Mannheim
{p.kaiser, s.klaus}@fh-mannheim.de

1 Das Praktikum

Im Bachelor-Studiengang für Informatik der FH Mannheim stellt ein einsemestriges Software-Engineering-Praktikum im vierten Fachsemester einen wesentlichen Bestandteil dar. Es hat das Ziel, das praktische Wissen für die erfolgreiche Entwicklung von Software zu vermitteln. Dazu gehören die Anwendung von Software-Prozessen, die Anwendung und das Verständnis von Software-Engineering- und Prozessmanagement-Methoden, das Führen kleiner Teams, sowie Projektpräsentationen.

Das Praktikum baut auf den üblichen Informatik-Vorlesungen auf, wobei dabei bereits ein Mini-Software-Projekt mit dem Fokus auf den Entwicklungsdokumenten durchgeführt wird. Im vierten Semester werden neben wenig anderen Vorlesungen Intensivkurse zu projektrelevanten Themen in Blockvorlesungen gehalten, 12 SWS widmen die Studierenden der Praktikumsaufgabe. Diese umfasst alle Entwicklungsphasen von der Anforderungsermittlung bis zur Pilotanwendung.

Teams à fünf Personen entwickeln die gleiche Web-basierte Applikation. Der Entwicklungsprozess – angelehnt an Spiralmodell und RUP – gibt die Iterationen und Hauptmeilensteine mit Ergebnissen und Terminen vor, lässt den Teams jedoch viel Freiheit. Ausgangspunkt sind – bewusst unklare – Anforderungen. Das Entwicklungsende (etwa „initial operational capability“) ist festgelegt auf zwei Wochen vor Vorlesungsende und wird gefolgt von dem Piloteinsatz.

In Workshops werden von den Teams die Details für Tätigkeiten und Entwicklungsdokumente erarbeitet, z.B. der Inhalt einer Architekturspezifikation. Die Teams können „Umwege“ einschlagen. Durch Statusberichte und Projektreviews vor allen Praktikumsteilnehmern, sowie Team-internen Gesprächen mit den Betreuern wird das Vorgehen diskutiert (reflektiert) und notfalls von diesen korrigierend eingegriffen.

Im SS2004 schlossen vier Teams ihre Aufgabe, ein Werkzeug zur Vorlesungs-evaluierung zu realisieren, mit einer erfolgreichen Pilotanwendung ab. Es gab jedoch erhebliche Funktions- und Qualitätsunterschiede.

2 Die Erfahrungen

Die Studierenden bewerteten den Lernerfolg des Praktikum als sehr hoch. Sie lobten die spannende, praxisnahe Aufgabe, die intensive Betreuung sowie den Lerneffekt bzgl. des Nutzens von guter Planung und gutem Software-Engineering. Negativpunkte waren aus ihrer Sicht die umfangreiche Dokumentation, deren Notwendigkeit erst spät erkannt wurde, sowie die hohe Arbeitsbelastung.

Im Folgenden werden Lessons Learned aus der Sicht der Lehrenden auszugsweise vorgestellt:

- Risiken wurden grundsätzlich nicht erkannt oder unterschätzt. So wurden die Einarbeitungszeiten für unbekannte Programmiersprachen und -umgebungen unterbewertet oder die Möglichkeit sich ändernder Anforderungen erst gar nicht als Risiko eingestuft. Bei der nächsten Durchführung werden daher Krisen wie Erkrankung eines Teammitglieds simuliert, um die Risikowahrnehmung zu verbessern.
- Prototypen wurden nicht ausreichend erstellt.
- Erst in der letzten Iteration wurde realistisch geplant (mit Aktionslisten) und die Planung anschließend auch durchgesetzt.
- In den Iterationen wurde jeweils nur ein Schwerpunkt bearbeitet, z.B. in der ersten Iteration nur die Anforderungsspezifikation, dagegen blieben Architektur, Test usw. außen vor.
- Unklaren Anforderungen standen die Studierenden zunächst sehr ratlos gegenüber.
- Der Umgang mit Projektrückschlägen kann in Vorlesungen nicht gelehrt werden. Um diese Erfahrung zu vermitteln, wurde eine didaktische Kombination zwischen Vorgaben und Endscheidungsfreiheit an vielen zentralen Stellen gewählt. Dies führte zum einen dazu, dass Notwendiges, wie etwa Spezifikationen, nicht aureichend ausgeführt wurde. Auf der anderen Seite wurde durch „Sackgasseneffekte“ ein hoher Lernerfolg erzielt.
- Sehr motivationssteigernd wirkte, dass das entwickelte Produkt tatsächlich benötigt wurde und von den Teams auch pilotmäßig eingesetzt werden musste.

