Software Engineering im Unterricht der Hochschulen

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.

Martin Glinz hat an der Rheinisch-Westfälischen Technischen Hochschule Aachen Mathematik und Informatik studiert und 1983 dort in Informatik promoviert. Danach war er rund zehn Jahre bei BBC/ABB in Baden, wo er sich in verschiedenen Positionen mit Forschung, Entwicklung, Schulung und Beratung im Gebiet Software Engineering beschäftigt hat. Daneben war er Lehrbeauftragter an der ETH Zürich und an der Universität Basel. Seit 1993 ist er Professor für Informatik an der Universität Zürich. Sein Forschungsgebiet ist Requirements Engineering, insbesondere Methoden und Werkzeuge zur Modellierung von Anforderungen. Seine übrigen Interessensgebiete, die er auch in der Lehre vertritt, sind Software Engineering und Software-Qualitätsmanagement.

Horst Lichter • Martin Glinz (Hrsg.)

Software Engineering im Unterricht der Hochschulen

SEUH 7 – Zürich 2001

dpunkt.verlag

Prof. Dr. Horst Lichter
RWTH Aachen
Informatik III
Ahornstr. 55
D-52056 Aachen
E-Mail: lichter@informatik.rwth-aachen.de

Prof. Dr. Martin Glinz
Institut für Informatik
Universität Zürich
Winterthurer Str. 190
CH-8057 Zürich
E-Mail: glinz@ifi.unizh.ch

Satz: Horst Lichter, Aachen
Herstellung: Birgit Dinter
Umschlaggestaltung: Helmut Kraus, Düsseldorf
Druck und Bindung: Koninklijke Wöhrmann B.V., Zutphen, Niederlande

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

1. Auflage 2001

Ringstraße 19
69115 Heidelberg

5 4 3 2 1 0

Vorwort

Die Workshop-Reihe SEUH widmet sich seit 1992 speziell Themen im Bereich der Software Engineering Ausbildung. Der Leser mag sich vielleicht fragen, warum es für diesen Ausbildungsbereich eine eigene Workshop-Reihe gibt. Die Gründe dafür sind vielschichtig. Obwohl es einen breiten Konsens über die grundsätzlichen Inhalte im Bereich Software Engineering gibt, steht doch jeder Dozent und jede Dozentin vor dem Problem, Vorgehensweisen, Techniken und auch Werkzeuge auszuwählen, die im Rahmen der Lehrveranstaltungen eingesetzt werden sollen. Aber nicht nur die Inhalte stehen zur Diskussion, sondern auch die Art und Weise, wie die gewählten Inhalte am besten vermittelt werden können. Dabei muss fast immer berücksichtigt werden, dass Nutzen und Tragfähigkeit vieler Konzepte, Methoden und Techniken erst im Kontext der Entwicklung großer Systeme deutlich werden. Diese Situation muss für die Hochschulausbildung simuliert oder künstlich hergestellt werden. Ein weiterer wichtiger Aspekt bei der Software Engineering Ausbildung besteht darin, dass nicht nur Fakten- und Methodenwissen, sondern auch Erfahrungswissen vermittelt wird. Dies alles führt dazu, dass klassische Lehrformen, wie Vorlesungen, nur bedingt geeignet sind, um den Studierenden die Probleme, die bei der industriellen Softwareentwicklung typisch sind, aufzuzeigen und um Lösungsansätze vorzustellen und zu bewerten.

Das zentrale Ziel, das die SEUH-Reihe verfolgt, besteht deshalb darin, Ausbildungsformen und Ausbildungsinhalte, die sich als erfolgreich erwiesen haben, zu präsentieren, zu diskutieren und an Kollegen und Kolleginnen weiterzugeben. Der Erfahrungsaustausch steht dabei im Vordergrund.

Der vorliegende Band enthält die Beiträge zum siebten Workshop SEUH 2001, der zum ersten Mal außerhalb von Deutschland, in Zürich, stattfindet. Aus den eingereichten Beiträgen hat das Programmkomitee neun Beiträge ausgewählt und diese in drei Gruppen eingeteilt:

- *Ausbildung und Praxis*
  Die ersten beiden Beiträge beschreiben sehr unterschiedliche Ansätze, wie praxisrelevante Inhalte in die Ausbildung eingebracht werden können. Der dritte Beitrag stellt eine industrielle Ausbildungsinitiative im Bereich Informatik vor.

- *Werkzeuge und Sprachen*
  Im Rahmen der Software Engineering Ausbildung spielen Werkzeuge und Sprachen ein wichtige Rolle. Zwei der Beiträge berichten über Erfahrungen beim Einsatz von Werkzeugen. Der dritte Beitrag diskutiert Vor- und Nachteile der Sprache Java aus der Sicht der Lehre.

- *Software-Praktika*
  Praktika sind eine wichtige Lehrform. Diese können jedoch je nach angestrebtem Ziel sehr unterschiedlich organisiert und durchgeführt werden. Die Beiträge in dieser Kategorie berichten über konkrete Erfahrungen.

Zusätzlich enthält das Programm zwei eingeladene Beiträge: Stefan Roock präsentiert erste Erfahrungen mit eXtreme Programming in der Ausbildung. David Parnas stellt alternative Konzeptionen für die Gestaltung neuer Software Engineering Studiengänge vor.

Der Workshop wird abgerundet durch zwei Diskussionsrunden. Die erste hat die Themen „Job und Studium“ und „Soziale Kompetenzen“ zum Inhalt. Die zweite Diskussionsrunde ist dem Thema Bachelor- und Master-Studiengänge im Bereich Software Engineering gewidmet.

Der siebte Workshop SEUH wird gemeinsam von den GI-Fachgruppen 2.1.1 (Softwaretechnik) und 2.1.6 (Requirements Engineering), der Fachgruppe Software Engineering der SI und dem German Chapter der ACM veranstaltet. Gastgeber ist die Universität Zürich.

Die Mitglieder des ständigen Organisationskomitees der SEUH sind Jochen Ludewig (Universität Stuttgart), Günter Riedewald (Universität Rostock) und Andreas Spillner (Hochschule Bremen).

Im Programmkomitee haben mitgearbeitet: Horst Lichter (Vorsitzender, RWTH Aachen), Martin Glinz (Tagungsleiter, Universität Zürich), Björn Dreher (FH Wiesbaden), Anne-Lene Groll (Hochschule Rapperswil), Eckhard Jaus (German Chapter of the ACM), Jochen Ludewig (Universität Stuttgart), Kurt Schneider (Daimler-Chrysler, Ulm), Debora Weber-Wulff (TFH Berlin), Heinz Züllighoven (Universität Hamburg).

Wir bedanken uns bei allen, die bei der Organisation und bei der Durchführung der SEUH 2001 mitgewirkt haben. Insbesondere bedanken wir uns bei folgenden Unternehmen und Institutionen, welche die SEUH in Zürich mit einem finanziellen Beitrag unterstützt haben: ABB • CREALOGIX • dpunkt.verlag • GLANCE • HSR Hochschule Rapperswil • Rational Software • sd&m • UBS • Zühlke Engineering

Zürich und Aachen, im Dezember 2000

Martin Glinz
Horst Lichter

Inhalt

Eingeladene Vorträge

Ausbildung und Praxis

Werkzeuge und Sprachen

Software-Praktika

XP lehren und lernen

Martin Lippert, Stefan Roock, Henning Wolf, Heinz Züllighoven
Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik
Email: {lippert | roock | wolf | zuelligh@informatik.uni-hamburg.de}
http://swt-www.informatik.uni-hamburg.de

Zusammenfassung

Der Arbeitsbereich Softwaretechnik der Universität Hamburg führt einmal im Jahr ein Grundstudiumspraktikum (GSP) durch, in dem Java, Objektorientierung und softwaretechnische Grundprinzipien vermittelt werden. In diesem Jahr haben wir das Vorgehen im GSP stark modifiziert, um den Anteil softwaretechnischer Grundprinzipien zu erhöhen. Dafür haben wir das GSP als XP-Projekt aufgesetzt.

Dieser Artikel beschreibt das Vorgehen im GSP und die daraus gezogenen Lehren, die sich unserer Meinung nach auch auf die Weiterbildung und Vermittlung von XP im kommerziellen Umfeld übertragen lassen. Das Vorgehen wird jeweils aus Sicht der Betreuer der jeweiligen Gruppe beschrieben.

1 eXtreme Programming

Das Praktikum sollte neben praktischen Erfahrungen mit softwaretechnischen Konzepten und der Programmiersprache Java die Grundzüge des eXtreme Programming (XP) vermitteln. Wir skizzieren an dieser Stelle sehr knapp, welche Konzepte wir dabei berücksichtigt haben (mehr zu XP in [Beck00]).

- Story-Cards, Kundenrolle: In XP werden Karteikarten verwendet, um Kundenanforderungen auf Ebene einzelner Features aufzuschreiben. Die Karten werden dann vom Kunden priorisiert und von den Programmierern umgesetzt. Dabei wird nur das an Änderungen und Erweiterungen erledigt, was für dieses Feature gerade benötigt wird. (siehe auch KISS und YAGNI).
- Engineering-Cards: Neben Story-Cards werden zu erledigende Umstellungen (sog. Refactorings) auf Engineering-Cards vermerkt, so dass die Aufgaben nicht vergessen werden, aber nicht sofort erledigt werden müssen.
- Refactoring: Auf Grundlage der Engineering-Cards soll das bereits lauffähige Programmsystem durch Refactoring im softwaretechnischen Sinne gesäubert werden. Voraussetzung für einfaches Refactoring sind Testfälle, da so sehr leicht überprüft werden kann, ob das System seine Funktionalität noch erbringt. Didak-

tisch setzen wir das Refactoring auch ein, um dem weit verbreiteten „never change a running system“ entgegenzuwirken.

- Integrationsrechner / Continuous Integration: Die Entwickler sollen die verschiedenen Ergebnisse so häufig wie möglich zu einem lauffähigen System integrieren, um größere Integrationsprobleme am Ende von mehreren Tagen Programmierarbeit zu vermeiden und möglichst die ganze Zeit auf dem aktuellen Stand des Systems zu arbeiten. Dies haben wir mehrfach täglich mit den 8 parallel arbeitenden Gruppen gemacht. Dabei wurde ein im Arbeitsbereich entwickelter Integrationsrechner benutzt.
- Testklassen: Für jede Klasse soll es auf der Ebene von Unit-Tests eine Testklasse geben. Als Test-Framework haben wir JUnit (siehe [junit]) verwendet.
- „Keep it simple stupid“ (KISS) und „You aren’t gonna need it“ (YAGNI) sind zwei XP-Konzepte, die verhindern sollen, dass die praktisch entwickelten Programme unnötig kompliziert werden. Dies passiert häufig, weil Entwickler gerne Änderungen antizipieren, die potenziell auftreten können. Es soll aber genau das realisiert werden, was auf der Story-Card steht und nicht mehr.
- Pair-Programming: Jeweils zwei Entwickler programmieren gleichzeitig an einem Rechner. Derjenige, der gerade die Tastatur hat, erläutert ständig was er gerade warum macht und der zweite beurteilt und kommentiert dies im Sinne eines ständigen Reviews. Dabei sollte häufig (mehrmals pro Stunde) die Tastatur gewechselt werden. Wir wollten den Studenten erfahrbar machen, dass dieses Entwickeln in Paaren große Vorteile bringt.

2 Das Grundstudiumspraktikum

Ein GSP ist in Hamburg verbindlicher Teil des Grundstudiums und wird im wesentlichen von Studierenden im dritten Semester besucht. Diese haben Lehrveranstaltungen über Konzepte der funktionalen und logischen sowie der objektorientierten Programmierung mit Java besucht.

Die Erfahrungen zeigen, dass die Studierenden nach den ersten beiden Semestern einen sehr unterschiedlichen Kenntnisstand zu Java und Objektorientierung mitbringen. Daher kombinieren wir in unserem GSP softwaretechnische Lernziele mit einer Vertiefung in den Bereichen Java und Objektorientierung.

Zum GSP wurden 32 Studierende zugelassen, die in zwei Gruppe à 16 Studierende aufgeteilt wurden. Die eine Gruppe wurde von einem wissenschaftlichen Mitarbeiter die andere von zwei Mitarbeitern im Wechsel betreut. Das GSP dauerte insgesamt fünf Tage (Montag bis Freitag, jeweils 9 bis 17 Uhr) und entspricht damit einer Lehrveranstaltung mit 4 Semesterwochenstunden.

In den vorangegangenen Jahren war jeder Tag des GSP in einen Plenumsteil und einen Übungsteil aufgeteilt. Morgens wurde ein Vortrag (1-2 Stunden) zu einem Thema (z.B. Swing) gehalten. Anschließend haben die Studierenden feste Paare ge-

bildet, die dann festgelegte Aufgaben übernommen haben. Wenn die Aufgaben für den jeweiligen Tag erledigt waren, konnten die Studierenden gehen.

In diesem Semester erhielt jede Gruppe (à 16 Studierende) eine gemeinsame Projektaufgabe: Die Studierenden sollten ein Multi-User-Dungeon programmieren. Dazu musste ein Labyrinth programmiert werden, in dem sich der Spieler bewegen und Geldsäcke einsammeln kann. Ausbaustufen betreffen vom Computer gesteuerte Spieler, Multi-User-Fähigkeit etc.

Da im GSP die XP-Praktiken und die Arbeit im Team vermittelt werden sollten, mussten die Studierenden die Paare möglichst häufig wechseln (mind. täglich). Plenumsvorträge aus dem traditionellen GSP wurden ersetzt durch das Angebot der Betreuer, situativ zu beliebigen Themen Vorträge zu halten. Wir machten deutlich, dass Vorträge nur dann gehalten werden, wenn dies von den Studierenden selbst gefordert wird. Dazu waren die Betreuer ständig mit Notebooks und Beamern ausgerüstet.

Das GSP wurde von Martin Lippert, Stefan Roock und Henning Wolf betreut. Alle drei haben langjährige Praxis in Entwicklungsprojekten mit Java und XP (siehe auch [Lip00]). Außerdem haben sie reichhaltige Vortragserfahrung in Lehrveranstaltungen sowie kommerziellen Schulungen und Beratungen.

3 Bericht über das Grundstudiumspraktikum

Im folgenden fassen wir die Aufzeichnungen der Betreuer zusammen, um einen Eindruck von der Vorgehensweise und den anstehenden Problemen zu geben.

Montag

Wir begannen den Montag mit einem Kurzvortrag (ca. 1 Stunde) zum Thema XP. Während des Vortrages waren alle 32 Studenten anwesend. Nach dem Vortrag wurden die Studenten auf zwei Gruppen zu je 16 Leuten aufgeteilt. Dabei haben wir jeweils jeden zweiten auf der Anmeldungsliste einer Gruppe zugeordnet, um möglichst häufig „bewährte“ Studier-Paare in unterschiedliche Gruppen zu platzieren. Es hat sich gezeigt, dass der Vortrag und die anschließende Betreuung ausreichte, um die XP-Techniken deutlich zu machen.

Die Betreuer agierten in den beiden Gruppen unterschiedlich -– ein Betreuer schreibt die Anforderungen an das Spiel auf Story-Cards und machte klar, dass niemand sonst entscheidet, welche Features in das Programm sollen und auf welche verzichtet wird. Der andere Betreuer gab nur allgemeine Vorstellungen über das Spiel vor und überließ die Detaillierung und den weiteren Prozess der Gruppendiskussion.

In der ersten Gruppen gingen die Studierenden recht zielgerichtet ans Werk. Sie erarbeiteten an der Tafel gemeinsam einen Entwurf. Der Entwurf hatte am Ende 5 Klassen, die im Kern der Anwendung auch bis zum Ende der Praktikums erhalten blieben: Dungeon, Raum, Geldsack, Spieler, Controller (für Ein-/Ausgabe).

Der Betreuer der ersten Gruppe schrieb systematisch Anforderungen und Aufgaben auf Story Cards und Engineering-Card. Die Paare orientierten sich an diesen Karten und fingen an, arbeitsteilig entsprechende Klasse plus Testklasse zu schreiben.

Da der Betreuer der zweiten Gruppe keine Story Cards geschrieben hatte, nahm die Gruppendiskussion über die Systemanforderungen und den ersten Entwurf viel Zeit in Anspruch und verlief recht chaotisch. Dennoch kristallisierten sich Teilaufgaben heraus, die dann arbeitsteilig von den einzelnen Paaren bearbeitet wurden.

Die erste Gruppe arbeitete nach kurzer Zeit mit einem Integrationsrechner, während die zweite Gruppe noch nicht dessen Sinn sah.

Die Feedback-Runden am Ende des ersten Tags brachten erwartungsgemäß unterschiedliche Ergebnisse. Übereinstimmend positiv wurde das Pair-Programming bewertet. Auch fiel den Studierenden auf, dass sie sehr viel miteinander reden mussten, um die Schnittstellen der Klassen abzustimmen. Die zweite Gruppe war aber mit der Selbstorganisation und dem Diskussionsprozess unzufrieden. Sie beschlossen, sich am nächsten Tag auf die erste lauffähige Version des Spiels zu konzentrieren und disziplinierter vorzugehen.

Dienstag

Die erste Gruppe entwickelte auf der Basis der existierenden Story- und Engineering-Cards weiter. Gegen Mittag lief die erste spielfähige Version des Programms auf dem Integrationsrechner. Die Integration der Klassen war reibungslos, wozu die Testklassen ganz wesentlich beigetragen haben.

Die zweite Gruppe begann mit einer Gruppendiskussion, die jetzt deutlich zielgerichteter lief. Alle waren sehr interessiert, rasch eine lauffähige Version zu entwickeln. Als Organisationsprinzip wurden täglich zwei Gruppendiskussion vereinbart, die von da ab recht konzentriert und strukturiert waren. Allerdings gelang es den Gruppen am zweiten Tag nicht, eine lauffähige Version zu integrieren. Dazu trugen auch die fehlenden Testklassen bei.

Das Pair-Programming mit wechselnden Paaren war in der ersten Gruppen fest etabliert, während sich in der zweiten Gruppe oft drei oder vier Entwickler um einen Rechner scharten.

In beiden Gruppen bestand die Tendenz, Technologie auf Vorrat in die Entwürfe einzubauen. Die Betreuer erläuterten noch einmal das KISS- und das YAGNI-Prinzip. Allerdings wurden die Rollen der Betreuer als „Kunden“ und als „OO-Experten“ nur wenig in Anspruch genommen. Einzelne Technologievorträge wurden aber gezielt angefordert.

In den Feedback-Runden wurde klar, dass die Studierenden der ersten Gruppe den Überblick über das Projekt verloren hatten. Jedes Paar wusste zwar, was es als nächstes zu tun hatte. Vielen war aber unklar, was die anderen Paare taten. Die Studierenden beschlossen daher, im weiteren immer mit einem aktuellen Entwurf an der Tafel zu arbeiten. Ähnliche Probleme hatte die zweite Gruppe nicht, da sie sich über

die Gruppendiskussion und die Tafel bereits ausreichend koordiniert hatten. Die zweite Gruppe stellte fest, dass sie am zweiten Tag deutlich produktiver gearbeitet hatte.

Mittwoch

Gruppe 1 entwickelte ein gemeinsames Tafelbild mit den einzelnen Komponenten, so dass danach jeder wieder einen Überblick über das Gesamtsystem hatte. Dazu wurde an der Tafel eine Liste mit allen Rechnern gemacht, in die jeweils eingetragen wurde, welches Paar gerade an welcher Aufgabe arbeitete. Diese Liste wurde im weiteren als sehr sinnvoll bewertet.

In der zweiten Gruppe lief die koordinierende Gruppendiskussion besser, da jetzt ein Moderator bestimmt wurde. Es gelang erst gegen Mittag, eine ersten lauffähige Version des Spiels zu integrieren. Wieder erwiesen sich die fehlenden Testklassen als Problem.

Vormittags wurde vom Betreuer der Gruppe 1 ein Code-Review an Kernklassen des Systems durchgeführt. Es zeigte sich, dass die Studierenden nicht selbst in der Lage waren, den Code anderer zu bewerten. Sie konnten jedoch Nutzen aus den Hinweisen des Betreuers ziehen. Sie haben dann selbst entschieden, welche Refactorings sich vom Aufwand her im GSP lohnen und welche nicht.

Beide Gruppen hatten Schwierigkeiten, die Anforderungen an den Multi-User-Betrieb des Spiels zu realisieren. Die entsprechenden Vorträge über RMI wurden nur von wenigen Studierenden verstanden.

In der Feedback-Runde zeigte sich die Gruppe 1 weitgehend zufrieden. Die zweite Gruppe war mit dem Projektfortschritt unzufrieden und führte dies auf die noch nicht reibungsfreien Diskussionen und Koordinationsaufwände zurück. Außerdem bemängelten sie eigene technische Wissenslücken.

Donnerstag

Gruppe 1 arbeitete parallel an zwei Versionen, der Einzelplatzversion mit weiteren Ausbaustufen und der Mehrbenutzerversion, die von einer RMI-Task-Force erstellt wurde. Die zweite Gruppe konzentrierte sich im wesentlichen auf die Einzelplatzverion.

In beiden Gruppen ist die Arbeit mit den Story- und Engineering-Cards mittlerweile etabliert. Auch die zweite Gruppe hatte ihre Bedeutung erkannt.

Der Integrationsrechner wurde nun ebenfalls von beiden Gruppen benutzt. Die zweite Gruppe merkte immer schmerzlicher, dass Testklassen fehlten. Da deshalb nicht regelmäßig integriert wurde, verlor eine Paar seine Änderungen, was zu einem emotional heftigen Lerneffekt führte.

Die Studierenden der ersten Gruppe erklärten während der Feedback-Runde, dass sie selbst überrascht seien wie weit sie schon gekommen waren. Die zweite Gruppe betonte den hohen Lerneffekt und die Tatsache, dass einige softwaretechnische Prin-

zipien und die Techniken des XP durch eigene Erfahrung als sinnvoll erkannt wurden.

Freitag

Am Vormittag konnte die erste Gruppe alle noch offenen Programmieraufgaben abarbeiten. Während die Präsentation von einigen vorbereitet wurde, feilten andere an zusätzlichen Features oder versuchten, die Multi-User-Version doch noch ans Laufen zu bringen. Insgesamt war die Stimmung eher euphorisch.

Die zweite Gruppe war vormittags hektisch mit Abschlussarbeiten beschäftigt. Die Koordination zwischen diesen Arbeiten und der Vorbereitung der Präsentation war schlecht, was der insgesamt guten Stimmung keinen Abbruch tat.

Die gemeinsame Präsentation der beiden Gruppenergebnisse war für bei Gruppen der Höhepunkt der Praktikumswoche. Präsentiert wurden die drei Teile: das laufende Programm, die Architektur, der Entwicklungsprozess.

Bei der Präsentation zeigte sich, dass beide Gruppen das Entwicklungsziel erreicht haben: Sie können ein lauffähiges System vorführen. Das Ergebnis der ersten Gruppe war dabei etwas ausgereifter.

Bei der Präsentation wurde deutlich, dass eine starke Konkurrenzsituation zwischen den beiden Gruppen entstanden war. Die Diskussion über Ergebnis und Prozess wurde zeitweise sehr emotional zwischen den Studierenden geführt. Für die Betreuer wurde deutlich, dass in beiden Gruppen eine gemeinsame zielorientierte Projektkultur entstanden ist.

4 Bewertung der XP-Elements

Testklassen

Nur Gruppe 1 hat ausgiebig Gebrauch von Testklassen gemacht. Dies hat sich sehr positiv auf die Integrationen und die wenigen Refactorings ausgewirkt.

Continuous Integration

Der explizite Integrationsrechners hat sich positiv ausgewirkt, da die Studierenden einen expliziten Schnitt zwischen „Entwickeln“ und „Integrieren“ machen mussten.

KISS + YAGNI

Auf diese beiden Prinzipien musste immer wieder hingewiesen werden. Insbesondere das Wissen um noch kommende Anforderungen verleitete die Studierenden dazu, „Technologie auf Vorrat“ zu konstruieren.

Aufgrund der mangelnden Erfahrung gab es zwischen Studierenden und Betreuern nicht immer sofort einen Konsens darüber, was „einfach“ bedeutet. So hatten Teile der Studierenden in beiden Gruppen zunächst die Idee, Räume, Geldsäcke und

Spieler in assoziativen Arrays zu repräsentieren. Hier haben die Betreuer deutlich darauf hingewiesen, dass diese Lösungen „zu einfach“ seinen. Am Ende des GSP hatte sich aber in Ansätzen eine gemeinsame Vorstellung darüber etabliert, was „einfach“ bedeutet.

Story-Cards + Engineering-Cards

Die explizite und sehr strikt gespielte Kundenrolle in Gruppe 1 hat dieser Gruppe einen klaren Rahmen und eine deutliche Zielvorstellung vermittelt.

Gruppe 2 hat ihre Story-Cards im wesentlichen selbst geschrieben und viel Zeit mit der Diskussion der zu schreibenden Story-Cards verbraucht. Außerdem war in Gruppe 2 die Tendenz erkennbar, Anforderungen zu definieren, welche sie Studierenden gerne umsetzen würden.

Die Koordination über Story- und Engineering-Cards lief in beiden Gruppen sehr gut.

Pair-Programming

Das Pair-Programming mit wechselnden Paaren hat in beiden Gruppen mit wenigen Ausnahmen sehr gut funktioniert. Es hat sich deutlich gezeigt, dass Pair-Programming ein mächtiges Mittel ist, um Wissen zwischen den Studierenden zu transferieren. So konnten sie sich häufig selber helfen.

Refactoring

In Gruppe 1 war das Code-Review die Ausgangsbasis für die durchgeführten Refactorings. Refactoring hat im GSP jedoch eher eine untergeordnete Rolle gespielt. Dies lag nach Einschätzung der Betreuer daran, dass das Projekt insgesamt eher klein war. Daher gab es keine wirkliche Notwendigkeit zum Refactoring. Außerdem fehlte den Studierenden die Programmiererfahrung, um wirklich effektiv Refactoring umsetzen zu können.

5 Gesamtbewertung

Die Auswertung einer Umfrage erbrachte, dass die Studierenden das GSP außerordentlich positiv bewerteten. Es wurde vielfach sogar gefordert, dass das GSP auf mindestens zwei Wochen auszudehnen.

Es zeigte sich, dass relativ große Projekte (mit 16 Personen) funktionieren können, wenn konsequent Pair-Programming mit wechselnden Paaren und Collective Code Ownership angewendet wird. Nach Einschätzung der Betreuer wären vier Gruppen à 8 Leuten wahrscheinlich effektiver gewesen, weil die Koordination viel einfacher gewesen wäre. Allerdings hatten die Studierenden in den großen Gruppen die Gelegenheit, eine Projektgröße im Grenzbereich mit den zugehörigen Problemen zu erleben.

Nach Einschätzung der Betreuer haben die Studierenden weniger Technologien (Swing, Threads, Exceptions, RMI) gelernt, als dies im traditionellen Praktikum der Fall war. Allerdings haben sie deutlich mehr in den Bereichen XP, Softwaretechnik, Architektur, Projektkultur und Entwicklungsprozess gelernt. Die Studierenden hatten letztlich aber einen größeren Lerneffekt, der sie in die Lage versetzen sollte, sich die noch fehlenden Technologien selbst anzueignen.

Das potenziell größte Problem dieser Form der Lehrveranstaltung dürfte die Qualifikation der Betreuer sein. Diese müssen sich nicht nur sehr gut mit Java und der verwendeten Programmierumgebung auskennen, sondern selbst einen breiten Erfahrungshintergrund zu XP und Entwicklungsprojekten haben. Sie müssen im Notfall sogar Projektleitungskompetenzen mitbringen, um ein kippendes Projekt wieder zu stabilisieren. Ein gescheitertes Projekt bietet sicher viel Stoff zum Lernen. Allerdings wäre dafür dann keine Zeit mehr. Daher sollte das Projekt erfolgreich sein.

Dazu kommt, dass die Betreuer in der Lage sein müssen, im Prinzip beliebige Vorträge situativ zu halten, ohne sich vorher noch darauf vorbereiten zu müssen. Die funktioniert nur, wenn die Betreuer die potenziell in Frage kommenden Vorträge bereits mehrfach gehalten haben.

Danksagung

Wir möchten an dieser Stelle den Studierenden danken, die durch ihre engagierte Teilnahme an dem Praktikum ganz wesentlich zu diesem Text beigetragen haben.

Literatur

[Bec00] K. Beck: eXtreme Programming explained: Embrace Change. Reading, Massachusetts, Addison-Wesley. 2000.

[Fow99] M. Fowler: Refactoring: Improving the Design of Existing Code. Reading, Massachusetts, Addison-Wesley, 1999.

[Gam96] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Entwurf-smuster – Elemente wiederverwendbarer objektorientierter Software. Übersetzung von D. Riehle, Bonn: Addison Wesley, 1996.

[Lip00] M. Lippert, S. Roock, H. Wolf, H. Züllighoven: JWAM and XP - Using XP for framework development. Proceedings of the XP2000 conference. Cagliari, Sardinia, Italy, 2000.

[Mer00] Peter Merel: eXtreme Hour. http://c2.com/cgi/wiki?ExtremeHour. 2000.

University Programmes in Software Development

David Lorge Parnas, P.Eng.
NSERC/Bell Industrial Research Chair in Software Engineering
Director of the Software Engineering Programme
Department of Computing and Software
Faculty of Engineering
McMaster University, Hamilton, Ontario, Canada - L8S 4K1

Abstract

Programmes in "Software Engineering" have become a source of contention in many universities. Computer Science departments, many of which have used the phrase "Software Engineering" to describe individual courses for decades, claim software engineering as part of Computer Science. Some Engineering faculties claim "Software Engineering" as a new member of a growing family of engineering disciplines. Other organisations seem to view "Software Engineering" as a Management discipline. These different viewpoints result in programmes that differ widelyin their style and content.

Any organisation thinking of starting a "Software Engineering Programme" at the University or Technical University level must begin by recognising that the term "Software Engineering" has been used so carelessly that it is almost meaningless. To design a meaningful educational programme, one must understand the alternative meanings of the phrase and choose a consistent viewpoint.

The talk will describe the main alternatives. Each is based on a different interpretation and vision of what the graduates will do. Some of them require students capable of doing well in a traditional university education. Others do not constitute a university education but the graduates should work under the supervision of university educated people. All can be useful if they are not confused with each other.

Praxisnähe durch Reverse Engineering-Projekte: Erfahrungen und Verallgemeinerungen

Klaus Bothe, Ulrich Sacklowski
Institut für Informatik
Humboldt-Universität zu Berlin

Zusammenfassung

Mit Reverse Engineering (RE)-Projekten kann im Hochschulbereich eine größere Praxisnähe erreicht werden als mit Projekten, die eine Neuentwicklung von Software zum Ziel haben. Über entsprechende Erfahrungen und Verallgemeinerungen eines für das RE typischen Anwendungsfalles berichtet dieser Artikel.

1 Einführung

Obwohl die Bearbeitung existierender Software-Systeme in der Praxis weitaus häufiger ist als die Neuentwicklung [4], spielen Veranstaltungen im Hochschulbereich, in denen existierende Software Gegenstand der Untersuchungen sind, eher eine untergeordnete Rolle.

Eine Reihe von Arbeiten beschreibt Projekte, die i. w. auf die Wartung existierender Software gerichtet sind [1, 3, 9, 11, 12]. Es geht in ihnen um die Modifikation und Erweiterung existierender Funktionalität [3, 9, 11, 12], aber auch um die Fehlersuche [9] und die Portierung [1]. Neben Projekten werden auch Vorlesungen angeboten, in denen existierende Software mit Reengineering-Techniken untersucht wird [6, 7, 10], die allerdings oft mit Akzeptanzproblemen konfrontiert sind.

Im vorliegenden Artikel beschreiben wir die Erfahrungen mit einem für universitäre Verhältnisse großen Reverse Engineering-Projekt eines realen nichtkommerziellen Kunden, das von den Studierenden als Ergänzung bzw. Umsetzung einer vorab gelaufenen Vorlesung 'Software Engineering' gewählt werden konnte. Dabei handelt es sich um ein Anwendungsgebiet außerhalb der Informatik, mit dem die in der Praxis immer wieder benötigte Einarbeitung in neues Fachwissen [4, 8] eingeübt werden kann.

Es stellten sich interessante Möglichkeiten dieser Projektform heraus, mit der Forderungen an die Ausbildung, Praxisnähe insbesondere auch über Projektarbeit im Curriculum umzusetzen [3, 4, 5, 8, 11], besonders nahegekommen werden kann.

Das Projekt begann im Wintersemester 1998/99 mit neun Studierenden; bei zwischenzeitlich wechselnden studentischen Mitgliedern waren 15 Teilnehmer im 4. Projektsemester (Sommersemester 2000) einbezogen.

Unsere grundlegenden Erfahrungen lassen sich in drei Hauptthesen zusammenfassen:

- **These 1**: RE-Projekte haben Vorteile gegenüber Projekten, die eine Neuentwicklung von Software (FE: Forward Engineering) zum Ziel haben: Studenten sind unmittelbar mit großer SW konfrontiert, während FE-Projekte erst langsam über längere Zeiträume wachsen können und deshalb im Hochschulbereich oftmals nur prototypisch entstehen [3, 9].
- **These 2**: SW großer Komplexität in einem RE-Projekt ist auch unter universitären Bedingungen beherrschbar. Wichtig sind
  - eine geeignete, auf einen längeren Zeitraum orientierte Projektorganisation und
  - eine problemorientierte Selektion von Teilaufgaben zur separaten Bearbeitung.
- **These 3**: Reverse Engineering wird in der Industrie in steigendem Maße der Neuentwicklung von SW vorgezogen, einfach deshalb, weil es gilt, den Entwicklungsaufwand der Alt-Software gewinnbringend in die Weiterentwicklung einzubeziehen [13]. Im wesentlichen aus demselben Grund können universitäre RE-Projekte weitaus ökonomischer ablaufen als FE-Projekte: Die zeitlichen Ressourcen in der Ausbildung sind noch knapper bemessen, so daß komplexe Neuentwicklungen kaum möglich sind [9]. Komplexe Alt-Software hingegen beinhaltet bereits von anderen geleisteten Arbeitsaufwand, von dem universitäre Projekte entlastet sind.

Im nächsten Kapitel werden wir zunächst auf verwandte Projekte im Hochschulbereich eingehen. Daran anschließend werden wir eine kurze Einführung in unser Projekt geben (Kapitel 3), den Ausgangszustand der von uns bearbeiteten Alt-Software charakterisieren (Kapitel 4) und die Schritte unserer RE-Strategie beschreiben (Kapitel 5). Schließlich werden verschiedene Seiten der von uns gemachten Erfahrungen zusammengefaßt: Probleme unseres RE-Projektmanagements in Kapitel 6 und Verallgemeinerungen in Kapitel 7.

2 Verwandte Projekte

In den letzten Jahren wurden in Forschungsberichten vereinzelt Projekte aus dem Hochschulbereich beschrieben, die nicht die Neuentwicklung von Software, sondern die Bearbeitung existierender Systeme zum Gegenstand hatten [1, 3, 9, 11, 12]:

- Pascal-S-Compiler [9]: Die Aufgabe bestand in einer Erweiterung der Funktionalität eines Compilers von 4000 LOC sowie der vorgegebenen Validation-Suite innerhalb eines Semesters. Das Projekt wurde über viele Jahre hinweg wiederverwendet.

- Krankenhaussystem [11]: In einer begrenzten Zeit (1 Woche) mußten die Studenten Vorschläge für eine veränderte Funktionalität machen sowie Subsysteme erkennen. Ausgangspunkt war ein real genutztes System.
- Telefonvermittlungssystem [12]: Neue Telefondienste waren neu einzubauen, die durch Lehrkräfte vorgegeben wurden. Ein lauffähiger Prototyp war das Ziel. Besonderes Gewicht wurde auf die Planung des Projekts gelegt.
- Mehrere Projekte an der Carnegie Mellon University [3]: Bearbeitung von studentischen Projekten, die vorab in Vorläuferprojekten entstanden sind, d. h. für die bereits Entwicklerdokumente vorlagen. Es gab keinen realen Auftraggeber (sondern: Lehrkräfte ersetzen Kunden).
- Mehrere Projekte an der University of Western Ontario [1]: UNIX-Software war entsprechend den Vorgaben von Lehrkräften zu modifizieren bzw. Software realer Kunden zu warten. Im ersten Fall entstanden Motivationsprobleme unter den Studenten, in der zweiten Aufgabenklasse wurden - obwohl anders geplant - lediglich Prototypen entwickelt.

All diese Projekte waren vornehmlich Wartungsprojekte, in denen es um die Modifikation existierender Funktionalität [9, 11], um die Erweiterung des Funktionsumfangs ([3, 12], die Fehlersuche [9] und die Portierung [1] ging. Hierin unterscheidet sich das von uns realisierte Projekt, das in erster Linie RE-Aufgaben zu bewältigen hatte. Unter RE verstehen wir in übereinstimmung mit gängigen Definitionen den Prozess der Analyse eines Systems zur Identifikation von Systemkomponenten und zur Erzeugung einer Repräsentation des Systems in einer anderen Form oder auf einem höheren Abstraktionsniveau [2].

Trotz dieser grundlegenden Orientierung auf das RE spielten auch bei uns Wartungsaufgaben eine Rolle, die sich aus Nutzerwünschen ableiteten. Weitere Probleme der o. g. Projekte sind die folgenden:

- Die Anforderungen an die Projektarbeit stammen vom Lehrpersonal und nicht von realen Kunden, wodurch leicht der Eindruck einer 'künstlichen' Aufgabenstellung bei den Studierenden entsteht [3, 9, 11, 12].
- Mit dem Fehlen eines realen Kunden kann der Umgang mit ihnen auch nicht eingeübt werden [1, 3, 9, 11, 12].
- In allen Fällen entstanden lediglich Prototypen mit mehr oder weniger offensichtlichen Defekten [1, 3, 9, 11, 12].
- Oftmals wurden nur ausgewählte Phasen bzw. Dokumente eines Softwareprojekts einbezogen [9, 11, 12].

Mit den hier angegebenen Defiziten verbunden ist oftmals auch eine verringerte Motivation der Studierenden, die wir mit unserem Projekt eines realen, jedoch nichtkommerziellen Auftraggebers überwinden konnten und in dem es um das Ziel eines produktionsreifen Systems ging, wobei alle Phasen und die entsprechenden Dokumente einbezogen wurden.

3 Entstehungsgeschichte: der Anfang

Alles begann mit einer Email:

> *"... ich wende mich an Sie mit einer Bitte um einen Rat.*
> *Wir sind eine experimentelle Arbeitsgruppe am Institut fuer Physik der Humboldt-Universität zu Berlin. In unserem Labor laufen rund 10 Rechner, die Röntgendiffraktometer steuern. Vor einigen Jahren hat ein Ingenieur ein Programm (C++) für Windows 3.1 geschrieben, mit dem diese Geraete gesteuert werden.*
> *Leider ist ... das Programm insgesamt nicht sehr uebersichtlich. Meine Frage ist nun folgende:*
> *Koennten Sie uns beraten, wie wir zu einer sauberen Software-Basis fuer die Zukunft kommen, die wir mit ertraeglichem Aufwand pflegen koennen?"*

In unserer Antwort erbaten wir die Klärung einer Reihe von Fragen zur Ausgangslage der Wartungs- bzw. RE-Situation. Es stellte sich heraus, daß der Programmentwickler nicht mehr anwesend war und neben den Programmquellen keinerlei weitere SW-Dokumente (Nutzerdokumentation, Pflichtenheft, Architektur usw.) vorhanden waren - also eine typische 'Reverse Engineering-Situation'. Noch typischer, was den Realitätsgehalt der Software anbelangt, war der vorliegende Umfang der Quellen: 27 000 LOC, was die Frage der Sinnfälligkeit der Aufgabe in einem universitären Ausbildungsprojekt aufwarf.

Nach ca. 2 Jahren Bearbeitungszeit kann diese Frage weitestgehend positiv beantwortet werden: Direktkontakte mit den Physikern führten zur Entscheidung, ein gemeinsames Projekt anzugehen, von dem beide Seiten profitieren können. Ein Projektseminar mit dem Thema 'Software-Sanierung' wurde im WS 1998/99 angeboten, das mit 9 Teilnehmern startete. Der Einstieg wurde mit der Vorführung des Systems vor Ort und mit Vorträgen von Physikern zur Fachthematik erreicht.

4 Charakterisierung des Ausgangszustands der Software und fachliche Projektziele

Jedes RE-Projekt sollte mit einer Bestandsaufnahme zum Ausgangszustand der vorgefundenen Software beginnen. Hierzu haben wir in den ersten Projektseminaren die Aufgabe vergeben, den vorgefundenen Programmcode mittels *Code Review* zu bewerten. Durch diese erste *statische Analyse* konnte eine Reihe von Defiziten ermittelt werden (Abbildung 1).

1. schlechte Softwarearchitektur (keine saubere Dekomposition der 27 KLOC Programmcode: z. B. Mischung von funktionalen Komponenten mit solchen der Oberfläche)
2. Mischung zwischen imperativem C-Code und objektorientiertem C++-Code,
3. z. T. unstrukturierter interner Programmcode (Layout)

4. toter Code: angedachte Erweiterungen
5. Instabilität des technischen Systems in ausgewählten Situationen (z. T. nicht reproduzierbar)
6. Verletzung von Prinzipien der Softwareergonomie in den Dialogfenstern
7. keinerlei Programmdokumentation und kaum Kommentierung, so dass alle relevanten Informationen aus den Programmquellen abgeleitet werden muessen
8. keine sonstige Dokumentation: Nutzerdokumentation, Pflichtenheft, Architektur usw. fehlen vollständig
9. veraltete Umgebung: Windows 3.11, Borland C++ 4.1

Abb. 1: Defizite unseres vorgefundenen Programm-Systems

Als besonders nachteilig für die studentische Projektarbeit stellte sich heraus, daß keine saubere Dekomposition der 27 KLOC Quellcode vorlag. Damit war zunächst eine Arbeitsteilung aufgrund fester Schnittstellen zwischen den Komponenten nicht möglich.

Es gab aber auch positive Seiten: konsistente Namenswahl, z. T. saubere imperative Schnittstellen (Funktionen), z. T. oo Klassenbildung mit sauberen Vererbungsbeziehungen, grundlegende Stabilität des Systems, einheitlicher Programmierstil.

Als fachliche Ziele unserer Projektarbeit ergaben sich nun folgende drei Aufgabengruppen:

- RE: Defizite des Projektzustandes beheben (fehlende Dokumente entwickeln: Pflichtenheft, Nutzerdokumentation, Design, Kommentierung, Testdokumente u. a.).
- Wartung (Nutzerwünsche erfüllen): Stabilität, Fehlerbeseitigung, Erweiterung (neue technische Geräte ansteuern), Portierung (modernes Betriebssystem, neue Compilerversion).
- allgemeine Ausbildungsziele: Team-Arbeit, Projekt als Ganzes, fachlich anderes Problemgebiet außerhalb der Informatik, Nutzerkontakte mit Nicht-Informatikern, Tooleinsatz ...

5 Schritte unserer RE-Strategie

RE bedeutet Analyse eines Systems zur Identifikation von Systemkomponenten und zur Erzeugung einer Repräsentation des Systems in einer anderen Form oder auf einem höheren Abstraktionsniveau [2]. Zu einer zentralen Aufgabe wird damit das Verstehen des Systems auf all seinen Abstraktionsebenen (Code, Architektur, Funktionalität).

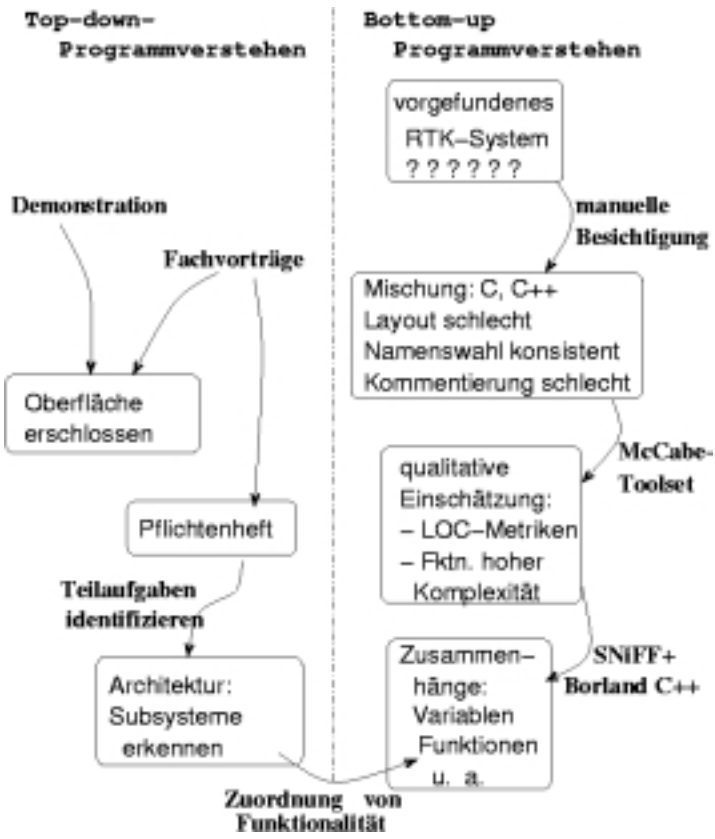

Abb. 2: Schritte zum Erkenntnisgewinn über das System

Im Mittelpunkt unserer Projektarbeit stand dabei i. w. eine Kombination von Tool-Einsatz zur Aufbereitung der Quellen und Arbeit an der Funktionalität des Systems. Abbildung 2 faßt die ausgeführten Schritte zusammen, wobei in Fettdruck die Projektaktivitäten, in den Kästen die Ergebnisse unterschieden werden. Links sind Maßnahmen im Rahmen eines Top-down-, rechts im Rahmen eines Bottom-up-Programmverstehens gruppiert. Im einzelnen sollen folgende Aspekte hervorgehoben werden:

- Das Projekt begann mit einer Demonstration des Systems vor Ort und mit Fachvorträgen von Physikern. Ohne Fachwissen hätte es kein Verständnis der Programmquellen geben können. Dasselbe trifft auch auf Software zu, die Disziplinen der Informatik zugeordnet werden können. Beispielsweise kann Compilersoftware nicht verstanden werden, ohne Kenntnisse zu Techniken des Compilerbaus zu haben (z. B. Parsing-Algorithmen, Symboltabellenaufbau usw.).
- Das Verständnis des Programms beginnt mit seiner Anwendung, d. h. aus Nutzersicht. Die Erschließung der Oberfläche und die Entwicklung eines Pflichten-

hefts waren auch in einem RE-Projekt zu Beginn grundlegend, um die Bedeutung des Programmcodes zu erschließen.

- Das Pflichtenheft wurde schrittweise, über mehrere Semester verteilt, entwickelt, wobei eine aufgabenorientierte Dekomposition zugrunde gelegt wurde. Einzelne Programmfunktionen entsprechen dabei Anwendungsfällen (Use Cases), für die jeweils Teile des Pflichtenheftes entwickelt wurden.
- Eine manuelle statische Analyse der Quellen ist nützlich (Bewertung der Qualität des Codes). Fortschritte beim Programmverstehen allein dadurch sind jedoch nicht zu erwarten.
- Code-Analyse-Tools (SNiFF+, McCabe) bringen Erkenntnisse zu statischen Programmeigenschaften, insbesondere quantitative Analysen, die jedoch das Verständnis des Programms ebenfalls kaum weiterbringen. Mittels des McCabe-Tools werden beispielsweise Funktionen ermittelt, die eine hohe zyklomatische Komplexität besitzen und somit Problemfälle darstellen.
- Architecture-Recovery-Tools: Hier lagen insbesondere Probleme der Anwendbarkeit auf unsere Alt-Software vor. So akzeptierte das Bauhaus-Tool [7] nur die C-, nicht aber die C++-Syntax.

6 Projektgruppenarbeit (RE-Projektmanagement)

Projektgruppenarbeit bzw. Projektmanagement unter universitären Bedingungen muß sich zwangsläufig von der im industriellen Bereich unterscheiden. Eine Vielzahl von Nebenbedingungen ist im Ausbildungsbereich wirksam: Belastung durch andere Lehrveranstaltungen, zeitliche Beschränkungen im Semestertakt, Motivierungsprobleme, Nebentätigkeit u. a. Diesen Einflüssen muß Rechnung getragen werden, will man komplexe Systeme nicht nur prototypisch bearbeiten, sondern sie als Ganzes beherrschen und durch Bearbeitung zu einer Produktionsversion machen. Folgende Aspekte waren hierbei für unser Projekt wesentlich:

- **Komplexe RE-Projekte: offenes Ende, beliebige Teilnehmerzahl ...**
  Komplexe Projekte an Hochschulen sind kaum in ein oder zwei Semestern abzuschließen [3, 9]. Aus diesem Grunde wurden keine Endtermine für das Gesamtprojekt gesetzt. Vielmehr wurden die in Abschnitt 5 beschriebenen RE-Schritte ohne Zeitdruck umgesetzt. Die Teilnehmerzahlen in den einzelnen Semestern schwankten dabei zwischen 9 und 15. Projekte ohne feste Endtermine sind allerdings nur möglich, hat man es - wie in unserem Fall - mit einem nichtkommerziellen Kunden zu tun.
- **Projektmodell: Abfolge von Einsteigerseminaren und Vertiefungsseminaren**
  Zu Beginn des Projekts nicht geplant, kristallisierte sich die in Abbildung 3 angegebene Folge von Seminaren heraus.

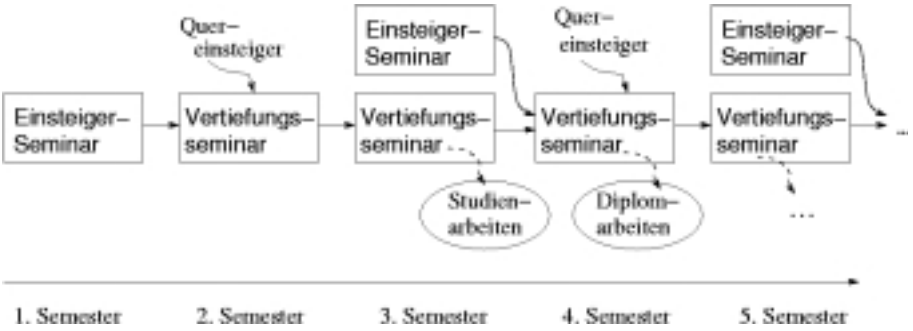

Abb. 3: Seminarabfolge

In den Einsteigerseminaren wurden die Teilnehmer an das Projekt durch gemeinsame Aufgaben herangeführt, während in den Vertiefungsseminaren Einzelaufgaben bearbeitet wurden. Daraus ergaben sich dann Themen für Studien- und Diplomarbeiten.

- **Nachnutzung des Projekts [3, 9]**
  Lehrkräfte (der Informatik) müssen sich in das Fremdprojekt einarbeiten (Fachwissen, Software-Architektur ...) und ein entsprechendes Projektmanagement aufbauen. Dieser hohe Aufwand sollte sich durch entsprechende Wiederverwendung in mehreren Semestern rentieren. Läuft ein Projekt - wie bei uns - über mehrere Semester, so zahlt sich dieser Aufwand aus. Das Projekt wird dabei von Semester zu Semester auf höherem Niveau wiederverwendet.

- **Alt-Mitglieder vs. Neueinsteiger im RE-Team**
  Zu lange Mitgliedschaft im RE-Team (> 3 Semester) führt zu Ermüdungserscheinungen. Neueinsteiger bringen hier frische Motivation und neue Ideen, wobei jeder Neueinstieg auf höherem Projektniveau erfolgt.

- **Rollenorientierte vs. problembezogene Aufgabendekomposition**
  Projekte, die dem rollenorientierten Ansatz folgen, zerlegen die Aufgaben der Teilnehmer vorrangig nach Phasen bzw. Rollen: Es gibt Anforderungsingenieure, SW-Designer, Codierer, Tester [1, 9]. Demgegenüber werden Projekte, die problembezogen dekomponiert sind, so organisiert, daß eine Gruppe von Entwicklern eine Teilaufgabe in allen Phasen abdeckt.
  Wir folgen vorrangig der problembezogenen Aufgabendekomposition (Abbildung 4): Hier lernen die Teilnehmer Dokumente mehrerer Phasen kennen, und die Aufgabe wird so vielfältiger. Außerdem sind Mehrsemesterprojekte über diesen Ansatz besser in den Griff zu bekommen. In einigen Fällen (Qualitätsmanagment: Test, Metriken) wurden die Aufgaben jedoch rollenorientiert vergeben.

- **Allgemeine Prinzipien des Projektmanagements**
  treffen (selbstverständlich) auch auf RE-Projekte zu. Hier wollen wir auf regelmäßig durchgeführte Projektgruppentreffen, dabei angefertigte Projektprotokolle zur Erinnerung an Beschlüsse und als Informationsquelle der Neueinsteiger (was

ist bisher gelaufen?) mit wechselnden Protokollanten, das im Laufe der Zeit aufgebaute zentrale Repository (alle Dokumente) sowie auf Richtlinien zur Struktur der Dokumente und zum Programmcode (Layout, Namenswahl, Kommentierung) verweisen. Ganz entscheidend ist ein Versionsmanagementsystem, in unserem Fall cvs.

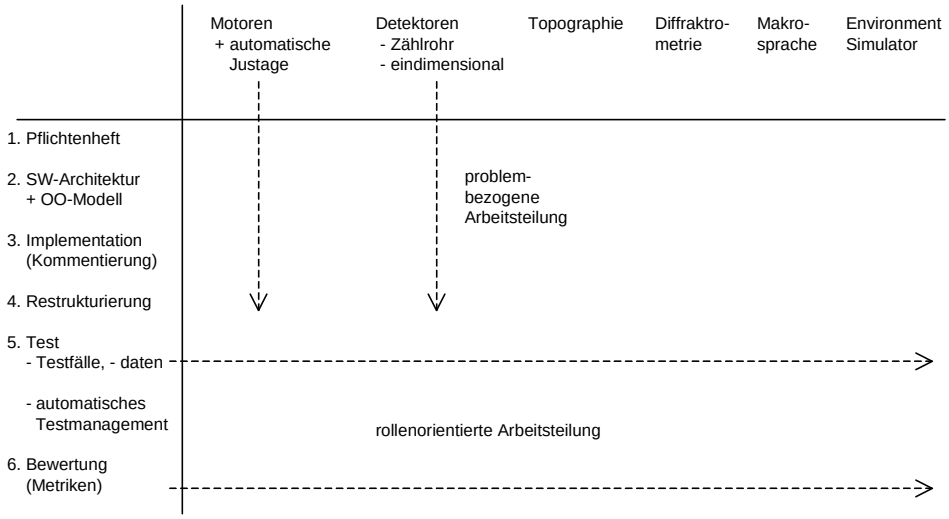

Abb. 4: Rollenorientierte- und problembezogene Aufgabenverteilung

7 Allgemeine Erkenntnisse aus unserem RE-Projekt

Nach der Darstellung des RE-Projektmanagements im 6. Kapitel sollen nun Verallgemeinerungen vorgestellt werden, die sich aus unserer Projektarbeit ergeben:

- **Ausgiebige Reverse Engineering-Phase**
  Eine ausgiebige Reverse Engineering-Phase erschließt die vorliegende Software (Analyse, Gewinnung von Information auf höherer Abstraktionsebene: SW-Architektur, Anforderungsdefinition). Erst danach können Modifikationen im Sinne eines Reengineering (z. B. Restructuring, Re-Architecturing) bzw. Wartungsaufgaben beginnen: Beginnt man zu zeitig mit Änderungen am System, ohne es hinreichend verstanden zu haben, können sich leicht Fehler einschleichen.

- **(Scheinbarer) Interessenkonflikt: Anwender - akademisches RE-Team**
  Um das Interesse des Anwenders (Kooperationsbereitschaft) zu erhalten, kann die Reverse Engineering-Phase nicht erst vollständig abgeschlossen werden. Ausgewählte Ergebnisse bzw. Erfolge sind wichtig. Oberstes Kundenziel sind Modifikationen, Beseitigung von Fehlern, Erweiterungen - also Wartungsaufgaben.

Demgegenüber ist die Analysephase (Reverse Engineering) für den Kunden ohne jede Bedeutung - es tut sich aus seiner Sicht nichts (Was hat er davon, wenn das Programm besser kommentiert vorliegt oder die SW-Architektur erschlossen wird ...).
In unserem Projekt erstreckte sich die Reverse Engineering-Phase über drei Semester, bevor Modifikationen an den Programmquellen vorgenommen wurden, und wird in bestimmten Programmteilen derzeit weiter fortgeführt.

- **Reverse Engineering vs. Forward Engineering (Neuentwicklung)**
  Studenten schlagen von Zeit zu Zeit immer wieder eine Neuentwicklung des Systems vor - es werde alles besser: Struktur, Namenswahl ... Es kann jedoch davon ausgegangen werden, daß eine Neuentwicklung unseres komplexen Systems unter universitären Bedingungen gescheitert wäre (Kapazitäts- und Zeitprobleme).
  Ein RE-Projekt kann selektiv Schwerpunkte setzen und gezielt Ergebnisse vorweisen - das Gesamtsystem bleibt funktionsfähig.

- **RE-Projekte bringen sofortige Konfrontation mit komplexer SW**
  Während Neuentwicklungen komplexer Systeme einige Zeit benötigen, konnten wir die Projektteilnehmer sofort mit 27 000 LOC konfrontieren: Die Notwendigkeit der Dekomposition, der arbeitsteiligen Teamarbeit wurde sofort offensichtlich. Die Überschaubarkeit der Software in all ihren Teilen durch einzelne ist kaum möglich.

- **Realer Auftraggeber (Kunde) bringt starke Motivation**
  Während durch Lehrkräfte definierte SW-Projekte von Studierenden meist als akademische (Spielzeug-)Projekte angesehen werden [1, 3], trägt ein realer Kunde zu einer starken Motivation und damit zum Projekterfolg bei ("mein Produkt" wird gebraucht).

- **Nicht-informatisches Anwendungsgebiet motivierend und herausfordernd**
  Einmal nicht mit der Entwicklung von Systemsoftware zu tun zu haben, sondern ein Anwendungsgebiet kennenzulernen, stellte ebenfalls eine Motivationsquelle dar, brachte aber auch Verständigungsprobleme mit den Fachleuten. So mußten die Physiker in Diskussionen gebremst werden, um die Informatiker nicht mit ihrer Fachsprache zu erschlagen.

7 Zusammenfassung

Während Projekte im Hochschulbereich auch heute noch meist FE-Projekte sind, bieten gerade Projekte, die auf existierender Software aufbauen eher praxisorientierte Bedingungen. Sie sind ökonomischer, da sie den Aufwand, der in existierender Software steckt (Implementations- und Testaufwand), nachnutzen. Damit ist die Bewältigung komplexer Software auch unter Hochschulbedingungen möglich.

In diesem Bereich gibt es an den Hochschulen vorrangig Erfahrungen bei Wartungsprojekten, während das von uns realisierte Projet schwerpunktmäßig auf den Prozeß des RE orientiert. Damit wird eine umfassendere Sicht auf die vorgefundene Software eingenommen als bei Wartungsprojekten, die eher selektiv auf die angestrebte Wartungsaufgabe gerichtet sind.

Literatur

1. J. Andrews, H. Lutfiyya: Experience Report: A Software Maintenace Project Course, 13th Conference on Software Engineering Education and Training, Austin, Texas 2000
2. R. Arnold: Software Reengineering, IEEE Computer Science Press 1993
3. B. Bruegge: From Toy Systems to Real Software Development: Improvements in Software Engineering Education, SEUH'94
4. GI 99: Empfehlungen der GI zur Stärkung der Anwendungsorientierung in Diplom-Studiengängen der Informatik an Universitäten, Informatik Spektrum, 6 (Dez.) 1999
5. J. Harrison: Enhancing Software Development Project Courses via Industry Participation, 10th Conference on Software Engineering Education and Training, Virginia Beach, 1997
6. K. Hildebrand: Re-Engineering und Re-Use von Software: Konzeption einer Veranstaltung und erste Erfahrungen in der Lehre, SEUH' 94
7. R. Koschke: Vorlesungen zum Thema Reengineering, Workshop Reengineering, Bad Honnef 2000
8. W. McMillan, S. R. Aprabhakaran: What Leading Practitioners Say Should Be Emphasized in Students' Software Engineering Projects, 12th Conference on Software Engineering Education and Training, New Orleans 1999
9. K. Pierce: Teaching Software Engineering Principles using Maintenance-Based Projects, 10th Conference on Software Engineering Education and Training, Virginia Beach, 1997
10. L. Schmitz: Reengineering als Lehrgegenstand: Zweck und Gestaltung, SEUH'94
11. K. Sikkel, T. Spil, R. van de Weg: Replacing a Hospital Information System: an Example of a Real-World Case Study, 12th Conference on Software Engineering Education and Training, New Orleans 1999
12. C. Wohlin: Meeting the Challenge of Large-Scale Software Development in an Educational Environment, 10th Conference on Software Engineering Education and Training, Virginia Beach, 1997
13. E. Yourdon: Die westliche Programmierkunst am Scheideweg, Hanser 1993

Qualifizierte Software-Projektmanager durch simulationsbasierte Ausbildung

Patricia Mandl-Striegnitz
Abteilung Software Engineering, Universität Stuttgart
mandlpa@informatik.uni-stuttgart.de

Zusammenfassung

Die Qualifikation des Projektleiters entscheidet häufig über den Erfolg oder Misserfolg eines Softwareprojekts. Wie aber können zukünftige Projektleiter auf ihre Rolle vorbereitet werden? Das Vermitteln von Fachkenntnissen alleine reicht nicht aus. Um ein Softwareprojekt erfolgreich leiten zu können, benötigen die Projektleiter vor allem auch praktische Erfahrung.

In diesem Artikel wird ein Ansatz zur Ausbildung von Projektleitern vorgestellt, durch den nicht nur das erforderliche Fachwissen vermittelt wird, sondern auch praktische Projektmanagement-Erfahrungen gewonnen werden können. Erste Erfahrungen mit diesem Ansatz werden präsentiert.

1 Einführung

Softwareprojekte erfolgreich durchzuführen, ist eine schwierige Aufgabe, die zu bewältigen auch mehr als 30 Jahre nach „Entdeckung" des Software Engineerings häufig misslingt. Auch in den neunziger Jahren wird der überwiegende Teil aller Projekte nicht innerhalb der gegebenen Zeit, innerhalb des vorgegebenen Kostenrahmens und mit Ergebnissen der geforderten Qualität abgeschlossen ([4]; [12]). Untersuchungen zeigen, dass das Scheitern eines Projekts häufig weniger auf technologische Probleme als vielmehr auf Schwächen des Projektmanagements zurückzuführen ist (z.B. [7]). Ein erfolgreiches Projekt zeichnet sich nach Jones [5] immer durch eine effektive Projektplanung und Kostenschätzung, eine effektive Fortschritts- und Qualitätskontrolle und kompetente Projektleiter aus.

Die Kompetenz des Projektleiters ist dabei der entscheidende Erfolgsfaktor: Er muss nicht nur in der Lage sein, die vielfältigen Aufgaben des Projektmanagements sicher, schnell und zuverlässig auszuführen, sondern in jeder Situation die richtigen Entscheidungen treffen können. Um diesen Anforderungen gerecht zu werden, benötigt der Projektleiter eine sehr gute Qualifikation: Er muss nicht nur über Fachwissen verfügen, sondern auch die notwendige Erfahrung besitzen, um auf überraschende Ereignisse sinnvoll zu reagieren und wichtige Projektmanagement-Aufgaben umsetzen zu können.

Viele Studienpläne, aber auch Ausbildungsprogramme in der Industrie, sehen jedoch häufig nicht einmal eine theoretische Ausbildung im Software-Projekt-management vor. Die Ausbildung bietet noch weniger die Möglichkeit, Gelerntes anzuwenden, Problemsituationen zu erfahren oder verschiedene Strategien auszuprobieren, um aus Fehlentscheidungen zu lernen und die (positiven und negativen) Konsequenzen des eigenen Handelns direkt zu erleben.

In diesem Artikel wird ein Ausbildungsansatz vorgestellt, durch den angehende Projektleiter nicht nur das erforderliche Fachwissen erwerben, sondern vor allem die Möglichkeit haben, praktische Projektmanagement-Erfahrungen zu sammeln.

2 Software-Projektmanagement: Stand der Praxis

Ein Blick in die Praxis des Software-Projektmanagements zeigt nicht nur erhebliche Defizite, sondern macht auch die Bedeutung des Projektmanagements als Schlüsselfaktor für den Erfolg oder Misserfolg eines Projekts deutlich.

Die folgenden Aussagen spiegeln den Stand der Praxis im Projektmanagement in einem industriellen Großunternehmen. Sie sind das Ergebnis einer Studie, die Ende der neunziger Jahre durchgeführt wurde [7].

Die meisten der befragten Projektleiter investierten lediglich 5 bis maximal 35% ihrer Arbeitszeit in das Projektmanagement. Aufgrund ihrer technischen Fähigkeiten zum Projektleiter befördert kannten sie weder ihre Aufgaben als Projektleiter, noch waren sie sich der Bedeutung ihrer Rolle bewusst. Folglich konnten sie diese Rolle nicht ausfüllen und bevorzugten inhaltliche Projektaufgaben. Der Projektverlauf zeigt die Konsequenzen: z.B. eingeschränkte Fortschrittskontrolle, mangelhaftes Aktualisieren des Projektplans und in der Folge hohe Abweichungen von Termin- und Kostenplänen. In keinem der untersuchten Projekte wurden systematisch wichtige Projektkenndaten wie beispielsweise Aufwand je Phase oder Tätigkeit erhoben. Folglich fehlten quantitative Informationen für eine zuverlässige Planung, Fortschrittskontrolle und Projektsteuerung. Außer auf fehlende quantitative Daten war die schlechte Qualität der Projektpläne und Statusberichte auch auf den fehlenden Einsatz von Kostenschätzverfahren oder Verfahren zum Risikomanagement zurückzuführen. Und letztlich war den meisten Projektleitern die Bedeutung wichtiger Elemente des Software Engineerings wie beispielsweise Qualitätssicherung und Dokumentation nicht bewusst. Sie haben diese Tätigkeiten weder bei der Planung entsprechend berücksichtigt, noch während des Projektverlaufs darauf geachtet, dass diese Tätigkeiten in erforderlichem Maße ausgeführt werden. Dass diese Schwächen im Projektmanagement keine Ausnahme darstellen, bestätigen weitere Untersuchungen in diesem Bereich (z.B. [3]; [13]).

3 Die Qualität des Software-Projektmanagements: Eine Frage der Ausbildung

Die Defizite im Projektmanagement sind im Wesentlichen auf die schlechte Qualifikation der Projektleiter zurückzuführen. Während technische Projektmitarbeiter selbstverständlich Schulungen erhalten, hat keiner der befragten Projektleiter jemals an einer Projektmanagement-Schulung teilgenommen [7]. Folglich kannten sie weder ihre Aufgaben als Projektleiter, noch Methoden und Techniken, um diese Aufgaben zu bewältigen. Vor allem aber fehlte ihnen die notwendige Erfahrung, um Problemsituationen beurteilen und zuverlässig darauf reagieren zu können.

Aufgrund der Bedeutung des Projektmanagements ist es deshalb entscheidend, zukünftige Projektleiter gezielt auf die tägliche Praxis vorzubereiten.

3.1 Konventionelle Ausbildung im Projektmanagement

Wir können Software-Engineering- und Projektmanagement-Kenntnisse (z.B. Techniken zur Planung und Fortschrittskontrolle und Maßnahmen der Qualitätssicherung) in konventionellen Lehrveranstaltungen wie beispielsweise Seminaren, Vorlesungen und Übungen vermitteln. Diese Veranstaltungsformen sind jedoch unzureichend, um Studierende und Praktiker auf ihre Rolle als Projektleiter vorzubereiten. Viele Aussagen erscheinen ohne Praxiserfahrung unmotiviert und trivial. Beispielsweise ist es schwierig, in einer Vorlesung die Bedeutung einer konsequenten Qualitätssicherung zu vermitteln, wenn die Studierenden nie erfahren haben, welche Konsequenzen es hat, Zwischenresultate ungeprüft zu verwenden. Außerdem bieten diese Veranstaltungsformen keine Möglichkeit, das Gelernte auch anzuwenden, um die über das theoretische Wissen hinaus notwendige praktische Erfahrung zu erwerben. Selbst wenn ein unerfahrener Projektleiter weiß, wie ein Statusbericht zu erstellen ist, wird er ohne Erfahrung Schwierigkeiten haben, den aktuellen Projektstatus sicher einzuschätzen, notwendige Informationen schnell zu erheben und zu beurteilen, inwieweit Abweichungen vom Projektplan vorliegen.

Auch typische Schwierigkeiten und Probleme, die in realen Softwareprojekten mit unschöner Regelmäßigkeit immer wieder auftreten, lassen sich nicht anhand von Lehrbüchern vermitteln. Projektleiter müssen möglichst viele Problemsituationen selbst erfahren und Übung im Lösen dieser Probleme gewinnen. Schwierigkeiten treten in einem Projekt nicht einzeln und unabhängig voneinander auf, sondern in jeder Situation spielen viele verschiedene Einflussfaktoren zusammen und beeinflussen sich in ihren Auswirkungen auf den Projektverlauf gegenseitig. Der Erfolg eines Projektleiters ist vor allem davon abhängig, ob es ihm gelingt, eine bestimmte (Problem-)Situation zu analysieren und ob er entscheiden kann, welches Vorgehen und welche Lösungsstrategie er in dieser Situation wählen sollte. Um diese Erfahrung zu gewinnen, müssten Projektleiter bereits in der Ausbildung viele (auch sehr verschiedene) Softwareprojekte zu Übungszwecken durchführen können. Dabei

müssen sie sowohl die Chance haben, Fehlentscheidungen zu treffen, als auch positive Erfahrungen sammeln können.

Typische studentische Projekte sind kaum geeignet, da Umfang und Teilnehmerzahl zu gering sind und der Schutz der Hochschule es verhindert, dass Effekte realer Projekte in der erforderlichen Komplexität auftreten. (Mehrere) umfangreiche, realistische Softwareprojekte bereits in der Ausbildung durchzuführen, scheitert am erforderlichen Aufwand für Studierende und Betreuer. Darüber hinaus besteht die Hauptaufgabe in diesen Projekten in der Lösung technischer Probleme, so dass die Studierenden kaum echte Management-Erfahrung sammeln können.

3.2 Software Engineering Simulation durch animierte Modelle

Grundidee und Zielsetzung

Eine mögliche Lösung des Problems kennen wir aus der Ausbildung von Piloten: die Simulation. Ebenso wie ein Projektleiter kann auch ein Pilot nicht alle möglichen schwierigen Situationen real erproben, um zu lernen, wie er sie am besten meistert. Ein Pilot muss die Konsequenzen möglicher Fehler verstehen, *bevor* er zum ersten Mal selbständig ein Flugzeug fliegen darf. Hier kommen die Flugsimulatoren ins Spiel. Entscheidend ist, dass die Simulationen ausreichend realistisch sind, so dass die Erfahrungen mit gleichem Ergebnis in der Realität anwendet werden können.

Aufgrund der Parallelen verfolgt die Abteilung Software Engineering an der Universität Stuttgart in ihrem SESAM-Projekt einen vergleichbaren Ansatz zur Ausbildung von Projektleitern. Ziel war es, ein System zur interaktiven Simulation von Softwareprojekten bereitzustellen. Die Idee dabei ist, dass man mit einem solchen System alle denkbaren Effekte und kritischen Situationen, die ein Projektleiter in der Praxis erleben kann, simulieren und damit für die Spieler erfahrbar machen kann [6]. Sie haben – ebenso wie ein Pilot am Flugsimulator – die Chance, schwere Fehler zu machen, ohne Schaden anzurichten. Der vom Spieler gewählte Verlauf bestimmt die Ergebnisse des simulierten Projekts. Die Spieler können somit verschiedene Strategien ausprobieren, bis es ihnen letztlich gelingt, ihr Projekt zum Erfolg zu führen.

Die Simulation basiert auf Modellen von Softwareprojekten. Der Modellbauer entscheidet, welche Objekte eines Softwareprojekts und mögliche Beziehungen zwischen diesen Objekten jeweils mit ihren relevanten Eigenschaften modelliert werden. Es gibt nur eine Ausnahme: Die Rolle des Projektleiters wird von einem realen Spieler, dem auszubildenden Projektleiter, übernommen. Dazu stellt ihm das Modell Aktionen zur Verfügung, mit denen er den Verlauf des simulierten Projekts interaktiv steuern und kontrollieren kann. Welche Änderungen des Projektzustands der Spieler durch seine Aktionen jeweils erwirkt, hat der Modellbauer in der Dynamik des Modells festgelegt.

Ein SESAM-Modell – das QS-Modell

Drappa [2] hat ein erstes umfassendes Simulationsmodell realisiert. Da Softwareprojekte sehr verschieden sind, ist es nicht möglich, ein universelles Modell zu entwickeln, das jeden beliebigen Projekttyp und Entwicklungsansatz unterstützt. Das Modell ist daher auf die Entwicklung von Software im Rahmen eines kleinen bis mittleren Auftragsprojekts beschränkt. Es unterstützt aktivitäten-orientierte Prozessmodelle, keine evolutionären oder zyklischen Vorgehensmodelle. Es konzentriert sich auf die Effekte von Qualitätssicherungsmaßnahmen (und wird daher im folgenden als QS-Modell bezeichnet).

An einem einzigen Simulationsmodell können nicht alle Aufgaben des Projektmanagements geschult werden. Durch das QS-Modell können im Wesentlichen die folgenden Projektmanagement-Funktionen trainiert werden (vgl. [2]): Projektplanung, Stellenbesetzung und Projektführung. Die Planungstätigkeiten müssen vom Spieler außerhalb des Systems durchgeführt werden. Mit Hilfe der vom Modell angebotenen Aktionen leiten die Spieler das simulierte Projekt entsprechend ihrer Planung. Dazu müssen sie die notwendigen Projektmanagement-Aufgaben durchführen, beispielsweise Mitarbeiter einstellen, ihnen (entsprechend ihrer Qualifikation und Erfahrung) Aufgaben zuweisen, den Projektfortschritt kontrollieren und ggf. korrigierend eingreifen. Die Management-Funktionen Organisation und Personalführung spielen in diesem Modell keine Rolle.

Erste Erfahrungen zum Einsatz von SESAM in der Ausbildung

Im Wintersemester 1992/93 wurde der Einsatz von SESAM in der Projektmanagement-Ausbildung untersucht [1]. Ziel war es festzustellen, ob es mit Hilfe des SESAM-Ansatzes wirklich gelingen konnte, Projektmanagement-Erfahrungen zu vermitteln. Die Studierenden erhielten die Aufgabe, in voneinander unabhängigen Zweierteams ein simuliertes Softwareprojekt in der Rolle des Projektleiters zu planen und durchzuführen. Für alle Teilnehmer galten dabei die gleichen Rahmenbedingungen und Zielvorgaben. Nach Abschluss der Simulationen erhielten die Teams von den Betreuern Feedback zu ihrer Vorgehensweise und den erzielten Resultaten.

Diese Untersuchung ergab folgendes: Das Ziel, Management-Erfahrungen zu vermitteln, war erreicht worden. Die Studierenden hatten ihre Fehler vielfach schon während des Projekts erkannt, spätestens jedoch bei der Analyse des Simulationslaufs zusammen mit den Betreuern verstanden [10]. Außerdem hatten sie sich derart intensiv mit dem simulierten Projekt und ihrer Rolle als Projektleiter identifiziert, dass sie während der Simulation typische Stimmungsschwankungen durchmachten. Der Lerneffekt wurde bei dieser Untersuchung jedoch nicht explizit gemessen. Zudem bezieht sich die Untersuchung auf ein sehr einfaches Simulationsmodell [11]. Da das SESAM-System außerdem zu diesem Zeitpunkt noch nicht implementiert war, fand die Simulation von Hand statt. Insofern können die erzielten Ergebnisse nicht ohne weiteres auf eine rechnergestützte Simulation mit einem realistischen, komplexen Simulationsmodell übertragen werden.

Nachdem das SESAM-System technisch realisiert ist und mit dem QS-Modell ein erstes umfassendes Simulationsmodell vorliegt, wurde erneut überprüft, ob sich unsere Annahmen über den Nutzen von SESAM in der Projektmanagement-Ausbildung bestätigen lassen. Zu diesem Zweck führte Notter [9] zwei Experimente zur „Evaluierung des SESAM-Systems in Bezug auf seine Eignung als Lehrmittel" durch. Dabei überprüft er die Hypothese „Ein Spieler lernt durch die Benutzung des SESAM-Simulators". Die Versuchspersonen hatten die Aufgabe, mehrere Softwareprojekte zu planen und anschließend am Simulator durchzuführen. Die Randbedingungen und Zielvorgaben variierten dabei von Projekt zu Projekt. Zur Messung des Lernerfolgs wurden die Projektpläne und die Ergebnisse der einzelnen Simulationsläufe miteinander verglichen und Fragebögen ausgewertet.

Die Auswertungsergebnisse bestätigen die aufgestellte Hypothese nicht. Ein Lernerfolg alleine durch die Simulation der Projekte kann nicht nachgewiesen werden. Der Vergleich der Simulationsläufe lässt erkennen, dass die Versuchspersonen nicht selbständig lernen konnten. Sie haben die simulierten Effekte offensichtlich nicht durchschaut. Folglich wurden Fehler in Folgeprojekten wiederholt, weil sie nicht als Ursache für den Misserfolg erkannt wurden [9]. Ein Simulator wie SESAM reicht offensichtlich nicht aus, um den angestrebten Lernerfolg zu erzielen.

4 Ein simulationsbasiertes Ausbildungskonzept

Wenn wir also weder mit konventionellen Ausbildungsformen noch durch den Einsatz von SESAM als eigenständiges Lehrmittel den gewünschten Lernerfolg erzielen können, benötigen wir ein Ausbildungskonzept, das beide Ausbildungsformen miteinander kombiniert. Bevor dieses Ausbildungskonzept festgelegt werden kann, müssen die Ausbildungsziele definiert werden. Aus diesen Zielen ergeben sich die einzelnen Schulungskomponenten.

4.1 Ausbildungsziele

Durch die Projektmanagement-Ausbildung sollen die angehenden Projektleiter sowohl das notwendige Fachwissen erwerben als auch reale Projektmanagement-Erfahrungen sammeln und auf diese Weise praktische Fähigkeiten als Projektleiter erlangen. Im einzelnen bedeutet das:

1. Die Studierenden sollen wichtige Aufgaben des Projektmanagements und Techniken zur Bewältigung dieser Aufgaben erlernen. Sie sollen lernen, diese Techniken anzuwenden, und erfahren, welche Konsequenzen es hat, Projektmanagement-Aufgaben durchzuführen oder zu vernachlässigen.
2. Die Studierenden sollen möglichst viele Schwierigkeiten und kritische Situationen erfahren und lernen, diese zu vermeiden oder zu meistern. Beispielsweise sollte jeder Projektleiter erkennen,

- wie schwierig es ist, die Größe und Komplexität eines Softwareprojekts abzuschätzen, um eine zuverlässige Projektplanung zu ermöglichen oder
- wie schwierig es ist, den Status zu erheben und Abweichungen anzugeben.

3. Die Studierenden sollen die komplexen Zusammenhänge in einem Softwareprojekt erfahren. Sie sollen lernen, in jeder Situation alle wichtigen Einflussfaktoren und ihre Auswirkungen auf den Projektverlauf zu berücksichtigen, um sinnvolle Entscheidungen zu treffen. Sie sollen erkennen, wie sich ihr Vorgehen auf die Ergebnisse auswirkt.
4. Die Studierenden sollen sich ausschließlich auf ihre Rolle als Projektleiter konzentrieren können. Nur so können sie sich auf ihre tägliche Praxis vorbereiten und erkennen, dass der Projektleiter über Erfolg oder Misserfolg eines Projekts entscheidet.
5. Und letztlich soll die Ausbildung vor allem Spaß machen. Je höher die Motivation der Schulungsteilnehmer ist, desto eher lässt sich ein Lernerfolg erzielen. Sie sollen durch die Schulung angeregt werden, sich auch darüber hinaus mit dem Thema Projektmanagement auseinanderzusetzen.

4.2 Schulungskomponenten

Um diese Ausbildungsziele erreichen zu können, müssen die Projektsimulationen eingebettet werden zwischen Komponenten konventioneller Lehre und die Aufbereitung der Simulationsergebnisse. Dem Konzept liegt die Annahme zugrunde, dass die Studierenden über Grundkenntnisse im Bereich Software Engineering und Projektmanagement verfügen. Ziel ist es dann, sie durch die Projektmanagement-Schulung auf ihre Rolle als Projektleiter vorzubereiten. Folgende Tabelle zeigt, welche Schulungskomponenten das Ausbildungskonzept vorsieht und gibt jeweils den Aufwand aus Sicht des Spielers und die zeitliche Abfolge der Komponenten an.

Schulungskomponenten	Aufwand	zeitl. Abfolge
Einführungsveranstaltung	2 Stunden	
Planen u. Durchführen eines Projekts am Simulator (1)	4 Stunden	
Analyse der Spielverläufe und der erzielten Resultate	2 Stunden	
Seminar zum Thema Projektmanagement	2 Stunden	
Planen u. Durchführen eines Projekts am Simulator (2)	4 Stunden	

Überblick über die zeitliche Abfolge und Dauer der Schulungskomponenten

Diese fünf Komponenten bilden jeweils einen sog. Schulungsblock. Eine konkrete Schulung kann aus einem oder mehreren solcher Schulungsblöcke bestehen (s. Kapitel 4.3 und 5).

Einführungsveranstaltung

Die Einführungsveranstaltung dient dazu, den Studierenden die Bedienung des Systems und das verwendete Simulationsmodell zu erläutern. Dabei werden auch die Aktionen erklärt, die für die Projektdurchführung zur Verfügung stehen.

Planen und Durchführen eines Softwareprojekts am Simulator (1)

Die Schulungsteilnehmer erhalten Informationen über das Softwareprojekt, das sie in der Simulation leiten sollen. Die Informationen umfassen einige wichtige Projektgrößen (z.B. geschätzter Projektumfang) sowie die Zielvorgaben (vorgegebener zeitlicher Rahmen, verfügbares Budget und Qualitätsanforderungen). Die Schulungsteilnehmer müssen ihr Projekt zunächst gemäß diesen Zielen unter Beachtung der Rahmenbedingungen planen und anschließend am Simulator durchführen. Die Studierenden sind sowohl bei der Planung als auch bei der Projektführung vollständig auf sich allein gestellt. Weder das System noch ein Tutor geben ihnen den Verlauf vor oder unterstützen sie dabei, ihre Aufgaben als Projektleiter wahrzunehmen und die richtigen Entscheidungen zu treffen. Dieser Aspekt ist besonders wichtig, denn nur durch selbständiges, eigenverantwortliches Handeln können sie die erzielten Projektergebnisse als Konsequenz ihres eigenen Vorgehens begreifen und die notwendigen Erfahrungen sammeln.

Aus den Ausbildungszielen ergeben sich Anforderungen an die Simulationsmodelle (vgl. [2]): Damit der Spieler seine Rolle als Projektleiter wahrnehmen und begreifen kann, müssen die Handlungsmöglichkeiten in der Simulation mit der Realität vergleichbar sein. Wie auch in der Realität darf der Spieler wichtige Projektinformationen nur auf Anfrage oder durch entsprechende Prüfungen erhalten. Je weniger Aufwand er betreibt, um diese Informationen zu gewinnen, desto ungenauer sind die Daten. Um die komplexen Zusammenhänge in einem Projekt erkennen zu können, müssen möglichst alle Phasen eines Softwareprojekts (von der Analyse bis zur Auslieferung) modelliert werden. Beispielsweise werden die Konsequenzen von Fehlern in den frühen Phasen erst sehr viel später sichtbar.

Analyse der Projektverläufe und der erzielten Resultate

Nachdem die Schulungsteilnehmer ihr Projekt abgeschlossen haben, werden die Projektverläufe und die erzielten Resultate von einem Tutor ausgewertet. In einer gemeinsamen Analyserunde erhalten die Studierenden Rückmeldungen zu ihren Stärken und Schwächen bei der Projektdurchführung. In der Analyserunde werden die Ergebnisse mehrerer Simulationen gegenübergestellt. Die unterschiedlichen Resultate werden dann aus dem jeweils gewählten Vorgehen heraus erklärt. Auf diese Weise wird den Studierenden das Zusammenwirken der vielen verschiedenen Einflussfaktoren und der Zusammenhang zwischen ihrem Vorgehen und den erzielten Ergebnissen deutlich. Die Studierenden sollen sich dabei aktiv an der Ursachenforschung beteiligen. Durch den Vergleich lassen sich die Auswirkungen von Fehlern auf den Projektverlauf oder auch positive Effekte anschaulich demonstrieren und gemeinsam diskutieren. Auf diese Weise erfahren die Studierenden Alternativen zu ihrem gewählten Vorgehen und den gewählten Lösungsstrategien.

Ebenso wie Schneider [10] sehe ich in der Analyse der Projektverläufe in Zusammenarbeit mit einem Tutor eine notwendige Voraussetzung, um die angestrebten Ausbildungsziele zu erreichen. Erst durch die Analyse können die Studierenden die

komplexen Zusammenhänge erkennen und ihre Lehren daraus ziehen. Schneider [10] bezeichnet die Nachbearbeitung des simulierten Projekts als ebenso entscheidend wie in realen Projekten; „Nur so können Fehler dingfest gemacht und das nächste Mal vermieden werden“.

Seminar zum Thema Projektmanagement

Nachdem die Spieler eigene Projekterfahrungen gemacht haben, bietet es sich an, in einem Seminar für ein erfolgreiches Projektmanagement relevante Themengebiete wie beispielsweise Software-Metriken oder Risikomanagement zu behandeln. Das Seminar kann und soll dabei durchaus über die erfahrbaren Effekte des Simulationsmodells hinausgehen.

Planen und Durchführen eines Softwareprojekts am Simulator (2)

Abgerundet wird das Ausbildungskonzept durch eine Wiederholung der Projektsimulation. Dabei gelten dieselben Rahmenbedingungen und Zielvorgaben wie im ersten Projekt. Diese Wiederholung der Simulation ist ein entscheidender Bestandteil des Ausbildungskonzepts. Die Studierenden erhalten so die Möglichkeit, ihre Erfahrungen aus der ersten Projektdurchführung umzusetzen, indem sie ein anderes Vorgehen wählen, in Problemsituationen alternative Lösungsstrategien ausprobieren und erneut die praktische Anwendung wichtiger Projektmanagement-Techniken üben. An besseren Projektresultaten können die Spieler den Lernerfolg unmittelbar erkennen. Das Erfolgserlebnis stabilisiert das veränderte Verhalten.

4.3 Einsatzmöglichkeiten und Zielgruppen

Das Konzept selbst gibt keine konkreten Schulungsinhalte (beispielsweise die Aspekte des Simulationsmodells oder die Inhalte des Seminars) vor. Vielmehr können auf Basis dieses Konzepts verschiedene Projektmanagement-Schulungen entwickelt werden, die jeweils den Bedürfnissen der Zielgruppe angepasst sind. Damit kann dieses Ausbildungskonzept sowohl an der Hochschule als auch in der Industrie umgesetzt werden. Es ist beispielsweise für die Projektmanagement-Ausbildung von Studierenden mit nur geringem theoretischen Fachwissen im Grundstudium, von Studierenden im Hauptstudium oder von erfahrenen Softwareentwicklern, die vor ihrer Beförderung zum Projektleiter stehen, gleichermaßen geeignet.

Daraus ergeben sich verschiedene Einsatzsituationen. Integriert in eine Semesterveranstaltung kann ein einzelner Schulungsblock beispielsweise wie folgt innerhalb von vier Wochen durchgeführt werden: In einer Übung erhalten die Studierenden die Einführung, danach haben sie eine Woche Zeit, die erste Simulation durchzuführen. Anschließend wertet der Tutor die Spielverläufe aus und gibt den Studierenden nach einer weiteren Woche (z.B. im Rahmen einer Übung) Feedback. Anschließend wird das Seminar durchgeführt und die Studierenden haben wiederum eine Woche Zeit für die zweite Simulation. Wird die Schulung hingegen als Indust-

rieseminar durchgeführt, könnten die Einführungsveranstaltung und die erste Simulation am ersten Schulungstag und die restlichen Komponenten am zweiten Schulungstag ca. eine Woche später durchgeführt werden. Soll die Schulung aus mehreren Schulungsblöcken bestehen, erhöhen sich der Aufwand und der zeitliche Rahmen entsprechend. Aufgrund des geringen Aufwands lassen sich die Schulungen jedoch ohne weiteres in bestehende Studienpläne und betriebliche Weiterbildungsmaßnahmen integrieren

5 Erfahrungen mit einer konkreten Schulung

Ist das Ausbildungskonzept erfolgreich? Lässt sich mit Projektmanagement-Schulungen, die auf diesem Konzept aufbauen, der gewünschter Lernerfolg erzielen? Macht es Schulungsteilnehmern Spaß, Softwareprojekte zu simulieren?

Um diese Fragen zu beantworten, habe ich zunächst eine konkrete Projektmanagement-Schulung bestehend aus einem Schulungsblock entwickelt (d.h. konkrete Schulungsinhalte und den organisatorischen Rahmen festgelegt). Als Simulationsmodell wurde das QS-Modell eingesetzt. Im Seminar wurden grundlegende Themengebiete des Software Engineerings und Projektmanagements behandelt. Anschließend wurde diese Schulung in zwei Experimenten eingesetzt. Am ersten Experiment nahmen sechs wissenschaftliche Mitarbeiter einer technischen Fakultät an der Universität Stuttgart teil. Die Vorkenntnisse der Versuchspersonen waren sehr unterschiedlich. Die Hälfte der Versuchspersonen verfügte sowohl über theoretische Fachkenntnisse als auch über praktische Erfahrung als Softwareentwickler und teils auch als Projektleiter, andere hatten nur theoretische Kenntnisse, nur praktische Erfahrungen oder keinerlei spezifische Kenntnisse in der Softwareentwicklung. Am zweiten Experiment nahmen zehn Studierende im Hauptstudium teil, neun aus dem Studiengang Informatik und ein Student mit Nebenfach Informatik. Bei dieser Experimentgruppe waren die individuellen Schwankungen in Bezug auf den Ausbildungsstand im Software Engineering und Projektmanagement sehr gering. Alle Versuchspersonen hatten die Vorlesung „Grundlagen des Software Engineerings" besucht und die Prüfung abgelegt. Außerdem hatten alle mindestens ein studentisches Projekt durchgeführt.

Beide Experimentgruppen erhielten die gleiche Aufgabenstellung. Sie sollten ein Softwareprojekt mit einem Umfang von 200 Adjusted Function Points zunächst planen und anschließend am SESAM-Simulator durchspielen. Ihnen stand dafür ein Budget von 450.000 DM zur Verfügung, die Gesamtlaufzeit des Projekts sollte neun Monate nicht überschreiten. Code und Handbuch sollten je mindestens 95% der Anforderungen enthalten. Die Fehlerdichte im Code sollte zwölf Fehler/KLOC nicht übersteigen, für das Handbuch waren 0,5 Fehler/Seite als Obergrenze angegeben. Nach der ersten Simulation wurden die Projektverläufe und die erzielten Resultate ausgewertet, es folgten die Analyserunde und das Seminar. Im Anschluss haben alle das gleiche Projekt ein zweites Mal geplant und durchgeführt (s. Kapitel 4.2).

Zur Messung des Lernerfolgs wurde zunächst folgende Annahme getroffen: Ein Lernerfolg zeigt sich erstens in der Fähigkeit der Versuchspersonen, ein Softwareprojekt erfolgreich durchzuführen, und zweitens in ihrem Bewusstsein für die Zusammenhänge in Softwareprojekten und die Probleme und Aufgaben des Projektmanagements. Zur Bewertung des Lernerfolgs wurden die erzielten Ergebnisse und das gewählte Vorgehen in der ersten und zweiten Simulation miteinander verglichen. Zur Messung des zweiten Aspekts wurden zusätzlich Fragebögen eingesetzt. Eine ausführliche Beschreibung der Auswertungskriterien und -ergebnisse findet sich in Mandl-Striegnitz [8]. Die Ergebnisse beziehen sich dort jedoch ausschließlich auf das erste Experiment.

Die Projektmanagement-Schulung und damit unser Ausbildungskonzept erwies sich in beiden Experimenten als sehr erfolgreich.

- In der Wiederholung erzielten beide Experimentgruppen wesentlich bessere Projektresultate. Besonders die Qualität der erstellten Dokumente war im zweiten Projekt deutlich besser als im ersten, wenn auch zum Teil geringfügig zum Nachteil der Termin- und Budgeteinhaltung. Interessant ist dabei auch folgendes: Die zweite Experimentgruppe erzielt aufgrund besserer Vorkenntnisse bereits im ersten Simulationslauf wesentlich bessere Ergebnisse als die Versuchspersonen des ersten Experiments. Unabhängig davon konnte jedoch durch die Schulung eine vergleichbare Wirkung erzielt werden. Die besseren Projektergebnisse sind im Wesentlichen auf ein besseres Vorgehen der Versuchspersonen in folgenden Aspekten zurückzuführen:
- Alle Versuchspersonen haben im zweiten Projektverlauf kontinuierlicher auf die Qualität der erstellten Dokumente geachtet. Sie haben nicht nur konsequent Qualitätssicherungsmaßnahmen zur Prüfung der (Zwischen-)Resultate eingesetzt, sondern vor allem auch die Organisation und Durchführung der Prüf- und Korrekturmaßnahmen deutlich verbessert.
- Die Analyse der Projektverläufe zeigt, dass die Versuchspersonen im zweiten Projekt nicht nur verstärkt auf die Qualität der erstellten Dokumente, sondern auch auf deren Konsistenz geachtet haben. Gefundene Fehler wurden umgehend korrigiert. Außerdem haben die Versuchspersonen verstärkt daran gedacht, nach erfolgter Prüfung nicht nur den Prüfling, sondern zusätzlich auch andere, logisch damit verknüpfte Dokumente nachzuführen.
- Bessere Ergebnisse wurden auch dadurch erzielt, dass der Kunde im zweiten Simulationslauf verstärkt in den Entwicklungsprozess einbezogen wurde (beispielsweise zur Prüfung der Spezifikation und des Handbuchs).
- Die Projektdaten zeigen, dass es allen Versuchspersonen gelungen ist, den Aufwand beim zweiten Mal in sinnvollere Tätigkeiten zu investieren als im ersten Projekt: Obwohl insgesamt deutlich bessere Ergebnisse erzielt werden, sinkt der Gesamtaufwand entweder oder er steigt nur minimal.

Die Auswertung der Fragebögen bestätigt diese Ergebnisse nicht nur, sie zeigt auch, dass die Versuchspersonen in einigen Aspekten mehr über Projektmanagement gelernt haben, als sie in ihrer Rolle als Projektleiter praktisch umsetzen konnten.

- Beispielsweise hatten die Versuchspersonen auch in der zweiten Simulation noch immer große Schwierigkeiten, die zur Verfügung stehenden Mitarbeiter entsprechend ihrer Qualifikation und Erfahrung einzusetzen.
- Außerdem hatten sie wiederholt große Schwierigkeiten, den Überblick über ihr Projekt zu behalten und den Prozess zu beherrschen. Informationen aus dem Projekt wurden selten und unregelmäßig erhoben und kaum genutzt, um beispielsweise Fehler im eigenen Vorgehen oder Abweichungen vom geplanten Verlauf zu erkennen und steuernd einzugreifen.

Die Antworten im Fragebogen zeigen dagegen deutlich, dass sie sich dieser Zusammenhänge und Schwierigkeiten im Projektmanagement – spätestens nach der Analyserunde – durchaus bewusst sind. Einige dieser Aspekte werden sogar bereits in der schriftlichen Befragung vor der Schulung (Pretest) angegeben. Beispielsweise betonen alle Versuchspersonen bereits im Pretest, dass es wichtig ist, den Kunden frühzeitig in den Prozess einzubeziehen. Sie erkennen jedoch erst durch die Erfahrung aus dem ersten Projekt, welche Konsequenzen es hat, dies nicht zu tun. Im zweiten Projekt wird diese Erfahrung dann konsequent genutzt. Ebenso bemerken viele Versuchspersonen bereits im Pretest, wie wichtig es ist, dass der Projektleiter regelmäßig den Projektfortschritt kontrolliert und den Überblick über sein Projekt behält. Es gelingt jedoch kaum jemandem, dieses Wissen im ersten Projekt anzuwenden. Die Versuchspersonen erkennen das Problem und weisen deshalb im Posttest (der vor dem zweiten Simulationslauf durchgeführt wurde) nochmals verstärkt darauf hin (beispielsweise betonen sie, dass der Projektleiter Informationen selbst einholen und mitprotokollieren muss). Jedoch gelingt auch im zweiten Projekt nur wenigen eine Verbesserung, einige verschlechtern sich sogar gegenüber dem ersten Simulationslauf. Man kann sagen „das Problem ist erkannt, jedoch noch nicht gebannt". An diesem Beispiel zeigt sich anschaulich, dass Wissen alleine nicht ausreicht. Erst durch die praktische Anwendung gewinnt der Projektleiter die notwendige Erfahrung, um dieses Wissen richtig umsetzen zu können.

Am Ende der Experimente wurden die Versuchspersonen nach ihrer subjektiven Einschätzung gefragt. Die Aussagen der Versuchspersonen bestätigen die Messergebnisse. Alle Versuchspersonen bewerten ihren Lernerfolg positiv. Bemerkenswert ist, dass fast alle Versuchspersonen angeben, sich während der Simulation mit der Rolle des Projektleiters identifiziert zu haben und durchaus den Ehrgeiz verspürt haben, ihr Projekt erfolgreich abzuschließen und Alternativen auszuprobieren. Besonders wichtig war ihnen deshalb die Möglichkeit zur Wiederholung des Projekts. Durch das Erfolgserlebnis konnten sie ihren Lernerfolg unmittelbar sehen. Sie waren begeistert, endlich einmal die Praxis erproben und Gelerntes anwenden zu können. Alle Versuchspersonen haben ausgesagt, dass sie Spaß bei der Durchführung der Projekte hatten. Sie sehen in der Simulation von Softwareprojekten eine sinnvolle

Ergänzung zur konventionellen Ausbildung. Eine häufig gemachte Aussage war beispielsweise „Man bekommt durch die Simulationen ein Gefühl für die Zusammenhänge in einem Softwareprojekt. Das Wissen ist weniger diffus." Dabei betonen sie, wie wichtig in diesem Zusammenhang auch die weiteren Schulungskomponenten sind, allen voran die Analyserunde: Die Versuchspersonen bezeichnen die Analyse des gewählten Vorgehens und der erzielten Resultate als einen unverzichtbaren Teil des Ausbildungskonzepts.

6 Fazit und Ausblick

Aus den Experimenten lässt sich folgendes Fazit ziehen: Das vorgestellte Ausbildungskonzept hat sich als sehr erfolgreich erwiesen. Alle Ausbildungsziele konnten erreicht werden. Doch auch wenn die Studierenden beim zweiten Mal erfolgreicher waren, konnten sie durch das Leiten von lediglich zwei Projekten nicht soviel Erfahrung sammeln, um allen simulierten Schwierigkeiten sinnvoll zu begegnen und alle möglichen Einflussfaktoren zu berücksichtigen. Außerdem bleibt die Frage offen, ob sie ihre Erfahrungen auch auf ein völlig neues Projekt mit geänderten Rahmenbedingungen und Anforderungen übertragen können. Spielen in einem weiteren Projekt zusätzliche Einflüsse eine Rolle, erweist sich eine zuvor bewährte Lösungsstrategie unter den veränderten Bedingungen evtl. als nicht sinnvoll. Die Fähigkeit, (neue) Problemsituationen und Schwierigkeiten in einem Projekt zu meistern, hängt in erheblichem Maß davon ab, wie viele Schwierigkeiten jemand erfahren hat und wie oft er die Möglichkeit hatte, verschiedene Lösungsstrategien auszuprobieren.

Außerdem hat sich gezeigt, dass die mögliche Komplexität der Zusammenhänge und die Zahl der Effekte, mit denen man (angehende) Projektleiter in einer ersten Projektmanagement-Schulung konfrontieren kann, begrenzt sein muss, um einen Lernerfolg erzielen zu können. Eher unerfahrene Teilnehmer mit nur geringem Fachwissen haben bereits das verwendete Simulationsmodell als sehr komplex und unüberschaubar eingestuft, während die zweite Experimentgruppe nach der zweiten Simulation eine Erweiterung des Modells um zusätzliche Effekte gefordert hat. Dass aber auch für sie die Komplexität des simulierten Projekts zunächst ausreichend war, zeigen ihre Schwierigkeiten, den Überblick über ihr Projekt zu behalten. Will man hier zu viel auf einmal vermitteln, bleibt der Lernerfolg wahrscheinlich – trotz Analyserunde – aus.

Um den Studierenden also ausreichende Übungsmöglichkeiten zu bieten und sie außerdem nicht zu überfordern, soll das Ausbildungskonzept – wie bereits angedeutet – erweitert werden: Die Schulungen können sich aus mehreren Schulungsblöcken zusammensetzen. Jeder Schulungsblock ist in sich abgeschlossen und folgt dem in Kapitel 4.2 vorgestellten Konzept. Die Schulungsinhalte variieren dann je Block. Beispielsweise könnte man im ersten Block ein einfaches Softwareprojekt simulieren (geringer Projektumfang, leichte Zielvorgaben und günstige Rahmenbedingungen), damit der Projektleiter sich zunächst einmal auf seine Funktion als Projektleiter, das

Anwenden von Techniken zur Projektplanung und Fortschrittskontrolle und seine eigenen Fehler konzentrieren kann, statt durch zusätzliche Probleme realer Projekte wie beispielsweise den Ausfall wichtiger Mitarbeiter gefordert zu sein. Mit jedem weiteren Block können dann die Anforderungen an das Projektmanagement verschärft werden. Dabei hat man die Möglichkeit, beispielsweise den Umfang des Projekts zu steigern, die Zielvorgaben zu verändern oder die Komplexität der dynamischen Zusammenhänge im modellierten Projekt durch zusätzliche Effekte beliebig zu erhöhen (letzteres setzt voraus, dass weitere Modelle entwickelt werden).

Eine Schulung bestehend aus mehreren Schulungsblöcken kann beispielsweise studienbegleitend eingesetzt werden. Angefangen mit einfachen Schulungsinhalten wird im Grundstudium bereits ein erster Schulungsblock durchgeführt. Die Studierenden werden so für die Lehren des Software Engineerings und Projektmanagements sensibilisiert. Im Verlauf des Studiums, wenn zusätzliches Wissen aus Vorlesungen und Praktika hinzukommt, werden weitere Schulungsblöcke mit wachsender Komplexität der verwendeten Simulationsmodelle durchgeführt. Im Seminar werden dann Spezialthemen behandelt. Auf diese Weise erhalten die Studierenden im Verlauf ihres Studiums eine qualifizierte Projektmanagement-Ausbildung.

Wie viele Schulungsblöcke letztlich insgesamt durchlaufen werden, hängt dabei von der verfügbaren Gesamtzeit, den Vorkenntnissen der Schulungsteilnehmer und den konkreten (inhaltlichen) Lernzielen ab. Es ist jedoch zu empfehlen, mindestens drei Schulungsblöcke durchzuführen. Mit einem Gesamtaufwand von ca. 45 Stunden lassen sich die Schulungen noch immer problemlos in bestehende Studienpläne aber auch in betriebliche Weiterbildungsmaßnahmen integrieren. Der Aufwand ist, verglichen mit dem Umfang studentischer Projekte oder technischer Schulungen in der Industrie, vergleichsweise gering. Die Kosten für eine solche Schulung sind ohnehin gering im Vergleich zu dem Schaden, den unerfahrene Projektleiter verursachen.

Die Experimente haben darüber hinaus weitere Ideen und Anregungen geliefert: Wir müssen den Projektleitern Unterstützung für die wichtigen Projektmanagement-Tätigkeiten Projektplanung und Fortschrittskontrolle bieten. Bislang müssen Projektpläne aufwändig von Hand oder unter Anwendung eines (für unsere Zwecke viel zu) komplexen Projektplanungswerkzeugs geführt werden. Derzeit wird deshalb für SESAM ein Assistent zur Planung und Fortschrittskontrolle entwickelt. Er erlaubt es den Spielern, einfache Projektpläne zu erstellen und zu aktualisieren sowie wichtige Projektdaten während des Projektverlaufs zu erfassen und anschaulich aufzubereiten. Damit der Projektleiter den Überblick über seinen Projektverlauf wahren kann, steht ihm zusätzlich ein Projekttagebuch zur Verfügung.

Da sich die Auswertung der Projektverläufe und der erzielten Resultate als sehr aufwändig erwiesen hat, wird außerdem ein Auswertungswerkzeug realisiert, mit dem die notwendigen Auswertungen automatisiert werden können. Dies ist wichtig, um den Schulungsteilnehmern möglichst schnell Rückmeldung zu ihrem Projektverlauf geben und die Schulungen durchaus auch in einer größeren Gruppe sinnvoll einsetzen zu können.

Ziel der zukünftigen Arbeiten in diesem Projekt ist, weitere konkrete Schulungen zu realisieren, die auf dem erweiterten Konzept aufbauen, und diese Schulungen in der studentischen Ausbildung, möglichst auch in der industriellen Praxis, einzusetzen. In diesem Zusammenhang wird derzeit eine komplexere Variante des QS-Modells entwickelt, mit der auch einige Aspekte der Personalführung trainierbar sein sollen.

Literatur

1. Deininger, M.; Schneider, K.: Teaching Software Project Management by Simulation: Experiences with a Comprehensive Model. in: Diaz-Herrera, J. L. (Hrsg.): **Proceedings of the 7th Conference on Software Engineering Education**, San Antonio, Texas, Springer, Berlin, 1994.
2. Drappa, A.: **Quantitative Modellierung von Softwareprojekten**. Shaker, Aachen, 2000.
3. Elzer, P.: Management von Softwareprojekten. **Informatik-Spektrum**, 12 (4), 1989, S. 181-197.
4. Gibbs, W.: Software's Chronic Crisis. **Scientific American**, Nr. 9, 1994, S. 86-95.
5. Jones, C.: **Software Systems Failure and Success**. International Thompson Computer Press, Boston, 1996.
6. Ludewig, J.: **SESAM: Grundidee und Überblick**. in: Ludewig, J. (Hrsg.): SESAM – Software-Engineering-Simulation durch animierte Modelle, Bericht Nr. 5/94, Universität Stuttgart, 1994.
7. Mandl-Striegnitz, P.; Lichter, H.: Defizite im Software-Projektmanagement – Erfahrungen aus einer industriellen Studie. **Informatik/Informatique - Zeitschrift der schweizerischen Informatikorganisationen, 5 (5)**, 1999, S. 4-9.
8. Mandl-Striegnitz, P.: Untersuchung eines neuen Ansatzes zur Projektmanagement-Ausbildung. in: Dumke, R.; Lehner, F. (Hrsg.): **Software-Metriken – Entwicklungen, Werkzeuge und Anwendungsverfahren**, Deutscher Universitätsverlag, Wiesbaden, 2000.
9. Notter, A.: **Eine Untersuchung zur Wirksamkeit der Projektmanagement-Ausbildung am Simulator**. Universität Stuttgart, Diplomarbeit, 1999.
10. Schneider, K.: Komm, wir spielen Projektleiter! – Ein Lehrspiel für Software Engineering. **Tagungsband des Workshops SEUH '94 (Software Engineering im Unterricht der Hochschulen)**, München, Germany, Teubner, Stuttgart, 1994.
11. Schneider, K.: **Ausführbare Modelle der Software-Entwicklung – Struktur und Realisierung eines Simulationssystems**. vdf, Zürich, 1994.
12. The Standish Group: **CHAOS Report.** Online verfügbar unter www.standishgroup.com/chaos.html, 1995.
13. Weltz, F.; Ortmann, R.: **Das Softwareprojekt: Projektmanagement in der Praxis**. Campus Verlag, Frankfurt, 1992.

Sidestep: Die Informatik-Initiative von sd&m

Johannes Siedersleben
sd&m AG, Thomas-Dehler-Str. 27, D 81737 München
johannes.siedersleben@sdm.de

Zusammenfassung

Das Softwarehaus sd&m, München, veranstaltet in enger Zusammenarbeit mit der TU München ein sechsmonatiges Ausbildungsprogramm für Quereinsteiger, die ein natur- oder ingenieurwissenschaftliches Studium absolviert haben. Wir beschreiben die Ziele, die Organisation und den Inhalt dieses Programms.

1 Übersicht

Alle Software-Häuser in Deutschland und anderswo klagen über Nachwuchsmangel. Der Erfolg oder Mißerfolg eines Unternehmens entscheidet sich heute vor allem beim Wettbewerb um junge, geeignete Mitarbeiter: Nur wer es schafft, die besten Absolventen in ausreichender Zahl an sich zu binden, wird auf dem Markt bestehen.

Software-Häuser brauchen nicht einfach Informatiker, sondern Informatik-Berater. Das sind zunächst einmal Informatiker, die sich ihre Kenntnisse im Studium oder anderswo angeeignet haben. Sie können aber mehr: Sie schreiben gute Texte, sie halten gute Vorträge, sie treten beim Kunden gewandt und diplomatisch auf, sie sind teamfähig, sie sind belastbar, sie behalten die Übersicht und sie organisieren sich selbst. Sie sind natürlich keine Übermenschen, aber sie sind *wirksam* im Sinn von F. Malik [3].

Solche Leute sind rar. Das Greencard-Programm hilft uns dabei wenig: Es beschert uns Informatiker aus anderen Kulturkreisen, die – allein schon wegen des Sprachproblems – in der Regel erst nach längerer Einarbeitungszeit als Informatik-Berater arbeiten können. Deshalb haben wir bei sd&m einen anderen Gedanken verfolgt:

Es gibt in Deutschland viele junge Leute mit der Eignung zum Informatik-Berater und einer systematischen mathematischen Grundausbildung, denen allerdings das Informatik-Wissen fehlt. Man findet sie unter den Absolventen der natur- und ingenieurwissenschaftlichen Studiengänge. Nun herrscht auch da nicht gerade Arbeitslosigkeit, aber erstens ist der Nachwuchsmarkt dort nicht ganz so leer gefegt wie in der Informatik, und zweitens ist die Informatik auch für viele Nicht-Informatiker attraktiv wegen der interessanten Aufgaben und der guten Karriere-

möglichkeiten. Wir hatten bei der Erschließung dieses Potenzials zwei Probleme zu lösen:

- Wie kann man geeignete Nicht-Informatiker in einem vertretbaren Zeitraum mit dem erforderlichen Informatik-Wissen ausstatten?
- Wie kann man geeignete Kandidaten ansprechen und gewinnen?

Oberstes Ziel war dabei die Qualität: Wir wollten ein erstklassiges Programm für erstklassige Kandidaten; Einstiegskurse auf dem Niveau von Windows-Klickerei gibt es schon genug. Hier nun das Konzept:

Wir bieten 25 Quereinsteigern (daher der Name "sidestep") ein 6-monatiges Ausbildungsprogramm von Januar bis Juni 2001. Dieses Programm wurde in enger Zusammenarbeit mit der TU München gestaltet, die den größten Teil der Dozenten stellt. Eine Dozentin kommt von der Ludwig-Maximilian-Universität München, die übrigen von sd&m Research, der Forschungs- und Schulungsabteilung von sd&m.

Das Programm findet statt in den Räumen von sd&m München; jeder Teilnehmer bekommt einen eigenen Arbeitsplatz (Schreibtisch, Laptop usw.). Erfahrene sd&m-Mitarbeiter kümmern sich als Mentoren um das fachliche und seelische Wohl der Teilnehmer. Bereits während der Ausbildungszeit wird ein Gehalt von 2800,- Euro pro Monat gezahlt. Die Teilnehmer erhalten eine Garantie auf weitere Anstellung nach erfolgreicher Abschlußprüfung, und sie verpflichten sich, mindestens 3 Jahre bei sd&m zu bleiben. Im Fall des vorzeitigen Ausscheidens ist eine Pönale fällig.

sd&m finanziert das Programm zu 100%. Allein die Gehälter der Teilnehmer belaufen sich im Ausbildungszeitraum auf 420.000,- Euro.

2 Das Programm

Wie kann man das Informatik-Studium auf 6 Monate verdichten? Fünf Schritte führen zum Ziel:

1. Man streicht die Mathematik-Grundausbildung, denn das können die Teilnehmer schon.
2. Man streicht das Nebenfach, denn das haben die Teilnehmer als Hauptfach studiert.
3. Man streicht die Diplomarbeit, denn die Teilnehmer haben in ihrem Hauptfach bereits eine geschrieben.
4. Man streicht solche Informatik-Fächer, die für uns als Software-Haus nur untergeordnete Bedeutung besitzen (z. B. Automatisches Beweisen und Problemelösen).
5. Man vertieft diejenigen Fächer, die uns besonders wichtig sind: Programmieren, Datenbanken, Software-Entwurf.

Das Ergebnis ist ein Programm, das 560 Vorlesungs- bzw. Übungsstunden à 45 Minuten in 20 Netto-Wochen unterbringt. Diese sind verteilt auf die ersten 26 Wochen

des Jahres 2001; die Lücken ergeben sich durch den Start des Programms am 8. Januar, Osterferien, diverse Feiertage und die Prüfungswoche am Schluß. Die resultierenden 28 Wochenstunden sind hart, aber machbar. Das Programm ist gedacht als zielgerichtetes Aufbaustudium für Teilnehmer, die schon wissen, wie man lernt. Es ist aufgeteilt in zwei gleich lange Abschnitte:

Fach	**Abschnitt 1 Stunden**	**Abschnitt 2 Stunden**	**Summe**	**Dozent**
Grundlagen der Informatik	60	60	120	Paul (TUM)
Programmier-praktikum	120	60	180	Siedersleben (sd&m)
Software-Engineering		60	60	Siedersleben (sd&m)
Grundlegende Algorithmen und Datenstrukturen	40		40	Steger (TUM)
Datenbanken	40	40	80	Kossmann (TUM)
Betriebssysteme	40		40	Baumgarten (TUM)
Rechnerarchitektur	20			Linhoff-Popien (LMU)
Rechnernetze		20	40	Linhoff-Popien (LMU)
Summe	300	260	560	

Tab 1: Vorlesungsplan

Die Stundenzahlen enthalten Vorlesungen und Übungen; das Verhältnis ist etwa 3:2. Die Veranstaltungen finden im wesentlichen vormittags statt; die Nachmittage dienen der Vor- und Nachbereitung.

Weiche Themen wie Vortragstechnik oder Gesprächsführung sind nicht enthalten, denn: Erstens stehen die Teilnehmer ab dem 1. Juli 2001 im Beruf und können diese Fähigkeiten täglich trainieren. Zweitens gibt es bei sd&m gerade in diesem Bereich spezielle Fortbildungsmaßnahmen, in deren Genuß jeder sd&m-er kommt, also auch die sidestep-Leute. Und schließlich waren diese Fähigkeiten wichtiges Kriterium bei der Auswahl der Kandidaten.

Der sidestep-Absolvent ist selbstverständlich kein voller Informatiker, aber er besitzt eine solide Kenntnis der Informatik-Grundlagen und wird von daher in der Lage sein, sich rasch in neue Gebiete einzuarbeiten. Die genannten 560 Vorlesungs- und Übungsstunden entsprechen nur etwa 60 Semesterwochenstunden oder 40 ECTS-Punkten[1], aber diese schematische Betrachtung wird dem sidestep-Programm nicht gerecht. Dafür gibt es drei Gründe:

[1] European Credit Transfer System

- Die sidestep-Teilnehmer sind nach strengen Maßstäben ausgewählt. Sie haben gute mathematische Vorkenntnisse, und sie sind – nicht zuletzt auch dank der Bezahlung – hoch motiviert.
- Die sidestep-Teilnehmer haben eine optimale Arbeitsumgebung. Sie sind keine Einzelkämpfer, sondern sind bereits während der Ausbildung als sd&m-Mitarbeiter integriert. Die persönlichen Mentoren helfen in schwierigen Situationen.
- Die Ausbildung ist zugeschnitten auf das Unternehmensziel von sd&m: Entwicklung großer betrieblicher Informationssysteme. Daher haben Themen wie Datenbanken oder Programmierpraktikum ein größeres Gewicht als in manchen Informatik-Studiengängen.

Der sidestep-Absolvent ist auf den Beruf als Informatik-Berater vielleicht besser vorbereitet als ein Informatiker, der die für uns relevanten Fächer nur im vorgeschriebenen Mindestumfang belegt hat.

3 Programmierpraktikum und Software-Engineering

Dieser Abschnitt beschreibt zwei Fächer, die uns im sidestep-Programm besonders am Herzen liegen: Programmierpraktikum und Software-Engineering. Die Grundlage dazu bilden die einwöchigen, intensiven Programmer- und Designer-Schools, die sd&m seit drei Jahren für junge sd&m-Mitarbeiter durchführt.

3.1 Programmierpraktikum

Jedes Code-Review bestätigt folgende These: Viele Programme sind zu kompliziert und zu lang für das, was sie eigentlich tun sollen. Softwarequalität beginnt mit der Qualität der Programme. Deshalb wollen wir den Teilnehmern zeigen, wie man einfache, klare und im Idealfall elegante Programme schreibt. Einige Stichpunkte dazu:

- Denken in einfachen mathematischen Begriffen (Mengen, Funktionen, Prädikate),
- Programmiermuster
- Objektorientierung mit Augenmaß; Schnittstellen statt Vererbung
- Denken im richtigen Paradigma (Generierung, Vererbung, Vorlage; vgl. Coplien[1]).
- Themenorientiertes Programmieren: Wie programmiert man einen Datenbank-Zugriff, wie schreibt man einen Dialog, wie nutzt man Middleware wie Corba?

Die verwendeten Programmiersprachen sind Java, C++ und Cobol; als Übungsmaterial dienen Projekte aus der sd&m-Praxis in angepaßter Form.

3.2 Software-Engineering

Zentraler Punkt der Vorlesung ist der Software-Entwurf. Hier gilt sinngemäß dasselbe wie bei der Programmierung: Viele Systeme sind schon vom Entwurf her zu kompliziert; Software-Architekten haben einen fatalen Hang zum Komplexen. Hier unser Ansatz: Es gibt eine Reihe von immer wiederkehrenden Entwurfsaufgaben, z.B.:

- Datenbankzugriff mit Standardthemen wie Caching oder Objektidentität
- Dialogprogrammierung mit Standardthemen wie Zustandskontrolle, Fehlerbehandlung, Feldplausibilitäten, Kreuzplausibilitäten
- Berechtigungen
- Regelauswerter
- Anwendungsthemen wie Stückliste, Workflow.

Zu diesen Entwurfsaufgaben entwickeln wir Musterlösungen (nur Text und Bilder, kein Code), die mit unterschiedlichem Aufwand in verschiedene Programmiersprachen (z.B. Java, Cobol) und Programmierumgebungen (z.B. EJB, CICS) übertragbar sind (Man vergleiche hierzu Fowler[2].) Diese Musterlösungen, die sich ständig weiterentwickeln, sind das Kernstück der Ausbildung zum Software-Designer. Anhand dieser Musterlösungen vermitteln wir das Erste Gebot des Software-Entwurfs: Denken in Komponenten und Schnittstellen.

4 Zwischenbilanz

Der erste Schritt unseres Projekts – die Erstellung des Programms und die Rekrutierung der Dozenten – verlief völlig reibungslos. Hier ist besonders die Hilfe von Prof. Broy (TU München) hervorzuheben.

Der zweite Schritt war die Rekrutierung der Teilnehmer: Die Prognosen lagen zwischen 10 und 1000 Bewerbungen. Es sind (Stand Anfang November) etwa 200, nachdem wir im Juli und im September Anzeigen im redaktionellen Teil großer Zeitungen plaziert hatten (FAZ, Süddeutsche und ZEIT, ferner auch Spektrum der Wissenschaften und Physikalische Blätter). Daneben gab es die üblichen Kanäle wie Internet und persönliche Kontakte.

Die Qualität der Bewerbungen war wesentlich höher als bei unseren normalen Rekrutierungsaktionen. Wir konnten 70 Kandidaten mit ausgezeichneter Papierform einladen. Daraus ergaben sich mühelos 25 Verträge mit festen Zusagen.

Ein häufig gehörter Einwand lautet: Fast alle Software-Häuser beschäftigen doch schon heute Nicht-Informatiker in großer Zahl und fahren nicht schlecht damit – wozu also sidestep? Es sind vor allem drei Gründe:

1. sidestep macht geeignete Bewerber, denen nur das nötige Grundwissen fehlt, fit für unseren Beruf.

2. sidestep eröffnet neue Mitarbeiterquellen: sd&m erreicht gute Leute, die sonst nicht zu sd&m gekommen wären.
3. sidestep erhöht den Anteil an Mitarbeitern mit solidem Basiswissen in Informatik.

Auch bei sd&m liegt der Informatiker-Anteil nur bei etwa 50%. Daneben gibt es zahlreiche Mathematiker, Physiker, Ingenieure und Absolventen anderer Fachrichtungen, die sich – ausgehend von mehr oder weniger soliden Grundlagen – nach und nach in die Informatik eingearbeitet haben. Als grobe Schätzung kann man sagen, daß von den 50% Nicht-Informatikern bei sd&m etwa die Hälfte ausreichende Informatik-Grundlagen mitbringen. Die Übrigen (bezogen auf alle Mitarbeiter sind das also 25%) gehen einen steinigen und risikoreichen Weg: Der Festkörper-Physiker, der seine Versuchsreihen mit Hilfe großer Fortran-Programme ausgewertet hat, ist trotz seiner Fortran-Kenntnisse für die Arbeit als Software-Ingenieur in der Regel nur unzureichend vorbereitet. Training-on-the-Job mit knappen oder unzureichenden Grundlagen ist mühsam, frustrierend für den Mitarbeiter und teuer für den Arbeitgeber.

5 Folgerungen für die Informatikausbildung an den Hochschulen

Unser Ziel war die Erschließung eines neuen Mitarbeiterpotenzials, nicht die Reform der Informatikausbildung. Insofern ist das sidestep-Programm klassisch; die angebotenen Vorlesungen gibt es in dieser oder ähnlicher Form wohl in jedem Informatik-Curriculum.

Belastbare Erkenntnisse, die auch auf den Unterricht an Hochschulen übertragbar sind, werden wir wir frühestens nach Abschluß des Programms im Juli 2001 besitzen, spätestens nach der Bewährung der sidestep-Absolventen in der Praxis, also im Jahr 2002.

Wir wagen heute zwei Hypothesen, die im weiteren Verlauf zu verifizieren sind:

1. Das sidestep-Programm insgesamt kann Grundlage sein für vergleichbare Aufbaustudiengänge an Hochschulen. Dabei wäre es auf zwei oder drei Semester zu strecken. Eine Spezialisierung für andere Bereiche (etwa Echtzeitanwendungen) ist sehr gut vorstellbar.
2. Die in Abschnitt 3.1 und 3.2 herausgestellten Elemente des Programmierpraktikums und des Fachs "Software-Engineering" sind eine Bereicherung für entsprechende Veranstaltungen an den Hochschulen.

Literatur

1. Coplien, J.: Multi-Paradigm Design for C++. Addison-Wesley, 1999
2. Fowler, M.: Analysis Patterns. Addison Wesley, 199
3. Malik, F.: Führen, Leisten, Leben. DVA, 2000

Erfahrungen mit einem Werkzeug zur Projektunterstützung

Andreas Spillner
Hochschule Bremen
Fachbereich Elektrotechnik und Informatik
Neustadtswall 30
D-28199 Bremen
spillner@informatik.hs-bremen.de

Zusammenfassung

Der Beitrag berichtet über Erfahrungen beim Einsatz des ProVista® method kit [1] in einem studentischen Ausbildungsprojekt im Fachbereich Elektrotechnik und Informatik der Hochschule Bremen.

ProVista® method kit ist von der Firma 3soft in Erlangen [2] entwickelt worden und beschreibt einen Software-Entwicklungsprozess. Es ist kein Werkzeug im üblichen Sinne, sondern eher eine Sammlung von nützlichen Hinweisen, Verfahren und Dokumenten. Es ist nach dem Baukastenprinzip aufgebaut, so dass die zentralen Elemente (Methoden, Phasen, Rollen und Dokumente) leicht auf die jeweiligen Projekterfordernisse und Anwendungsgebiete anpassbar, kombinierbar und erweiterbar sind. Die Darstellung erfolgt auf Basis der World-Wide-Web-Technologie.

Neben den Beschreibungen und Erörterungen sind Dokumentmuster direkt verfügbar (in MS-Office-Formaten), die sich sehr gut als Vorlagen für die zu erstellenden Dokumente eignen.

Durch das ProVista® method kit konnte das studentische Projekt, von dem hier berichtet wird, einen Rahmen zur Unterstützung der Projektabwicklung finden, ohne das es zu langwierigen Diskussionen über eher unwichtige Fragen kam.

1 Einleitung

Die Ausbildung in Softwaretechnik sollte insbesondere an einer Fachhochschule sehr praxisnah und Ingenieur-gemäß erfolgen. Eine Schwierigkeit bei der Vermittlung des Lehrinhalts besteht oft darin, dass die Studierenden den Praxisbezug nicht direkt erkennen. Meist haben sie auch aus ihrer Tätigkeit neben dem Studium andere Vorgehensweisen - meist keine - bei der Softwareentwicklung kennengelernt. Es gilt, einen strukturierten Entwicklungsprozess zu vermitteln und nachzuweisen, dass dieser auch in der Praxis angewendet wird.

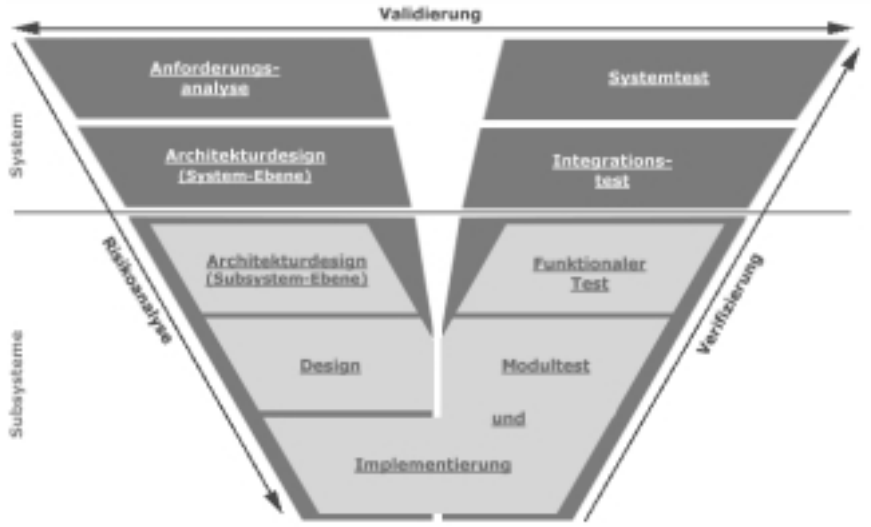

Abb. 1: Phasenmodell

Das in der Lehre verwendete Werkzeug ProVista® method kit ist ein Produkt, welches von der Firma 3soft vertrieben wird und in mehreren Firmen (ca. 50, Stand Okt. 2000) nach einer entsprechenden Anpassung an deren Bedürfnisse eingesetzt wird. Den Studierenden wird durch das Werkzeug eine Vorgehensweise der Softwareentwicklung vermittelt, die auch in der Praxis ihre Anwendung findet.

Studierende wurden auf dem SQM-Kongress 1998 in Köln auf das Werkzeug aufmerksam. Nach der Kontaktaufnahme mit der Firma 3soft wurde der Hochschule eine Version für Lehrzwecke zur Verfügung gestellt. Ein Vergleich mit anderen Werkzeugen zur Unterstützung des Software-Entwicklungsprozessese fand nicht statt.

2 ProVista® method kit

"ProVista® method kit dient der umfassenden Definition von Software-Entwicklungsprozessen und geht nach dem Baukastenprinzip vor. ProVista® method kit beschreibt alle Phasen des Entwicklungsprozesses, verteilt Rollen und Aufgaben, stellt standardisierte Dokumentenvorlagen und Checklisten zur Verfügung, benennt und erklärt alle notwendigen Entwicklungsmethoden. Diese Elemente werden nicht wie sonst üblich in Papierform, sondern auf Basis der World-Wide-Web-Technologie im HTML-Format präsentiert." [1]

Das ProVista® method kit besteht aus vier Hauptteilen, die allerdings untereinander in enger Verbindung stehen. Es gibt Phasen, Rollen, Methoden und Dokumente. Die vier Teile sollen im folgenden kurz vorgestellt werden.

2.1 Phasen

Die Methoden und Dokumente werden in ein Phasenmodell eingeordnet. Das Modell (s. Abb. 1) orientiert sich an dem V-Modell und wird von 3soft selbst für ihre Entwicklungen eingesetzt. Es werden die Phasen Anforderungsanalyse, Architekturdesign auf System- und Sub-System-Ebene, Design auf Modul-Ebene und die Implementierung unterschieden. Beim Test wird zwischen Modul-, Funktions-, Integrations- und Systemtest-Phasen differenziert.

Abb. 2: Übersicht zur Phase Anforderungsanalyse

Ausführliche Beschreibungen zu allen Phasen lassen sich durch Anklicken der entsprechenden Verweise erreichen. Zu jeder Phase sind, neben einer kurzen Beschreibung der Ziele, die Ein- und Austrittskriterien genannt sowie die bei dieser Phase erstellten Arbeitsergebnisse zusammen mit den einzusetzenden Methoden. Abbildung 2 zeigt die Übersicht der Phase Anforderungsanalyse. Die weiterführenden Informationen sind über die Verweise erreichbar.

2.2 Rollen

In der Softwareentwicklung sind unterschiedliche Aufgaben zu erledigen, die bestimmten Rollen zugeordnet werden können. Im ProVista® method kit sind zehn unterschiedliche Rollen benannt. Zu jeder Rolle gibt es eine Kurzbeschreibung der Aufgabe, Vorgaben, die erfüllt sein müssen, die Beschreibung der Aktivitäten und eine Liste der Dokumente, die für diese Rolle relevant sind.

Für die Rolle "Entwickler" gibt es folgende Beschreibung mit der entsprechenden Möglichkeit, direkt weitere Informationen abzurufen:

Kurzbeschreibung
Der Entwickler bearbeitet einen Teilbereich (z.B. ein oder mehrere Module, Subsystem, ...) und ist dort insbesondere für das detaillierte Design und die anschließende Realisierung dessen zuständig.
Vorgaben
Der Entwickler erhält
- den Anforderungskatalog für seinen Teilbereich in Form eines Pflichtenheftes,
- Terminvorgaben aus dem Projektplan.

Aktivitäten
Der Entwickler
- erstellt nach einer entsprechenden Analyse ein Design für seinen Teilbereich,
- modelliert unter möglichst weitgehender Wiederverwendung existierender Module,
- führt die Implementierung der Module durch und führt auch erste Modultests durch,
- fügt die im Rahmen seiner Tätigkeit entstandenen Dokumente der Dokumentation für das Gesamtprojekt hinzu,
- informiert den Projektleiter regelmäßig über den Stand seiner Arbeiten.

Dokumente
Der Entwickler ist verantwortlich für
- Moduldesign
- Quellcode

Er arbeitet bei der Erstellung der folgenden Dokumente mit:
- System-Design
- Release Notes

2.3 Methoden

Die im ProVista® method kit beschriebenen Methoden sind wie folgt gegliedert und mit entsprechenden Verweisen versehen (s. Kasten).

Projektmanagement
- Projektplanung
- Berichterstattung
- Meilensteintrendanalyse
- Risikomanagement

Zu jeder Methode gibt es eine Kurzbeschreibung, die Erörterung des Vorgehens, Erläuterungen zur Notation und Richtlinien zur Anwendung der Methode.

- Anforderungsmanagement
  - Anforderungsanalyse
  - Requirement Keys
  - Anforderungsverfolgung
- Design
  - Unified Modeling Language
- Implementierung
  - Programmierrichtlinien
- Integration & Test
  - Integrationsplanung
  - Teststrategien
- Support & Organisation
  - Konfigurationsmanagement
  - Change Management
- Review-Techniken

In der Kurzbeschreibung wird Sinn und Zweck der Methode beschrieben. Im Vorgehen wird detailliert dargelegt, wie und unter welchen Randbedingungen die Methode eingesetzt werden soll. Die Notation wird mit Beispielen veranschaulicht. So sind zum Beispiel bei den Programmierrichtlinien für die Programmiersprachen C, C++ und Assembler jeweils ausführliche Informationen abrufbar.

Als vierter Punkt bei den Methodenbeschreibungen sind Richtlinien angegeben, die konkrete Hinweise zum Gebrauch der jeweiligen Methode geben.

2.4 Dokumente

ProVista® method kit stellt Informationen zu folgenden Dokumenten zur Verfügung (s. Kasten).

- Projektmanagement
  - Projektplan
  - Statusbericht
  - Risikoanalyse
- Anforderungsmanagement
  - Lastenheft
  - Pflichtenheft
- Design
  - System-Designdokument
  - Modul-Designdokument
- Implementierung
  - Quellcode
- Integration & Test
  - Testplan
  - Modul-Testspez./-dokumentation
  - Funktionale Testspez./-doku.
  - Integrations-Testspez./-doku.
  - System-Testspez./-doku.
- Support & Organisation
  - Release-Notes
  - Review-Protokoll

Zu jedem Dokument gibt es jeweils eine Kurzbeschreibung, die einzusetzende Methode und die jeweiligen Verantwortlichen. Daneben sind meist eine Mustervorlage und eine Checkliste abrufbar.

Die Kurzbeschreibung verdeutlicht den Nutzen der Dokumente und gibt auch eine knappe inhaltliche Richtung vor. Die einzusetzende(n) Methoden zur Erstellung der Dokumente werden ebenso beschrieben wie die Rollen, die verantwortlich an der Erstellung der Dokumente beteiligt sind.

Besonders hilfreich bei den Dokumenten sind Mustervorlagen und Checklisten, die bei fast allen Dokumenten zur Verfügung stehen.

Die Mustervorlagen sind in MS-Office-Formaten, umfassen meist eine Standard-Gliederung und enthalten zu den einzelnen Kapiteln Hinweise zu den jeweiligen Inhalten. Die Checklisten sind eine Zusammenstellung von Fragen, die helfen, die Dokumente vollständig zu erstellen.

3 Einsatz im studentischen Ausbildungsprojekt

Im Fachbereich Elektrotechnik und Informatik der Hochschule Bremen haben die Studierenden ein Projekt zu absolvieren, das zwei Semester in Anspruch nimmt (insgesamt 8-12 SWS). Im Projekt entwickeln ca. zehn Studierende einen Prototypen eines Softwaresystems oder eines Systems, das aus Hard- und Software-Anteilen besteht. Dabei wird meist der gesamte Entwicklungsprozess durchlaufen. Im Projekt, das ProVista® method kit eingesetzt hat, wurde ein Editor zur Erstellung von UML-Klassendiagrammen realisiert. Neben der graphischen Darstellung war eine Komponente für die Umsetzung der Diagramme in einen Code-Rahmen zuständig und eine weitere kontrollierte die Einhaltung von Richtlinien in Bezug auf Namensgebung und anderer Festlegungen.

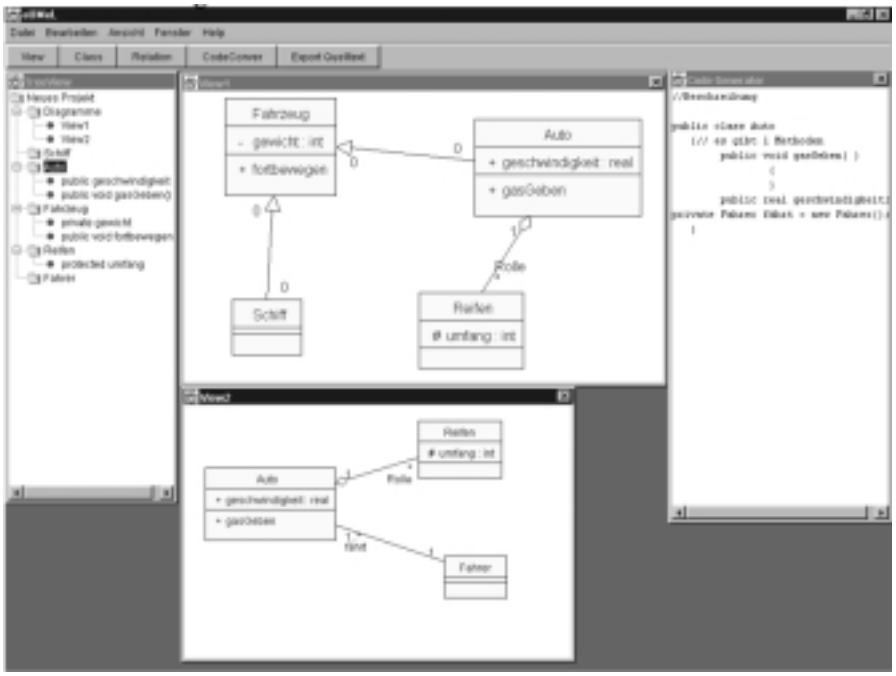

Abb. 3: Graphische Oberfläche des Prototyps

Im Projekt wurde festgelegt, welche Teile des ProVista® method kit sinnvollerweise übernommen und eingesetzt werden können. Es wurde also eine Art Tailo-

ring durchgeführt, allerdings eher parallel zum Projektverlauf und nicht durchgängig für alle Phasen zu Beginn des Projektes.

Im folgende werden einige Teile exemplarisch vorgestellt, um die Verwendung des Werkzeugs im Projekt zu veranschaulichen.

3.1 Das Pflichtenheft

Das Pflichtenheft für das Ausbildungsprojekt wurde anhand der Mustervorlage erstellt. Neben der Standardgliederung sind in der Mustervorlage folgende Angaben für das Deckblatt des Pflichtenheftes zu entnehmen: Titel, Verteiler, Beschreibung, Autor, Co-Autor, Ablageort, Version, Status, Datei und Datum. Da die Gliederung und das Deckblatt des Pflichtenheftes in der Praxis verwendet wird, gab es keinen Diskussionsbedarf über Sinn und Zweck der einzelnen Angaben. Zu den jeweiligen Gliederungspunkten ist in der Mustervorlage angegeben, welche Inhalte unter der Gliederungsüberschrift einzufügen sind.

Unter Punkt 1.3 Allgemeine Beschreibung sind folgende Hinweise angegeben:

> Die Zielsetzung enthält die Erklärung dafür, warum ein neues System zu entwickeln ist. Ggf. kann auch ein IST-Zustand beschrieben werden, der durch die Einführung des neuen Systems verbessert werden soll.
>
> Die Beschreibung sollte enthalten:
>
> - die Festlegung des offiziellen Namens,
> - die Ziele, die mit dem Projekt verfolgt werden,
> - die Einordnung in ein evtl. bestehendes Gesamtkonzept,
> - eine Problembeschreibung,
> - eine Abgrenzung zu evtl. vorhandenen anderen Projekten.

Die einzelnen Punkte wurden entsprechend der Projektvorgabe ausgefüllt. Gliederungspunkte, die für das Projekt nicht sinnvoll erschienen, wurden nach einer kurzen Diskussion weggelassen. Die Erstellung des Pflichtenheftes konnte zügig durchgeführt werden und die Diskussion beschränkte sich auf die inhaltlichen Punkte und Angaben. Eine wenig sinnvolle Diskussion über Layout-Fragen, Reihenfolgen und Aufnahme einzelner Punkte, kam durch die Verwendung der Vorlage gar nicht erst auf.

3.2 Rollen

Der nächste Schritt im Projekt betraf die Verteilung der Rollen. Jeder Studierende hat sich näher mit einer Rolle beschäftigt und die jeweiligen Aufgaben der Rolle bei einem Projekttreffen vorgestellt. Es wurde auf Grundlage der Informationen aus der Softwaretechnik-Vorlesung entschieden, ob die jeweilige Rolle im Projekt sinnvoller Weise durch eine Person besetzt werden sollte.

Die Rollen Designer, Entwickler und Tester sollten von allen Beteiligten wahrgenommen werden, da es sich um ein Projekt handelt, das die Ausbildung und nicht die Produktentwicklung als Schwerpunkt hat. Für die Projektleitung und die Systemadministration wurden jeweils zwei Studierende vorgesehen. Ein Student erhielt die Rolle des Qualitätsmanagers. Die verbleibenden vier Rollen (Technischer Redakteur, Sicherheitsbeauftragter, Change Manager, Konfigurationsmanager) wurden als nicht relevant für das Projekt angesehen. Dabei war die Rolle des Konfigurationsmanagers umstritten. Einige Studierende waren der Meinung, dass diese Rolle und ein unterstützendes Werkzeug notwendig sind. Sie konnten sich allerdings bei einer Abstimmung nicht durchsetzen. Allgemein wurde erwartet, dass nicht so viel unterschiedliche Versionen entstehen werden. Während des Projektes wurden die Rollen nicht gewechselt. So konnten sich die Studierenden intensiv mit einer Rolle vertraut machen und tiefere Kenntnisse erlangen.

3.3 Projektplanung

Im Werkzeug sind die grundlegenden Aktivitäten zur Projektplanung beschrieben. Diese wurden allerdings für das Ausbildungsprojekt auf das Notwendigste reduziert. Es gab eine grobe zeitliche Planung der einzelnen Aktivitäten. Die Einhaltung der festgelegten Bearbeitungszeiträume wurde von der Projektleitung kontrolliert. Eine Anpassung der Planung an die Realität wurde zur Mitte des zweiten Projektsemesters vorgenommen. Die in der ersten Planung freigelassenen zwei Wochen am Ende des Semesters mussten als Projektzeit hinzu genommen werden.

3.4 Reviews

Reviews werden nach ProVista® method kit zur Qualitätssicherung nahezu aller erstellten Dokumente verwendet. Die Studierenden haben das beschriebene Review-Verfahren auf die Projektbedürfnisse angepasst. Ebenso ist mit der Protokoll-Vorlage verfahren worden.

Reviews wurden von den fünf studentischen Arbeitsgruppen am intensivsten in der Codierungsphase eingesetzt. Jeweils eine Gruppe führte ein Review der vorliegenden Dokumente einer anderen Gruppe durch. Die Protokolle dienten der Informationsvermittlung, allerdings gab es zusätzlich eine Reihe von direkten Diskussionen unter den Gruppen. Einhellige Meinung war, dass Reviews sehr sinnvoll sind und zu einer erheblichen Qualitätssteigerung geführt haben.

3.5 Statusberichte und Protokolle

Im Projektmanagement sind Statusberichte der einzelnen Gruppen in bestimmten Zeiträumen vorgesehen. Bei studentischen Projekten gibt es leider oft Schwierigkeiten, den Fortschritt der einzelnen Gruppen nachzuvollziehen. Aussagen wie „Wir haben 80% des Programms fertig“ oder „Wir sind voll im Zeitplan“ sind nur schwer bzw. gar nicht kontrollierbar.

Die Statusberichte haben folgenden Inhalt:

1. Historie
2. Inhaltsverzeichnis
3. Abstrakt
4. Gestellte Ziele im Berichtzeitraum
5. Aktueller Projektstand
6. Probleme/ Störungen / Risiken/ Korrekturen
7. Offene Punkte
8. Stellungnahme des Qualitätsmanagers
9. Ziele bis KW

Durch die Verwendung der Statusberichte waren die oben erwähnten Aussagen nicht mehr möglich.

Es konnte frühzeitig auf sich abzeichnende Probleme reagiert werden. So wurde rechtzeitig vor Ende der Implementierungsphase von der Projektleitung dafür gesorgt, dass Teile der Arbeit einer Gruppe von einer anderen übernommen wurden.

3.6 Phasenmodell

Das im Werkzeug vorgestellte Phasenmodell (s. Abb. 1) wurde nicht angewendet. Das Modell legt eine sequentielle Entwicklung nahe. Diese erschien für das Ausbildungsprojekt nicht angemessen, da frühzeitig eine erste Version des Programms vorliegen soll. Andere Modelle der Softwareentwicklung werden im ProVista® method kit nicht unterstützt. Generell lassen sich aber alle zur Verfügung gestellten Methoden, Dokumentenvorlagen und Rollen auf die Belange eines Projektes anpassen. Das Tailoring wird dann in Kooperation mit dem Kunden von der Firma 3soft durchgeführt. Die entsprechende Anpassung für das Ausbildungsprojekt wurde von den Projektbeteiligten selbst durchgeführt.

Im Projekt wurde inkrementell entwickelt. Grundlage war der Entwicklungsprozess in Iterationen, wie er in [3, Kapitel 4.4] beschrieben wird. Es gab während des Projektes zwei Iterationen, also aufeinander aufbauende Ausbaustufen des Editors. Die erste Iteration wurde allerdings aus Zeitgründen nicht ausgiebig getestet.

4 Bewertung

ProVista® method kit ist sicherlich nicht als Werkzeug im engeren Sinne anzusehen. Es ist eine sehr sinnvolle Zusammenstellung der Phasen, Rollen, Methoden und Dokumente, die in der Softwareentwicklung von Bedeutung sind. Durch die Aufbereitung der Informationen im HTML-Format und die Bereitstellung von Mustervorlagen in MS-Office-Formaten ist keinerlei oder nur eine sehr geringe Einarbeitungszeit notwendig. Die Informationen sind mit jedem Browser abrufbar. Diese Art der Aufbereitung wird von den Studierenden gerne angenommen und viel intensiver genutzt als ein entsprechendes Buch mit gleichem Inhalt.

Im Fachbereich Elektrotechnik und Informatik steht den Studierenden bereits eine Web-basierte Lernplattform [4] zur Verfügung, so dass sie es gewohnt sind, mit

dem Medium zu arbeiten. In dieser Lernplattform wurden auch alle Dokumente des Projektes verwaltet.

Durch das Werkzeug wurden Standards für Vorgehensweisen und Dokumente gesetzt, die von den Studierenden ohne große Diskussion akzeptiert und auch eingehalten wurden. Die Vorlagen sind ohne Aufwand auf die jeweilige Projektsituation anpassbar, was nahezu für alle Dokumente vorgenommen wurde und die Akzeptanz insgesamt weiter erhöhte. Die Studierenden hatten dadurch nie das Gefühl, unnütze Dokumente erstellen zu müssen; im Gegenteil, die Bedeutung und Notwendigkeit von schriftlichen Festlegungen wurde allen klar.

ProVista® method kit gibt den Studierenden einen Leitfaden durch die einzelnen Aktivitäten und die Rollenverteilung während der Softwareentwicklung. In der abschließenden Bewertung des Projektes wurde von den Studierenden einhellig festgestellt, dass der Verzicht auf ein Konfigurationsmanagement eine Fehlentscheidung war. Auch bei relativ kleinen Projekten ist die Aufgabe des Konfigurationsmanagers nicht zu vernachlässigen.

Zusammenfassend lässt sich feststellen, dass beide Seiten – Lehrende und Lernende – das ProVista® method kit sehr positiv bewerteten.

Einige Teile aus dem Werkzeug werden inzwischen auch in der Vorlesung Softwaretechnik verwendet, um die Praxisrelevanz zu verdeutlichen.

Literatur und Hinweise im Netz (Stand 9.2000)

1. http://www.3soft.de/d/provista/index.php3?select=provista
2. http://www.3soft.de/
3. Oesterreich (Hrsg.), B.; Hruschka, P.; Josuttis, N.; Kocher, H.; Krasemann, H.; Reinhold, M.: Erfolgreich mit Objektorientierung. Oldenbourg, München Wien, 1999
4. http://www.weblearn.hs-bremen.de/

Projektdaten

Das Projekt eUMeL - editor for Unified Modeling Language - wurde an der Hochschule Bremen im Fachbereich Elektrotechnik und Informatik im SS99 und SS00 durchgeführt. Projektbeteiligte waren: Bayram Alpamar, Frank Breidbach, Norman Helm, Wolfgang Meyer, Swen Moczarski, Henning Plump, Sven Riemann, Wiebke Schröder, Enno Siebels und Holger Viehoefer. Betreut wurde das Projekt von Prof. Dr. Andreas Spillner und Prof. Dr. Ulrich Breymann (im SS99).

Copyrightangaben

Der Beitrag von Groupware-Applikationen zur methodengestützten Abwicklung von studentischen Projektseminaren

Michael Röthlin
Universität Bern, Institut für Wirtschaftsinformatik, Engehaldenstrasse 8, CH-3012 Bern,
roethlin@ie.iwi.unibe.ch

Zusammenfassung

Die Durchführung von studentischen Projektseminaren stellt sowohl an Betreuende als auch Studierende hohe Anforderungen. Auf Grund der oft mangelnden Erfahrung der Beteiligten ist der Aspekt der Projektführung besonders zu beachten; zur Unterstützung des Projektmanagements empfiehlt sich die Verwendung eines detaillierten Vorgehensmodells. Weiterhin stellt die Team-Kommunikation eine wichtige Herausforderung in der Projektarbeit dar. Ein wesentliches Mittel dabei sind Dokumente, die einzelne Projektmitarbeitende erstellen und andere weiter verwenden; bei dieser kollektiven Problembearbeitung verspricht der Einsatz von Groupware eine gewichtige Unterstützung. Um den genannten Aspekten in Projektseminaren besser gerecht zu werden, wurde am IWI der Universität Bern im Sommersemester 2000 die Groupware Lotus Notes in Kombination mit der Vorgehensmethode HERMES eingesetzt. Der vorliegende Erfahrungsbericht beschreibt die allgemeinen arbeitstechnischen Problemstellungen in Projektseminaren, die Implementierung des Vorgehensmodells und der Groupware im Projekt, deren gegenseitige Abhängigkeiten und ihren Beitrag zur Projektabwicklung.

1 Einleitung – das Projektseminar GIS

1.1 Einbettung des Projektseminars in das Studienprogramm

Im Rahmen des Hauptstudiums der Wirtschaftswissenschaften an der Universität Bern wird vom Institut für Wirtschaftsinformatik (IWI) das Fachprogramm GIS (Gestaltung von Informations-Systemen) angeboten. Es besteht aus vier Teilen:

- Vorlesung I und II, insgesamt 3 SWS (Semesterwochenstunden).
- Übungen I (CASE-unterstützte Systementwicklung), 3 SWS.
- Übungen II (Programmierkurs *Microsoft Visual Basic*), 2 SWS.
- Projektseminar, 4 SWS.

In der Vorlesung werden die grundlegenden Methoden und Werkzeuge für die Entwicklung von betrieblichen Informationssystemen vermittelt und in den Übungen I und II vertieft. Das Projektseminar wird in der Regel mit einem Praxispartner durchgeführt, wobei ein möglichst direkter Bezug der Problemstellung zu den Vorlesungsinhalten angestrebt wird.

1.2 Umfang und Ziele des Projektseminars

Mit dem drei Monate dauernden Seminar, an welchem jeweils ca. 20 Studierende unter der Leitung einer Assistenzkraft mitwirken, werden folgende Ziele verfolgt:

- Anwendung der in den Vorlesungen und Übungen vermittelten Methoden des Software-Engineerings.
- Praktische Projektarbeit anhand einer konkreten Problemstellung der Wirtschaftsinformatik.
- Entwicklung der Sozial- und Projektmanagementkompetenzen der Beteiligten.

Auf Grund der Verschiedenartigkeit der Problemstellungen existieren keine generellen Vorgaben bezüglich der Arbeitsweise; die Wahl eines geeigneten Vorgehensmodells liegt im Ermessen des Projektteams und letztlich der Assistenzkraft.

1.3 Herausforderungen in Projektseminaren

Die Herausforderungen studentischer Projektseminare im Rahmen des Fachprogramms GIS lassen sich wie folgt charakterisieren (siehe dazu auch Myrach in [6]):

- Unerfahrenheit bezüglich Methoden, Werkzeuge, Vorgehensweisen und Problemstellung: Für die Mehrzahl der Studierenden ist das Projektseminar der erste Kontakt mit einer realitätsnahen Projektsituation der Wirtschaftsinformatik. Bei den betreuenden Assistenzkräften gestaltet sich die Lage ähnlich; ihnen fehlen zumeist Projektmanagementerfahrungen.
- Enge zeitliche Begrenzung der Seminare: Die Abwicklung eines derart kurzen, zu Beginn schwach dokumentierten Projektes mit externen Partnern stellt an sich bereits eine grosse Herausforderung dar. Hinzu kommt, dass die Beteiligten während des Projektseminars auch andere Lehrveranstaltungen besuchen (müssen). Diese Konkurrenzsituation erfordert, dass Studierende während der Laufzeit des Seminars mit weitgehender zeitlicher und örtlicher Autonomie arbeiten können.

1.4 Inhalte und Besonderheiten des Projekts *ADAM*

Das hier besprochene Projektseminar hatte die Entwicklung einer Softwarelösung zum Ziel, welche den Datenaustausch zwischen einem auf *Lotus Notes* basierenden Verkaufsprojekt-Verfolgungssystem und einer auf *SAP R/3* basierenden Materialwirtschaft automatisieren sollte. Das Projekt *ADAM* (*A*utomation des *D*atenflusses zwischen *A*bsatzvorschau und *M*aterialbedarfsplanung) wurde in Zusammenarbeit mit einem Industriepartner durchgeführt, bei welchem die realisierte Lösung zukünf-

tig zur Stabilisierung und Verbesserung der Materialbedarfsplanung eingesetzt werden soll.

2 Einsatz der Projektführungsmethode *HERMES*

2.1 Herkunft, Ziele und Umfang von *HERMES*

HERMES, ein Instrument zur Abwicklung von Informatikprojekten der Schweizer Bundesverwaltung, befindet sich seit 1975 im praktischen Einsatz [4]. Als Ziele von *HERMES* werden die Steigerung der Qualität von Informationssystemen, die Verbesserung der Kommunikation zwischen Fachbereichen und Informatikabteilung sowie die Verringerung von Projektrisiken und -kosten genannt.

Die Methode umfasst ein phasenorientiertes Modell für Informatikprojekte, die Definition der Aufgaben, Kompetenzen und Verantwortlichkeiten aller Beteiligten, ein standardisiertes Berichtswesen sowie Massnahmen zur Qualitätssicherung. Zahlreiche Elemente anerkannter Standards, insbesondere des *V-Modells*, wurden in das Werk übernommen. Die Anwendung der Methode ist für alle Informatikprojekte bei der Schweizer Bundesverwaltung verbindlich; *HERMES* kann für den persönlichen Gebrauch vergütungsfrei genutzt werden.

2.2 Umsetzung der Methode im Projektseminar

HERMES unterscheidet bei der Abwicklung von Informatikprojekten fünf Phasen: Initialisierung, Voranalyse, Konzeptphase, Realisierung und Einführung. Im Rahmen des Projektseminars wurden nur die ersten drei Phasen abgearbeitet. Ausserdem wurde auf die Verwendung einer detaillierten Projektablaufstruktur wie zum Beispiel *HERMES SWE* (siehe [5]) verzichtet; einzelne der dort beschriebenen Vorgehensweisen und Dokumentenvorlagen fanden jedoch Verwendung. Am Ende jeder der drei implementierten Phasen wurde eine Zusammenkunft sämtlicher Studierender und Vertreter des Partnerunternehmens anberaumt; diese umfasste jeweils Kurzpräsentationen der Arbeiten und Ergebnisse durch die beteiligten Teams, gefolgt von einer Diskussion und einem Ausblick auf die nächsten Schritte.

Für alle Projektergebnisse spezifiziert *HERMES* die jeweiligen Beteiligten, deren Soll-Aktivitäten und die zu erstellenden Dokumente. Aus den durch das Modell definierten mehr als 50 Ergebnisbeschreibungen wurden fünfzehn ausgewählt und bearbeitet: Arbeitsauftrag, Detailstudie, Projekthandbuch, Prüfspezifikation, QS-Plan, Systemziele, usw.

Die Submodelle von *HERMES* definieren die Ergebnisse, Aktivitäten und Rollen der Beteiligten. Beschrieben werden Projektrollen (PR), Projektmanagement (PM), Qualitätssicherung (QS) und Konfigurationsmanagement (KM), wobei die letzten drei dem Standard *V-Modell* entnommen sind.

Sämtliche Projektrollen von *HERMES* (beinhaltend die Projektorganisation mit ihren Kompetenzen, Aufgaben und Verantwortungen) wurden im Rahmen des Pro-

jektes implementiert: Koordinations- und Kontrollstellen, Entscheidungs- und Steuerungsfunktionen, Projektleitung, Projektsachbearbeitung, Qualitätssicherung, Konfigurationsmanagement und Projektunterstützung.

2.3 Implikationen für das Projektseminar

Durch die Vorgabe einer Projektmethode wurde als organisationsbezogenes Lernziel innerhalb des Projektseminars nicht die Suche nach einer optimalen Organisationsstruktur definiert, sondern das Ausgestalten eines vorgegebenen, bereits beschriebenen Rollenmusters. Bezüglich Teamaufgaben und -organisation erzwingt *HERMES* keine eigenen Strukturen, so dass im Laufe des Projektes zwei Organisationsformen ohne Änderung des Vorgehens implementiert werden konnten.

3 Der Groupware-Einsatz

3.1 Zweck des Groupware-Einsatzes

Die Idee, die Teamarbeit im Projektseminar durch geeignete Informatik- und Kommunikationsmittel zu unterstützen, liegt nahe. Auf dem Markt werden seit Jahren verschiedenste Produkte zur Computerunterstützung der Teamarbeit angeboten. Unter Groupware-Applikationen versteht man aus Software und eventuell spezifischer Hardware bestehende Systeme, durch welche Gruppenarbeit unterstützt oder ermöglicht wird [7].

Der Bedarf nach Kommunikationsunterstützung lag im Projektseminar sowohl im zeitlich verschobenen Austausch von dokumentenbezogenen Informationen (Holprinzip) als auch in der Verteilung von Aktualitäten zwischen Einzelnen oder innerhalb von Gruppen (Bringprinzip). Der erwartete Beitrag zur Verbesserung der Koordination umfasste Hilfestellungen bei der Termin- und Aufgabenplanung, dem Erteilen von Aufträgen und der Darstellung von Ergebnissen in strukturierter Form.

Da sich die Systementwicklung in Informatikprojekten oft im Formalisieren, sprich Dokumentieren von Problemstellungen und Lösungsansätzen niederschlägt, war eine Software gefragt, welche die Kooperation bei der Dokumentenerstellung im Team unterstützt. Hier sind die folgenden Forderungen zu nennen: die Möglichkeit zur Arbeit über das Internet, die Verfügbarkeit von Hypertext-Verknüpfungen, die Verwaltung mehrerer Versionen und ein Schutzmechanismus auf Dokumentenebene, welcher Datenverluste bei Mehrfachzugriff verhindert.

3.2 Auswahl der Groupware-Lösung

Da das Produkt *Lotus Notes* bereits als Entwicklungs- und Zielplattform der zu entwickelnden Lösung vorgegeben war, lag die Verwendung von *Notes* auch als Groupware-Applikation nahe. Versuche, Projektdokumente mit Office-Applikationen wie *Microsoft Word* zu erstellen, wurden nach Beginn des Projektseminars

aus Gründen der besseren Integration in die Gesamtdokumentation und eines einfacheren Erstellungsprozesses rasch zu Gunsten von *Notes* aufgegeben.

3.3 Die Groupware *Lotus Notes* – ein Überblick

Die *Notes*-Familie der Firma *Lotus Development* ist mit derzeit weltweit über 50 Mio. Installationen Marktführerin im Bereich der kollaborativen Office-Autorenwerkzeuge [3]. Das System *Lotus Notes* basiert auf verteilten, gemeinsam nutzbaren Dokumentendatenbanken [2]. Jede dieser Datenbanken besitzt ein Berechtigungsschema, wobei Zugriffe rollenbezogen eingeschränkt werden können. Zusammengehörige Daten (Datensätze) werden in *Notes* als Dokumente in Datenbanken abgespeichert. Dokumente können mit Hilfe von Ansichten und Masken dargestellt und bearbeitet werden. *Notes* erlaubt das Einfügen von Verweisen in Dokumente, welche so untereinander, mit lokalen Dateien oder mit Internet-Ressourcen verknüpft werden können. Schließlich sorgt die Software mit einem Automatismus dafür, dass bei der Speicherung eines Dokumentes, welches von mehreren Personen gleichzeitig bearbeitet wird, keine Datenverluste auftreten.

Der *Domino*-Server verwaltet *Notes*-Datenbanken und wickelt den Datenverkehr zwischen diesen ab. Nebst dem Serverbetrieb im Zusammenspiel mit *Notes*-Clients unterstützt ein *Domino*-Server auch Internet-Protokolle, so dass er Datenbankelemente wie Dokumenteninhalte und Ansichten als Web-Dokumente für den Zugriff über Browser bereitstellen kann. Der *Notes*-Client kann sowohl auf *Notes*-Datenbanken als auch Web-Inhalte zugreifen. Nebst dem *Notes*-Client stehen für die Server-Administration und die Gestaltung von Datenbanken jeweils spezialisierte Applikationen zur Verfügung.

E-Mails werden von *Lotus Notes* als Dokumente behandelt und können so verschiedene Gestaltungselemente wie Grafiken, Tabellen oder Verweise auf andere *Notes*-Dokumente, -Ansichten oder -Datenbanken enthalten. In die Mail-Datenbank integriert befindet sich die Aufgaben- und Terminverwaltung. Aufgaben und Termineinträge werden von *Notes* ebenfalls als spezialisierte Dokumente behandelt.

4 Ablauf des Projektseminars

4.1 Projekt-Meilensteine

Aufbau der Infrastruktur und Schulung

Die Aktivitäten vor dem Beginn des Projektseminars umfassten Abklärungen und Vereinbarungen mit dem Industrie-Partnerunternehmen sowie den Aufbau der Kommunikations- und Arbeitsinfrastruktur.

Auf einem Server des Instituts wurde eine vollständige *Domino*-Server-Konfiguration und auf den Arbeitsstationen des den Studierenden zur Verfügung stehenden Seminarraums je ein *Notes*-Client installiert (Release 5.0.1). Zudem wurde für

alle Projektbeteiligten je eine Mail-Datenbank angelegt und der Internetzugriff auf die relevanten Datenbanken mit Passwortschutz freigeschaltet. Leider war der Versand von Mails aus den *Notes*-Clients in Richtung Internet aus betrieblichen Gründen nicht realisierbar; diese Funktionalität wäre insbesondere für die Kommunikation mit externen Stellen nutzbringend gewesen.

Zur Erstellung und zentralen Ablage der während der Projektabwicklung behandelten Dokumente (insbesondere aller *HERMES*-Dokumente) wurde eine Datenbank mit der *Notes*-Schablone *TeamRoom* erstellt. Diese vordefinierte Struktur erlaubt es, mit einer einzigen Datenbank einen Grossteil der Anforderungen bezüglich Kommunikation, Koordination und Kooperation zu erfüllen. Namentlich lassen sich Dokumente gleichzeitig mehreren Kategorien sowie Teams oder Einzelpersonen mit Angabe eines Fälligkeitstermins zuordnen. Die via Web-Browser über Internet zur Verfügung stehende Funktionalität zur Bearbeitung von Dokumenten ist dabei nahezu identisch mit derjenigen des *Notes*-Clients. Erweitert wurde die Standardfunktionalität der *TeamRoom*-Anwendung durch eine News-Rubrik, in welcher sowohl Projektaktualitäten als auch teamspezifische Dokumente wie Sitzungsberichte oder weitere organisatorische Mitteilungen aufbereitet wurden.

In Ermangelung eines eigentlichen CASE-Werkzeuges für die Entwicklung unter *Lotus Notes* wurde von Studierenden und Assistenten ein einfaches Data Dictionary konstruiert. Dieses besteht aus einer *Notes*-Datenbank, welches die Verwaltung von Feldern, Masken und Ansichten erlaubt und über Verweise direkt mit anderen *Notes*-Dokumenten verknüpft werden kann. Mit Hilfe dieses Werkzeugs wurden sämtliche Definitionen von Daten, Datenflüssen, Prozessen usw. vorgenommen.

Da die Studierenden bei Projektstart nur sehr vereinzelt über einschlägige Kenntnisse verfügten, wurde zu Beginn des Projektseminars ein zweitägiger Kurs durchgeführt, in welchem die grundlegenden *Notes*-Komponenten vorgestellt wurden und der Umgang mit E-Mail, Terminplaner und selbst erstellten Datenbanken anhand von praktischen Übungen erlernt werden konnte. Als erstes Teilprojekt wurde dabei von den Studierenden ein Internetportal realisiert, über das in der Folge die Kommunikation mit dem Partnerunternehmen abgewickelt wurde.

Aufbau der Projektorganisation

Anlässlich des "Kick-Off"-Meetings des Projektseminars wurden die Studierenden in das Modell *HERMES* eingeführt und mussten sich für die Übernahme einer der zur Verfügung stehenden *HERMES*-Rollen entscheiden. Die Leitung des Projektes oblag einem Management-Team, welches sich aus dem betreuenden Assistenten und drei Studierenden zusammensetzte. Im mindestens zweiwöchentlichen Rhythmus wurden der Stand der Arbeiten und das weitere Vorgehen besprochen und Folgedokumente wie Aufträge, Hinweise und Communiqués im *TeamRoom* abgelegt.

4.2 Grundkonstrukte der Zusammenarbeit

Um den Einsatz der *Notes*-Groupware im Projektseminar-Alltag zu illustrieren, soll deren Verwendung anhand dreier repräsentativer Abläufe erläutert werden.

Aufgabenmanagement: Nach der Abstimmung von Ziel und Inhalt einer Aufgabe durch das Management-Team wurden die nötigen Angaben jeweils in einem Aufgabendokument im *TeamRoom* festgehalten. Die Verantwortlichkeiten wurden mit Hilfe des *Notes*-Benutzernamens von einzelnen Studierenden oder Gruppen im entsprechenden Dokument eingetragen, ebenso der Fälligkeitstermin der Aufgabe. Ergebnisse und allfällige Kommentare (wie beispielsweise Bewertungen) konnten dabei als Antwortdokumente neben den Aufgaben dargestellt werden, was eine strukturierte, hierarchische Ansicht von Fragestellungen und Ergebnissen ermöglichte.

Event Management: Die Organisation von Sitzungen mit einer grossen Anzahl von Teilnehmenden wurde werkzeuggestützt mit Hilfe der elektronischen Agenden der Studierenden durchgeführt. Projekt-Meilensteine wurden im *TeamRoom*-Kalender aufgeführt. Die gezeigten Unterlagen, die gefassten Beschlüsse und verteilten Aufgaben wurden nach der Durchführung der Veranstaltung in der *TeamRoom*-Datenbank festgehalten.

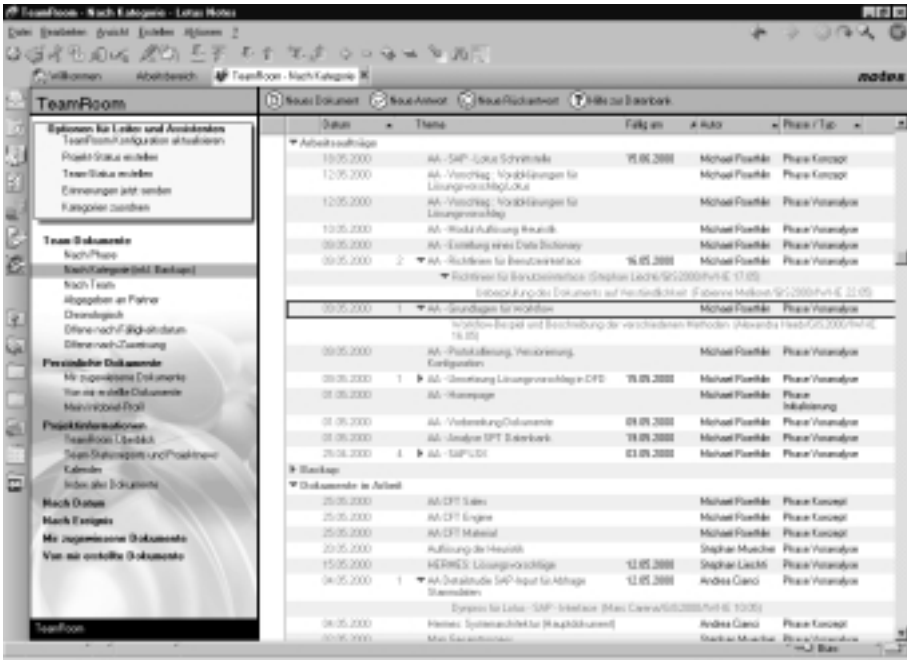

Abb. 1: Aufgaben-, Termin-, und Dokumentenmanagement im *TeamRoom*.

Dokumentenerstellung: Um die Erstellung der von *HERMES* geforderten Dokumente zu erleichtern, wurden vor dem Beginn der Bearbeitung Vorlagen sämtlicher Doku-

mententypen erstellt. Da *TeamRoom*-Dokumenten generell Fälligkeitstermine und Verantwortlichkeiten zugeordnet werden konnten, wurden sie zusammen mit den eigentlichen Arbeitsaufträgen in einer Gesamtsicht im *TeamRoom* dargestellt (siehe Abb. 1).

5 Bewertung der Verwendung der Groupware *Lotus Notes*

Im Folgenden sollen die Abhängigkeiten zwischen Projektführungsmethode und Groupware-Applikation und die Auswirkungen auf die Arbeitsweise von Studierenden und Assistenten dargestellt werden.

5.1 Abhängigkeiten zwischen Methode und Groupware

Mit der Implementierung wichtiger Ergebnisvorlagen und modellspezifischer Darstellungen in *Lotus Notes* konnte die Akzeptanz des Projektführungssystems *HERMES* gefördert werden, ohne dass deswegen die beiden Instrumente in einer strikten Abhängigkeit voneinander gestanden hätten. Die Freiheiten bei der Gestaltung der Projektorganisation oder der Verwendung der einzelnen Elemente der Methode wurden durch die Groupware nicht beschnitten. Beispielsweise konnten neue Teams im System jederzeit frei definiert und Aufgaben sowohl an Teams als auch an Einzelpersonen zugeteilt werden.

5.2 Auswirkungen auf die Arbeitsweise der Beteiligten

Der Groupware-Einsatz verschaffte den Studierenden im Projektseminar folgende Vorteile:

- Der Informationsaustausch wurde erleichtert; sämtliche relevanten, elektronisch gespeicherten Fakten standen im *TeamRoom* strukturiert zur Verfügung.
- Die Dokumentenerstellung wurde durch die Möglichkeiten der verteilten und asynchronen Bearbeitung vereinfacht, Referenzen auf existierende Dokumente konnten über *Notes*-Hyperlinks *(DocLinks)* sehr einfach erstellt werden.
- Mit der Kategorisierung nach "Team", "Thema" oder "Phase" einerseits und der Zuweisung der Verantwortlichkeiten der Dokumentenerstellung auf Personen andererseits konnten die Studierenden sehr rasch einen Überblick über die anstehenden Aufgaben und deren Fälligkeit gewinnen.

Für den beteiligten Assistenten resultierte aus der Verwendung der *Lotus Notes*-gestützten Werkzeuge eine wesentliche Unterstützung bei der Projektführung. Mit wenigen Mausklicks konnte mittels Terminierung und Verantwortlichkeitskennung der Dokumente ein Überblick über den Stand der Arbeiten gewonnen werden.

Als negative Punkte sind vereinzelte technische Probleme bei der Konfiguration der Internet-Browser sowie der Lernaufwand beim Einsatz der Textbearbeitungs- und Kommunikationsfunktionen von *Notes* aufzuführen.

5.3 Eignung von *Lotus Notes* als Arbeitsplattform

Das Groupware-System verschaffte den Beteiligten einen Dispositionsfreiraum bei der Lösung von dokumentierenden Aufgaben; es ermöglichte sowohl eine zeitliche als auch (über den Internet-Zugang) örtliche Entkopplung der Aktivitäten.

Die Möglichkeiten zur gemeinsamen Erstellung von Dokumenten unter *Lotus Notes* wurden sehr positiv beurteilt. Insbesondere die vielfältigen textlichen Gestaltungsmöglichkeiten schnitten in der Gunst der Studierenden gut ab; die Erstellung einer Hypertext-Dokumentation war für die meisten eine neue, bereichernde Erfahrung. Der Einsatz der *TeamRoom*-Datenbank als Dokumentenspeicher und Team-"Gedächtnis" wurde von den Beteiligten als nutzbringend erachtet. Bereits dokumentierte Lösungen wirkten als Vorbilder und beschleunigten den Lernprozess.

Nebst der direkten Unterstützung der Dokumentenerstellung im Team wurden Planungsaktivitäten wie das Aufgaben-Management wirkungsvoll unterstützt. Verantwortlichkeiten für Aufgaben waren durch die Zuweisung von Dokumenten an Personen oder Gruppen, zusammen mit der Angabe des Fälligkeitstermins, sehr einfach abzubilden. Auf diese Weise konnte auf die Verwendung einer eigentlichen Projektüberwachungs-Software verzichtet werden.

Aus Sicht der IT-Infrastruktur eignete sich die *Lotus Notes*-Umgebung auf Grund ihrer kompakten Form hervorragend für den raschen Aufbau einer kollaborativen Arbeitsumgebung. In einem einzigen Installationsvorgang konnten die E-Mail-, Datenbank- und Web-Serverfunktionen aufgesetzt werden. Ein kritischer Punkt war dabei allerdings die Integration der externen Kommunikation: Die nur limitiert nutzbare, weil aus betrieblichen Gründen nicht mit dem Internet integrierte Mail-Funktionalität der *Lotus Notes*-Installation führte dazu, dass das *Notes*-Mail nur selten und unzuverlässig benutzt wurde (siehe dazu auch Bullen und Bennett in [1]).

5.4 Grenzen des Einsatzes von Groupware-Systemen

Der kritische Prozess der Zuteilung von Aufgaben unter Berücksichtigung von Kriterien wie Vorwissen, Arbeitsauslastung oder Präferenz musste im Führungsteam abgestimmt werden und konnte nicht der "Maschine" überlassen werden.

Schöpferische Gruppenarbeiten wie das Sammeln, Weiterentwickeln und Bewerten von Lösungsansätzen wurden nie über die Groupware abgewickelt – in dieser Situation bewährten sich Arbeitstreffen, denen der Assistent weitgehend passiv beiwohnte. Als wertvoll wurden von den Studierenden die Plenarsitzungen zu wichtigen Projekt-Meilensteinen empfunden, an denen Resultate und geplante Vorgehensweisen durch die Teammitglieder vorgestellt wurden, Fragen gestellt und diese direkt beantwortet werden konnten.

Eine grosse Herausforderung für das einzelne Teammitglied lag trotz der detailgenauen Dokumentation letztlich darin, die Übersicht über den Stand des Gesamtprojektes und das Ziel der laufenden Arbeiten zu bewahren. Weiterhin ist die durch

die *HERMES*-Richtlinien sichergestellte formale Qualität der Dokumentation noch kein hinreichendes Kriterium für die semantische Qualität der Arbeitsergebnisse.

Auch beim Einsatz einer Groupware-Lösung bleibt somit die engagierte und reflektierende Mitarbeit aller Beteiligten der Schlüsselfaktor für den Erfolg eines Projektseminars.

6 Schlussfolgerungen

Die Groupware-Lösung *Lotus Notes* (Release 5) eignet sich sehr gut für die Unterstützung von verteilt wirkenden Arbeitsgruppen. Sowohl bei der Installation als auch der Anpassung an Vorgehensmethoden wie *HERMES* und insbesondere dessen Dokumententypen ist das System einfach handhabbar.

Ein Groupware-System wie *Notes* bietet angesichts der Herausforderungen von Projektseminaren ausreichende Hilfsmittel, was die Planung und Beherrschung der Komplexität betrifft; allerdings müssen Studierende systematisch an den Umgang mit ungewohnten Hilfsmitteln wie einem zentralen Dokumentensystem herangeführt werden, da die üblicherweise benutzten Kommunikationsmittel wie E-Mail auf anderen Prinzipien beruhen. Die bei derartigen Projekten besonders wichtigen Aktivitäten der (An-)Leitung und Kontrolle werden durch *Notes* wirkungsvoll unterstützt.

Unter Berücksichtigung der genannten Einschränkungen kann die Verbindung einer Projektführungsmethode wie *HERMES* und der Groupware *Lotus Notes* für den Einsatz in studentischen Projektseminaren empfohlen werden. Zum einen können so Lösungsansätze für praktisch alle Herausforderungen einer Projektsituation gefunden und in kurzer Zeit effektive Projektstrukturen und Informationskanäle aufgebaut werden. Zum anderen hält der Einsatz derartiger Hilfsmittel die Studierenden zu einer kritischen Auseinandersetzung mit gebräuchlichen Techniken einer professionellen Projektabwicklung und in der Geschäftswelt verbreiteten Kommunikationshilfsmitteln an.

Literatur

1. Bullen, C. V., Bennett, J. L.: Learning from User Experience with Groupware. In: Marca, D., Bock, G. (Hrsg.): Groupware: Software for Computer-Supported Cooperative Work. IEEE Computer Society Press, Los Alamitos, 1992, S. 11–22.
2. Fricke, R.: *Lotus Notes* / *Domino* R5. bhv Verlags GmbH, Kaarst, 1999.
3. Hase, M.: Bringt der Rabe Glück? In: INFORMATIONWEEK (2000) 21, S. 20–27.
4. Bundesamt für Informatik: *HERMES* - Führung und Abwicklung von Informatikprojekten (Ausgabe 1995). EDMZ, Bern, 1996 (Bezug über http://www.edmz.ch).
5. Bundesamt für Informatik: *HERMES SWE* - Vorgehensmodell für die Software-Entwicklung (Ausgabe 1996). EDMZ, Bern, 1996 (Bezug über http://www.edmz.ch).
6. Myrach, T.: Die Rolle von CASE-Werkzeugen im Ausbildungskonzept für Wirtschaftsinformatik an der Universität Bern. In: Spillner, A., Breymann, U. (Hrsg.): Software Engineering im Unterricht der Hochschulen / SEUH ’95. B. G. Teubner, Stuttgart, 1995, S. 48–58.
7. Teufel, S. et al.: Computerunterstützung für die Gruppenarbeit. Addison-Wesley, Bonn, 1995.

Can you Java? Ein Erfahrungsbericht

Ulrike Jaeger
Studiengang Software Engineering
FH Heilbronn
ulrike.jaeger@fh-heilbronn.de
http://www.se.fh-heilbronn.de/~jaeger

Zusammenfassung

Dieser Erfahrungsbericht schildert Erfahrungen mit der Grundausbildung in Programmiersprachen. Soweit es ging, konnten Studenten aktiv lernen. Auch die Leistungskontrolle sollte einen aktiven Charakter bekommen: gleichzeitig von den Studenten aktiv erlebt und gestaltet werden, ihnen aber auch Feedback geben und am Ende benotet werden. Dabei erwies sich die klassische Klausur als ungeeignet. Der Beitrag schildert die ersten Schritte in Richtung aktiv erlebter Feedback-Veranstaltungen.

1 Programmiersprachen im Studiengang Software Engineering

Der Studiengang Software Engineering an der FH Heilbronn wurde im Wintersemester 1996 gegründet. Er ist bewußt als praxisorientiertes Fachhochschulstudium mit Betonung auf ingenieursmäßige Herangehensweise konzipiert. Objektorientierung, Teamarbeit und projektbezogenes Denken werden in vielen Fächern von Anfang an vermittelt. Das Curriculum für Programmierung bietet in den ersten beiden Semestern jeweils 4+2 SWS "*Programmiersprache I*", die am Ende gemeinsam in einer Klausur geprüft werden. Im Hauptstudium gibt es eine 4-stündige Vertiefungsvorlesung "*Programmiersprache II*".

1.1 Erste Erfahrungen

1996 bis 1998 wurde im ersten Jahr C++ als Einführungssprache gewählt. Die Vertiefung im Hauptstudium bot dann eine Java-Vorlesung nach [Eckel00].

C++ erwies sich als zu speziell und umfangreich für eine Einführung in die Programmierung. Versuche, ein weitere, nicht kommerzielle Sprache [Pemberton87] parallel zur Relativierung mit zu unterrichten, verwirrte die Studenten. Besser erwies sich ein an Modula angelehnter Pseudocode [Appelrath95]. Die vielbeschworene Praxisrelevanz von C++ traf zumindest für unsere Industriepraktika nicht zu: kaum ein Student arbeitete an Projekten in C++.

Nach dem Praxissemester war der Wiedereinstieg sehr mühsam. Die C++ Kenntnisse waren schon wieder halb vergessen, die Java Vorlesung baute aber nach

[Eckel00] darauf auf. Statt das vergessene C++ als Referenz für alles, was in Java anders ist zu verwenden, beschlossen wir, die Reihenfolge umzudrehen. Im Wintersemester 1999/2000 starteten wir mit Java als erste Programmiersprache.

1.2 Anpassung der Lehrinhalte

Durch den Wechsel haben sich einige Vorlesungen verändert. Die Grundvorlesung mit ihren zweimal 4+2 Stunden basierte nun auf Java. Im 4. Semester gibt es eine zweistündige Vorlesung "Programmiersprachen II", die hauptsächlich C++, aber auch andere Sprachen streift, sowie eine eigene Vorlesung zum Thema "Entwicklungsumgebungen". Das schafft die Möglichkeit, sich in den ersten beiden Semestern ganz auf Java in einfachster Umgebung zu konzentrieren.

2 Aktives Lernen

Wir wollten in dieser ersten Programmiervorlesung keine Sprach-Spezialisten ausbilden, sondern Java lediglich als Vehikel zum Verständnis von Modellier- und Programmierprinzipien benutzen.

2.1 "Java Day"

Im ersten Semester wurden über die Woche verteilt seminaristische Vorlesungen mit Übungsterminen am Rechner kombiniert (4+2). Im zweiten Semester gab es stattdessen einen sogenannten *„Java Day"* pro Woche, an dem in sich lockerer Folge Theorie und Übungen am Rechner abwechselten. Das ermöglichte uns, Fragen, Varianten und Lösungsideen der Studenten sofort praktisch zu untersuchen.

2.2 Dokumentendatenbank

Zur Verbreitung von Material, Folien, Aufgaben und Lösungen wurde eine *Lotus Notes™ Domino* Dokumentendatenbank eingerichtet. Sie bot vollständige Information, leichte Verwaltung und geeignete Stichwortsuche. Im Gegensatz zu früheren Modellen über ftp-Server waren die Studenten dieses Mal mit der Bereitstellung von Unterlagen sehr zufrieden. Die Bedienung war attraktiv, die Stichwortsuche wurde eifrig genutzt. Einzig die Verfügbarkeit des Servers litt unter der Last der Zugriffe.

2.3 Interaktives Programmieren

Für die Grundlagenvorlesung suchten wir Literatur, die weder andere Sprachkenntnisse voraussetzt, noch einen Crashkurs für die legendären 21 Tage bietet, und nicht nur eine reine Beschreibung der Sprache Java ist. Wir entschieden uns für den Ansatz von Lynn Andrea Stein [Stein00].

Das Konzept dieses Buches geht von Interaktiver Programmierung aus. Es baut die Vorstellung vom Programmieren auf dem Paradigma des gegenseitigen Vertrags

von Teilnehmern auf. Solche Teilnehmer können Objekte sein, auf tieferen Ebenen dann Methoden mit ihren Schnittstellen. Das Denken in Vereinbarungen, Garantien und Ausnahmen läßt sich aus alltäglichen Situationen ableiten und führt schnell auf Interaktionsmodelle und Protokolle hin.

Zusätzlich wurde [Bishop99] empfohlen und im zweiten Semester auch [Eckel00] und [Gamma96] für die Diskussion von *Design Patterns*. Alle Studenten wurden ermutigt, für sich persönlich weitere Bücher zu prüfen.

Die Programmierung wurde eng mit der gleichzeitigen vierstündigen Vorlesung "*Grundlagen der Informatik*" verzahnt. Im ersten Semester geht es um den Algorithmusbegriff und Modularisierung. Im zweiten Semester werden dann Interaktion, Ereignisse mit Verteilten Systemen gekoppelt.

Wir benutzten keine komfortable Entwicklungsumgebung. Zur Modellierung wurde zwar die *UML*-Notation verwendet, aber ebenso oft intuitive Skizzen mit Strichmännchen und Eigenkreationen der Studenten. Wir wollten die Studenten vorerst an das Notieren selbst gewöhnen, aber noch nicht auf eine bestimmte Notation einschwören.

Schon in den ersten Wochen wurden die Studenten mit den *JDK Packages* von Sun™ konfrontiert. Lediglich für komfortables I/O haben wir die üblichen Vereinfachungen für die Studenten angeboten, alles andere war Original *JDK1.2*. Das anfangs mühsame Lesen und Verstehen von Dokumentationen wurde einfacher, als mitten im Semester auch die Folien vollständig auf Englisch umgestellt wurden. Offensichtlich wurde danach eher in Englisch "gedacht".

2.4 Didaktische Konzepte

Angeregt durch einen Gastaufenthalt an der Universität von Amsterdam, die bei ihren Medizin-Informatikern das *Problem Based Learning PBL* [PBL97] einsetzt, wollten wir die Studenten so aktiv wie möglich beteiligen. Persönlich Erlebtes wird eher verstanden und besser behalten als passiv Gehörtes. *PBL* geht außerdem davon aus, daß die Studenten sehr wohl in der Lage sind, Lösungen und Antworten selbst zu erarbeiten. Obwohl wir mit 45 Studenten deutlich überbelegt waren, versuchten wir, die Studenten zum Diskutieren, Erklären und Ausprobieren zu ermutigen. Grundsätzlich wurden die Übungsstunden aufgeteilt, so daß es zwei Großgruppen im Programmierpraktikum gab. Zusätzlich bildeten sich Kleingruppen von 3-4 Teilnehmern, die Aufgaben und den Stoff während der Vorlesungen bearbeiteten, im Plenum vortrugen und diskutierten. Ganz nebenbei wurde damit auch das Arbeiten im Team, Vortragen und Moderieren geübt.

Im zweiten Semester wurden die Kleingruppen strukturiert. Die Teilnehmer hatten (wechselnde!) Rollen:

- Mediator:
  - vertritt die Gruppe nach außen
  - spricht Schnittstellen mit anderen Gruppen ab

- entscheidet bei Konflikten innerhalb der Gruppe

- Designer:
  - prüft, ob es bereits fertige Teillösungen gibt
  - entwirft die Struktur der Lösung
  - entwirft die Struktur der Lösung

- Hacker:
  - testet fragliche Varianten
  - zeigt generelle Machbarkeit von Einzelproblemen

Die Vorlesungen waren nicht immer für das gesamte Plenum gedacht. Es gab Sonderveranstaltungen nur für Designer (z.B. *Observer Design Pattern*) oder nur für Hacker (z.B. *Reflection* in Java). Diese Spezialisten hatten dann die Aufgabe, das Gelernte den anderen Gruppenmitgliedern zu erklären.

2.5 Ein Beispiel

Über das zweite Semester hin wurde u.a. ein Taschenrechner entwickelt. Dazu benötigten sie die sehr hilfreiche Diskussion in [Stein00] über Entscheidungen, die von Verzweigungen bis zur Modellierung von entsprechenden Objekten führt. Als Informatik-Grundlagen boten wir die *Stack*-Datenstruktur für Argumente und Zwischenergebnisse, und Zustandsautomaten mit Ein- und Ausgaben.

In Kleingruppen arbeiteten die Studenten an immer anspruchsvoller werdenden Fassungen des Taschenrechners:

1. Fassung: ohne GUI, der arithmetische Ausdruck wird einfach als String von der Konsole gelesen. Einige Gruppen implementieren das Einlesen und Interpretieren des Ausdrucks (*IO-Interface*), andere Gruppen implementieren die Arithmetik selbst.

2. Fassung: einige Gruppen stecken ein *GUI* an das *IO-Interface*, dabei nutzen sie das Adapter Design Pattern.

3. Fassung: Schnittstellenanpassung, so daß die *GUI* und Arithmetikteile der einzelnen Gruppen beliebig gegeneinander austauschbar werden.

4. Fassung: Erweiterung des Taschenrechners um weitere Funktionen.

Nach der zweiten Fassung erarbeiteten die meisten Gruppen ein gründliches Redesign ihrer Lösung. Sie wußten wieviel Arbeit es macht, jedesmal von vorn anzufangen. Sie identifizierten logische Einheiten, die unverändert bleiben konnten, und begannen ernsthaft, sich mit den anderen Gruppen über die Schnittstellen zu unterhalten.

3 Feedback und Prüfen

Über den langen Zeitraum eines Jahres ging es uns zwar auch um handwerkliche Fähigkeiten in Java, aber vor allem um eine Erziehung zum objektorientierten und interaktiven Denken, das sich prinzipiell auf andere Sprachen und Probleme übertragen läßt. Es fragte sich, wie wir diese "Erziehung" prüfen konnten.

3.1 Aktives Feedback

Im ersten Semester wurde ein unbenoteter Schein aus drei kleineren Aufgaben gebildet:

- individuell: schriftliche Ausarbeitung zum Thema: Namen, Typen und Schnittstellen
- individuell: *online* Programmieraufgabe
- Kleingruppenarbeit: Erarbeiten von Prüfungsfragen, Musterlösungen und Begründung für die Frage (z.B. Was wird überprüft? Was soll gelernt werden?) und Präsentation im Plenum. Diese Aufgabe wurde einheitlich für die ganze Gruppe beurteilt.

Im zweiten Semester wünschten sich die Studenten zur eigenen Leistungskontrolle nun freiwillige Aufgaben ähnlich dem ersten Semester. Alle Aufgaben der Übungen waren Gruppenaufgaben, alle Musterlösungen stammten ausschließlich von den Studenten selbst.

Zur Klausurvorbereitung am Ende gab es wieder auf vielfachen Wunsch eine Session mit von den Studenten erarbeiteten und diskutierten Prüfungsfragen. Dieses Mal waren die Fragen an die spezielle Situation einer schriftlichen Klausur angepaßt. Es gab also sehr kleine Programmieraufgaben zur Analyse oder zum Ergänzen, *Multiple Choice* Fragen und Verständnisfragen. Die Studenten waren sehr engagiert und nutzten die Session zur intensiven Wiederholung und Diskussion. Irrtümer wurden selbständig korrigiert. Die Dozentin hatte wie schon im ersten Semester nur sehr selten etwas zu ergänzen.

3.2 Klausur

Am Ende des zweiten Semesters wurde der gesamte Stoff in einer schriftlichen Klausur geprüft. Hier zeigte sich ein ernstes Problem: während des gesamten Jahres waren die Studenten gewohnt, sich die Informationen aus dem Netz zusammenzusuchen. In der Klausur gab es plötzlich nur Bücher und Skripten als Hilfsmittel. Die Programmieraufgaben mußten nun auf einfache Beispiele reduziert werden, alle notwendigen Klassen mußten als lange Texte auf Papier zur Verfügung gestellt werden. Dieser Bruch im Denken und Handeln wurde von allen sehr beklagt.

Als Alternative wäre eine *online* Klausur denkbar gewesen. Es gibt aber in unserem Haus keinen hinreichend großen Rechnerraum. Eine Aufspaltung in Gruppen

hätte unvergleichbare Nebenbedingungen geschaffen, die nicht benotet werden können.

