Erfahrungen mit Pair-Teaching für Software Engineering: Kooperation von Hochschule und Industrie

Axel Böttcher Matthias Utesch* und Austin Moore***

* Hochschule für angewandte Wissenschaften (FH) München, Fakultät für Informatik und Mathematik
Lothstr. 34, D-80335 München

** namics (deutschland) gmbh, Watzmannstraße 1a, D-81541 München

Abstract

In diesem Artikel stellen wir eine Variante des Team Teaching vor, die in Anlehnung an das Pair Programming als Pair Teaching bezeichnet wird. Während der vergangenen zwei Jahre setzten wir Pair Teaching im Rahmen unserer Software-Engineering- Ausbildung in dem Kurs Software-Architektur ein.

Wir lehren Software-Architektur mit verteilten Rollen: Ein Hochschuldozierender als „Theoretiker" und ein Senior Software Developer aus der Industrie als „Praktiker". Diese gestalten gemeinsam den Kurs bestehend aus einer Vorlesung und einem Praktikum.

Angeregt durch das Rollenspiel der beiden Dozierenden haben wir Synergie-Effekte erlebt. Die Studierenden fühlten sich besser betreut, und schätzten insbesondere die „super gute" Praxisnähe.

Gerade im Zuge der Umsetzung des Bologna-Prozesses sind praxisbezogene Lehrmuster im Hinblick auf neue Steuerungsmodelle für den Einsatz von Lehrleistung zu diskutieren.

1 Einleitung

Beginnend mit dem Wintersemester 2001/2002 starteten wir als eine der ersten Fakultäten für Informatik die Umsetzung des Bologna-Prozesses. Heute haben ca. 400 Absolventen ihren Bachelor und 60 Absolventen ihren Master erhalten. Die neuen Abschlüsse waren für uns auch Anlass, unsere Pädagogik und Didaktik zu reflektieren.

Der Kurs Software-Architektur ist Teil der Software-Engineering-Ausbildung im Bachelor-Studiengang Informatik an der Hochschule (FH) München. Der Kurs besteht aus einer klassischen Vorlesung und einem Praktikum mit je zwei Semesterwochenstunden. Lernziel ist, dass die Studierenden fähig sind, komplexe Software-Systeme

zu entwickeln, zu warten und zu betreiben. Sie lernen und trainieren, die dazu erforderlichen Werkzeuge einzusetzen.

Im didaktischen Design des Kurses [Flechsig/Haller 1987] gilt es unserer Erfahrung nach vor allem zwei Fragen zu beantworten:

1. Wie lehren wir in unserem Kurs die relevante Theorie über Architektur großer und komplexer Software-Systeme?
2. Wie lernen unsere Studierenden, sich nach Abschluss des Studiums in diesem dynamischen Gebiet in der Praxis zu bewähren [Schaller 1979]?

Unsere Antwort auf diese Fragen ist eine innovative Variante des Team Teaching, die wir in den vergangenen Semestern erprobt haben [Huber 2000]. Kern unseres didaktischen Designs ist das Lehren mit verteilten Rollen: Theoretiker und Praktiker. Die Rolle des Theoretikers wird von einem Hochschullehrer, die Rolle des Praktikers von einem Senior Software-Entwickler aus der Industrie wahrgenommen. Auf diese Weise wird der Unterricht bereichert durch eine wertvolle, nicht-deterministische kommunikative Komponente [Rustemeyer 1985]. Unsere Lernumgebung ist multimedial [Kerres 1998]: Der Praktiker entwickelt live am Rechner in der Software-Entwicklungsumgebung Eclipse. Der Theoretiker kommentiert diesen Vorgang, formuliert Fragen aus studentischer Sicht an den Praktiker (Experteninterview, siehe [Waldherr/Walter 2008]) und regt damit die Kommunikation und Interaktion mit den Studierenden an. Zeitweise haben wir auch gemeinsam entwickelt und damit Pair Programming vorgelebt.