3 Ausblick

Zusammenfassend ist ein positives Resümee dieses Praktikums zu ziehen. Die hochgesteckten Erwartungen bei der Konzeption des Praktikums konnten erfüllt werden. Das Praktikum wird in dieser Form mit einigen Verbesserungen und Erweiterungen, unter anderem ein expliziteres Risikomanagement und höhere Anforderungen an die Dokumentation, in Sommersemester 2005 erneut durchgeführt werden.

Software-Engineering-Unterricht online

Ilse Schmiedecke, Debora Weber-Wulff
TFH Berlin, Labor Online-Learning, Luxemburger Str. 10, 13353 Berlin, *schmiedecke@tfh-berlin.de*
FHTW Berlin, Internationale Medieninformatik, Treskowallee 8, 10318 Berlin *weberwu@fhtw-berlin.de*

1 Ist Online-Unterricht für Software Engineering geeignet?

Der Bedarf an E-Learning-Modulen wächst ständig. So wurde auch im Rahmen des Bachelor-Studienganges Medieninformatik an der Virtuellen Fachhochschule (VFH[1]) ein Online-Modul Software Engineering benötigt. Für das Online-Lehrmaterial galten die bekannten Anforderungen: an Semesterwochen orientierte Lerneinheiten, Tests zur Selbstkontrolle und vor allem die Nutzung der Hyper- und Multimediamöglichkeiten und des Internets zur didaktisch optimalen Präsentation des Lernstoffs [GöM 02].

Die größere Herausforderung war die angemessene Gestaltung des Online-Unterrichts. Denn Software-Engineering-Unterricht muss über reines Wissen hinaus auch fachspezifische „Soft Skills" trainieren: Fachkommunikation, Team-Arbeit und -Organisation und gemeinschaftliche Problemlösung. Das Studium an der VFH unterscheidet sich vom klassischen Fernstudium vor allem dadurch, dass die Studierenden ihren Mentor und die Lerngruppe kennen und online miteinander kommunizieren, etwa im Chat, in dem der Mentor oder die Mentorin mit der gesamten Lerngruppe bestimmte Themen aufgreift und vertieft. Die Frage ist, ob die eingeschränkten Kommunikationsmöglichkeiten in einem virtuellen Lernraum ausreichen, um die erforderlichen Soft Skills zu trainieren. Da wir hier selbst skeptisch waren, möchten wir über unsere Erfahrungen mit dem Modul berichten.

2 Asynchrone Kommunikation

Voraussetzung für echte Teamarbeit ist ein synchronisierter Lernfortschritt, was gut durch bewertete terminierte Einsendeaufgaben zu erreichen ist. Wir sind an dieser Stelle noch eine Schritt weiter gegangen, indem wir anstelle der Einsendung die Veröffentlichung im Forum des virtuellen Lernraums gefordert haben. Bewertet wurde nur die fristgerechte Veröffentlichung – die Inhalte wurden zur Diskussion gestellt, im Forum und im wöchentlichen Chat. Dabei lernten die Studierenden einerseits, ihre guten Ideen

[1] BMBF Leitprojekt Virtuelle Fachhochschule: *http://www.oncampus.de*

preiszugeben und von denen anderer zu profitieren, und andererseits entwickelte sich teilweise im Forum eine sehr fruchtbare fachliche Diskussion, auf die der Mentor oder die Mentorin dann im Chat zurückgreifen konnte.