4 Ausblick

4.1 Nächste Schritte

Für die Zukunft haben wir deshalb einige Veränderungen geplant:

- Verwendung eines *Lotus Notes™ Learning Space*, der freiwillige Leistungskontrolle, Feedback durch halbautomatische Tests besser unterstützt, Eigenbeteiligung der Studenten im Learning Space attraktiver macht
- Keine Klausur mehr, nur noch vorlesungsbegleitende, unbenotete Scheine
- Für spezielle, fachübergreifende Aufgaben: *Board of Advisors*: Dozenten anderer Fächer fungieren als Berater für bestimmte Fragestellungen. Beispiele:
  "Verteilte Systeme" erklärt Protokolle für die *thread*-Programmierung
  "Grundlagen des Software-Engineering" berät bei Notationen und Analyse
  "Organisations-Psychologie" berät bei Gruppenproblemen

Um die Studierenden eher auf die objektorientierte, verteilte Anwendungswelt vorzubereiten, werden wir in Zukunft bereits im ersten Semester die folgenden Prinzipien beachten:

- Verteiltes Programmieren von Anfang an
- Design Patterns von Anfang an
- Zweigleisige Aufgaben: Gleichzeitig Top Down und Bottom Up Aufgaben stellen, z.B. Bausteine der Programmiersprache einüben, aber auch GUI-unterstützte Anwendungen zum Konfigurieren anbieten (nach [Stein00]).

4.2 Erkenntnisse

Der Umstieg auf Java war eine Gelegenheit, die Vorlesung neu zu konzipieren. Die positive Resonanz liegt nicht nur an der deutlich klareren Sprache Java selbst, sondern auch an den begleitenden Maßnahmen.

Der Stoff wird weniger im Frontalunterricht vermittelt, sondern anhand von konkreten Aufgaben durch Fragen der Studenten erschlossen. Dieses didaktische Prinzip ist seit längerem allgemein anerkannt, aber die Umsetzung scheitert allzuoft an den Realitäten einer Hochschule. Bei uns zum Beispiel war es sehr schwierig, sowohl einen Hörsaal als auch einen Rechnerpool für den Javatag im zweiten Semester freizuhalten.

Die Veranstaltung war das gesamte Jahr über von Neugierde und Experimentierfreude der Studenten geprägt. Scheinaufgaben, Gruppenarbeit und Organisation wurden einhellig gelobt. Die vorgeschriebene Klausur war dann der Einbruch der trockenen Hochschulwirklichkeit in ein lebendiges Experiment. Zwar bestanden fast alle Stu-

denten, aber die Noten blieben im mittleren Bereich. Es fragt sich, ob diese Noten der wirklichen Lernleistung entsprechen Die Leistung sollte auf dieselbe Art überprüft werden wie der Stoff bewältigt wird: lebendig, kreativ und diskussionsfreudig. Wir plädieren deshalb für unbenotete Scheine, deren Bestandteile während der Veranstaltung entstehen.

Andere Hochschulen können sicher Ähnliches berichten, wenn sie aktives Lernen verwirklichen wollen. Seit vielen Jahren wisssen wir alle, was man besser machen könnte, aber allein die angespannte Personal- und Raumsituation genügt schon, um gute Ansätze im Keim zu ersticken.

Literatur

[Appelrath95] Hans-Jürgen Appelrath, Jochen Ludewig: Skripum Informatik — eine konventionelle Einführung. Teubner, 1995.

[Bishop99] Judy Bishop: Java lernen. Anfangen, Anwenden, Verstehen. Addison Wesley, 1999.

[Eckel00] Bruce Eckel: Thinking in Java, Prentice Hall, 2nd Edition, 2000.

[Gamma96] Erich Gamma: Design Patterns, Addison Wesley, 1996.

[PBL97] PBL Conference 97: Integrity, Innovation, Integration, 3 - 6 December 1997, Australian Catholic University, Macauley Campus, Brisbane.

[Pemberton87] Steven Pemberton: An Alternative Simple Language and Environment for PC's, IEEE Software, Vol. 4, No. 1, Januar 1987, pp 56-64.

[Stein00] Lynn Andrea Stein: Interactive Programming in Java, Morgan Kaufman, to appear (http://www.mkp.com/ipij/)

Die Rolle der Reflexion in Softwarepraktika

Claus Lewerentz, Heinrich Rust[*]
Lehrstuhl Software-Systemtechnik, BTU Cottbus

Zusammenfassung

Softwarepraktika dienen in vielen Informatikstudiengängen zur Einübung von Softwaretechnikfertigkeiten. Weil die spezifischen Techniken in der Softwaretechnik rasch veralten, ist es wichtig, dass die Studierenden lernen, ihre eigene Arbeitsweise immer wieder zu verbessern und neu vorgeschlagene spezifische Techniken dahingehend zu bewerten, ob sie in ihrer Arbeitssituation hilfreich sind. Es wird eine zyklische Organisationsform für ein Softwarepraktikum vorgestellt, in dem die Studierenden immer wieder zur Selbstbeobachtung und zur Reflexion der von ihnen gemachten Erfahrungen angehalten werden. Diese wiederholten expliziten Reflexionen münden in Konsequenzen, die im jeweils folgenden Zyklus in der Praxis erprobt werden und am Ende des Zyklusses sowie am Ende des Gesamtprojektes bewertet werden können.

1 Aufgaben und Formen eines Softwarepraktikums

Softwarepraktika sind in unterschiedlicher Form und Dauer Bestandteil verschiedener Studiengänge mit Softwaretechnik-Anteilen. Solchen Softwarepraktika liegen unter anderem folgende Ausbildungsanliegen zugrunde: (1) Die typischerweise zuvor in Vorlesungen gelernten Einzeltechniken werden im Zusammenhang angewendet. (2) Sogenannte „soft skills" werden eingeübt. (3) Studierende erfahren, dass es in der softwaretechnischen Praxis nicht ohne Dilemmata und, in der Folge, ohne Kompromisse abgeht. (4) Der Transfer von softwaretechnischem Wissen in die Praxis wird erleichtert, wenn in man einen Zwischenschritt einschiebt zwischen die (oft bloß rezipierte und in ihrer Relevanz nicht praktisch erfahrene) Theorie und die (oft nicht systematisch reflektierte) Praxis.

An der Brandenburgischen Technischen Universität in Cottbus muß im Grundstudium der Studiengänge Informatik bzw. Informations- und Medientechnik ein Softwarepraktikum absolviert werden. Dieser Artikel beschreibt, wie wir in Cottbus diese Praktika gestalten.

[*] BTU Cottbus, Postfach 10 13 44, D-03013 Cottbus, Deutschland;
Tel.: +49 (355) 69-38 81, Fax: -38 10; (lewerentz, rust)@informatik.tu-cottbus.de

Unser Ansatz beruht auf einer einfachen Kernidee, die wichtige Anteile an der Verantwortung für den Lernerfolg den Studierenden zuweist: Es ist die Aufgabe der Dozenten, für die Studierenden Gelegenheiten zu generieren, Erfahrungen zu machen; und es ist die Aufgabe der Studierenden, ihre Erfahrungen explizit zu machen, sie zu interpretieren und Konsequenzen daraus zu ziehen, oder kurz: sie zu reflektieren. Reflexion ist ein wesentliches Element in unserem Ansatz der Gestaltung eines Softwarepraktikums im Grundstudium, und dieses Element sehen wir auch als ein wichtiges Hilfsmittel für das Lernen in der späteren beruflichen Praxis.

Wir sind im Laufe der Zeit zu einem zunehmend detaillierten Vorgehensmodell gekommen, das unter anderem folgende Elemente enthält: (1) Es wird verlangt, dass das Projekt eine zyklische Struktur hat, wobei ein wasserfallartiger Prozess drei bis vier Male durchlaufen wird. (2) Wir verlangen von den Studierenden detaillierte Protokolle der Aufwände für ihre praktikumsspezifischen Tätigkeiten, um Planungen und wirkliche Aufwände vergleichen zu können und Konsequenzen für kommende Planungsarbeiten zu ziehen. (3) Wir verlangen zum Abschluss jedes Teilprojekts und zum Abschluss des Gesamtprojekts Reflexionspapiere. Diese enthalten zum einen Vergleiche von geplanten und tatsächlichen Aufwänden für das betrachtete Projekt und zum anderen persönliche Beurteilungen des Ablaufes des reflektierten Teilprojektes, wobei auch qualitative Einschätzungen eingehen dürfen. Wir erwarten für die verschiedenen reflektierten Punkte die Ableitung von Verhaltenskonsequenzen für die Folgeprojekte. (4) Jede Gruppe präsentiert ihrem Betreuer wöchentlich ihren Projektstand. Jedes dieser vier Elemente unterstützt die Reflexion.

2 Aufbau des Cottbuser Softwarepraktikums

Der „Personal Software Process" und der „Team Software Process" (PSP/TSP, [3,4]), und geringerem Maße auch der Ansatz des „Extreme Programming" (XP [1,2]), der Ansatz von Schön zum reflektierten Handeln [6,7,8] und Paschs Ideen zum Softwareentwurf im Team [5] haben uns bei der Ausgestaltung des Cottbuser Softwarepraktikums inspiriert. Schließlich sind neben den schon angedeuteten Erfahrungen in früheren Durchläufen des Praktikums Erfahrungen aus der industriellen Praxis eingeflossen: Hier werden neue Mitarbeiter nicht selten in ihr erstes Projekt geworfen wie in kaltes Wasser, unterstützt allein durch den Ratschlag: Schwimme! Wir wollen unseren Studierenden Fähigkeiten mitgeben, mit einer solchen Situation so produktiv umzugehen wie es eben möglich ist. Sie sollen ihre Arbeit so organisieren können, dass die Auswirkungen von Fehlern begrenzt werden und aus früheren Fehlern Konsequenzen für die späteren Projektphasen gezogen werden können. Ein zyklischer Projektablauf mit expliziten Reflexionselementen ist hierfür der angemessenste Rahmen.

Die für unseren Ansatz spezifischen Lernziele sind:

- Eigene Erfahrungen explizieren, interpretieren und für Folgetätigkeiten nutzen: Die eigenen Erfahrungen sind deshalb in der Softwaretechnik besonders wichtig, weil sie nicht so schnell wie das technische Lehrbuchwissen veralten – zumindest solange die Studierenden bereit sind, zuvor gemachte Erfahrungen auch in Frage zu stellen und neue Erfahrungen zu akzeptieren. Im Softwarepraktikum versuchen wir, den Studierenden den Wert ihrer eigenen (wenn auch im Lehrumfeld nur beschränkten) Erfahrungen deutlich zu machen. Wir versuchen dies, indem wir verlangen, dass die Praktikumsteilnehmer ihre Erfahrungen zum Teil explizit und damit für andere nachvollziehbar machen, dass sie sie interpretieren, und dass sie Konsequenzen daraus ziehen.
- Ein Projekt lernfreundlich organisieren: Wir bemühen uns, im Softwarepraktikum das Schlagwort „das Lernen lernen" mit einem präzisen Sinn zu versehen. Wir vermitteln den Praktikumsteilnehmern, welchen Wert es hat, wenn sie ihre Arbeitsprozesse so organisieren, dass sie in früheren Phasen Erfahrungen machen und diese später anwenden können. So versuchen wir, sie mit der Fähigkeit auszustatten, den eigenen Arbeitsprozess lernfreundlich zu organisieren.

Um anschaulich zu machen, mit Hilfe welcher Art von Aufgabenstellungen wir die angedeuteten Ziele zu erreichen versuchen, skizzieren wir zwei Beispiele, die wir im Sommersemester 2000 verwendet haben. Die erste Aufgabe betrifft die Entwicklung eines Klassenhierarchie-Browsers, wobei Symboltabellendaten aus einer Datenbank entnommen und mittels einer grafischen Benutzeroberfläche dargestellt werden sollten. Sie wurde von drei Gruppen mit je vier Personen bearbeitet. Die Aufwände bewegten sich insgesamt im Bereich von ca. 430 bis 480 Stunden, pro Person von etwa 107 bis 120 Stunden. Die zweite Aufgabenstellung betrifft die Programmierung eines kleinen mobilen Roboters, der mit einer Anzahl von optischen Abstandssensoren ausgestattet ist. Es ist ein Labyrinth zu vermessen, es sind Hindernisse in einem abgegrenzten Gebiet zu lokalisieren, und es ist ein gefahrener Weg zurückzuverfolgen. Diese Aufgabe hat sich auch in anderen Jahren bewährt; sie wurde zum wiederholten Male vergeben. Die Aufwände bewegten sich hier von 352 Stunden bei drei Personen bis 889 Stunden bei vier Personen, also von ca. 117 bis ca. 222 Stunden pro Person. Die letzte Zahl ist allerdings ungewöhnlich groß.

Insgesamt scheint ein Aufwand von etwa acht bis zehn Semesterwochenstunden für das Softwarepraktikum typisch zu sein. Dieser Umfang ist auch für diese Veranstaltung vorgegeben.

Wir geben den Studierenden vor, dass sie im Softwarepraktikum einen zyklischen Entwicklungsprozess benutzen müssen. Wir verlangen, dass diese einzelnen Teilprojekte möglichst vollständig sind, dass sie also alle Tätigkeiten enthalten, die in einem Wasserfallmodell vorkommen.

Die in den Teilprojekten gesammelten Arbeitsergebnisse sind jeweils adäquat zu dokumentieren, wie es die Studierenden in früheren Vorlesungen gelernt haben. Allein eine explizite Reflexionsphase, die jedes Teilprojekt abschließt, ist den Studie-

renden aus den vorbereitenden Vorlesungen nicht vertraut. Mit diesen Dokumenten weisen die Studierenden den Projektverlauf und ihren Lernfortschritt nach.

Die Planung ist zuweilen ein Stiefkind der vorbereitenden Vorlesungen, wohl mit Recht, weil Planungsfähigkeiten so schwer rein theoretisch zu vermitteln sind und nicht im Rahmen einer isolierten Hausaufgabe eingeübt werden können. Allerdings sind die Gegenüberstellungen von Planungen und wirklichem Projektverlauf eine der besten Reflexionsgelegenheiten. Daher wollen wir genauer erörtern, worauf es uns im Softwarepraktikum bei der Planung ankommt.

2.1 Vorgehen bei der Planung und Projektverfolgung

Die Planung wird aufgeteilt in Aufwands- und Ressourcenplanung. Die Aufwandsplanung enthält eine Aufteilung der in einem Teilprojekt zu leistenden Arbeit in Arbeitspakete und eine Aufteilung der in jedem Arbeispaket zu leistenden Arbeit in Teilaufgaben, die einzelnen Mitgliedern zugewiesen werden und für die die aufzuwendenden Ressourcen (hier: Arbeitszeiten) geschätzt werden sollen. Zusammen mit den Angaben der Ressourcenplanung werden daraus geplante Abschlussdaten für die Arbeitspakete festgelegt.

Die Ressourcenplanung enthält für jedes Gruppenmitglied eine Planung der aufwendbaren Arbeitsstunden über die nächsten Wochen der Projektlaufzeit. Typischerweise vergessen die Studierenden hier zunächst, die Verminderung der von ihnen aufwendbaren Zeit einzuplanen, die durch Klausurphasen und Feiertage entstehen. Dies wird ihnen aber bewusst, wenn aus solchen Gründen Verzögerungen auftreten.

Die erste Aufwands- und Ressourcenplanung des Teams kann nur „über den Daumen“ erfolgen, weil zunächst keine auswertbaren Erfahrungen vorliegen. Bei späteren Planungen sollten dann zuvor gemachte Erfahrungen einfließen. Abbildung 1 enthält ein Beispiel für die Auswertung von Plan und tatsächlichem Ablauf in einer Praktikumsgruppe, wie sie einem Reflexionspapier entnommen ist. Die erste Spalte enthält die Kurzbeschreibung des Arbeitspaketes und der darin enthaltenen Teilaufgaben. Die zweite sind Nummern; so heißt „K2.3.1”: Es handelt sich um das zweite Teilprojekt, darin das dritte Arbeitspaket, und darin wiederum die erste Teilaufgabe. Die letzten beiden Spalten stellen geplante und tatsächliche Aufwände für die Teilaufgaben und das gesamte Arbeitspaket einander gegenüber. Die Daten in der Kopfzeile sind geplante und tatsächliche Abschlussdaten des Arbeitspaketes. Die Ergänzung „(nf)” (für „nicht fertig”) gibt an, dass das Paket nicht wirklich abgeschlossen wurde.

Ein hinreichender Detaillierungsgrad ist bei der Aufwandsplanung nötig, damit man (1) überhaupt eine Chance hat, im Vorhinein zu beurteilen, ob der Plan realistisch ist; damit man (2) den Projektfortschritt während des Projektablaufs verfolgen kann, denn bei einer allzu geringen Zahl von Arbeitspaketen gibt es schlicht nicht genügend Abschlusstermine, deren Nichteinhaltung eine Verzögerung gegenüber dem Plan nachweist; und damit man (3) die Verantwortung für Teilaufgaben einzel-

nen Gruppenmitgliedern zuordnen kann. Im Laufe der Teilprojekte lernen die Studenten diese Konsequenzen aus der Gestalt eines guten Aufwandsplans zu schätzen.

		Planung	Tatsächlich
Theoretisch Algorithmen entwickeln	**K 2.3**	**29.05**	**29.05 (nf)**
Algorithmus 1. Teilaufgabe	K 2.3.1	10 h	9:00
Algorithmus 2. Teilaufgabe	K 2.3.2	6 h	5:00
Algorithmus 3. Teilaufgabe	K 2.3.3	6 h	6:00
Pseudo-Code für alle Teile mit abstrakten Roboterbefehlen	K 2.3.4	8 h	0:00
		= 30 h	= 20:00

Das Paket wurde fristgerecht abgeschlossen, ohne dass der geforderte Pseudocode angefertigt worden ist. Das wurde ins Paket 2.6.1 verschoben. Die Unvollständigkeit dieses Pakets ist begründet durch einen zu hohen Ansatz der möglichen Ressourcen (Zeit). Es wurde nicht beachtet, dass die dieses Paket ausführenden Gruppenmitglieder in dieser Zeit Klausuren hatten und somit die fristgerechte Erfüllung dieses Pakets nicht möglich war. Da die folgenden Pakete den Abschluss dieses Pakets erforderten, haben wir den unerfüllten Teil in ein späteres Paket hineingelegt.

Abb. 1: Gegenüberstellung von Planung und wirklichem Aufwand

Die Studierenden lernen, dass es hilfreich ist, die geplanten Teilaufgaben einzelnen Gruppenmitgliedern oder wenigstens Paaren zuzuweisen, weil in vielen Teams allein schon das Unterlassen der Dokumentation von Aufgabenzuweisungen an Teammitglieder zu Missverständnissen führt. Typischerweise bewährt es sich zudem nicht, wenn für eine Aufgabe für alle Gruppenmitglieder der gleiche Aufwand eingeplant wird, weil auch in diesem Fall auch eine klare Verantwortungszuweisung unterbleibt.

2.2 Projektverfolgung

Ebenso wie das Planen ist die Verfolgung des Projektfortschritts ein notorisches Problem, weil in Softwareentwicklungsteams allzu oft der Wunsch die realistische Einschätzung trübt. Dabei gehört eine realistische Einschätzung des Stands im Projekt zu den wichtigsten Elementen erfolgreicher Softwareprojekte. Die Studierenden erkennen, dass es hierzu wichtig ist, von allen Gruppenmitgliedern regelmäßig Rückmeldungen über Arbeitsergebnisse abzufordern, und sie lernen so, den Wert von kleinen, klar von Folgearbeiten abgrenzbaren Arbeitspaketen zu schätzen, weil sich nur solche Arbeitspakete für die zuverlässige Verfolgung des Projektfortschritts eignen. So lernen sie, den Wert zahlreicher und aussagekräftiger Meilensteine für das Wissen über den Fortschritt im Projekt zu schätzen. Abbildung 2 gibt einen entsprechenden Auszug aus einem Reflexionspapier.

Die Projektverfolgung ist auf zwei Abstraktionsebenen möglich: (1) Die zyklische Projektstruktur sorgt dafür, dass es Meilensteine am Ende jedes Teilprojektes

gibt, und (2) innerhalb jedes Zyklusses werden Arbeitspakete definiert, die wieder jeweils einen eindeutig geplanten Endtermin haben. Hierdurch wird der Projektfortschritt innerhalb eines Teilprojektes verfolgbar.

Ein weiteres Problem in unserer Planung waren „fehlende Ergebnisse". D.h. es gab Teilaufgaben, an deren Ende kein konkretes Ergebnis gefordert war. So sollten selbständig Klassen entwickelt werden. Da die Umsetzung erst später erfolgen sollte, dachte auch keiner vorher an eine ausführliche Ausarbeitung der Klassen und der damit verbundenen Schnittstellen. Es musste so viel Arbeit nachgeholt werden, die eigentlich schon fertig sein sollte. Für unseren Gesamtplan war dies natürlich schlecht, da sich vieles verschob. Ergo
→ in Zukunft nur Meilensteine mit Vorzeigbarem planen.

Abb.2: Erkenntnis des Wertes klarer Abschlusskriterien von Arbeitspaketen

2.3 Prozessverbesserung durch Reflexion

Die Studierenden müssen in unserem Softwarepraktikum viele Aspekte ihres Projektes dokumentieren. Wir machen den Studierenden deutlich, dass diese Dokumentation der Nachweis darüber ist, dass sie während des Praktikums gelernt haben, ein mäßig umfangreiches Softwareentwicklungsprojekt im Team zu planen und durchzuführen und dabei wichtige Daten zu erheben, diese auszuwerten und Konsequenzen aus den gemachten Erfahrungen zu ziehen.

Das anzufertigende Produkt ist dabei von geringerer Bedeutung; es ist allerdings insofern wichtig, als eine allzu flagrante Verletzung der ursprünglichen Anforderungen, womöglich noch ohne rechtzeitige Erkenntnis, dass die ursprüngliche Aufgabenstellung nicht einzuhalten ist, ein Hinweis darauf ist, dass die Gruppe noch nicht verstanden hat, worauf es bei einem mäßig umfangreichen Softwareentwicklungsprojekt ankommt.

Wir legen Wert darauf, dass die Studierenden lernen, ihren Softwareentwicklungsprozess lernfreundlich zu organisieren. In unserem Ansatz ist ein wesentliches Element die Reflexion von Erfahrungen aus früheren Zyklen mit dem Ziel zur Verbesserung der zu benutzenden Prozesse in den späteren Zyklen. Grundlagen der Reflexion können Messungen und Beobachtungen bei der Prozessausführung in verschiedenster Form sein. Im Cottbuser Softwarepraktikum nutzen wir die folgenden Elemente:

- **Zeitprotokolle** der Gruppenmitglieder, insbesondere als Grundlage für den Vergleich von Plänen und tatsächlichem Ablauf. Es hat sich gezeigt, dass dies eine der fruchtbarsten Reflexionsgelegenheiten war. Dies hat wohl den Grund, dass die Gegenüberstellung von stark voneinander abweichenden harten Zahlen einen unabweisbaren Erklärungsbedarf anzeigt. Insbesondere schult dies die wichtigen Planungsfähigkeiten.

  Die Zeiterfassungen machen den Studierenden zudem bewusst, für welche Tätigkeiten im Rahmen der Softwareentwicklung welche Zeitanteile anfallen.

Dafür muss man aber natürlich die Aufwände entsprechend klassifizieren und auch quantitativ hinreichend exakt erfassen.

Die Studierenden dokumentieren ihre Reflexion bezüglich geplanter und wirklicher Aufwände pro Arbeitspaket und pro Person durch einen Reflexionsbericht am Ende jedes Teilprojektes und am Ende des Gesamtprojektes. Dieser enthält einen Vergleich von geplanten und tatsächlichen Aufwänden pro Arbeitspaket, und von geplanten und tatsächlich aufgewendeten Ressourcen pro Person, jeweils bestehend aus (a) Darstellung, (b) Interpretation und (c) der Angabe von daraus zu ziehenden Konsequenzen. Diese Reflexion betrifft die Genauigkeit der Aufwands- und Ressourcenplanung. Wir erwarten, dass die Gruppe sich über diese Punkte einen Konsens erarbeitet; dieser Teil des Reflexionspapiers ist also ein Gruppenergebnis. Hier werden harte Daten, nämlich die protokollierten Aufwandsdaten, interpretiert und Konsequenzen aus ihnen gezogen. Die schon erwähnte Abbildung 1 zeigt einen Auszug aus diesem Teil eines Reflexionspapiers.

- **Gruppenbesprechungen**. Diese sind eine zweifache Quelle von Reflexionsgelegenheiten: Einerseits werden gewisse Fragen direkt in den Gruppenbesprechungen reflektiert, andererseits dienen die Protokolle dem späteren Rückblick auf den Projektverlauf.

  Die Explizierung und Reflexion von Beobachtungen und Erkenntnissen, die in den wöchentlichen Gruppenbesprechungen zum Thema werden, liegt stark in der Hand der Betreuer. Hier kann die Aufmerksamkeit auf Probleme insbesondere der Kommunikation und der Kooperation in der Gruppe geleitet werden, die so gruppenspezifisch sind, dass sie im vorgegebenen Prozess nicht berücksichtigt sind.

  Wenn Ergebnisprotokolle der Gruppenbesprechungen sorgfältig geführt werden, so sind sie ein gutes Mittel, um bei der Abschlussreflexion festzustellen, wo man Lerngelegenheiten gut oder auch nicht gut genutzt hat, wenn man etwa feststellt, dass sich gewisse Probleme immer wieder wiederholen.
- Die Erarbeitung von **Coding-Styleguides und Review-Checklisten** und ihre Erprobung in der Praxis. Hier müssen sich die Studierenden gründlich über eher technische Fragen Gedanken machen und einen Konsens erzielen, und es wird deutlich, wie schwierig es ist, die wesentlichen Qualitätsanforderungen in eindeutigen Checklisten-Fragen zu formulieren.
- Zusätzlich zum gemeinsam zu erstellenden Reflexionsbericht, der die harten Daten auswertet, verlangen wir von jedem einzelnen Studierenden am Ende jedes Teilprojektes einen **persönlichen Reflexionsbericht**, in dem insbesondere subjektive und eher qualitative Erfahrungen darzustellen und auszuwerten sind.

Die Zweiteilung der an jedem Teilprojektende abzuliefernden Reflexionsberichte in einem gemeinsamen, auf harten Daten beruhenden Teil und einen persönlichen, auch „weiche“ Erfahrungen auswertenden Teil ermöglicht es den Studierenden, zum einen den Umgang mit harten Daten zu üben und den Wert der von ihnen gesammelten Aufwandsdaten schätzen zu lernen, und zum anderen aber auch eher intuitive Er-

kenntnisse festzuhalten und für das Vorgehen in Folgeprojekten zu nutzen. Die persönlichen Reflexionspapiere sichern zudem, dass jedes Gruppenmitglied sich ein paar Gedanken über den Prozessverlauf macht, und dass die Reflexionsarbeit nicht nur an einem oder zwei Gruppenmitgliedern hängenbleibt, was leicht geschieht, wenn nur ein Gruppenpapier verlangt wird.

Die sorgfältige Nutzung von selbstgemachten Erfahrungen ist nicht nur während der Ausbildung wichtig, sondern ist auch von eminenter Bedeutung für die Praxis, die gewöhnlich durch Turbulenz, also wechselnde Arbeitsbedingungen gekennzeichnet sind, besonders im expandierenden Beratungsgeschäft, wo kein Projekt ist wie das andere: (1) Die Zusammensetzung von Entwicklerteams zwischen Projekten und während Projekten wechselt. (2) Die Auftraggeber wechseln, und sie haben wechselnden Stärken und Schwächen bezüglich der Anforderungsdefinition. (3) Die Ansprechpartner beim gleichen Auftraggeber mit wechselnden Projektaufträgen wechseln. (4) Die zu verwendenden Werkzeuge wechseln durch Innovation.

2.4 Spezifische Techniken für Entwurf, Review und Test werden ausprobiert und ihre Wirksamkeit wird reflektiert

Die in Vorlesungen vorgestellten Techniken müssen von den Studierenden nicht einfach als wirksam akzeptiert werden, indem man der Autorität des Dozenten folgt. Vielmehr können die Leistungen der Techniken direkt erfahren und die für diese Wirkungen nötigen Vorbedingungen im Anschluss an eine Verwendung reflektiert werden. Abbildung 3 bietet ein Beispiel für einen Auszug aus einem Reflexionspapier, in dem deutlich wird, dass der Wert einer spezifischen Technik der Softwaretechnik, nämlich von Detaildesigns erkannt wurde.

> Besonders schlecht ist in diesem Teilprojekt meine persönliche Arbeit erfolgt. Die Implementierung der Teilanlage 3 wurde zweimal von mir abgebrochen und nahezu vollständig von vorn begonnen. Ursache ist sicherlich die mangelnde Planung, es reicht eben, wie ich erfahren durfte, nicht aus, sich zu einer Klasse wohlklingende Methoden als Schnittstelle auszudenken, solange man nicht genügend Zeit in die Erkennung von Zusammenhängen investiert.

Abb. 3: Einsicht in den Wert von Detaildesigns

Hierbei kann ein Problem darin bestehen, dass in einem Softwarepraktikum die entsprechenden Anwendungsbedingungen, etwa aufgrund noch fehlender Erfahrung der Studierenden mit einer Technik, nicht gegeben sind. Dies kann wohlgeprüfte und unter entsprechenden Bedingungen leistungsfähige Techniken, etwa Qualitätssicherungsmaßnahmen wie Reviews oder systematische Tests, in den Augen der Studierenden ungerechtfertigterweise diskreditieren. Dieses Problem können wir nicht lösen, wenn wir die eigenen Erfahrungen der Studierenden betonen, sondern können ggf. nur darauf hinweisen, daß die gemachten Erfahrungen möglicherweise nicht repräsentativ sind.

2.5 Koordinations- und Kommunikationsprobleme

In den (Teil-)Projektberichten wird erwartet, daß Koordinations- und Kommunikationsprobleme in der Gruppe dargestellt und reflektiert werden.

Viele Probleme in den Softwarepraktikumsprojekten hängen mit einem Mangel an Koordination zusammen: Die Gruppen treffen sich nicht regelmäßig, oder sie thematisieren nicht die jeweils relevanten Probleme. In verschiedenen Projektphasen haben solche Probleme typischerweise verschiedene Gestalt: (1) am Anfang, wenn sich die Gruppenmitglieder noch nicht kennen; (2) in Motivationstiefs, wenn kein rechter Fortschritt erkennbar ist; und (3) nach „Projektpausen“ (etwa Klausurzeiten, Feiertagen), wenn ein Projekt erst langsam wieder anläuft. Die Betreuer achten auf diese Probleme, machen sie explizit, versuchen, Konsens zu erzielen, und verlangen, dass sie im Ergebnisprotokoll festgehalten werden, damit die Erfahrung später im Rückblick ausgewertet werden kann.

Kommunikation

In den Abbildungen 4 und 5 finden sich Auszüge aus Reflexionspapieren mehrerer Mitglieder einer Gruppe, in der nach anfänglicher Dominanz des Entwurfsgeschehens durch eine Person ein ausgeglichenerer Entwurfsprozess hergestellt werden konnte. Im Auszug der Abbildung 4, der von einem Studenten mit relativ viel Entwicklungserfahrung stammt, wird auch deutlich, dass Entwurfshilfsmittel auch als wichtige Kommunikationsmittel erkannt wurden. Dass sich die Urteile in der Gruppe decken, belegt ein Reflexionsbericht eines anderen Gruppenmitglieds, von dem ein Auszug in Abbildung 5 wiedergegeben ist.

Anfangs gab es viele Schwierigkeiten, die durchaus gelegentlich in Verzweiflung resultierten. Kommunikation war zu diesem Zeitpunkt innerhalb der Gruppe eher eine Pflicht als eine Hilfe. Zusammenarbeit bestand im Wesentlichen darin, dass ich einen Vorschlag machte, keiner (wirklich) etwas dazu sagte und der Vorschlag schließlich angenommen wurde. Gelegentlich hatte ich das Gefühl, Diktator zu sein, der seinen Untergebenen nur sagt, was sie zu tun haben. Natürlich führte das dazu, dass die meisten Gruppenmitglieder keinen Überblick über das Projekt erhielten. Schließlich war die entworfene Klassenstruktur aus meiner Sicht für die meisten eher unverständlich als schlüssig.
Und so kam es, wie es kommen musste: Während der Auswertung nach Abschluss des Teilprojektes ließ dann jeder einmal so richtig Luft ab. Nachdem ich dabei relativ viel abbekam, aus meiner jetzigen Sicht sogar großteils gerechtfertigt, ergab sich als Resultat übereinstimmend, dass an der Kommunikation innerhalb der Gruppe wesentlich gearbeitet werden muss. Aus diesem Grund hielten wir es für erforderlich, auch die Moderation der Gruppentreffen abwechselnd verschiedenen Mitgliedern zuzuweisen. Allerdings war es dabei am Anfang relativ schwierig, viele Schweigesekunden zu überbrücken und ein Thema so lange zu vermeiden, bis es der Moderator angesprochen hatte.
Dieses Konzept hat sich jedoch bewährt. Die im ersten Teilprojekt aufgetretenen Probleme waren seit dieser Besprechung wie weggeblasen. Plötzlich hatte jeder Lust, sich intensiv in das Projekt einzubringen, es kamen Vorschläge zu Entwürfen, organisatorische Sachen und Umsetzungen. Man könnte meinen, diese Besprechung wäre eine Art Befreiungsschlag gewesen, nachdem keiner mehr Angst hatte, seine Meinung zu den verschiedensten Themen zu sagen.

Abb. 4: Einsicht in die Folgen fehlender Kommunikation über Entwürfe.

Probleme, die sich im Softwarepraktikum ergaben, waren am Anfang durch die großen Unterschiede der Programmiererfahrungen und den verschiedenen Herangehensweisen jedes Einzelnen in der Gruppe an das Praktikum. Ich hatte am Anfang das Gefühl, überrannt zu werden, da ich nicht hinterher kam. Dies kam vor allem, daß F. sofort mit einem Klassendiagramm kam und ich mir erst einmal über ein Vorgehen in einem Softwareprojekt klar werden musste. Nachdem wir F. erst einmal gebremst hatten und er sich offensichtlich auch selbst ein wenig zurückgehalten hatte, wurde mir einiges klar und das Softwarepraktikum konnte beginnen.

Abb. 5: Bewertung von Entwurfsgeschehen und Reviews von Seiten eines weniger erfahrenen Studierenden

Konflikte

Ein typischer Fall für ein Kooperationsproblem ist, dass ein Gruppenmitglied sich nicht genügend engagiert. Studentische Solidarität führt oft dazu, dass dies gegenüber dem Betreuer nicht transparent gemacht wird. Die Gegenüberstellung von geplanten und tatsächlichen Aufwänden macht es den Betreuern erfahrungsgemäß etwas leichter, solche Phänomene nach Abschluss eines Teilprojektes zu erkennen und ihre Behandlung sowie die Dokumentation der Entscheidung über Konsequenzen in einem Projektbericht zu verlangen. Wird ein solcher Missstand in einem frühen Teilprojekt deutlich, dann hat die Gruppe auch Gelegenheit, ihn in späteren Teilprojekten abzustellen oder angemessene Konsequenzen zu ziehen.

Die persönlichen Reflexionspapiere sind ein Prozesselement, das in manchen Gruppen dazu genutzt wird, Frustrationen über andere Gruppenmitglieder Ausdruck zu geben. Hier haben wir auch schon den Fall erlebt, dass der Betreuer der Gruppe dabei helfen konnte, Missverständnisse, die den Konflikten zugrundelagen, aufzuklären und damit gewisse Konflikte zu entschärfen.

Auch die Konfliktfähigkeit wurde geprobt. In allen Gruppen treten Konflikte auf. In den Gruppensitzungen besteht die explizite Gelegenheit, zu reflektieren, ob und wie man diese Konflikte thematisieren kann, um in Folgeteilprojekten die schädlichen Folgen unbearbeiteter Konflikte zu vermeiden und die (womöglich vorhandenen) fruchtbaren Konfliktursachen produktiv zu nutzen. Abbildung 6 liefert einen Auszug aus einem persönlichen Reflexionspapier eines Mitglieds einer Gruppe, in der ein Gruppenmitglied zuweilen „ausfiel“.

Der eigentlich unangenehmste Teil dieses Teilprojektes war jedoch nicht inhaltlicher sondern organisatorischer Art. Ein Gruppenmitglied, dessen Motivation bzw. Zuverlässigkeit bereits aus vorherigen Teilprojekten bekannt war, ließ uns in diesem Teilprojekt besonders hängen. Am ärgerlichsten war hier jedoch nicht seine Abwesenheit, sobald sich die Woche dem Ende näherte (meistens so ab Mittwoch), sondern die Tatsache, dass er uns bei dem ersten durchzuführenden Review hat vergeblich warten lassen. Eine Verschiebung des Termins war somit die Folge.

Abb. 6: Hinweise auf einen Konflikt in der Gruppe

2.6 Erfolgskriterien

Ein für die Studierenden wichtiger Punkt in vielen Lehrveranstaltungen ist die Frage, wie sie nachweisen sollen, dass sie die Veranstaltung erfolgreich absolvieren. Im Praktikumsleitfaden, der den Studierenden am Anfang der Veranstaltung ausgeteilt wird, finden die Studierenden den in Abbildung 7 wiedergegebenen Passus. Es hat sich bewährt, die Studierenden während der Projektlaufzeit immer wieder auf diesen Passus hinzuweisen, weil dies die Motivation, die entsprechenden Dokumente anzufertigen und die Protokolle sorgsam zu führen, erheblich stützte.

Für den Scheinerwerb sind wenigstens folgende Dokumente vorzulegen:
Aussagekräftige Zeitprotokolle jedes einzelnen Gruppenmitglieds über die gesamte Projektlaufzeit
Stilfibel für das Kodieren und für das Dokumentieren
Je einen Plan für jedes der Teilprojekte, je mit inhaltlicher Beschreibung, Aufwands abschätzung, Abschlussdatum und Datum der Planung.
Nach jedem Teilprojektabschluss: Gegenüberstellung von Planungen und tatsächlichem Projektablauf mit Erläuterungen in einem Reflexionspapier, das auch die explizite Lehren enthält, die die Teilnehmer aus den Abweichungen gezogen haben.
Bei Änderungen von Plänen ausstehender Teilprojekte: Überarbeitete Pläne, Begründung für die Veränderung
Getestete, dokumentierte und der Stilfibel entsprechende Programme für jedes Teilprojekt
Wenigstens ein Protokoll über die Durchführung eines Reviews
Die Ergebnisprotokolle der gemeinsamen Sitzungen, über die gesamte Projektlaufzeit
Ein zusätzliches Reflexionspapier von jedem einzelnen Gruppenteilnehmer zum Projektschluss.

Weitere notwendige Bedingungen sind:
Kontinuierliche Mitarbeit an den Teilprojekten über die gesamte Projektlaufzeit
Regelmäßige Teilnahme an den Plenums- und den Betreuerterminen
Eingehende Kenntnis nicht nur der selbst angefertigten Teile von Programm und anderen Arbeitsergebnissen, sondern auch der Teile, die von anderen Gruppenmitgliedern angefertigt sind.

Abb. 7: Erfolgskriterien im Cottbuser Softwarepraktikum

An anderer Stelle des Leitfadens wird beschrieben, wie etwa ein Ergebnisprotokoll, ein Reviewprotokoll oder ein Testplan zu gestalten ist.

3 Erfahrungen und Ausblicke

Die Sichtweise von Softwareentwicklung als Tätigkeit mit expliziten Reflexionsaspekten als Leitline für die Gestaltung des Softwarepraktikums ist bei uns im Laufe der Semester immer relevanter geworden. Wir haben die Erfahrung gemacht, dass eine immer konsequentere Gestaltung des Softwarepraktikums mit der Maßgabe, Softwareentwicklung als reflektiertes Handeln zu organisieren, zu immer befriedigenderen Lernergebnissen geführt hat. Auch die Leistungen erfolgreicher Prozesselemente, die wir ursprünglich aus anderen Gründen in das Softwarepraktikum eingeführt hatten, etwa die Projektplanung, die Aufwandsprotokollierung und die

Durchführung von Reviews, ließen sich oft nachträglich auch damit begründen, dass sie das Wechselspiel von Praxis und Reflexion unterstützen.

Wir selbst haben erst mit der Zeit gelernt, wie Reflexionselemente in die Organisation eines Softwarepraktikums eingefügt werden können. Bei unserem ersten Entwurf für einen Leitfaden für das Softwarepraktikum vor drei Jahren haben wir noch einen einfachen Durchlauf durch ein Wasserfallmodell der Softwareentwicklung empfohlen und keine Anfertigung von Reflexionspapieren verlangt. Dieser Leitfaden war in der Ausbildungspraxis aber auch wenig erfolgreich. Wir haben die in diesem und allen späteren Durchläufen durch das Softwarepraktikum gesammelten Erfahrungen in Zusammenarbeit mit den Praktikumsteilnehmern ausgewertet, Konsequenzen daraus gezogen und diese durch Überarbeitung des Leitfadens fixiert.

Unser Hauptziel ist der Lernerfolg der Studierenden. Es ist nicht eindeutig, wie man diesen messen müsste. Nach objektiven Kriterien, etwa hinsichtlich der Planungsgenauigkeit, sind wir wohl nicht erfolgreich. Dafür ist die Zahl von (typisch etwa) drei Zyklendurchläufen wohl auch zu klein, um schon einen Lernerfolg konkret nachzuweisen. Allerdings gibt es eine Vielzahl weniger harter Kriterien, die uns an einen Erfolg glauben lassen. Inbesondere die Reflexionsberichte weisen darauf hin, dass die Studierenden lernen; allerdings lässt sich aus diesen nicht ablesen, inwiefern der Transfer aus der (theoretischen) Einsicht in die Praxis gelingt.

Der bisher von den Betreuern des Praktikums erbrachte Aufwand wird mit steigenden Studierendenzahlen nicht mehr tragbar sein. In Erwartung eines solchen Anstiegs in den kommenden Jahren haben wir im Sommersemester 2000 versucht, den Betreuungsaufwand zu vermindern und gleichzeitig Studierenden aus dem Hauptstudium eine neuartige Lerngelegenheit zu verschaffen. Wir haben im Rahmen eines Seminars im Hauptstudium Studierende als Berater zur Betreuung einiger Gruppen eingesetzt. Sie haben den Projektverlauf beobachtet und die Gruppen in technischen, vor allem aber in organisatorischen Fragen beraten. Neben dieser allgemeinen Beratung gehörte es zu den Aufgaben der Seminarteilnehmer, sich ein spezielles Problemgebiet auszusuchen, hinsichtlich dessen die Praktikumsgruppe besonders untersucht werden sollte, und hinsichtlich dessen sie ihre eigene Beratungstätigkeit reflektieren sollten. Für dieses Problemgebiet sollten Beobachtungen gemacht und Daten gesammelt, dokumentiert und interpretiert werden, eigene Interventionen sollten beschrieben und hinsichtlich ihres Erfolgs bewertet werden. Beispiele für solche Problemgebiete waren Planung und Projektverfolgung, Entstehung und Behandlung von Konflikten, und die Rollenverteilung in der betreuten Arbeitsgruppe.

Die Aufgabe der Berater bei der Betreuung war es also, der Gruppe Probleme bewusst zu machen, sie bei der Lösung der Probleme zu beraten, und ihre eigene diesbezügliche Tätigkeit zu reflektieren. Allerdings verfügen die Berater nicht über formale Autorität, und entsprechend haben sie auch keine offizielle Verantwortung für den Gruppenerfolg.

Wir nutzten das Softwarepraktikum im Sommersemester 2000 mithin in zweierlei Weise: (1) Es war eine Möglichkeit für die Praktikumsteilnehmer im Grundstudium, Erfahrungen zu machen und daraus zu lernen. (2) Es war eine Möglichkeit für Stu-

dierende im Hauptstudium, den Ablauf von Softwareentwicklungsprojekten am Beispiel zu erleben, Schwierigkeiten der Einflussnahme kennenzulernen, und beides zu reflektieren.

Unsere ersten positiven Erfahrungen sind, dass die Seminarteilnehmer lernten, die Mitglieder der von ihnen betreuten Praktikumsgruppe in verschiedener Hinsicht sehr differenziert zu beurteilen.

Eine Überraschung, die wir bei der Durchführung der Praktika in den letzten Jahren erlebten, bezog sich auf die Unterschiedlichkeit der Erfahrungen, die die Studierenden in den verschiedenen Teilprojekten machten. Es hat sich herausgestellt, dass die meisten Gruppen die Gesamtaufgabe nicht in die empfohlenen vier, sondern nur in drei Teilprojekte aufgeteilt haben. Dabei hat sich gezeigt, dass in den verschiedenen Teilprojekten verschiedene Probleme und Lernbereiche dominieren:

1. Teilprojekt: Planungsprobleme werden explizit, Aufwandsprotokollierung und Reflexionselemente werden, zuweilen mit Schwierigkeiten, eingeführt.
2. Teilprojekt: Konsolidierung und erste Anwendung des im ersten Teilprojekt Gelernten; es treten Motivationstiefs auf, die expliziert und behandelt werden müssen, sowie andere Probleme der Arbeitsorganisation, insbesondere Konflikte unter den Gruppenmitgliedern; erste schwerwiegendere technische Probleme treten auf und müssen gelöst werden.
3. Teilprojekt: Es entsteht Projektdruck durch den festen Endtermin, aufgrund dessen man keine Teilaufgaben mehr in spätere Teilprojekte verschieben kann, wie das am Ende der früheren Teilprojekte noch möglich gewesen war. Die technischen Schwierigkeiten spitzen sich gemeinsam mit den organisatorischen zu.

Unsere Perspektive für das Softwarepraktikum ist, dass wir den Studierenden zunehmend effektiv, aber auch effizient vermitteln wollen, wie sie ihre eigene Arbeit sowohl während des Studiums als auch später im Beruf als Lernprozesse organisieren können. Insbesondere wollen wir vermitteln, wie man Softwareentwicklungsprojekte als Lernprojekte organisiert. So bleibt der oft gehörte Slogan „das Lernen lernen" für sie nicht abstrakt, sondern wird mit einer konkreten Vorgehensweise unterlegt.

Wir haben die Erfahrung gemacht, daß die expliziten Reflexionsdokumente zu einer wesentlich qualifizierteren Diskussion sowohl zwischen den Praktikumsteilnehmern als auch mit den Auftraggebern und Beratern führten und eine sachliche Analyse von positiven und negativen Projektverläufen erlaubten. Die Studierenden haben gelernt, gezielt Maßnahmen zur Veränderung ihrer Arbeitsprozesse zu ergreifen und die Effekte zu bewerten. Dadurch entstand vor allem eine erheblich realistischere Einschätzung der grundsätzlichen Komplexität von Software-Entwicklungsprozessen und der Schwierigkeiten von Teamarbeit.

Im Zusammenhang mit dem Softwarepraktikum gibt es Lernende auf verschiedensten Ebenen: Die Teilnehmer des Softwarepraktikums lernen die Anwendung der konkreten Techniken und die reflektive Auswertung der dabei gemachten Erfahrungen; fortgeschrittene Studierende unterstützen als Moderatoren diese Reflexion-

prozesse und reflektieren selbst ihre Moderationstätigkeit; und Dozenten und wissenschaftliche Mitarbeiter sammeln Erfahrungen mit der Durchführung von Praktika und reflektieren diese ebenfalls mit dem Ziel der Prozessverbesserung.

Danksagung

Wir danken den Teilnehmern des Cottbuser Softwarepraktikums der letzten Jahre und insbesondere unserem Kollegen Hans-Gerd Köhler für Mitwirkung an unserem ständigen Lernprozess.

Literatur

1. Anderson, A.; Beattie, R.; Beck, K.; Bryand, D.; DeArment, M.; Fowler, M.; Fronczak, M.; Garzaniti, R.; Gore, D.; Hacker, B.; Hendrickson, Ch.; Jeffries, R.; Joppie, D.; Kim, D.; Kowalsky, P.; Mueller, D.; Murasky, T.; Nutter, R.; Pantea, A.; and Thomas, D.: Chrysler goes to „extremes". *Distributed Computing*, S. 24 - 28, October 1998.
2. Beck, K.: *Extreme Programming Explained: Embrace Change*. Addison Wesley Longman, Reading/Massachusetts, 1999.
3. Humphrey, W. S.: *A Discipline for Software Engineering.* Addison-Wesley, Reading/Massachusetts, 1995.
4. Humphrey, W. S.: *Introduction to the Team Software Process.* Addison-Wesley, Reading/Massachusetts, 2000.
5. Pasch, J.: *Softwareentwicklung im Team.* Springer-Verlag, Berlin, 1994.
6. Schön, D. A.: *The Reflective Practitioner.* Basic Books, New York, 1982.
7. Schön, D. A.: *Educating the Reflective Practitioner.* Jossey-Bass Publishers, San Francisco, 1986.
8. Schön, D. A. (Hrsg.).: *The Reflective Turn. Case Studies In and On Educational Practice.* Teachers College Press, 1991.

Ein Prozessmodell für das Software-Praktikum

Doris Schmedding
Fachbereich Informatik, Lehrstuhl für Software-Technologie, Universität Dortmund
Doris.Schmedding@cs.uni-dortmund.de

Zusammenfassung

Es wird ein besonders für die Lehre in Form eines Praktikums geeignetes auf UML und Java basierendes Prozessmodell vorgestellt. Dazu werden spezielle Anforderungen an ein Prozessmodell für die Lehre im Grundstudium herausgearbeitet. Während des ersten Einsatzes wurde die Eignung des Modells durch eine Evaluation überprüft und Verbesserungsvorschläge erarbeitet.

1 Einleitung

Die Umstellung der Informatik-Grundausbildung an der Universität Dortmund auf die Programmiersprache Java machte eine grundsätzliche Umgestaltung des zum Grundstudium gehörenden Software-Praktikums (SoPra) nötig. Ziel des Praktikums ist eine handlungsorientierte Einführung in Methoden und Verfahren der Software-Technik, indem die Studierenden im Team Software-Projekte durchführen. Das Software-Praktikum ist eine Pflichtveranstaltung für alle Informatikstudierenden und wird in der Regel als sechswöchige Blockveranstaltung in der vorlesungsfreien Zeit durchgeführt. Das Praktikum findet nach dem 3. Semester statt, nachdem die Programmierausbildung abgeschlossen ist.

Es werden pro Jahr drei Praktika angeboten, an denen in der Regel 96 Studierende teilnehmen, die in 12 Gruppen zu je 8 aufgeteilt werden. Die Gruppen führen nacheinander zwei Software-Projekte durch, wobei alle Teams die gleichen Aufgaben lösen. Jedes Entwicklerteam wird von einer studentischen Hilfskraft oder einem wissenschaftlichen Mitarbeiter betreut, die den Gruppen beratend zur Seite stehen.

Als Modellierungssprache wurde UML [3] [5] [9] gewählt. Die Modellierung mit UML wird durch das relativ einfach zu bedienende Modellierungswerkzeug TogetherJ unterstützt. Die Suche nach einem für die Lehre geeigneten Prozessmodell gestaltete sich recht schwierig, denn die meisten der in der Literatur vorgestellten Prozessmodelle [7] schienen wegen ihrer Komplexität nicht geeignet zu sein. Dies gilt insbesondere für den speziell auf UML abgestimmten Rational Unified Process (RUP) [6]. Aufgrund der speziellen Anforderungen in der Lehre, die im folgenden Kapitel näher erläutert werden, wurde ein besonders einfaches SoPra-Prozessmodell

entwickelt, dessen besondere Eignung für die Lehre durch eine Evaluation überprüft wurde.

Die Evaluation wurde im Rahmen einer Diplomarbeit [1] praktikumsbegleitend durchgeführt. Dazu wurden einerseits die TeilnehmerInnen mittels Fragebögen und Stundenzettel über Lernerfolg, Zufriedenheit und Arbeitsbelastung befragt, andererseits auch ihre Modellierungsergebnisse untersucht. Da 12 Gruppen parallel die gleichen Aufgaben gelöst haben, ergab sich die einmalige Gelegenheit, die Ergebnisse verschiedener Gruppen zu vergleichen.

2 Anforderungen an das Prozessmodell

Um zu einem geeigneten Prozessmodell zu gelangen, muss zunächst die Zielsetzung des Praktikums herausgestellt werden. Folgende Lernziele bilden die Grundlage des Konzepts der Lehrveranstaltung:

- Kennen lernen von Projektarbeit,
- Arbeiten im Team,
- Anwendung von Software-Entwicklungsmethoden und
- Einsatz von Software-Entwicklungswerkzeugen.

Das eingesetzte Prozessmodell muss möglichst optimal auf die beteiligten Personen, ihre Fähigkeiten, ihr Vorwissen und ihre Motivation, auf die eingesetzten Methoden und Verfahren und auf die zu verwendenden Entwicklungsumgebung mit ihren Werkzeugen abgestimmt sein [8]. Nur dann können mit diesem Entwicklungsprozess vorhersagbare Resultate erzielt werden.

Bei der Definition des Prozessmodells muss deshalb zunächst das Vorwissen der EntwicklerInnen berücksichtigt werden. Die TeilnehmerInnen am Software-Praktikum bringen nur recht rudimentäre Kenntnisse der UML mit, die sie im Rahmen ihrer Programmierausbildung erworben haben. Eine fundierte Ausbildung in Software-Technik erfolgt in Dortmund erst im Hauptstudium.

Zum Praktikum gehört eine Vorlesung im Umfang von nur einer Semesterwochenstunde, in der das Prozessmodell vorgestellt werden muss, die Sprachkonstrukte der UML und Themen aus der Programmierung wie z.B. die graphischen Benutzungsschnittstellen vertieft werden müssen, die in der Programmierungsvorlesung nur am Rande behandelt werden. Ganz neue Themen wie das Testen kommen hinzu.

Daraus ergeben sich folgende Anforderungen an ein Prozessmodells für das Software-Praktikum:

- Es muss auch ohne Vorwissen leicht erlernbar sein.
- Es muss klar strukturiert und damit leicht überschaubar sein, damit die Studierenden immer wissen, was als Nächstes zu tun ist.
- Es muss viele Hilfsangebote für die Studierenden beinhalten.

- Das Prozessmodell soll die studentischen EntwicklerInnen unterstützen, sie dürfen sich aber nicht gegängelt fühlen. Sie sollen das Prozessmodell als hilfreich und sinnvoll erleben.
- Das Prozessmodell muss auf die begrenzten zeitlichen Ressourcen des Software-Praktikums Rücksicht nehmen.

Die Komplexität der UML als Sammlung vieler Notationen bereitet große Probleme beim Einsatz in einer Lehrveranstaltung. Welche Notationen sollen in welcher Tiefe gelernt werden? Selbst wenn Zeit genug vorhanden wäre, alle Diagrammtypen und Sprachkonstrukte komplett vorzustellen, braucht es viel Erfahrung in der Modellierung, um entscheiden zu können, welcher Diagrammtyp bei der Lösung welcher Detailfrage besonders hilfreich ist. Also müssen die Lehrenden eine Teilmenge der UML festlegen, die ihrer Meinung nach groß genug ist, die Aufgaben zu lösen, und die Kreativität der EntwicklerInnen nicht einschränkt, andererseits klein genug ist, um von den Lernenden nach kurzer Zeit beherrscht werden zu können.

Auch auf den begrenzten zeitlichen Rahmen des Software-Praktikums muss bei der Definition des Prozessmodells Rücksicht genommen werden. Für ein Projekt stehen in einem Blockpraktikum nur drei Wochen zur Verfügung. Diese Zeit sollte optimal ausgenutzt werden, so dass kein Leerlauf entsteht, also alle Teilnehmer in allen Entwicklungsphasen Aufgaben haben und möglichst keine Entwickler auf Ergebnisse anderer Entwickler warten müssen.

Daneben sollte ein Prozessmodell immer möglichst gut auf die damit zu erstellenden Produkte zugeschnitten sein. In einer Lehrveranstaltung bietet sich im Gegensatz zur Industrie, wo zur vorgegebenen Aufgabenstellung das optimale Prozessmodell gesucht wird, die Möglichkeit, durch geschickte Wahl der Aufgabenstellung Prozessmodell und Aufgabe in Einklang zu bringen. Wir haben deshalb zunächst auf die Erstellung verteilter Software-System verzichtet und konnten so auch auf die Verteilungsdiagramme der UML verzichten.

Bei der Festlegung der Rollen im Prozessmodell werden in der Lehre andere Ziele verfolgt als bei der kommerziellen Software-Entwicklung, wo sich alle Entscheidungen am wichtigsten Ziel Wirtschaftlichkeit orientieren müssen. In der Lehre und besonders in der Grundausbildung sollte bei der Verteilung der Teilaufgaben an die Gruppenmitglieder nicht das effiziente Spezialistentum im Vordergrund stehen, sondern alle Lernenden sollten möglichst weitgehend mit allen im Prozessmodell vorgesehenen Aktivitäten Erfahrungen sammeln.

Auch bei Werkzeugauswahl und Einsatz müssen die besonderen Anforderungen einer Lehrveranstaltung berücksichtigt werden. Auch hier steht die leichte Erlernbarkeit und einfache Bedienung im Vordergrund der Auswahlkriterien.

3 Das SoPra-Prozessmodell

Angelehnt an das ISP-Modell (*Irrational Separated Process*) von Hitz und Kappel [5], das seinen Namen zur Abgrenzung gegenüber dem *Rational Unified Process*

(RUP) [6] trägt, wurde unter Berücksichtigung der oben definierten Anforderungen das Sopra-Prozessmodell definiert. Um den Lernenden die Orientierung zu erleichtern, wurde im Gegensatz zum RUP die Abfolge der Aktivitäten recht starr festgelegt, die Aktivitäten wurden genau bestimmten Phasen zugeordnet. Erfahrene Entwickler würden sich durch ein derart starres Schema unnötig gegängelt fühlen.

Abbildung 1 zeigt den Phasenablauf des Prozessmodells, dessen Grobstruktur weitgehend vom ISP-Modell übernommen wurde. Die Anforderung, dass leichte Erlernbarkeit und Überschaubarkeit für einen in der Lehre einsetzbaren Prozess wesentliche Merkmale sein müssen, rechtfertigt seine an das Wasserfallmodell angelehnte Struktur. Am Ende jeder Phase präsentieren die Studierenden im Rahmen eines Reviews ihre Ergebnisse. Außerdem werden die von den Studierenden erstellten Dokumente von den Gruppenbetreuern korrigiert. So soll sichergestellt werden, dass nur nach vollständiger Bearbeitung einer Phase in die nächste gewechselt wird

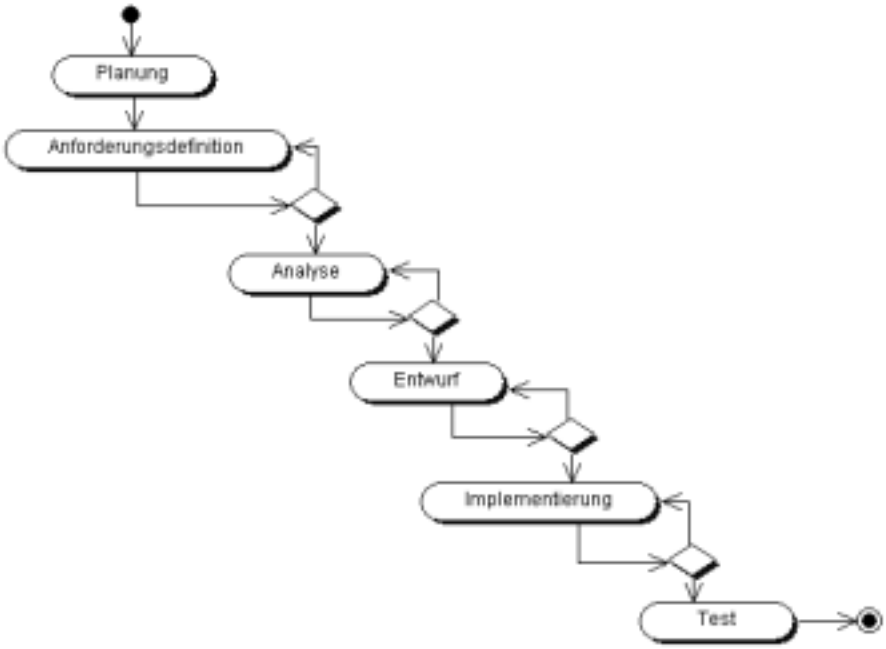

Abb. 1: Einteilung des Prozesses in Phasen

Normalerweise enthält ein Prozessmodell wie z.B. auch das ISP-Modell vage Anweisungen beispielsweise derart, dass besonders komplexe Anwendungsfälle durch Aktivitätsdiagramme näher zu untersuchen sind oder dass die Zustände von Objekten mit nicht trivialem Lebenszyklus durch Zustandsübergangsdiagramme modelliert werden sollten. Entscheidungen, wo sich welcher Aufwand lohnt, um zu neuen Erkenntnissen zu gelangen, können aber Lernende im Grundstudium mangels

Erfahrung noch nicht treffen. Deshalb werden diese Arbeitsaufträge durch Anweisungen derart ersetzt, dass z.B. für alle Anwendungsfälle Aktivitätsdiagramme zu erstellen sind. So erhalten alle Gruppenmitglieder die Chance, sich an der Erstellung der Aktivitätsdiagramme zu beteiligen und diese Fertigkeit zu erlernen.

Die **Planung** eines Projekts, die neben der Aufgabenbeschreibung auch die Erstellung eines Zeitplans beinhaltet, ist allein Aufgabe der Praktikumsveranstalter. Die Studierenden wären zumindest im ersten Projekt mit der Erstellung eines Zeitplans überfordert.

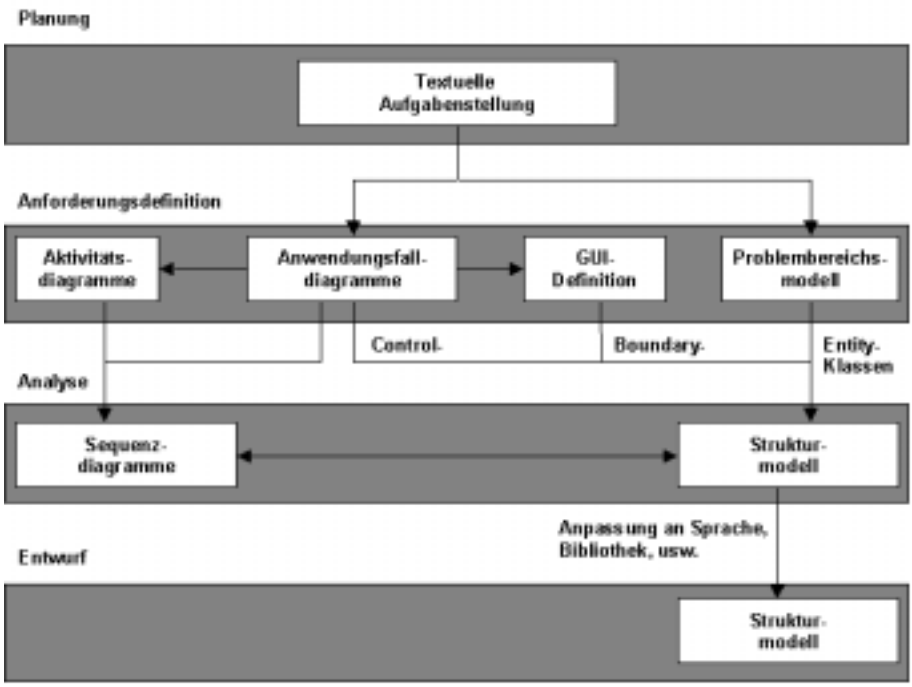

Abb. 2: Abhängigkeiten zwischen den in den Phasen eingesetzten UML-Diagrammen

Abbildung 2 zeigt, welche UML-Diagramme ausgehend von der textuellen Aufgabenbeschreibung in welchen Phasen zu entwickeln sind und welche inhaltlichen Zusammenhänge zwischen den Diagrammen bestehen. Nachfolgend wird die Zielsetzung der einzelnen Entwicklungsphasen und die darin zu erledigenden Aktivitäten näher erläutert.

Die Ausgangsbasis für die **Anforderungsmodellierung** bildet das aus den textuellen Aufgabenstellung zu erarbeitende Anwendungsfalldiagramm. Die einzelnen Anwendungsfälle werden durch Aktivitätsdiagramme näher beschrieben. Ein erstes Klassendiagramm stellt das Datenmodell des Problembereichs dar, das auch auf den Informationen des Aufgabentextes basiert. Mit Hilfe eines GUI-Builders wird ausgehend von den Anwendungsfällen die graphische Benutzungsschnittstelle des Produkts gestaltet. Die Benutzungsschnittstelle und die Funktionalität des Produkts wird

im Benutzungshandbuch beschrieben. Während in der Anforderungsdefinitionsphase alle im RUP- und auch im ISP-Prozess vorgesehenen UML-Diagramme erstellt werden, wurde in den nachfolgenden Phasen auf einige Diagramme verzichtet.

In der **Analysephase** werden die Klassen des Problembereichsmodells als sogenannte Entity-Klassen in das Strukturmodell, das Klassendiagramm der Analyse, übernommen. Zur Realisierung der Anwendungsfälle werden Control-Klassen in das Strukturmodell aufgenommen, wobei pro Anwendungsfall eine Methode vorzusehen ist. Zusätzlich kommen aus der graphischen Gestaltung der Benutzungsschnittstelle die GUI-Klassen als Boundary-Klassen hinzu.

Zu jedem Anwendungsfall, der durch ein Aktivitätsdiagramm näher beschrieben wird, wird ein Sequenzdiagramm erstellt. Mit Hilfe der Sequenzdigramme wird untersucht, welche Methoden benötigt werden. Da Sequenz- und Kollaborationsdiagramme beide die Interaktion der Objekte darstellen, wurde zugunsten der übersichtlicheren Sequenzdiagramme auf die Kollaborationsdiagramme verzichtet. Die Sequenzdiagramme wurden bevorzugt, weil sie wegen der vorgegebenen Struktur für Ungeübte leichter zu erstellen und zu verstehen sind, horizontal werden die am Anwendungsfall beteiligten Objekte dargestellt und die vertikale Dimension entspricht der Zeitachse. Der einzige Vorteil der Kollaborationsdiagramme wäre gewesen, dass sie neben den Interaktionen auch die aus dem Strukturdiagramm übernommen Objektbeziehungen darstellen. Werkzeuge sind in der Lage, diese beiden Diagrammtypen ineinander zu überführen.

Auch auf die Erstellung von Zustandsübergangsdiagrammen, mit denen Übergange zwischen den Zuständen eines Objekts modelliert werden können, wurde verzichtet. Der ISP-Prozess von Hitz und Kappel [5] sieht sie in der Analysephase vor. Zustandsübergangsdiagramme machen zwar für bestimmte komplexe Objekte Sinn, die einen nicht trivialen Lebenszyklus aufweisen, - dies gilt z.B. für manche Klassen der GUI und die Systemschnittstellenklassen - aber der Aufwand für die Vermittlung und Einübung einer weiteren graphischen Notation hätte meiner Meinung nach in keinem Verhältnis zum didaktischen Gewinn gestanden.

In der **Entwurfsphase** wird das Strukturmodell zur Vorbereitung der Implementierung an die eingesetzte Programmiersprache und die verwendeten Bibliotheksklassen angepasst. Dazu wird der Detailentwurf der Klassen vorgenommen, im wesentlichen werden dabei die Parameter der Methoden und die Sichtbarkeit der Attribute und Methoden festgelegt. In der Regel kommen im Entwurf weitere Boundary-Klassen hinzu, um z.B. die Persistenz der Daten zu realisieren. Die Entwickler können in interessanten Fällen noch einmal Sequenzdiagramme einsetzen, deren Notation bereits bekannt ist, um das Zusammenwirken der neu hinzugekommenen mit den alten Klassen zu untersuchen. Verlangt werden Sequenzdiagramme im Entwurf aber nicht.

Zu den weiteren Aufgaben der Entwurfsphase gehört die Strukturierung und Partitionierung des Strukturmodells. Falls das Modell unüberschaubar geworden ist, sollte es in Pakete aufgeteilt werden.

Für das **Implementieren** und **Testen** wurde die Regel festgelegt, dass alle Gruppenmitglieder mindestens eine Klasse selbständig implementieren und nach einem vorgegebenen Verfahren testen müssen. Auf die Erstellung von Komponenten- und Verteilungsdiagrammen wurde verzichtet, sie eher der Software-Dokumentation dienen. Ihr Nutzen wird erst in der im Praktikum fehlenden Wartungsphase sichtbar, so dass sie in einer Lehrveranstaltung wie dieser schlecht zu motivieren sind.

4 Die Evaluation des Prozessmodells und einige Ergebnisse

Um den Reifegrad des vor dem ersten Software-Praktikum mit Java und UML definierten Prozessmodells gemäß Capability Maturity Model (CMM) [8] zu erhöhen, wurde eine Evaluation des Prozessmodells durchgeführt. So wurden Verbesserungsvorschläge erarbeitet, die teilweise bereits umgesetzt wurden, bzw. noch umgesetzt werden sollen. Obwohl die PraktikumsteilnehmerInnen im Software-Praktikum zum ersten Mal Software mit Unterstützung eines Prozessmodells entwickelten, befindet sich der Prozess nicht auf der initialen Reifestufe, denn die langjährige Erfahrung der Veranstalter, die den Prozess definiert haben und überwachen, verhindert chaotischen Verhaltensweisen, die typisch für diesen Entwicklungsstand sind.

Gemäß [10] lässt sich die durchgeführte Evaluation folgendermaßen klassifizieren: Evaluationsobjekt ist das neu definierte Prozessmodell des Software-Praktikums. Die Evaluation wurde praktikumsbegleitend im Februar und März 2000 in summativer Form durchgeführt. Ziel der Evaluation war herauszufinden, wie weit die in Kap. 2 definierten Anforderungen vom definierten Prozessmodell realisiert werden. Die Evaluation wurde mittels Stundenzetteln, Fragebögen und Messungen der erstellten UML-Modelle mit Hilfe einer für diesen Zweck entwickelten Metrik [1] durchgeführt.

In der industriellen Software-Entwicklung spiegelt sich die Qualität des Prozesses in erster Linie in der Qualität der erstellten Produkte wieder. Da aber das Ziel eines Praktikums nicht das Produkt, sondern der Lernfortschritt der Studierenden ist, muss nicht nur die Qualität des Endprodukts, sondern auch die Qualität der Zwischenprodukte, der während des Entwicklungsprozesses erstellten UML-Diagramme und anderen Dokumente, bewertet werden. Einerseits werden die erstellten Diagramme und Dokumente von den Praktikumsbetreuern korrigiert, andererseits wurde versucht, die Qualität der Diagramme durch die Definition einer Metrik messbar zu machen.

Angelehnt an das Goal-Question-Metric-Paradigma [2] wurden im Rahmen einer Diplomarbeit [1] die drei Maße visuelle Größe, informative Größe und Komplexität definiert, wobei mit der visuellen Größe die Überschaubarkeit der Diagramme und mit der informativen Größe der Informationsgehalt eingeschätzt werden soll. Zur Beurteilung der Komplexität der verschiedenen Diagrammtypen wurden jeweils eigene Bewertungskriterien entwickelt. Neben dieser Bewertung der erstellten Modell von außen mit Hilfe der Metrik und durch die Betreuer wurden die Studierenden am

Ende des Praktikums zu einer eigenen Einschätzung ihres Lernerfolgs aufgefordert. Nachfolgend werden einige Ergebnisse vorgestellt:

Erhebliche Produktivitätssteigerung. Wöchentlich wurde in beiden Projekten durch Stundenzettel die Arbeitsbelastung der Gruppenmitglieder abgefragt. Im 2. Projekt wurde für die Modellierung nur halb so viel Zeit wie im erste Projekt benötigt, obwohl die Aufgabenstellung im 2. Projekt umfangreicher war. Die Steigerung der Produktivität im 2. Projekt ohne den Einarbeitungsaufwand in die neuen Methoden und Werkzeuge wurde von den Veranstaltern erheblich unterschätzt.

Lernfortschritte. Mittels Fragebögen wurde der Kenntnisstand, der Lernfortschritt und die Zufriedenheit der Studierenden mit dem Prozessmodell, mit den Methoden und den eingesetzten Werkzeugen ermittelt. Die Studierenden bewerteten selbst ihre Kenntnisse im Bereich der Software-Entwicklung, der Prozessmodelle, der UML, in der Programmierung allgemein und in der Programmierung in Java vor und nach dem Praktikum mit Schulnoten. In allen Bereichen wurden die Kenntnisse nachher als verbessert beurteilt, den größten Zuwachs erzielten die Gebiete Software-Entwicklung, Prozessmodelle und UML mit einer Verbesserung von 0,85, 1,3 bzw. 1,4 Punkten. Dieses Ergebnis entspricht genau der Zielsetzung des Praktikums, das Software-Technikinhalte und nicht Programmieren lehren will.

Prozessmodell. Für die Studierenden war das definierte Prozessmodell trotz einführender Vorlesung nur schwer durchschaubar. Im ersten Projekt gab ein Drittel der Teilnehmer an, nicht immer gewusst zu haben, was als nächstes zu tun sei. Insbesondere die von ihnen erwarteten Ergebnisse und die einzusetzenden Notationsmittel waren ihnen unklar. Selbst im 2. Projekte herrschte hier noch viel Ratlosigkeit.

Aktivitätsdiagramme. Neben Fragen mit vorbereiteten Antworten enthielt der am Ende des Praktikums ausgeteilte Fragebogen auch offen formulierte Fragen, nach Eindrücken und Meinungen zum Praktikum. Die Antworten der TeilnehmerInnen machten deutlich, dass das Erstellen der Aktivitätsdiagramme als besonders lästig und wenig nutzbringend empfunden wurde. Einerseits scheint die Bedeutung dieses Diagrammtyps für den gesamten Entwicklungsprozess nicht richtig deutlich geworden zu sein, andererseits wurden weisungsgemäß sehr viele, auch winzig kleine Aktivitätsdiagramme erstellt. Insgesamt wurden 245 Aktivitätsdiagramme entwickelt, pro Gruppe und Projekt 10 bis 15 Diagramme.

Sequenzdiagramme. Die Modellierung der Sequenzdiagramme wurde als besonders aufwendig und schwierig empfunden. In der ersten Version waren sie meist unvollständig oder enthielten Fehler. Die mangelnde Qualität spiegelte sich auch in den gemessenen Größen wieder, die informative Größe und die Komplexität der zunächst erstellten Diagramme waren gering. Deshalb mussten sehr viele Sequenzdiagramme überarbeitet werden. Die Studierenden gaben an, für die Modellierung der Sequenzdiagramme etwa gleich viel Zeit wie für die Klassendiagramme benötigt zu haben, obwohl an den Klassendiagrammen in allen Phasen gearbeitet wurde, während die Sequenzdiagramme nur in der Analysephase eingesetzt wurden.

Anwendungsfalldiagramme. Die Anwendungsfalldiagramme haben sich als Einstiegspunkt in die Modellierung bewährt. Sie wurden als nützlich und wenig auf-

wendig empfunden, ihre Modellierung kostete die wenigste Zeit von allen Diagrammtypen.

Klassendiagramme. Der Nutzen der Modellierung der Klassendiagramme wurde von allen Teilnehmern eingesehen. Unterstützt wurde dieser positive Eindruck sicher vom eingesetzten Modellierungswerkzeug TogetherJ, da zu jeder Klasse schon während ihrer Modellierung ihr Java-Code angezeigt wird. Die gemessenen Größen zeigten an, dass der Informationsgehalt der Klassendiagramme von Phase zu Phase wuchs.

Warum wurden manche Diagrammtypen abgelehnt?

Die Evaluation zeigte einige Stärken und Schwächen der vorgestellten Vorgehensweise auf. Die von den Studierenden geäußerte Unzufriedenheit machte sich im Wesentlichen an den Aktivitätsdiagrammen und an den Sequenzdiagrammen fest, während Klassendiagramm und Anwendungsfalldiagramm als nützlich und den Entwicklungsprozess unterstützend akzeptiert wurden.

Aktivitätsdiagramme bereiten den Studierenden Schwierigkeiten, da der in der Vorlesung vorgestellte Sprachumfang nur eine geringe Teilmenge der vom Tool angebotenen und in Lehrbüchern [5] behandelten Ausdrucksmöglichkeiten darstellt. Während wir Aktivitätsdiagramme recht informal in der Anforderungsdefinitionsphase zur Beschreibung von Anwendungsfällen einsetzen, können sie auch in der Analyse und im Detailentwurf zum Entwurf von Prozeduren einsetzt werden [6]. Entsprechend mächtig ist die Notation, die zur Beschreibung der Abläufe verwendet werden kann, beispielsweise sieht die Notation grafische Elemente für das Senden und Empfangen von Ereignissen, Objekte und Objektfluss vor. Dieser Widerspruch zwischen der gelehrten und der vom Tool angebotenen Notation scheint für die Verunsicherung der Studierenden mitverantwortlich zu sein.