2 Der Spagat zwischen Theorie und Industrie-Praxis

Talented people are the most important element in any software orgnization …. The better and more experienced they are, the better the chance of producing first-class results. [Humphrey 1990]

Talent benötigt mehrere Jahre Erfahrung, um sich zu entwickeln. Schlüssel dafür ist auch ein Umfeld mit guten und kooperativen Kollegen, von denen man lernen kann und mit denen gemeinsam die Fähigkeiten und Fertigkeiten ständig verbessert werden können. Technologien und Werkzeuge entwickeln sich zudem rasant weiter; gute Entwickler bleiben in dieser Landschaft auf dem Laufenden [Lehner 1991].

Auch der Hochschullehrer muss die Herausforderung bewältigen, in dieser Landschaft auf dem Laufenden zu bleiben. Selbst wenn Hochschullehrer aus dem Umfeld der industriellen Software Entwicklung kommen, sind sie in ihrem Alltag an der Hochschule oft verhältnismäßig weit davon entfernt. Auch dann, wenn man in Forschungsprojekten oder –Kooperationen mit Software-Entwicklung zu tun hat, kommt die Praxis häufig notgedrungen zu kurz.

Andererseits steht der Praktiker aus der Industrie, wenn er lehren will, unserer Erfahrung nach vor der Herausforderung, seinen Stoff didaktisch aufzubereiten. Aspekte dabei sind:

- Aufbereiten des Stoffes
- Finden geeigneter Übungsaufgaben
- Betreuung von Praktika
- Formulieren und korrigieren von Prüfungen

Dieses gerade ist die Kernkompetenz des Hochschullehrers, der im Laufe der Zeit sehr viel Erfahrung gewonnen und ein gutes Gefühl dafür entwickelt hat, wie man die Lernprozesse der Studierenden am besten befördern kann.

In der Vergangenheit haben wir mit zwei verschiedenen Lösungsansätzen gearbeitet: Eine Möglichkeit ist, dass ein Mitarbeiter aus einem Unternehmen einen Lehrauftrag für die Durchführung eines kompletten Kurses übernimmt. In der Praxis ist es oft schwierig, geeignete Lehrbeauftragte zu finden, die bereit sind, den großen Aufwand auf sich zu nehmen, den es bedeutet, jede Woche eine Vorlesung mit Praktikum abzuhalten – neben ihrer eigentlichen Industrie-Tätigkeit.

In der zweiten Variante haben wir die klassische Unterrichtsform, durchgeführt von einem Dozierenden, durch einen gelegentlich teilnehmenden Praktiker aus einem Unternehmen bereichert. Unsere Erfahrungen damit sind nicht durchgängig positiv, ähnlich wie in [Waldherr/Walter 2008, S. 30] beschrieben. Hinzu kommt, dass es für diesen externen Dozenten nicht so gut möglich ist, seine didaktischen Fähigkeiten weiter zu entwickeln.

Als Ergebnis dieser Erfahrungen entstand unser didaktisches Design in Analogie zum *Pair Teaching* von Andersson und Bendix [Andersson/Bendix 2005].

3 Pair Teaching

The child who receives a hammer for Christmas will discover that everything needs pounding [Weinberg 1985]

Wir wollen unseren Studierenden bewusst das Werkzeug „Teamarbeit" im Software Engineering beibringen und leben dieses Element deshalb in unserer Lehre vor. Ein etablierter Ansatz dieser Teamarbeit ist das Pair Programming. Es wird z.B. für die Praxis beschrieben in [Beck 2000] und hinsichtlich verschiedener Effizienzkriterien wissenschaftlich untersucht in [Williams 2000]. Die Komplexität, die in unserem Fachgebiet Software Engineering aus rasantem Fortschritt sowohl in Theorie als auch der Praxis entsteht, wird personalisiert durch die Rollen „Theoretiker" und „Praktiker" und durch deren live-Kommunikation und Interaktion dargestellt und damit im Sinne des Systems-Thinking-Ansatzes nach Senge bewältigt [Senge 1990]. Jede der Rollen wird durch einen Lehrenden personalisiert [Andersson 2005]. Wir haben diese Methode insofern erweitert, als wir je einen Dozierenden aus Hochschule und aus der Industrie zusammen bringen.

Mediale Präsentation von Inhalten durch zwei Personen ist eine in verschiedenen Bereichen durchaus übliche Praxis:

- Fernsehsendungen, die man dem Bildungsfernsehen zuordnen kann, werden gelegentlich von einem Paar bestehend aus einem Moderierenden sowie einem Experten gestaltet. Ein Beispiel hierfür ist die c't magazin-Sendung (siehe www.cttv.de). Hier spielt der Moderator die Rolle eines Lernenden. Ähnlich verhält es sich in Sendungen des IT-Kompaktkurses im Bildungskanal BR-Alpha (siehe www.br-alpha.de).
- Die Pro- und Contra-Debatte. Diese wird vor allem als Moderationstechnik eingesetzt. Sie wird zudem oft in Printmedien eingesetzt, um gegensätzliche Standpunkte zu verdeutlichen.

Wir beschreiben im Folgenden drei Beispiele für unsere Umsetzung von Pair Teaching im Kurs Software-Architektur.

3.1 Der Kurs Software Architektur

Unser Kurs Software-Architektur besteht aus einer Vorlesung und einem Praktikum zu je zwei Semesterwochenstunden. Unser didaktisches Design in diesem Kurs orientiert sich an [Bass et al. 2003]. Wir behandeln zunächst Qualitätsmerkmale von Software [Liggesmeyer 2002]. Mit dem Qualitätsmerkmal Funktionalität sind die Studierenden bereits vertraut und sie betrachten dies als das zentrale Merkmal. Deshalb führen wir – nach einer theoretischen Einführung – im Praktikum einige Refactoring-Übungen durch, um die Bedeutung von Merkmalen wie Testbarkeit, Erweiterbarkeit (Modifiability) und Benutzerfreundlichkeit (Usability) am praktischen Beispiel zu verdeutlichen.

Im weiteren Verlauf der Veranstaltung müssen die Studierenden Plugins für Eclipse[1] entwickeln. Ein vorhandenes Framework zu verwenden, hat den Vorteil, dass bereits komplexe Strukturen vorhanden sind, die durch eigenen Code erweitert werden müssen [Böttcher 2006]. Hinreichend komplexe Software selbst zu entwickeln, um daran Architektur studieren zu können, erschien uns zu wenig zielführend für ein Praktikum mit nur zwei Stunden pro Woche. Aufgrund des Pattern-orientierten Designs von Eclipse [Gamma/Beck 2004] lässt sich Software-Architektur anhand des Einsatzes von Patterns sehr praxisnah unterrichten.

3.2 Beispiel 1: Vorlesung zum Thema „Refactoring"

Refactoring ist eine Basistechnik, um die Qualität von Software zu verbessern. Ziel der Vorlesung zum Refactoring ist, die Studierenden mit dem theoretischen Werkzeug und der praktischen Vorgehensweise beim Refactoring vertraut zu machen und Ihnen ein Gefühl zu geben, für die Beurteilung existierenden Quellcodes.

In der Refactoring-Vorlesung wird das Rollenmodell Theoretiker/Praktiker wie folgt umgesetzt:

Der Praktiker führt die Software-Entwicklung mit einer Entwicklungsumgebung vor, nimmt damit also die Rolle des Drivers im Pair Programming wahr [Walther

[1] www.eclipse.org

2005]. Der Theoretiker hat das Auditorium im Blick und gibt zusätzliche Kommentare ab, die das Verständnis erleichtern. Gleichzeitig spielt er die Rolle des Navigators, also des zweiten Pair Programmierers. Wie der Navigator beim Pair Programming Abstand zur aktuellen Arbeit hat, kann er die aktuelle Strategie zur Problemlösung kommentieren, gemeinsam mit dem Auditorium reflektieren, überprüfen und verbessern. Probleme werden so im Dialog zwischen Navigator und Driver in Interaktion mit dem Auditorium gelöst.

Unser Ansatz vermeidet damit die Ermüdung der Studierenden, die das Vorführen von Software-Entwicklung mit einer Entwicklungsumgebung für die Studierenden oft mit sich bringt. Denn insbesondere das dauernde Wechseln von Editor-Fenstern erfordert hohe Konzentration der Studierenden. Ebenso ermöglichen wir den Dozierenden die gleichzeitige Konzentration auf die Entwicklungsumgebung sowie das Auditorium. Unsere Vorlesung mittels Pair Teaching ist somit zugleich eine praktische Demonstration von Pair Programming.

3.3 Beispiel 2: Praktikum zum Thema „Refactoring"

Die Studierenden sollen in der Lage sein, Code zu beurteilen und eine gewisse Basisqualität herzustellen. Lernziel ist es, das Gefühl der Studierenden zu entwickeln und Erfahrung dafür zu geben, wo Refactoring beendet werden kann, weil weitere Schritte nicht mehr wirtschaftlich sind, oder weil es dafür keinen Kunden mehr gibt.

Ausgangspunkt ist ein Java Programm, welches das Spiel Vier-Gewinnt realisiert. Dieser von einem Programmier-Anfänger geschriebene Code ist so offensichtlich von schlechter Qualität, dass die Notwendigkeit von Verbesserungen unmittelbar ins Auge springt. Die erste Übung besteht darin, die Software testbar zu machen. Dazu geben wir Unit-Tests vor, die aber erst nach einigen Basismodifikationen überhaupt zum Laufen kommen. Diese Abweichung von der reinen Lehre eines Test-Driven Ansatzes nehmen wir bewusst in Kauf, da die Aufgabenstellung, eine Software testbar zu machen, durchaus praxisrelevant ist. Als Basis für die Beurteilung haben wir einen Katalog mit Punkten formuliert, der sich abprüfen lässt, der aber den Studierenden nicht vorab bekannt gegeben wird.

- Wir erwarten die Beseitigung statischer Methoden, soweit möglich.
- Wir erwarten eine Umstrukturierung gemäß objektorientierter Prinzipien.
- Die Software enthält einen Fehler, der im Refactoring zutage tritt.
- Wir erwarten die Beseitigung duplizierten Codes.
- Wir erwarten eine weitgehende Testabdeckung, die mit dem Coverage-Werkzeug Emma[2] gemessen wird.

Nach rein akademischer Ausbildung neigen die Studierenden dazu, Lösungen zu bauen, die unnötig kompliziert sind, wie das Patterns-Happy-Beispiel aus [Kerievsky 2004, S. 23 ff]. Der Dozierende in der Rolle „Praktiker" bremst hier die Studierenden rechtzeitig, so dass sie nicht eine vollständig flexible Lösung erarbeiten, die allen

[2] `emma.sourceforge.net`

Prinzipien objektorientierten Designs genügen, aber unnötig viel Aufwand erfordern (dies ist als yagni-Prinzip – you ain't gonna need it – bekannt). Der „Praktiker" wird dies auch aus seiner Erfahrung heraus sehr genau und glaubhaft begründen können. Der „Theoretiker" erörtert das Problem des Abbruchs von Refactoring mit dem „Praktiker" und den Studierenden. An dieser Stelle wird Bewusstsein dafür geschaffen, dass sich keine eindeutigen Aussagen formulieren lassen, vielmehr „Gespür" erforderlich ist.

3.4 Beispiel 3: Praktikum zum Thema „Eclipse Plugins"

Schwerpunkt unseres Praktikums „Eclipse Plugins" ist es, erste persönliche Erfahrungen mit komplexer Software zu sammeln: Die Studierenden sollen in der Lage sein, komplexe Software unter Berücksichtigung von best practices agiler Ansätze selber zu schreiben.

Ein erstes Beispiel zeigt unser Lernpfad für den Einsatz von Patterns in komplexer Software. In unserem Fall gilt es, ein Plugin für das Verwalten und Abspielen von Bitströmen im MP3-Format zu erstellen. An einer Stelle kommt dabei das Composite Pattern ins Spiel. Fig. 1 skizziert ein Klassendiagramm für das Composite Pattern [Gamma et al. 1995]. Das Bild verdeutlicht die beiden möglichen Platzierungen der Methoden für die Verwaltung der Kinder (`addChild, getChildren` etc.). Beide Lösungen haben spezifische Vor- und Nachteile; vgl. Dazu etwa [Gamma et al. 1995] und [Gamma/Beck 2004].

Ein Dozierender kann dieses Pattern erläutern – aber: unser Lernziel ist, den Studierenden nicht Musterlösungen, sondern in Diskussionskultur beizubringen, die Fähigkeit zu vermitteln, abzuwägen und zu entscheiden. Deshalb diskutieren wir dieses Pattern, seine möglichen Ausprägungen und Konsequenzen im Dialog zweier Dozierender. So aktivieren wir die Studierenden, mit zu diskutieren.

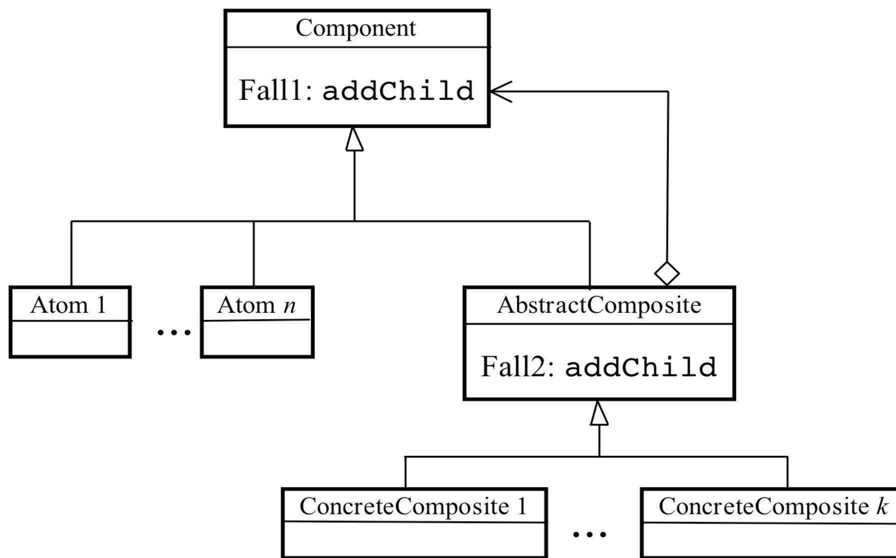

Fig. 1: Skizze eines Klassendiagramms zum Composite Pattern. Die Methoden zur Verwaltung der Kinder können entweder in Component oder in AbstractComposite implementiert werden (mit Fall1 und Fall2 gekennzeichnet).

Ein zweites Beispiel ist unser Lernpfad für die Schlüsselwörter `volatile` und `synchronized`. Die Rolle des Theoretikers ist, den Studierenden dafür die Definition und Syntax zu vermitteln. Reichen aber diese Kenntnisse aus, damit unsere Studierenden konkrete Probleme lösen können? Hier zeigt der Praktiker aus seinem Alltag des Software Engineering beispielhafte Anwendungen auf. Gemeinsam diskutieren dann Theoretiker und Praktiker die jeweiligen Auswirkungen dieser Schlüsselwörter in konkreten Programmen. Anschließend üben die Studierenden selbstständig. Die Dozierenden unterstützen gemäß der Idee des Pair Programming.

Grundsätzlich macht das didaktische Design des Praktikums es möglich, den Studierenden am konkreten Beispiel zu zeigen, dass in der Arbeit mit einer Entwicklungsumgebung immer wieder nicht vorhersehbare Probleme auftreten können. Wir können so aufzeigen, dass Werkzeuge oft Spezifika können, die sich einem logisch denkenden Menschen nicht unmittelbar erschließen und dessen Bedienung Spezialkenntnisse durch praktische Erfahrung erfordert.

Hier beweist sich unser Pair Teaching durch die Kommunikation zwischen den Dozierenden und durch die Summe der Erfahrungen eines Theoretikers und eines Praktikers. Wenn solche Probleme auftreten, kann einer der beiden Dozierenden den Praktikumsbetrieb fortsetzen, während der andere versucht, diese Einzelprobleme zu lösen und zu dokumentieren.

4 Bewertung

Bei der Bewertung unseres didaktischen Designs des Pair Teaching orientieren wir uns im Grundsatz an den Punkten von [Williams 2000]. Wir verwenden auch dessen englische Bezeichnungen. Diese Kriterien wurden gewählt, da wir unsere Evaluierung derzeit auf qualitative Aussagen und den Dialog aller am Lehr-/Lernprozess Beteiligten stützen, und erst im nächsten Schritt eine passende quantitative Evaluierung entwickeln werden. Wir bewerten also das Pair Teaching mit Hilfe folgender Punkte:

Pair Pressure Gemeinsames Arbeiten am selben Projekt erhöht die Motivation der Lehrenden, oder wie Williams formuliert: „*they do not want to let their partner down*” [Williams 2000, S. 39]. Diese Erfahrung haben wir selber gemacht: Man will den Kollegen nicht „hängen lassen”. Daraus entsteht ein positiver psychologischer Druck, gemeinsam die Qualität des Lehr-/Lernprozesses zu verbessern. Ein gutes Beispiel hierfür bietet das Thema der Synchronisation mehrerer Threads im Rahmen unserer MP3-Aufgabe. Wir konnten uns gegenseitig sehr gut helfen, die Fakten dazu ins Gedächtnis zu rufen und live gemeinsam den Studierenden beim Lösen der Probleme helfen.

Pair Think Bei Aufgaben, die das konkrete Design einer Lösung nicht vorgeben, kommen verschiedene Gruppen von Studierenden zu unterschiedlichen Lösungen: „*A system with multiple actors possesses greater potential*” [Williams 2000, S. 41].

Das Ändern des API bei einem Refactoring bietet hierfür ein konkretes Beispiel: Der Theoretiker erläutert, dass die Signaturen öffentlicher Methoden von existierender, laufender Software nicht geändert werden dürfen. Der Praktiker zeigt Fälle auf, in denen die Änderung sogar unabdingbar ist, nämlich dann, wenn ein API sehr schlecht ist. Die an solchen Stellen mit den Studierenden geführten Diskussionen erlebten alle Beteiligten als hilfreich.

Pair Relaying Pair Programmierer berichten, dass sie gemeinsam Probleme lösen konnten, die keiner der beiden allein hätte lösen können: „*together they can evolve solutions to unruly or seemingly impossible problems”* [Williams 2000, S. 42]. Dieser Effekt ähnelt den Erfahrungen, die über Brainstorming und Teamarbeit im allgemeinen berichtet werden. Senge beschreibt dies als Team Learning [Senge 1990, S. 9]. Hier ergänzen sich Praktiker und Theoretiker durch ihre unterschiedlichen Sichtweisen.

Wir haben im Pair Teaching diesen Effekt im Praktikum mehrmals an Stellen erfahren, an denen einer der Dozierenden einen Sachverhalt nicht schnell genug auf den Punkt bringen konnte. Dieser Aspekt wurde in den Evaluierungen mehrfach positiv erwähnt.

Pair Reviews Begründet in den Arbeiten von Fagan ist das Pair Reviewing eine Standard-Methode im Software Engineering [Fagan 1976], [Fagan 1986]. Wir Autoren dieses Artikels haben selber in unserer industriellen Praxis erfolgreich Reviews bei großen Systemen eingesetzt [Utesch/Thaler 1996]. Tom Gilb beschreibt den Kern eines Reviews: „*Reviews (…) are typically peer group discussion activities*”

[Gilb/Graham 1993, S. 5]. Wie lehren wir Studierenden die Kultur der Diskussion? Unsere Antwort darauf ist: Mit den Rollen „Theoretiker” und „Praktiker” leben wir Diskussionen in unserem Kurs vor.

Das persönliche Fazit der Dozierenden ist, dass durch das sozial-emotionale Element „Dialog zweier Dozierender” ein positiveres Erleben der universitären Umwelt ermöglicht wird. Denn einer Längsschnittstudie zu den Bildungslebensläufen von über 2.300 deutschen Abiturienten zufolge, die den Bedingungen und subjektiven Begründungen von Studienabbrüchen nachgegangen ist, sind Gründe für einen Studienabbruch neben Lernschwierigkeiten und schlechten Berufsaussichten, vor allem Praxisferne [Gold/Kloft 1991]. Nach dieser Studie wiesen die Nichtabbrecher insbesondere ein positiveres Erleben der universitären Umwelt und eine bessere Informiertheit über universitäre Angelegenheiten auf. Die genannten Themen „Praxisnähe“ und „Berufsaussichten“ werden von uns adressiert durch einen eigenen Dozierenden aus der Praxis. Wir bringen den „Beruf“ und die „Praxis“ personifiziert in die Hochschule.

Wir haben unsere Studierenden im Rahmen einer qualitativen Evaluierung gefragt, ob das Pair Teaching in dieser Form weiter gepflegt und entwickelt werden soll. Repräsentativ sind folgende Rückmeldungen:

- „Super gute Praxisnähe durch die Zusammenarbeit.”
- „Besonders manche Fallbeispiele/Kommentare sind interessant wenn man hört, wie es Austin in seiner Firma bewerkstelligt.”
- „Ich fand, dass die Dozierenden sich sehr gut ergänzt haben.”
- „Die Dozierenden konnten sich mehr Zeit nehmen für Fragen, oder um zu diskutieren, was in einem Feld wie der Software Architektur durchaus notwendig ist.”

Am meisten beeindruckte die Lernenden, dass sie in die Kommunikation und Interaktion der dozierenden Experten mit einbezogen wurden – und zwar auf „Augenhöhe“: Angeregt durch das Rollenspiel zwischen Theoretiker und Praktiker, äußerten die Studierenden frei ihre Meinung und begannen schrittweise selber die Verantwortung für das Gelingen der Lehrveranstaltung mit zu übernehmen. Wie bei [Utesch 2008] konstatiert, verstärkt das sowohl Engagement als auch Durchhaltevermögen der Lernenden.

5 Ausblick

Wir haben in dieser Arbeit das Pair Teaching als eine Variante des Team Teaching am Beispiel unseres Kurses Software-Architektur an der Hochschule München vorgestellt.

Pair Teaching wird aus Sicht der Dozierenden wie auch der Studierenden als wünschenswert und realitätsnah bewertet. Aus Sicht heute gültiger Verordnungen zur Berechnung von Lehraufwänden mag eine Bewertung nicht so positiv ausfallen: Pair

Teaching benötigt zwei Dozierende pro Kurs, was nach Kapazitätsverordnung [Kap-VO 2005] sowie Verordnung über die Lehrverpflichtung [LUFV 2007] nur in Ausnahmefällen zulässig ist bzw. nicht voll auf das jeweilige Lehrdeputat angerechnet werden kann. Vielleicht bietet hier die Diskussion um Skill-Modelle und Teaching Points [Handel et. al. 2005] im Zuge der Umsetzung des Bologna-Prozesses einen Rahmen, in dem statt Kapazitätsverordnung und Verordnung über die Lehrverpflichtung Erkenntnisse innovativer didaktischer Designs mit neuen Steuerungsmodellen für den Einsatz von Lehrleistung verbunden werden. Gerade in diesem Fall haben wir Pair Teaching als äußerst wertvoll erlebt, da diese Methode auf Kommunikation und Interaktion aller Beteiligten setzt und somit gemäß dem Bolognaprozess den Erwerb von Kompetenzen und nicht den Erwerb von Fachwissen in den Vordergrund stellt. So gelingt es uns, unseren Studierenden schon an der Hochschule Praxis zu bieten. Deshalb sind wir überzeugt, dass auch andere Veranstaltungen von diesem Modell profitieren könnten. Neue Modelle zur Festlegung und zum Einsatz von Lehrleistung müssen dies künftig begleiten.

Nothing new ever works, but there´s always hope that this time will be different
[Weinberg 1985, S. 142]

Literatur

[Andersson/Bendix 2005] R. Andersson, L. Bendix: Pair Teaching - an eXtreme Teaching Practice. Proceedings of 4:e Pedagogiska Inspirationskonferensen, LTH, Lund University, Juni 2006

[Bass et al. 2003] L. Bass, P. Clements, R. Kazman: Software Architecture in Practice. SEI Series in Software Engineering, Addison-Wesley, Boston. 2003

[Beck 2000] K. Beck: eXtreme Programming explained. Addison Wesley, Boston, San Francisco. 2000

[Böttcher 2006] A. Böttcher: Eclipse in der Hochschul-Lehre. iX-Konferenz Bessere Software, Frankfurt, 2006.

[Fagan 1976] M. E: Fagan: Design and Code Inspections to Reduce Errors in Program Develpoment. IBM Systems Jounal, 15(3), 182-211. 1976

[Fagan 1986] M. E. Fagan: Advances in Software Inspections, IEEE Transactions on Software Engineering. SE-12(7), July, 744-751. 1986

[Flechsig/Haller 1987] K. H. Flechsig, H.-D: Haller: Einführung in didaktisches Handeln. Stuttgart: Klett. 1987

[Fowler 2005] M. Fowler, Refactoring oder: wie Sie das Design vorhandener Software verbessern. Addison-Wesley, München. 2005

[Gamma et al. 1995] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software, Addison Wesley, Boston, 1995

[Gamma/Beck 2004] E. Gamma, K. Beck: Contributing to Eclipse. Addison-Wesley, Boston. 2004

[Gilb/Graham 1993] T. Gilb, D. Graham: Software Inspection. Addison-Wesley. 1993

[Gold/Kloft 1991] A. Gold, C. Kloft: Der Studienabbruch: Eine Analyse von Bedingungen und Begründungen. Zeitschrift für Entwicklungspsychologie und Pädagogische Psychologie 23, S. 265-279. 1991

[Handel et al. 2005] K. Handel, Y. Hener, L. Voegelin: Teaching Points als Maßstab für die Lehrverpflichtung und Lehrplanung. CHE-Arbeitspapier Nr. 69, 2005

[Huber 2000] B. Huber: Team-Teaching – Bilanz und Perspektiven, Verlag Peter Lang, Frankfurt, 2000

[Humphrey 1990] W. S. Humphrey: Managing the Software Process. Addison-Wesley New York. 1990

[KapVO 2005] Verordnung über die Kapazitätsermittlung, die Curricularnormwerte und die Festsetzung von Zulassungszahlen in Bayern (Kapazitätsverordnung - KapVO) vom 1. Januar 2005.

[Kerievsky 2004] J. Kerievsky: Refactoring to Patterns. Addison Wesley, Boston. 2004

[Kerres 1998] M. Kerres: Multimedia und telemediale Lernumgebungen. Oldenbourg Verlag, München. 1998

[Lehner 1991] F. Lehner: Softwarewartung – Management, Organisation und methodische Unterstützung. Carl-Hanser-Verlag, München, Wien 1991.

[Liggesmeyer 2002] P. Liggesmeyer: Software-Qualität - Testen, Analysieren und Verifizieren von Software. Heidelberg, Spektrum Akademischer Verlag, 2002.

[LUFV 2007] Verordnung über die Lehrverpflichtung des wissenschaftlichen und künstlerischen Personals an Universitäten, Kunsthochschulen und Fachhochschulen in Bayern (Lehrverpflichtungsverordnung - LUFV) vom 14. Februar 2007.

[Rustemeyer 1985] D. Rustemeyer: Kommunikation oder Didaktik? Aporien kommunikativer Didaktik und Konstruktionsprobleme kommunikativer Bildungstheorie. Pädagogische Rundschau 39, S. 61-85, 1985

[Schaller 1979] K. Schaller: Pädagogik der Kommunikation. In K. Schaller, Erziehungswissenschaft der Gegenwart. Prinzipien und Perspektiven moderner Pädagogik (S. 155-181). Bochum: Kamp. 1979.

[Senge 1990] P.M. Senge: The Fifth Discipline: the art & practice of the learning organization. London: Century Business - an imprint of Random House UK Ltd. 1990.

[Utesch/Thaler 1996] M. Utesch, M. Thaler: Effektivität und Effizienz von Softweare Inspektionen. In Liggesmeyer, P. (Hrsg.): Test, Analyse und Verifikation von Software. München: Oldenbourg Verlag. 1996.

[Utesch 2008] M. C. Utesch: Leistung braucht Netzwerke: Soziales Lernen und fachliche Leistung. In B. S. Kultus, Werte machen stark - Praxishandbuch zur Werteerziehung (pp. 94-96). Augsburg: Brigg Pädagogik, 2008.

[Waldherr/Walter 2008] F. Waldherr, C. Walter: Methodensammlung für die Lehre an Hochschulen. Ingolstadt: Didaktikzentrum DIZ, 2008.

[Walther 2005] T. Walther: Pair Programming. Ausarbeitung im Rahmen des Seminars, Agile Softwareprozesse. FU Berlin, 2005 (online unter www.tilman.de/uni/PairProgramming.pdf).

[Weinberg 1985] G. M. Weinberg: The secrets of consulting, Dorset House Publishing, New York, 1985

[Williams 2000] L. Williams: The Collaborative Software Process. Dissertation an der University of Utah, 2000 (online unter http://collaboration.csc.ncsu.edu/laurie/Papers/dissertation.pdf).