3 Fortgeschrittene synchrone Kommunikation

Was die synchrone Kommunikation betrifft, mussten wir feststellen, dass ein reiner Text-Chat für unsere Zwecke sehr unbefriedigend war. Die meisten Diskussionen im Rahmen des Software-Engineering-Unterrichts drehen sich um Fragen der Modellierung. Die Studierenden lernen schnell, dass es das „richtige" Modell für ein Problem nicht gibt, sondern nur je nach Intention mehr oder weniger gute Modelle. Verschiedene Alternativen asynchron zu diskutieren, ist ermüdend und schließt selten die ganze Lerngruppe ein.

Daher wird ein synchrones Kommunikationswerkzeug benötigt, das Gruppendiskussionen am Flipchart simuliert. Ein Whiteboard parallel zum Text-Chat, wie es die meisten Lernumgebungen anbieten, ist eine Hilfe, aber zumeist demotivierend schwerfällig und unflexibel. Die Autorinnen haben deshalb oft Zusatzmaterial auf ihre Homepage gestellt – teils auch während des Chats –, auf das sich der Chat beziehen konnte. Aber allein die textuelle Bezugnahme ist unangemessen mühsam.

Im Netucate-Lernraum[2] haben wir eine empfehlenswerte Alternative gefunden: Netucate bietet einen Audio-Chat (Video ist zuschaltbar) mit einem zusätzlichen Text-Chat für Einwürfe und Zwischenfragen. Zusätzlich zum Whiteboard gibt es einen „Shared Pointer", mit dem die Teilnehmer zeigen können, worauf sie sich beziehen, und vor allem das sog. „Application Sharing": Jeder Teilnehmer kann der Gruppe eine beliebige Applikation online zur Verfügung stellen. Modelldiskussionen verlaufen dann fast wie am Flipchart: Jemand präsentiert einen Modellvorschlag, der gemeinsam modifiziert wird. Darüber hinaus lassen sich Sitzungen aufzeichnen.

4 Persönliche Begegnungen

Technisch und didaktisch lässt sich also auch der „Soft Skill"-Aspekt des Software-Engineering-Unterrichts online mindestens so gut abdecken wie im Präsenzstudium. Dennoch empfehlen wir, wenigstens zwei Präsenztermine vorzusehen, da die persönliche Begegnung die Online-Kommunikation nachhaltig verbessert[3].

Literatur

[GöM 02] G. Görlitz und S. Müller: Didaktisches Design für eine Online-Programmierausbildung. GI Jahrestagung, Dortmund. 2002

[2] *http://www.netucate.com*

[3] Siehe auch *http://www.f4.fhtw-berlin.de/~weberwu/papers/SoftEngUnterrOnline.doc.*

Selbstgesteuertes Lernen und interdisziplinäre Projektarbeit – neue Wege der Kompetenzentwicklung im Software Engineering

Hans-Georg Hopf
Georg-Simon-Ohm-Fachhochschule Nürnberg
Kesslerplatz 12, 90489 Nürnberg
hans-georg.hopf@fh-nuernberg.de

1 Interdisziplinarität

Defizite in der Software-Engineering-Ausbildung werden häufig beklagt. Der im Jahr 2001 neu an der Georg-Simon-Ohm-Fachhochschule im Fachbereich „Elektrotechnik/ Feinwerktechnik/Informationstechnik“ eingeführte Ingenieurstudiengang Medientechnik bot die Möglichkeit, neu über Innovation in der Software-Engineering-Ausbildung auf dem Gebiet elektronische Medien nachzudenken. Der Studiengang ist eng verzahnt zum Studiengang Mediendesign konzipiert, der im Fachbereich Gestaltung angesiedelt ist. Das Konzept ist darauf ausgerichtet, die Identität der Disziplin Gestaltung bzw. Technik zu erhalten, jedoch die Sprachfähigkeit durch Vermittlung der Eigenarten der jeweils anderen Disziplin herzustellen. Auf diese Weise werden kompetente Fachleute ausgebildet, die sich im Team erfolgreich den Aufgaben der Praxis im Bereich elektronischer Medien stellen können.