Hinzu kommt, dass die Festlegung einer geeigneten Granularität der modellierten Aktivitäten für Ungeübte schwierig ist. Wird zu grob modelliert, sind die Diagramme wenig hilfreich beim nächsten Modellierungsschritt. Wird die Granularität der Aktivitäten zu fein gewählt, ist die Erstellung der Diagramme sehr aufwendig. Die Studierenden haben zwar sehr viele aber sehr kleine Diagramme modelliert. Um den Aufwand für diesen Diagrammtyp insgesamt für sie akzeptabel zu halten, haben sie, da sie alle Anwendungsfälle durch Aktivitätsdiagramme beschreiben sollten, sehr grob modelliert. Da die Aktivitätsdiagramme zu einem sehr frühen Zeitpunkt in unserem Entwicklungsprozeß erstellt werden, sind den Studierenden vielleicht die Abläufe noch nicht klar genug, um tiefer vordringen zu können.

Sequenzdiagramme dienen dazu, die Dynamik des Systems zu untersuchen. Die Betrachtung der dynamischen Abläufe wird von den Studierenden grundsätzlich als schwieriger empfunden als die Untersuchung statischer Strukturen. In einem Sequenzdiagramm wird beschrieben, wie ein Methodenaufruf bei einem Objekt weitere Methodenaufrufe bei anderen Objekten hervorruft, die wiederum Methodenaufrufe bei weitern Objekten bewirken. Bei der Korrektur der Diagramme fiel auf, dass bei

der Modellierung nicht weit genug in die Tiefe des Systems vorgedrungen wurde. Außerdem bereitet den Studierenden die Darstellung von Iterationen Probleme.

Sowohl aus den Aktivitätsdiagrammen als auch aus den Sequenzdiagrammen wird kein Code erzeugt, so dass ihr direkter Nutzen für die Produktentwicklung nicht offensichtlich ist.

5 Konsequenzen aus den Ergebnissen

Da das Praktikum handlungsorientiert angelegt ist, besteht die Gefahr, dass die Studierenden die Zielsetzung missverstehen. Das Ziel ist nicht ein Software-Produkt, sondern das Erlernen des systematischen Vorgehens bei der Software-Entwicklung. Wenn das allen Teilnehmern klar ist, sollte auch die Motivation der UML-Diagramme, die nicht direkt zu Java-Code führen, leichter fallen.

Die berechtigte Forderung nach mehr Information über den Ablauf des Prozessmodells führte zu einer detaillierten Beschreibung des Modells auf den WWW-Seiten des Software-Praktikums (http://ls10-www.informatik.uni-dortmund.de/LS10/Pages/sopra-specials/Phasenmodell.html). Beispiele zu den einzelnen Diagrammtypen zeigen, wie die Notation eingesetzt wird und bis zu welchem Detaillierungsgrad modelliert werden soll. Da in der Vorlesung aus Zeitgründen und aus Gründen der Überschaubarkeit nur kleine Beispiele vorgestellt werden können, wirkt der Einsatz bestimmter Diagrammtypen dort übertrieben, denn die Zusammenhänge lassen sich auch ohne diese Diagramme leicht durchschauen. Komplexe Beispiele zum Selbststudium sind deshalb zur Motivation wichtig.

Von den Studierenden kann verlangt werden, dass sie sich selbst besser auf die Aufgaben im Praktikum vorbereiten. Um eine intensivere Nachbereitung der Vorlesungsinhalte anzuregen, wurden Präsensaufgaben zu Beginn der einzelnen Entwicklungsphasen eingeführt: In Multiple-Choice-Tests wird der Vorlesungsstoff abgefragt und die Lösungen in der Gruppe besprochen.

Die Gruppenbetreuer sollen die Gruppen in Fragen des Methoden- und Werkzeugeinsatzes beraten. Wenn unter den Studierenden große Unsicherheit herrschte über den Sinn bestimmter Diagramme und die einzusetzenden Notationsmittel, müssen auch die Gruppenbetreuer fachlich besser auf ihre Aufgaben vorbereitet werden. Ihnen wurde deshalb zur Aufgabe gestellt, die oben beschriebenen Multiple-Choice-Tests auszuarbeiten.

Auf jeden Fall sollen auch weiterhin im Praktikums zwei Projekte durchgeführt werden, denn so erleben die Studierenden den Produktivitätsgewinn und die Entlastung durch Erfahrung. Erst im zweiten Projekt können sie die neu erlernten Notationen wirklich kreativ zur Beschreibung ihrer Vorstellungen einsetzen.

Als Konsequenz aus dem überraschend hohen Produktivitätsgewinn wurde im nachfolgenden Praktikum der Umfang der ersten Aufgabe auf ein Minimum reduziert und die Komplexität der zweiten Aufgabe erhöht. Man könnte auch neue Inhalte in das 2. Projekt verlagern, etwa die Entwicklung eines verteilten Systems als Aufgabe stellen.

Die erstmals für die Evaluation des Prozessmodells eingesetzten Stundenzettel wurden in den nachfolgenden Praktika beibehalten. Sie können den Veranstaltern und den Teilnehmern wichtige Informationen über den Schwierigkeitsgrad der Aufgabenstellungen, über den Zeitplan des Projekts, die Gleichmäßigkeit der Aufgabenverteilung in der Gruppe usw. liefern.

Obwohl die aus didaktischen Gründen vorgenomme Einschränkung der UML mitverantwortlich ist für einige der aufgezeigten Probleme, sehe ich keine Alternative zu diesem Vorgehen. Aus Zeitgründen kann nicht die vollständige UML gelehrt werden. Die Studierenden wären auch nicht in der Lage, alles bei ihrer Modellierung einzusetzen. Der Verzicht auf die wenig geliebten Diagrammtypen kann keine Lösung sein, vielmehr muss versucht werden, ihren Nutzen für den Entwicklungsprozess deutlicher hervorzuheben.

Insgesamt bestätigen die Ergebnisse der Evaluation die Richtigkeit der Strategie, die UML weitgehend einzuschränken und ein möglichst einfaches Vorgehensmodell für das Praktikum auszuwählen.

Danksagung. Ganz herzlich bedanken möchte ich mich bei Heiko van Elsuwe, der im Rahmen seiner Diplomarbeit die Evaluation durchgeführt hat, die einige Schwachpunkte des definierten Prozessmodells offengelegt und Verbesserungsmöglichkeiten aufgezeigt hat.

Literatur

1. Heiko van Elsuwe: Optimierung eines Prozessmodells durch Evaluation, Diplomarbeit am Fachbereich Informatik, Universität Dortmund, 2000.
2. Norman E. Fenton, Shari Lawrence Pfleeger: Software Metrics - A Rigorous & Practical Approach. International Thomson Computer Press, 1996
3. Martin Fowler: UML Distilled - Applying the Standard Object Modelling Language. Addison-Wesley, 1997.
4. Watts S. Humphrey: A Discipline for Software Engineering. Addison- Wesley, 1995.
5. Martin Hitz, Gerti Kappel: UML@Work - Von der Analyse zur Realisierung. dpunkt-Verlag, 1999.
6. Phillipe Kruchten: The Rational Unified Process: An Introduction. Addison-Wesley, 1998.
7. Gunter Müller-Ettrich: Objektorientierte Prozessmodelle. Addison-Wesley, 1999.
8. Mark C. Paulk, Charles V. Weber, Bill Curtis, Mary Beth Chrissis: The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, 1995.
9. James Rumbaugh, Ivar Jacobson, Grady Booch: The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.
10. Heinrich Wottawa, Heike Thierau: Evaluation. Verlag Hans Huber, 1990.

Einsatz von Standardprozessen bei der Gestaltung von Lehrveranstaltungen

Niko Kleiner, Stefan Sarstedt
Abteilung Programmiermethodik und Compilerbau
Fakultät für Informatik, Universität Ulm
89069 Ulm
email: {niko,stefan}@bach.informatik.uni-ulm.de

Zusammenfassung

Dieser Artikel beschreibt, wie mit Hilfe von Standardprozessen ein Softwarepraktikum im Hauptstudium an der Universität Ulm gestaltet wurde. Dieses Praktikum sollte nicht nur Entwicklungstätigkeiten schulen, sondern insbesondere auch auf begleitende Tätigkeiten wie Konfigurations- und Qualitätsmanagement eingehen. Der Leser erhält einen kurzen Überblick über den Aufbau des Praktikums, die Durchführung im SS2000 und einen Einblick in die dabei gewonnenen Erfahrungen.

1 Einleitung

1.1 Motivation und Lehrziele

Auf dem Markt befinden sich inzwischen eine Reihe von Standardprozessmodellen für die Softwareentwicklung [6,7]. Die Industrie verspricht sich von diesen Modellen kompaktes Erfahrungswissen großer Organisationen, das als Vorlage für die eigene Firma dienen kann. Dabei bieten diese Modelle in mehrerlei Hinsicht Hilfe: Einerseits geben sie an, wann etwas zu tun ist (Prozess) und andererseits bieten sie Hilfestellung, wie man diese Einzelschritte am effizientesten ausführt (Methodik). Dabei unterscheiden sich Prozessmodelle in Art und Aufbau der Dokumentation. Das V-Modell 97 [1] trennt z.B. die Aspekte Prozess und Methodik völlig und ist damit methodenunabhängig; der Rational Unified Process [8] (kurz: RUP) hingegen ist methodenspezifisch, indem er direkt eine OO-Methodik integriert.

Die Anwendbarkeit dieser Modelle wird oft diskutiert [6]. In diesem Artikel beschreiben wir, wie wir an der Uni Ulm im SS2000 ein Softwarepraktikum im Hauptstudium mit Hilfe von Standardprozessen gestaltet haben und welche Erfolge und Probleme wir dabei im Hinblick auf die Lehre erfahren haben. Im Einzelnen haben wir dabei den Schwerpunkt auf folgende Fragen gelegt:

1. Sind Standardprozessmodelle als Vorlage für eine Lehrveranstaltung geeignet?
2. Ist die methodische Unterstützung ausreichend?

3. Inwieweit verbessert der Einsatz eines Standardprozesses die Teamarbeit und damit die Qualität der erstellten Dokumente?

Die letzte Frage ergab sich aus dem Gedanken, Teamaspekte in Prozessen mit zu berücksichtigen. Gerade bei verteilter Zusammenarbeit wird diesem Faktor zu wenig Rechnung getragen.

Neben diesen Fragestellungen soll in diesem Artikel auch darauf eingegangen werden, inwiefern diese Lehrform unseren Lehrzielen (siehe Ende des Abschnitts) zuträglich war. Den Studenten sollte eine Möglichkeit geboten werden, umfassend die sogenannten „best practices" der Softwareentwicklung ohne Zeit- und Kostendruck auszuprobieren und in die Lage versetzt werden, sich eine auf eigenen Erfahrungen basierende Meinung über deren Anwendbarkeit bilden zu können. Wir formulierten drei Lehrziele:

1. Es sollte die Notwendigkeit entwicklungsbegleitender Tätigkeiten wie Qualitäts- und Konfigurationsmanagement verdeutlicht und verstanden werden.
2. OO-Modellbildung und Abstraktion sollten geschult werden.
3. Die Grundregeln von Reviews sollten erlernt werden. Insbesondere sollte hierbei erfahren werden, dass nicht nur die technischen Aspekte eines Reviews von Nutzen sind, sondern auch die Auswirkungen auf das Entwicklungsteam selbst.

1.2 Einbettung in das Curriculum

Während des Informatik-Grundstudiums an der Universität Ulm besuchen alle Studenten eine „Einführung in die Konzepte der Objektorientierung und des Softwareentwurfs". Im Anschluss daran wird während des 3. Fachsemesters ein „Softwaregrundpraktikum" durchgeführt. Als Lehrziele stehen dabei die Vermittlung erster Kenntnisse und Erfahrungen bei der methodischen Entwicklung großer Software-Systeme im Team im Vordergrund [9]. Aspekte wie Qualitätssicherung und Konfigurationsmanagement sind zweitrangig.

Als Grundlage für weiterführende Veranstaltungen im Hauptstudium dient die Vorlesung „Softwaretechnik". Sie soll einen möglichst umfassenden Überblick über Themen des Software Engineering geben. Darauf aufbauend gibt es zur Vertiefung Spezialvorlesungen zu den Themen „Requirements Engineering", „Software Qualität" sowie „Management von Softwareprojekten". Zur praktischen Umsetzung des Erlernten dienen schließlich das in diesem Bericht angesprochene Praktikum „Objektorientierte Softwarekonstruktion" sowie das Praktikum „Experimentelles Software Engineering", bei dem es hauptsächlich um die Modellierung des Ablaufs und die Untersuchung von Prüfverfahren eingebetteter Systeme geht [2].

Alle Veranstaltungen stehen sowohl im Rahmen des traditionellen Diplomstudiengangs Informatik als auch des vor kurzem eingeführten, auf einem Leistungspunktesystem basierenden Bachelor-/Master-Studiengangs zur Verfügung [3].

2 Aufbau des Praktikums

In diesem Abschnitt wird der Hintergrund vermittelt, unter dem das Praktikum durchgeführt wurde. Dazu beschreiben wir kurz das Projekt, den von uns vorgegebenen Prozess und die tatsächliche Durchführung während des Semesters.

2.1 Projektbeschreibung LuxOR

Im Vorfeld des Praktikums wurde zunächst nach möglichen Themen für das Projekt gesucht. Das zu erstellende System sollte sich von den üblichen universitären Problemstellungen wie beispielsweise „Literaturdatenbank“ oder „Bibliotheksverwaltung“ abheben und real sein, um die Teilnehmer zu motivieren. Schließlich haben wir uns auf ein Reservierungs- und Verkaufssystem für Kinokarten, genannt LuxOR (Lux Online-Reservierungssystem[1]), entschieden. Die Hauptmotivation für die Entwicklung eines solchen Systems liegt in der Verbesserung des Kundenservices des Kinos, der Möglichkeit der Sicherung eines festen Kundenstamms und einer wesentlichen Entlastung der Kassenmitarbeiter. Einige wichtige Features von LuxOR sind:

- Abrufen des Kinoprogramms über das Internet.
- Graphische Anzeige der Kinosäle und darauf interaktive Auswahl von Plätzen.
- Reservierung und Kauf von Kinokarten über das Internet oder an der Abendkasse.
- Verwaltungsfunktionen für Filme, Kinoprogramm, Kundenstamm etc.

2.2 Prozessübersicht

Aufgrund fehlender Erfahrungsberichte über ein Praktikum, in dem Standardprozesse zum Einsatz kamen, musste die Lehrveranstaltung von Grund auf neu gestaltet werden. Um die Risiken eines solchen Neustarts zu mildern, wurde eine Vorplanungsphase von drei Monaten und ein Probelauf von nochmals zwei Monaten eingeplant und durchgeführt. In der Vorplanung wurde aus dem RUP [8] und der Methode von Craig Larman [4] ein möglichst einfacher, aber vollständiger OO-Entwicklungsprozess extrahiert (genaueres dazu folgt im Abschnitt 0).

Abbildung 1 zeigt die Dokumente, die während des Prozesses zu erstellen waren (aus Platzgründen verzichten wir auf eine detaillierte Beschreibung des Prozesses). Pfeile geben Abhängigkeiten zwischen den Dokumenten an. Zuerst wurde ein Pflichtenheft (ein sogenanntes „Vision“-Dokument in der RUP Terminologie) erstellt, welches eine Problembeschreibung, die Beschreibung der Interessengruppen und Features des zu entwickelnden Systems enthielt. Daraus wurde ein Use Case Modell und Beschreibungen für die einzelnen Use Cases abgeleitet. Zur Analyse wurde ein konzeptuelles Modell erarbeitet, der Ablauf der Use Cases in Sequenzdiagrammen formalisiert und die dadurch gefundenen Systemoperationen mittels Kon-

[1] Lux ist der Name des Zusammenschlusses mehrerer lokaler Kinos

trakten (Beschreibung der Vor- und Nachbedingungen) spezifiziert. Jede Systemoperation wurde schließlich durch eine Kollaboration realisiert, die angab, wie verschiedene Objekte zusammenarbeiten, um die Systemoperation durchzuführen. Aus den Kollaborationen konnte dann ein Klassenmodell abgeleitet werden. Die Kollaborationen und das Klassenmodell beschrieben zusammen das Design.

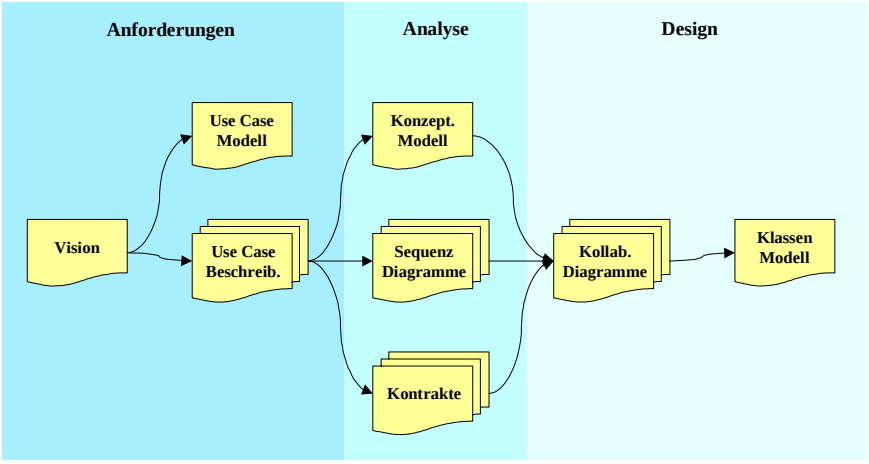

Abbildung 1: Dokumentenfluss

2.3 Organisation und Durchführung

Für die Durchführung des Praktikums standen uns im Sommersemester 13 Sitzungen zur Verfügung. Es wurden nur Modelle erstellt und auf die Implementierung verzichtet, da letztere erfahrungsgemäß zeitraubend ist und nichts zu den in diesem Praktikum angestrebten Lernzielen beigetragen hätte. Dieses Vorgehen wurde von Seiten der Studenten sehr honoriert.

Das Betreuerteam bestand aus drei Doktoranden und drei wissenschaftlichen Hilfskräften (kurz: Hiwis). Es nahmen neun Studenten teil, die in drei Dreiergruppen aufgeteilt wurden und einen Hiwi zur Seite gestellt bekamen. Alle arbeiteten an demselben Projekt.

Aus dem Vorlauf wurde eine Paketaufteilung des Projekts übernommen, so dass jedes Team an einem separaten Paket parallel arbeiten konnte.

Tabelle 1 zeigt die Abbildung des Prozesses auf die 13 Sitzungen (tatsächlicher Verlauf). Die Wochenaufgabe zu einer Sitzung gibt an, welche Aktivitäten mit welchen Ergebnissen im Laufe der Woche bis zur nächsten Sitzung durchzuführen bzw. zu erstellen waren.

Sitzung	Wochenaufgabe	Aufwand (in Std.)
1	Einarbeitung	7-9
2	Interviews, Vision erstellen	7-9
3	Use-Case Modell, Kurzbeschreibungen für Use-Cases	6-8
4	Ablaufbeschreibungen für Use-Cases	8-10
5	Konzeptuelles Modell pro Team, Sequenzdiagramme	6-8
6	Review vorbereiten	3-5
7	Konzeptuelle Modelle zusammenführen, Korrekturen aus Review	4-6
8	Kontrakte	8-10
9	Kollaborationen, Klassenmodell pro Team	10-12
10	Review vorbereiten	4-6
11	Klassenmodelle zusammenführen, Korrekturen aus Review	4-6
12	Abschlusspräsentation vorbereiten	3-5

Tabelle 1: Grober Projektplan (tatsächliche Durchführung)

Das konzeptuelle Modell und das Klassenmodell wurden zunächst von den Gruppen für ihre Pakete separat erstellt (in Wochenaufgabe 5 und 9) und mussten dann zu einem gemeinsamen Paket zusammengeführt werden (in Wochenaufgabe 7 und 11). Die Reviews wurden so platziert, dass jeder Teilnehmer mindestens ein Modell eines anderen Teams inspizieren musste und somit eine Übersicht über die Arbeit der Kommilitonen bekam. Dies sollte die Erstellung des Gesamtmodells erleichtern. So waren im ersten Review (7. Sitzung) das Use Case Modell, die Use Case Beschreibungen, die Formalisierungen davon in Form von Sequenzdiagrammen und die von den einzelnen Gruppen erstellten konzeptuellen Modelle zu inspizieren. Im zweiten Review (11. Sitzung) wurden Kontrakte, Kollaborationen und die einzelnen Klassenmodelle untersucht.

Als Konfigurationsmanagementwerkzeug setzten wir Rational ClearCase, für das Änderungsmanagement Rational ClearQuest und zur Modellierung Rational Rose ein.

3 Erfahrungen

3.1 Vorplanung und Probelauf

Sowohl die Vorplanung als auch der Probelauf haben sich in Bezug auf die resultierende Qualität der Lehre gerechnet. Rückblickend waren dabei folgende Erfahrungen am wichtigsten:

- Bei einem völlig neu aufzusetzenden Praktikum ist es zu Beginn beinahe unmöglich, die Aufwände für einzelne Aktivitäten abzuschätzen. Durch einen Probelauf erhält man erste Ausgangsdaten. Dadurch, dass wir im Probelauf den Hiwis den

Prozess und die angewandten Methoden näher bringen mussten, erhielten wir auch Hinweise, an welchen Stellen besondere Verständnisschwierigkeiten auftreten und konnten dies bei der nachfolgenden Planung bereits berücksichtigen.
- Ein vom gesamten Betreuerteam durchgeführter Probelauf (ohne Vorplanung) bringt ein gemeinsames Verständnis des Projekts, des Prozesses und der Methoden. Widersprüchliche Aussagen und Meinungen von der Betreuerseite werden dadurch während der Durchführung des Praktikums von vornherein vermieden.
- Die Werkzeugumgebung ist vor dem Praktikum bereits getestet. Damit wird unnötige, durch Probleme mit einem Werkzeug verursachte Zeitverschwendung vermieden.

3.2 Durchführung

Bei der Durchführung hat sich die parallele Erstellung von Modellen als Hauptproblem herausgestellt (Konzeptuelles Modell, Klassenmodell, siehe Abschnitt 0). Obwohl Reviews vor einem Zusammenführen von Teilmodellen zu einem Ganzen so platziert und organisiert wurden, dass jeder Teilnehmer mindestens ein Modell eines anderen Teams inspizieren musste und damit genügend Wissen für die Erstellung des Gesamtmodells haben sollte, gelang es den Teilnehmern nicht, ein homogenes und verständliches Gesamtmodell zu erstellen. Wesentliche Mängel waren dabei:

- Es wurde kein gemeinsames Abstraktionsniveau gefunden.
- Die Teilnehmer haben zu wenig auf Verständlichkeit geachtet und waren zu sehr auf eine möglichst vollständige Beschreibung aus.
- Es wurden Teile aus den Ursprungsmodellen einfach weggelassen, weil sie bei der Zusammenführung hinderlich waren (diese wurden in späteren Phasen dann wieder vermisst).
- Die Teilnehmer haben bei Überlappungen um Positionen gefeilscht, anstatt die Vereinigung aller Teilmodelle als gemeinsames Ziel zu begreifen.

Diese Schwierigkeiten traten im Probelauf nicht auf, sondern erst bei einer parallelen Arbeitsaufteilung. Auch Konfigurationsmanagement und ein Notationstreue erzwingendes Werkzeug wie Rose können diese Probleme nicht lösen, da diese semantischer und nicht syntaktischer Natur sind.

3.3 Sitzungen

Während der wöchentlichen Sitzungen wurden mehrere Workshops durchgeführt. Wir haben festgestellt, dass die Diskussion in der Gruppe schon bei 9 Personen eher zäh verlief. Deshalb änderten wir im Laufe des Praktikums die Organisation solcher Workshops: Jede Gruppe musste einen temporären Teamleiter bestimmen, der jeweils alleine zur Sitzung kam. Die Teamleiter wurden dann geschult und mussten zusammen mit ihrem Hiwi das Gelernte an das eigene Team weitergeben. In jeder Sitzung wechselte die Teamleiterrolle. Von Seiten der Studenten wurde diese Form als motivierend und teamstärkend empfunden.

3.4 Prozessgestaltung

Rational Unified Process

Zunächst war geplant, als Prozessvorlage auf das V-Modell 97 zurückzugreifen, was sich jedoch schnell als nicht handhabbar herausstellte. Insbesondere ist eine Anpassung und Durchführung für ein konkretes Projekt unserer Meinung nach nur mit Werkzeugunterstützung effizient zu schaffen. Schließlich haben wir uns entschlossen, eine eigens angepasste Version des Rational Unified Process zu verwenden. Dabei haben uns die folgenden Punkte die Gestaltung des Praktikums erleichtert:

- Die HTML-Dokumentation erleichtert das Finden von Informationen wesentlich.
- Alle wichtigen Dokumente liegen in Form von Templates vor (z.B. die Use Case Beschreibungen), wobei neben der Struktur auch Hinweise auf die jeweiligen Inhalte der einzelnen Abschnitte vorhanden sind. Die Templates waren besonders hilfreich, da die Teilnehmer sich dadurch keine Gedanken über die Struktur machen mussten. Des weiteren wurden die Reviews (vgl. Abschnitt 0) durch gruppenübergreifend standardisierte Dokumente wesentlich vereinfacht.
- Es werden Arbeitshilfen in Form von Anleitungen für Workshops, Interviews sowie Review- und Brainstormingsitzungen gegeben. Darauf haben wir insbesondere beim Finden der Use Cases zurückgegriffen.

Während sich der RUP in den frühen Entwicklungsphasen (speziell bei der Anforderungserfassung und -dokumentation) als sehr hilfreich erwiesen hat, stellte sich die weitere Verwendung in der Analyse- und Designphase als zusehends schwierig heraus. Dies hatte u.a. folgende Gründe:

- Die Beschreibung der Analyse- und Designphase wurde als nicht mehr hinreichend praktikable Hilfestellung empfunden. Die Beispiele in der RUP-Dokumentation sind nicht durchgängig und größtenteils zu trivial.
- Als verwirrend hat sich im Verlauf von Vorbereitungs-Workshops die Unterscheidung der Analyse-Objekte in Boundary-, Controller- und Entity-Objekte herausgestellt. Diese Differenzierung ist unserer Meinung nach für den Aufbau eines Verständnisses über objektorientierte Analyse und Entwurf kontraproduktiv, da die Gefahr besteht, dass z.B. Controller-Objekte zu einer eher funktionsorientierten und Entity-Objekte zu einer datenorientierten Sichtweise des Systems verleiten. Hinweise auf die Wichtigkeit einer adäquaten Zuweisung von Verantwortlichkeiten werden im RUP nicht gegeben.

Nicht zu vernachlässigen ist auch die enorme Einarbeitungszeit für einen so komplexen Entwicklungsprozess. Aufgrund der Nachteile wurde im Verlauf der Planungsphase klar, dass eine Verwendung der Analyse- und Designkapitel des RUP für unsere Zwecke nicht in Frage kam.

Larman

Für die weiteren Phasen erwies sich das von Craig Larman vorgeschlagene Vorgehensmodell [4] als sinnvoll. Das Buch enthält eine umfangreiche und leicht verständliche Einführung in objektorientierte Analyse und Design. Alle Aktivitäten werden anhand eines durchgängigen Beispiels erläutert. Besonders nützlich für die Vorbereitung und Durchführung unserer Lehrveranstaltung waren folgende Punkte:

- Larman geht auf die unterschiedlichen Sichtweisen von Use Case Beschreibungen ein. Dies war insbesondere wichtig, weil anfangs nicht klar war, ob überhaupt Oberflächen-Aspekte in Use Cases beschrieben werden sollen. Den richtigen Abstraktionsgrad bei Use Cases zu finden war ein oft diskutiertes Problem während der Vorbereitung des Praktikums.
- Als ideal haben sich Larman's „GRASP Patterns" (General Responsibility Assignment Software Patterns) erwiesen. Die Patterns geben Hinweise, wie Verantwortlichkeiten „richtig" (im OO-Sinne) auf die Objekte verteilt werden.

Die Beschreibung der Anfangsphasen der Softwareentwicklung ist bei Larman relativ dünn, weshalb dieser Teil fast vollständig vom RUP übernommen wurde.

3.5 Reviews

Als Qualitätssicherungsmaßnahme wurden begleitend zum Praktikum Reviews durchgeführt. Dort mussten sich die Teilnehmer mit den Meinungen anderer zu ihren Dokumenten und Modellen auseinandersetzen, was auch im Feedback als besonders positiv empfunden wurde. Durch ein Rotationsprinzip war jeder Teilnehmer einmal Inspektor bzw. Dokumentenersteller, die Inspektoren waren grundsätzlich immer aus den anderen Praktikumsgruppen. Reviews waren aus den folgenden Gründen sehr hilfreich für die Teilnehmer und den Verlauf des Praktikums:

- Durch die Sitzungen konnte immer festgestellt werden, ob die Qualität der Dokumente ausreichend für die nächsten Prozessschritte war, oder ob eine Überarbeitung notwendig wurde.
- Die Reviews halfen den Inspektoren, ein Verständnis über die Arbeit der anderen Gruppen und somit über das Gesamtsystem aufzubauen.
- Das Feedback der Inspektoren wiederum gab dem Dokumentenersteller nützliche Hinweise auf Fehler und Optimierungsmöglichkeiten.
- Sie waren eines der wenigen Mittel zur Einzelbewertung der Teilnehmer.

Reviews sollten formal, organisiert und mit Moderatoren (Praktikumsbetreuer oder Hiwis) durchgeführt werden. Von den Studenten selbst durchgeführte Reviews verfehlen meist die inhaltlichen Ziele [2].

4 Fazit

Rückblickend können wir die zu Anfang aufgeworfenen Fragen nun folgendermaßen beantworten:

Der Rational Unified Process half einen Rahmen für das Praktikum zu stecken (was sind die Inhalte, welche Workshops etc.) und bot für einzelne Aktivitäten, wie z.B. den Use Case Workshop, gute Checklisten und Hinweise. Außerdem erleichtern die Dokumentenvorlagen (Templates) die Vorbereitung wesentlich. Allerdings ist der Aufwand, um einen ersten, übersichtlichen Prozess zu gestalten, zu hoch. Es werden auch keine Hinweise gegeben, wie man einen Minimalprozess aus dem generischen Modell gewinnen kann. Prozessmodelle würden vielleicht eine größere Akzeptanz erfahren, wenn sie gerade umgekehrt aufgebaut wären, d.h. ein Minimalprozess vorgegeben würde und an entsprechenden Stellen Hinweise auf problemspezifische Erweiterungsmöglichkeiten gegeben würden.

In der methodischen Unterstützung ist der RUP eher dünn. Insbesondere kommt man nach der Use Case Modellierung kaum mehr ohne weiterführende Dokumentation voran. An dieser Stelle schließt Larman sehr gut an den RUP an.

Ein Standardprozessmodell hilft vorab nicht unbedingt, Teamarbeit zu unterstützen und besser zu koordinieren. Erst wenn ein schlanker, konkreter Prozess erstellt und an alle in einer verständlichen Form kommuniziert wird, hilft er den Beteiligten, eine bessere Übersicht zu bekommen und verbessert dadurch auch die Zusammenarbeit im Ganzen. Allerdings bleiben Probleme der Art, wie sie in Abschnitt 0 und 0 diskutiert wurden, bestehen.

Die gewählte Lehrform war für die Erreichung der Lehrziele erfolgreich. Durch den Einsatz von Konfigurationsmanagement und Reviews wurde den Studenten hinreichend klar, dass Softwareentwicklung im Großen ohne entwicklungsbegleitende Tätigkeiten nicht zu schaffen ist. Für die Schulung von OO-Modellbildung und Abstraktion hat der RUP selbst keine Hilfe geleistet. Larman's GRASP-Patterns schlagen unserer Meinung nach für diesen Zweck die richtige Richtung ein. Durch den Einsatz von Reviews und die damit verbundene Beschäftigung der Teilnehmer mit der Arbeit der Anderen werden oft erst die Grundlagen für Diskussionen gelegt, die die wesentlichen Probleme an den Tag befördern.

Die Organisation des Praktikums hat sich im Großen und Ganzen also bewährt und ausreichend Antworten für unsere Fragen abgeworfen. Mit dem nun vorhandenen Prozess, den Vorlagen und den erhobenen Aufwandsdaten können wir für folgende Semester relativ schnell und genau ein Folgepraktikum planen. Auch wenn die Aufspaltung eines Projekts und die Bearbeitung von Teilsystemen mit je einer Gruppe (vgl. auch Abschnitt 0) wesentlich mehr Koordinationsaufwand und Probleme mit sich bringt (siehe insbesondere Abschnitt 0), würden wir in Folgeveranstaltungen wieder diese Form wählen. Zwar leidet die Qualität des Ergebnises darunter (z.B. inhomogene Modelle), jedoch erkennen die Studenten unserer Meinung nach nur so diese Probleme und lernen daraus nachhaltig. Allerdings würden wir weniger Be-

treuer einsetzen, da 6 Betreuer auf 9 Studenten mehr Koordinationsaufwand als Nutzen ist. Als bleibende Schwierigkeiten sehen wir den Umfang und die Detailliertheit der begleitenden Literatur und insbesondere einen objektiven Maßstab für eine Einzelbewertung.

Literatur

1. Dröschel, W.; Wiemers, M.: Das V-Modell 97. Oldenbourg Verlag, München, 1999.
2. Ernst, D.; Schulte, W.; Houdek, F.; Schwinn, T.: Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme. Ulmer Informatik Berichte Nr. 97-13, 1997.
3. Gehring, W.: Ein Rahmenwerk zur Einführung von Leistungspunktesystemen. Ulmer Informatik Berichte Nr. 2000-04, 2000.
4. Larman, C.: Applying UML and Patterns. Prentice Hall, 1997.
5. Latinen, M.; Boddie, J.: Scaling Down Is Hard to Do / Do We Ever Really Scale Down?. IEEE Software, Band 17, Nummer 5, S.78-81, 2000.
6. Müller-Ettrich, G.: Objektorientierte Prozessmodelle. Addison-Wesley, 1999.
7. Noack, J.; Schienmann, B.: Objektorientierte Vorgehensmodelle im Vergleich. Informatik Spektrum 22, S.166-180, 1999.
8. Rational Unified Process. http://www.rational.com
9. Schwarz, M.; Schulte, W.: Realistische Aufgabenstellungen für das Softwaregrundpraktikum. Berichte des German Chapter of the ACM, Band 48, S.94-104, B.G. Teubner, Stuttgart, 1997.

2000, 313 Seiten, gebunden
DM 79,00 / öS 577,00 / sFr 71,50
ISBN 3-932588-83-5

»Schon jetzt lässt sich sagen, dass es sich um einen wertvollen, wissenschaftlich fundierten, kritisch-konstruktiven und vor allem praxisnahen Beitrag zur gegenwärtigen Diskussion um ein erfolgreiches IT-Projektmanagement handelt.
Wohltuend wirkt die sehr gute logische-strukturelle, didaktische und instruktiv-methodische Anlage des Buches, ohne dass dadurch das wissenschaftliche Profil beeinträchtigt wird.«
(Dr. Wandschneider)

H.-J. Etzel, H. Heilmann
R. Richter (Hrsg.)

IT-Projekt-management

Fallstricke und Erfolgsfaktoren

Erfahrungsberichte aus der Praxis

Wann gelingen IT-Projekte, wann nicht? Darum geht es in den Erfahrungsberichten dieses Buches, in denen reale Projekte beschrieben werden – so wie sie wirklich verlaufen sind, mit ihren Erfolgen und Misserfolgen. Vorgestellt werden typische Projektsituationen, wie Einsatz von Standardsoftware, Individualentwicklungen, Softwaremigrationen oder die Hinzunahme externen Know-hows. Alle Berichte werden von Herausgeberseite mit Blick auf die folgenden Fragen kommentiert:

– Welche Fehler wurden gemacht, wie wurden sie gelöst?
– Wurden die Probleme rechtzeitig erkannt?
– Wie hätte man den Misserfolg vermeiden können?
– Was lässt sich daraus für künftige IT-Projekte lernen?

Ein einführender Beitrag gibt außerdem einen zusammenhängenden Überblick über die Thematik.

dpunkt.verlag

Ringstraße 19 • 69115 Heidelberg
fon 0 62 21/14 83 40
fax 0 62 21/14 83 99
e-mail hallo@dpunkt.de
http://www.dpunkt.de

2000, 510 Seiten, gebunden
DM 99,00 / öS 723,00 / sFr 90,00
ISBN 3-932588-77-0

»...eine Fundgrube für Fachwissen und Erfahrungen zu einem hochaktuellen Informatik-Thema. Ganz eindringlich erinnert es Informatiker (und Bibliothekare) an ihre Verantwortung als Dienstleister für die Informations- und Wissensgesellschaft.«
(Prof. Dr. Wolffried Stucky, Universität Karlsruhe, ehemaliger Präsident der Gesellschaft für Informatik/GI)

Albert Endres, Dieter W. Fellner

Digitale Bibliotheken

Informatik-Lösungen für globale Wissensmärkte

Digitale Bibliotheken sind die Verwalter und Vermittler von wissensrelevanter Information im globalen Netz. Sie stellen in zunehmendem Maße die primäre Wissensquelle für wissenschaftliches und technisches Arbeiten dar. Auf Firmenebene bilden digitale Bibliotheken die Basistechnologie für das Wissensmanagement. Die angebotenen Inhalte erstrecken sich auf alle Medienformen und decken alle Fachgebiete ab.
Das Buch gibt einen umfassenden Überblick über verschiedene Formen digitaler Bibliotheken und vermittelt das notwendige Wissen für Anbieter und Nutzer. Ausführlich werden die Technologien zur Speicherung, Sicherung, Darstellung und Nutzung digitaler Dokumente dargestellt. Eingegangen wird außerdem auf die wirtschaftlich attraktive Gestaltung der Angebote einschließlich der effizienten Abrechnung aller Dienstleistungen.

dpunkt.verlag

Ringstraße 19 • 69115 Heidelberg
fon 0 62 21/14 83 40
fax 0 62 21/14 83 99
e-mail hallo@dpunkt.de
http://www.dpunkt.de