2 Konzeption der Software-Engineering-Ausbildung

Wie an Fachhochschulen üblich ist ein praktisches Studiensemester in den Studienverlauf integriert. Nach der theoretisch orientierten Grundausbildung soll mit dem praktischen Studiensemester der industrielle Kontext erschlossen werden, bevor im weiteren Hauptstudium die in der Praxis gewonnenen Erfahrungen in die Ausbildung einfließen und reflektiert werden können. Dazu dient das stundenmäßig hervorgehobene Multimedia-Design-Projekt. Für das Multimedia-Design-Projekt sind über zwei Semester insgesamt 18 SWS vorgesehen. Die Projektarbeit vermittelt die erforderlichen Kenntnisse, Methoden und Fertigkeiten *anwendungsbezogen und problemorientiert.* Das Projekt integriert Lernen und Arbeiten nach dem Prinzip *Lernen „near-the-job“*. Dementsprechend anspruchsvoll sind die Projektthemen. Projekte werden von Medientechnik- und Mediendesign-Studenten gemeinsam durchgeführt. An einem Projekt nehmen ca. zehn

Personen teil. In der pädagogischen Arbeit mit den Projektteilnehmern kommen unter weitgehender Orientierung am Prinzip des selbstgesteuerten Lernens verschiedene Methoden zur Anwendung (Methodenmix). Selbstgesteuertes Lernen wird als einheitlicher Sammelbegriff für unterschiedliche konzeptionelle Ansätze verwendet. Die Gemeinsamkeit dieser Ansätze besteht darin, dass der lernende Mensch Initiator und Organisator seiner eigenen Lernprozesse ist. Orientierung am individuellen Lernbedarf und individualisiertes Lernen bieten Möglichkeiten für ein nachhaltiges und effektives Lernen ([CG 04]). E-Learning-Anteile können auch hier genutzt werden, um Wissen verfügbar zu machen bzw. Wissen zu vertiefen. Mit der Projektarbeit wird individuelles Lernen mit Lernen im Team kombiniert. Individuell wird jedes Team-Mitglied sich seiner übernommenen Aufgabe stellen und sich Wissen, Kenntnisse, Fähigkeiten und Fertigkeiten auf seinem Spezialgebiet aneignen (Selbstlernen). In besonderen Problemsituationen kann im Team diskutiert und auf den Erfahrungshintergrund der anderen Team-Mitglieder zurückgegriffen werden (Teamlernen). Lernen und Arbeiten verschmelzen zu einem Prozess. In dieser Kombination verläuft der Wissenserwerb am effektivsten und effizientesten ([See 98]). Der Dozent ist nicht mehr nur ausschließlich Wissensvermittler. Er hat zusätzlich die Rollen des Managers des Lernprozesses, des Beraters im Lernprozess und des beispielgebenden Senior Engineers (Lernteam Coaching) übernommen.

3 Erfahrungen aus der ersten Durchführung

Fünf Projektgruppen haben sich vom SS2004 bis zum WS2004/05 im Multimedia-Design-Projekt gebildet. In allen Projektgruppen wurden neben den Ingenieurstudenten Studenten auch aus Gestaltungsstudiengängen aufgenommen. Es wurden insgesamt fünf Entwicklungsprojekte durchgeführt. Aus zwei Projekten soll näher berichtet werden. Ein Projekt beschäftigte sich mit der Entwicklung eines Orientierungssystems für die Hochschule. Ein weiteres Projekt hatte die Aufgabe, ein Werkzeug zur Verwaltung von Web-Content der Hochschule zur Verfügung zu stellen. Beide Systeme werden ab Frühjahr 2005 an der Hochschule eingeführt. Wie erfolgreich ist dieses Lehrkonzept? Die Standish Group ([Sta 01]) gibt Erfolgsfaktoren für IT-Projekte an. Die (Lern-)Projekte sollen durch „Abklopfen“ dieser Erfolgsfaktoren bewertet werden. Es wird berichtet, inwieweit die zehn genannten Erfolgsfaktoren in der Projektarbeit thematisiert und bewusst gemacht wurden.

4 Literatur

[CG 04] I. Cavalieri, H. Geupel, Selbstgesteuertes Lernen eine – das Lernen aktivierende – Alternative zur Vorlesung, Seminar, Nürnberg, 9./10. Februar 2004.

[See 98] J. Seel, Georg-Simon-Ohm-Fachhochschule, Nürnberg, private Mitteilungen, 1998.

[Sta 01] Standish Group International inc., Extreme Chaos, 2001, *www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf*