12. SEUH 2011

Kurzbericht und Übersicht der Beiträge

Jochen Ludewig, Holger Röder, Universität Stuttgart

ludewig@informatik.uni-stuttgart.de

Hintergrund

1992 fand in Stuttgart die SEUH (Software Engineering im Unterricht der Hochschulen) statt, im Rückblick die *erste* SEUH, aus der in den folgenden Jahren eine Institution wurde. Am 24. und 25.2.2011 gab es in der Hochschule München die zwölfte Auflage. Im Programmkomitee wirkten mit:

- Axel Böttcher, Hochschule München
- Ralf Bruns, Fachhochschule Hannover
- Marcus Deininger, HFT Stuttgart
- Ulrike Jaeger, Hochschule Heilbronn
- Jochen Ludewig, Universität Stuttgart
- Barbara Paech, Universität Heidelberg
- Axel Schmolitzky, Universität Hamburg
- Kurt Schneider, Leibniz Univ. Hannover
- Silke Seehusen, Fachhochschule Lübeck
- Olaf Zukunft, HfAW Hamburg

Da die SEUH traditionell mit einem kleinen Budget auskommt, erstmals auf den gedruckten Tagungsband verzichtet wurde und zudem eine Spende der Firma TNG Technology Consulting einige der Kosten abdeckte, waren die Tagungsgebühren wohltuend niedrig und reichten trotzdem aus, um auch den fröhlichen Abend im Löwenbräu-Keller zu finanzieren.

Programmübersicht (chronologisch)

Folgende Beiträge bildeten den Inhalt des Workshops:

- Oliver F. Nandico: Effizientere Software-Entwicklung durch Industrialisierung der Prozesse (eingeladener Vortrag)
- Georg Hagel, Jürgen Mottok: Planspiel und Briefmethode für die SE-Ausbildung - ein Erfahrungsbericht
- Doris Schmedding: Teamentwicklung in studentischen Projekten
- Axel Böttcher et al.: Softwaretechnik-Lehre an der Hochschule München
- Axel Schmolitzky, Chris Stahlhut: Die Metapher vom "Konsumieren und Produzieren" von Klassen in BlueJ
- Podiumsdiskussion (Leitung Jochen Ludewig): Agile Software-Entwicklung in der Lehre
- Daniel Kulesz: Ein Dashboard für Learning-Management-Systeme
- Axel Böttcher et al.: Kompetenzorientierte Lehre im Software Engineering
- Heinz Züllighoven: Praxis lehren und Forschung umsetzen: Wie können Hochschullehrer diese Kluft überbrücken? (eingeladener Vortrag)
- Stephan Kleuker, Frank Thiesing: Vier Jahre Software-Engineering-Projekte im Bachelor – ein Statusbericht
- Annabel Schmidt, Ralf Sommer: Rückblick der Studierenden

Fast die Hälfte der Zeit war für die Diskussion der Beiträge reserviert; diese Zeit wurde gern und gut genutzt.

Inhalte

Der Schwerpunkt des Programms lag bei den studentischen Projekten. Denn im traditionellen Unterricht können wir nur Grundlagen vermitteln, die dann praktisch erprobt, eingeübt und stabilisiert werden müssen. Weitere Beiträge befassten sich mit dem durchgängigen Beispiel in der Lehre, einer Entwicklungsumgebung für die Programmierausbildung und einer Software zur Verwaltung der Lehrveranstaltungen.

Im ersten eingeladenen Vortrag von Oliver F. Nandico (CapGemini) ging es um die Software-Prozesse in der Praxis und um die Forderungen, die daraus für die Lehre abzuleiten sind, im zweiten von Heinz Züllighoven (Universität Hamburg) um die Chancen für die Lehre, die entstehen, wenn der Hochschullehrer und seine Mitarbeiter auch ein Software- und Beratungsunternehmen betreiben.

Eine Podiumsdiskussion behandelte die Bedeutung und Behandlung agiler Prozesse in der Lehre. Vorn saßen als Vertreter der Praxis Oliver F. Nandico (CapGemini) und Gerhard Müller (TNG Technology Consulting), als Vertreter der Lehre

Gerd Beneken (FH Rosenheim) und Guido Gryczan (Universität Hamburg). Die Praktiker brachten zum Ausdruck, dass sie von den Absolventen zwar keine speziellen Kenntnisse, aber Erfahrungen mit Prozessen und Entwicklungswerkzeugen erwarten. Die Dozentinnen und Dozenten im Saal zeigten sich für die agilen Ansätze, insbesondere Scrum, sehr offen, haben aber einige Probleme damit, in den oft sehr kurzen Projekten das Vorgehen einzuüben. Zudem fehlen Experten, die etwa als Scrum Master beratend mitwirken können. Von der Industrie kam das Angebot, die Hochschulen zu unterstützen, beispielsweise durch Vorträge und durch die Einstellung von Praktikanten.

Eine Neuerung im Programm der SEUH war die Vorstellung der gastgebenden Hochschule. Eine Dozentin, zwei Dozenten und eine Studentin erklärten und begründeten das Lehrangebot der Hochschule München im Themenfeld Softwaretechnik.

Lange Tradition hat dagegen die Kritik durch Studierende. Annabel Schmidt (Hochschule München) und Ralf Sommer (Universität Stuttgart), beide kurz vor Abschluss ihres Studiums, hatten aufmerksam teilgenommen und konnten am Ende über ihre Wahrnehmungen berichten. Die Kritik fiel für die Kritisierten sehr glimpflich aus; zunächst brachte Frau Schmidt zum Ausdruck, wie erfreulich sie es fand zu erfahren, dass sich die Lehrenden wirklich sehr um eine gute Lehre bemühen. Dann kamen neben wichtigen Details (kleine Schrift und allzuviel Text auf den Folien) Anregungen, vor allem in Richtung auf eine Auseinandersetzung mit den Ideen der Didaktik. Wie alle Beiträge zuvor wurde auch dieser lebhaft diskutiert.

Der – vor zwei Jahren bestimmte – Termin dieser SEUH fiel unglücklich mit dem der SE'11 in Karlsruhe zusammen. Der neu zu formierende SEUH-Steuerkreis wird dafür sorgen, dass sich dieser Konflikt nicht wiederholt: Die SEUH wird ausweichen oder in das Rahmenprogramm der SE-Tagung umziehen.

Nachfolgend finden Sie die Zusammenfassungen der eingeladenen Vorträge und einseitige Kurzfassungen aller referierten Beiträge; der virtuelle Tagungsband ist als Vol. 695 bei den CEUR-Workshop Proceedings erschienen und frei zugänglich: *http://ceur-ws.org/Vol-695* (oder über die ständige Webseite der SEUH: *http://www.seuh.org/*)

Effizientere Software-Entwicklung durch Industrialisierung der Prozesse (eingeladener Vortrag)

Oliver F. Nandico, CapGemini

Zusammenfassung

Es gibt heute keine Branche, kein Unternehmen und keine Geschäftsprozesse, bei denen nicht „ein Stück Software" involviert ist. Wir können uns eine ganze Reihe alter und vor allem neuer Geschäftsmodelle ohne IT-Unterstützung gar nicht mehr vorstellen.

Trotz oder gerade aufgrund dieser Alltäglichkeit befindet sich die Entwicklung von Software in einem grundsätzlichen Wandel als Reaktion auf widersprüchliche Anforderungen. Verfallende Marktpreise und steigender Produktivitätsdruck verlangen die Industrialisierung der Entwicklungsprozesse. Das bedeutet Wiederholbarkeit, Standardisierung und Automatisierung. Gleichzeitig fordern die Anwender von Software die schnelle, innovative und ganz spezifische Reaktion des Softwareentwicklers auf ihre besondere Aufgabenstellung und ihr Geschäftsmodell. So erzielen sie ihrerseits einen Vorsprung im Wettbewerb vor der Konkurrenz.

Softwareentwicklung muss also heute Standardisierung und Automatisierung geeignet mit Flexibilität, Reaktionsfähigkeit und Nutzernähe kombinieren.

Diese Veränderung in der Softwareentwicklung wirkt sich konsequenterweise auf die Tätigkeit und das Berufsbild des Software-Ingenieurs aus. Zu diesem Wandel formuliert der Artikel sieben Thesen.

Praxis lehren und Forschung umsetzen: Wie können Hochschullehrer diese Kluft überbrücken? (eingeladener Vortrag)

Heinz Züllighoven, Univ. Hamburg und C1 WPS

Zusammenfassung

Die Softwaretechnik an Hochschulen versteht sich meist als anwendungsorientierte (Ingenieur-)Wissenschaft. Was sind aus heutiger Sicht ihre wissenschaftlichen Fundamente für die Lehre, und wie stabil sind sie?

Professionelle Softwareentwicklung sollte sich an wissenschaftlichen Konzepten und Ergebnissen orientieren. Umgekehrt sollte die Praxis auch Impulse für die anwendungsorientierte Wissenschaft geben. Welche Impulse sind dies für die Softwaretechnik?

Der Versuch, anwendungsorientierte Wissenschaft und forschungsorientierte Praxis in Forschung und Lehre zusammenzubringen, läuft in Hamburg seit zwanzig Jahren. Was ist dabei herausgekommen? Was könnte man auf andere Standorte übertragen?

Planspiel und Briefmethode für die Software Engineering-Ausbildung – ein Erfahrungsbericht

Georg Hagel, Hochschule Kempten, Jürgen Mottok, LaS³, Hochschule Regensburg

georg.hagel@fh-kempten.de, juergen.mottok@hs-regensburg.de

Zusammenfassung

Die konstruktivistische Perspektive unterstützt einen Paradigmenwechsel der akademischen Lehre hin zu einer Lerner- und Lernprozesszentrierung. Dabei thematisieren Selbstgesteuertes Lernen und aktivierende Lehre das studentische Lernen neu.

Planspiel und Briefmethode, die hier vorgestellten konstruktivistischen Methoden aus dem Konstruktivistischen Methodenpool (Reich, 2008), lassen sich auf ein „Lebenslanges Lernen" von Software Engineering (SE) im Berufsumfeld übertragen.

Planspiel

In Planspielen im Fach SE sollen Studierende durch Simulation einer Praxissituation einen möglichst realistischen und praxisbezogenen Einblick in Probleme und Zusammenhänge der methodischen Softwareentwicklung gewinnen, eigene Entscheidungen treffen und Konsequenzen ihres Handelns erfahren. Planspiele erfordern zudem eine hohe Partizipation aller Beteiligten. Sie sollten auf eine Erhöhung der Handlungsfähigkeit in dem Sinne zielen, dass sie Konsens und Dissens, Entscheidungsabläufe und Transparenz bei der Bildung von Gruppenentscheidungen aufdecken und diskutierbar werden lassen (Reich, 2008).

Die einzelnen Phasen des Planspiels bestehen aus:

1. Spieleinführung
2. Informations- und Lesephase, Rollenverteilung
3. Meinungsbildung und Strategieplanung innerhalb der Gruppe
4. Interaktion zwischen den Rollen
5. Vorbereitung eines Plenums / Konferenz
6. Durchführung eines Plenums / Konferenz
7. Spielauswertung

Briefmethode

Die Briefmethode wird häufig im Deutschunterricht der Schulen, aber auch in Geschichte und Literatur eingesetzt. Wir wollten untersuchen, ob sich diese Methode auch für den Einsatz in einem technischen Fach, wie SE an einer Hochschule eignet. Dabei wurden seitens der Dozierenden mehrere Ziele verfolgt: Die Studierenden sollten

- einen Sachverhalt, den sie sich vorher erarbeitet hatten, wiedergeben können und
- lernen, Briefe mit technischer Information zu verfassen.

Das wissenschaftlich fundierte Darstellen, z.B. das Erstellen technischer Dokumentation, kommt aus Sicht der Autoren in den Lehrveranstaltungen des SE zu kurz und beschränkt sich meist auf die Erstellung von UML-Diagrammen.

In der Vorbereitung muss zunächst ein hinreichend komplexer Sachverhalt gefunden werden, der sich in Briefform gut vermitteln lässt. In unserem Beispiel bat der Dozierende die Studierenden um ein Antwortschreiben auf einen fiktiven Brief mit einer technischen Fragestellung. Der Dozierende gab auf die Antwortschreiben individuell Feedback.

Ergebnisse

Planspiel und Briefmethode sind zwei konstruktivistische Methoden, die nach Meinung der Autoren sehr gut für die Ausbildung des SE geeignet sind. Die Lernenden lassen sich für das Planspiel einfacher motivieren als für die Briefmethode.
Planspiele im SE konfrontieren möglichst realistisch mit einer Praxissituation. Die Studierenden können dabei zum kreativen, weitgehend autonomen und selbstorganisierten Handeln in Bezug auf konkrete Probleme und deren Lösung motiviert werden und nehmen dabei unterschiedliche Positionen in einem komplexen Softwareentwicklungsprozess ein.

Die Briefmethode führt zu besserem Verfassen technischer Dokumentation und eignet sich sehr gut, um sich ein Thema anzueignen, oder zu wiederholen.

Beide Methoden führten in den durchgeführten Evaluationen zu positiver Rückmeldung.

Literatur

Reich, K. (2008): Konstruktivistische Didaktik – Lehr- und Studienbuch mit Methodenpool, 4. Aufl., Beltz Verlag, methodenpool.uni-koeln.de.

Teamentwicklung in studentischen Projekten

Doris Schmedding, TU Dortmund

Doris.Schmedding@tu-dortmund.de

Zusammenfassung

Teamfähigkeit ist eine wichtige Anforderung an Software-Entwickler und -Entwicklerinnen. Unter Teamfähigkeit wird verstanden, mit anderen zusammen sozial zu agieren und sich und sein Können im Sinne einer Gruppenaufgabe optimal einzubringen. Studentische Software-Entwicklungsprojekte bieten die Möglichkeit, neben den technischen Fähigkeiten auch die Softskills der Studierenden auf dem Gebiet der Teamarbeit zu trainieren. Dieser Artikel stellt in einem Software-Praktikum erfolgreich erprobte Methoden vor, mit denen ohne großen Aufwand die Teamentwicklung in studentischen Projekten unterstützt und die Reflexion der Erfahrungen anleitet werden kann, um daraus neue Impulse für eine bessere Zusammenarbeit zu gewinnen.

Teamentwicklung

Gute Zusammenarbeit ist eine wichtige Voraussetzung für das Gelingen eines Projekts. Die Entwicklung von einer zufällig zusammengestellten Gruppe von Studierenden mit heterogenem Vorwissen zu einem gut funktionierenden Team läuft in einer Abfolge von Phasen ab, die gerne mit den eingängigen Begriffen *Forming, Storming, Norming* und *Performing* bezeichnet werden. Diese Phasen werden bis auf die Forming-Phase bei neuen Aufgaben oder Zielen, oder wenn ein neues Mitglied zur Gruppe hinzukommt, wieder neu durchlaufen. Anstelle des Forming tritt dann die Reorganisation des Teams, das *Reforming* (Vigenschow u.a., 2009).

In der Anfangsphase, die von großer Unsicherheit geprägt ist, können Übungen zur Teambildung helfen, das Kennenlernen zu erleichtern und die Abläufe im Team zu veranschaulichen. Besonders geeignet für Software-Entwicklungsteams sind Übungen, die einen Projektcharakter haben und an denen sich die Analogie zu Software-Projekten gut zeigen lässt.

Im Artikel finden Sie zwei konkrete Vorschläge für die Gestaltung eines Teamentwicklungsprojekt, das sich leicht in die erste Doppelstunde einer Lehrveranstaltung integrieren lässt, also wenig Zeit kostet, für das man keine aufwendigen Vorbereitungen treffen muss und an dem sich sehr gut die Abläufe bei der Teamarbeit und die Phasen in einem Projekt veranschaulichen lassen.

Angeleitete Reflexion

Für das Lernen ist es wichtig, die eigenen Erfahrungen mit den neuen Arbeitstechniken zu reflektieren und bei Bedarf die Vorgehensweisen zu verändern. Nur so lassen sich daraus Konsequenzen für die weitere Arbeit, auch im Beruf, ziehen. In (Lewerentz u. Rust, 2001) wird ausführlich auf die Bedeutung der Reflexion in der Projektarbeit eingegangen. Wir setzen eine angeleitete Reflexion in der Mitte des Praktikums ein, um mit den Studierenden unter Einhaltung der Feedback-Regeln (Vigenschow u. Schneider, 2007) über ihre Erfahrungen zu diskutieren. So erhalten sowohl die Studierenden als auch wir als Lehrende wertvolle Hinweise, was wir noch verbessern können.

Fazit

In (Fleischmann u.a., 2005) und in (Lewerentz u. Rust, 2001) wird gezeigt, wie mit relativ großem zeitlichem und personellem Aufwand erfolgreich die Softskills der Studierenden verbessert werden. Hier wird erläutert, dass auch mit kleinen, gut platzierten Maßnahmen die Teamarbeit unterstützt werden kann. Die vorsichtige Integration einiger Inhalte in eine bestehende und etablierte Veranstaltung, wie ein Kurzvortrag über Teamarbeit, ein Teamentwicklungsprojekt und die angeleitete Reflexion der Erfahrungen, ist einfach zu realisieren, kostet fast keine zusätzliche Zeit, lohnt sich aber auf jeden Fall, denn diese Investition wird durch ein besseres Arbeitsklima belohnt und verbessert die Vorbereitung der Studierenden auf den Beruf.

Literatur

Fleischmann, A., Spies, K., Neumeyer, K. (2005): Teamtraining für Software-Ingenieure. In: Löhr, K.-P., Lichter, H. (Hsg.): SEUH 9, 26-40.

Lewerentz, C., Rust, H. (2001): Die Rolle der Reflektion in Softwarepraktika. In: Lichter, H., Glinz, M. (Hsg.): SEUH 7, 73-86.

Vigenschow, U., Schneider, B. (2007): Soft Skills für Softwareentwickler. dpunkt.verlag.

Vigenschow, U., Schneider, B., Meyrose, I. (2009): Soft Skills für IT-Führungskräfte und Projektleiter. dpunkt.verlag.

Dynamische Klassendiagramme – Nutzung der Metapher vom „Konsumieren und Produzieren“ in BlueJ

Axel Schmolitzky und Chris Stahlhut, Universität Hamburg
{schmolit,6stahlhu}@informatik.uni-hamburg.de

Zusammenfassung

In diesem Artikel stellen wir eine Erweiterung der für die Lehre der objektorientierten Programmierung entwickelten Entwicklungsumgebung BlueJ vor. Diese Erweiterung führt die bereits in BlueJ implizit vorhandene Unterstützung der Metapher vom *Konsumieren und Produzieren* (kurz: K&P-Metapher) konsequent fort, indem sie auch im BlueJ-Klassendiagramm dynamisch die Unterscheidung zwischen dem Konsumieren und dem Produzieren einer Klasse ermöglicht.

K&P in der Programmierung

In diesem Artikel liegt aus Sicht der K&P-Metapher der Fokus auf dem Konsumieren und Produzieren von *Klassen in der objektorientierten Programmierung*. Bei diesen Klassen gibt es klar definierte Eigenschaften, von denen einige primär für die Klienten (Konsumenten) einer Klasse relevant sind und andere nur für die Entwickler (Produzenten) einer Klasse. Dies macht sie einer automatisierten Darstellung zugänglich.

Ein zentraler Gedanke der Objektorientierung (und der Softwaretechnik) ist, dass bei einer Klasse zwischen ihrer *Schnittstelle* und ihrer *Implementation* unterschieden werden kann. Über ihre Schnittstelle bietet eine Klasse ihre Dienstleistungen an; in der Implementation wird dieses Angebot umgesetzt. In Java wird systematisch zwischen der Schnittstelle einer Klasse und ihrer Implementation unterschieden, indem die mit dem Werkzeug *javadoc* erzeugte Dokumentation einer Klasse üblicherweise ihre Schnittstelle darstellt, während der Quelltext der Klasse ihre vollständige Implementation wiedergibt.

Eine Klasse zu konsumieren erfordert nur die Kenntnis ihrer Schnittstelle; eine Klasse zu produzieren erfordert zwingend Einblick in ihre Implementation. Beide Sichten auf eine Klasse finden sich auch in *BlueJ* wieder, allerdings nur unvollständig. BlueJ ist eine für die Lehre des objektorientierten Programmierparadigmas entworfene Entwicklungsumgebung für Java. Der in BlueJ integrierte Editor zum Bearbeiten von Klassendefinitionen bietet zwei Sichten: eine konsumierende, die die von *javadoc* für die Klasse erstellte (und nicht interaktiv änderbare) Schnittstellenbeschreibung darstellt; und eine produzierende Sicht, die den vollständigen Quelltext der Klasse darstellt und bearbeitbar macht.

Dynamische Klassendiagramme

Eine von uns entwickelte Erweiterung von BlueJ namens *BlueJ-CP* ermöglicht ein interaktives Umschalten zwischen der Klientensicht und der Produzentensicht auf eine Klasse *auch im Klassendiagramm*. Abhängigkeiten zu anderen Klassen, die sich ausschließlich aus der Implementation einer Klasse ergeben (also nicht Teil ihrer Schnittstelle sind), können im Diagramm ausgeblendet werden. Transitiv werden dann auch ganze Klassen ausgeblendet, wenn alle auf sie verweisenden Abhängigkeiten ausgeblendet sind.

Diese grundsätzliche Möglichkeit zweier Sichten auf jede Klasse führt dazu, dass es nicht nur ein statisches, sondern eine Vielzahl von möglichen Klassendiagrammen für dasselbe Projekt gibt. Bei einem Projekt mit n Klassen ergeben sich bis zu 2^n potenziell verschiedene Klassendiagramme. Durch die Möglichkeit des interaktiven Umschaltens wird das statische Klassendiagramm von BlueJ somit zu einem *dynamischen Klassendiagramm*.

In dem Artikel motivieren wir die K&P-Metapher für die objektorientierte Programmierung und stellen unsere Erweiterung von BlueJ an einem einfachen Beispiel vor. Abschließend diskutieren wir die Einsatzmöglichkeiten dynamischer Klassendiagramme in der Lehre der objektorientierten Programmierung und in der Visualisierung von Software.

Literatur

[Kölling u. a. 2003] KÖLLING, M. ; QUIG, B. ; PATTERSON, A. ; ROSENBERG, J.: The BlueJ system and its pedagogy. In: *Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology* (2003)

[Späh u. Schmolitzky 2007] SPÄH, C. ; SCHMOLITZKY, A.: „Consuming before Producing“ as a Helpful Metaphor in Teaching Object-Oriented Concepts. In: *Eleventh Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts*, 2007

[Stahlhut 2010] STAHLHUT, C.: *Die Metapher „Konsumieren und Produzieren“ in BlueJ*, Universität Hamburg, Bachelorarbeit, 2010

Ein Dashboard für Learning-Management-Systeme

Daniel Kulesz, Institut für Softwaretechnologie, Universität Stuttgart
daniel.kulesz@informatik.uni-stuttgart.de

Zusammenfassung

In der Softwaretechnik-Lehre werden üblicherweise Lehrveranstaltungen durchgeführt, in denen studentische Teams gemeinsam Übungsaufgaben bearbeiten. Dabei werden auch studentische Hilfskräfte als Tutoren für die Korrektur der Übungsaufgaben eingesetzt.

Gängige Learning-Management-Systeme (LMS) wie Moodle oder ILIAS eignen sich aufgrund einiger Defizite nur bedingt zur Unterstützung solcher Lehrveranstaltungen. Dieser Beitrag beschreibt einen in unserer täglichen Lehrpraxis bewährten Ansatz, der diese Lücken prototypisch schließt. Dazu wurde ein bestehendes LMS um ein als Online-Spreadsheet realisiertes Dashboard erweitert.

Learning-Management-Systeme (LMS)

Learning-Management-Systeme (LMS) sind komplexe Softwaresysteme zur Bereitstellung von Lerninhalten und zur Organisation von Lernvorgängen. LMS sollen administrative Belange wie die Ausgabe der Aufgabenstellung, die Abgabe der Lösungen, die Verteilung der Korrekturaufgaben und die Rückmeldung an die Studenten vereinfachen.

Laut (Oevel u. Lange, 2009) stellt mittlerweile jede Hochschule ihren Mitgliedern ein LMS zur Verfügung. Die LMS werden dabei überwiegend als Unterstützung und nicht als Ersatz für die Präsenzlehre verwendet (Steffens u. Reiss, 2009).

Unterstützung von Lernszenarien

Wir wollten die Vorteile eines LMS vor allem in Lernszenarien der Softwaretechnik-Lehre nutzen, in denen studentische Tutoren für die Korrektur von Übungsaufgaben eingesetzt werden, die von studentischen Teams bearbeitet wurden. Die beiden populärsten Open-Source-LMS ILIAS und Moodle sind für solche Lernszenarien aufgrund von zwei großen Defiziten jedoch kaum geeignet: Die LMS ermöglichen den Lehrenden keinen Überblick über die Gesamtsituation der einzelnen Abgaben. Zudem bieten sie kaum Möglichkeiten für eine effiziente Kollaboration mit studentischen Tutoren.

Unser Verbesserungsansatz

Um die beschriebenen Lernszenarien trotzdem realisieren zu können, setzen wir die in ILIAS vorhandenen Bausteine ‘Diskussionsforum‘ und ‘Gruppe‘ stark zweckentfremdet ein: Wir erstellen für jede Übungsgruppe ein separates, abgeschottetes ‘Abgabenforum‘, in dem die Abgabe und Bewertung der Übungsaufgaben erfolgt.

Gruppe/ Ü-Leiter/ Tutor			Meilenstein	M2				M3		M
			Abgabe	Analysenot.		Projektplan		Spezifikation		k
			Datum	29.10.	05.11.	29.10.	05.11.	16.11.	23.11.	3
			Versuch	I.	II.	I.	II.	I.	II.	I.
1	D	A		MO		MO		MO		
3	D	M		AO		AO		AO		
5	D	A		AO+		AO+		AR	AO	
7	D	M		AO		AO		AO+		
9	D	A		AO+		AO+		AO-		
11	D	M		DR	AO	DR	AO	AR	AO-	
13	D	A		AO+		AO+		AO		
15	D	M		MO		MO		I1		
17	D	A		MO-		MO		I3		
19	D	M		MO+		MO+		I2		
21	D	A		DR	DO	DR	ME1	DO		

Abbildung 1: Ausschnitt aus dem Dashboard

Darüberhinaus führen wir gemeinsam mit unseren studentischen Tutoren ILIAS-extern das in Abbildung 1 dargestellte Dashboard. Es visualisiert den Zustand von Abgaben und Bewertungen durch festgelegte Kürzel und farbliche Markierungen. Dadurch lässt sich die Qualität der bisherigen Abgaben immer auf einen Blick erfassen.

Erfahrungen

Mit dem vorgestellten Ansatz lassen sich Gruppenübungen in einem LMS realisieren, das eigentlich keine Gruppenübungen unterstützt. Die Kollaboration unter den Korrektoren wird durch das externe Dashboard stark verbessert. Das Dashboard könnte jedoch noch enger in ein LMS integriert werden, wenn man es als interne Komponente reimplementierte.

Literatur

[Oevel u. Lange 2009] Oevel, Gudrun ; Lange, Gerald: ZKI LMS-Umfrage. (2009). http://www.doodle.com/uyvcg2wz6s4bwv6v

[Steffens u. Reiss 2009] Steffens, Dirk ; Reiss, Michael: Blended Learning in der Hochschullehre. Vom Nebeneinander der Präsenzlehre und des E-Learning zum integrierten Blended-Learning-Konzept. In: *HSW: Das Hochschulwesen* (2009), Nr. 4, S. 115–123

Kompetenzorientierte Lehre im Software Engineering

Axel Böttcher und Veronika Thurner, Hochschule München
ab | thurner@cs.hm.edu
Gerhard Müller, TNG Technology Consulting GmbH
gerhard.mueller@tngtech.com

Motivation und Definition von Lernzielen

Software Engineering ist eine komplexe Aufgabe, die von den Beteiligten ausgeprägte Kompetenzen in vielen verschiedenen Bereichen fordert. Durch die zunehmende Entwicklung im Team werden insbesondere auch Fähigkeiten wichtig, die sich mit der Positionierung einer Einzelperson im Team sowie mit der Integration und Interaktion befassen. Die Vermittlung dieser Kompetenzen über das reine Fachwissen hinweg ist eine zentrale Aufgabe, die eine Software-Engineering-Ausbildung leisten muss, um Absolventen für den Einsatz in der Praxis zu qualifizieren.

Wir definierten zunächst kompetenzorientierte Lernziele für die SE-Ausbildung. Diese decken unterschiedliche Wissensgebiete ab und bereiten die Studierenden auf die Übernahme verschiedener Rollen vor. Als Grundlage verwendeten wir den „Software Engineering Body of Knowledge“, die Ergebnisse aus Absolventenbefragungen, sowie Anforderungen von erfahrenen Praktikern aus der Industrie.

Lehrkonzept

Ausgehend von den relevanten Kompetenzen vermitteln wir in unserer Lehrpraxis mit Hilfe eines möglichst einfach gehaltenen, aber so komplex wie nötig gestalteten durchgängigen Beispiels eine Einführung in die komplexen Tätigkeiten des Software Engineerings. Wir konzipierten dieses Beispiel als eine moderne Webapplikation, um dem Curriculum der von uns bedienten Studiengänge „Informatik“ und „Wirtschaftsinformatik“ gerecht zu werden. Wir verwenden das Beispiel bislang in folgenden Modulen mit je zwei SWS seminaristischem Unterricht und zwei SWS Praktikum, bei fünf ECTS:

- Bachelor Wirtschaftsinformatik, Module „Software Engineering I/II“ im 3. und 4. Semester.
- Bachelor Informatik, Modul „Software-Architektur“ im 4. Semester.

Für einen ersten fachlichen Schnitt werden Planung, Analyse- und Entwurfsdokumente, Implementierung und umfangreiche Tests den Studierenden zur Verfügung gestellt. Diese dienen sowohl als veranschaulichendes Beispiel als auch als Vorlage für die von den Studierenden durchzuführenden Projektarbeiten. Ein zweiter fachlicher Schnitt ist ebenfalls ausgearbeitet, je nach Fokus der Lehrveranstaltung aber nur teilweise oder gar nicht für die Studierenden verfügbar. Bei Bedarf ist das Beispiel um zusätzliche fachliche Schnitte erweiterbar.

Abgleich Lehrkonzept vs. Lernziele

Nach den bisherigen Erfahrungen adressiert unser Lehrkonzept eine Vielzahl der Schlüsselkompetenzen:

Sachkompetenz: Es werden viele zentrale Aspekte des Software Engineerings von der Machbarkeitsanalyse bis hin zum Abnahmetest gezeigt. Der besondere Fokus liegt dabei auf der Integration der einzelnen Disziplinen. Dadurch werden für die Lernenden die Zusammenhänge zwischen einzelnen Arbeitsschritten und Artefakten klar ersichtlich und nachvollziehbar. Insbesondere werden dadurch die Auswirkungen einzelner Entwicklungsentscheidungen greifbar verdeutlicht. Das Beispiel und die zugehörigen Lernmaterialien ermöglichen so einen ganzheitlichen Einstieg in die komplexe Materie des Software Engineering.

Methodenkompetenz: Dadurch, dass das Beispiel hinreichend komplex ist und die Aufgabenstellung nicht alle Entscheidungen vorgibt, müssen die Studierenden an verschiedenen Stellen abwägen und entscheiden, wie weit sie bestimmte Kerntechniken anwenden. Durch stets knappe Zeit wird die Notwendigkeit der Abwägung zwischen Perfektionismus und pragmatischer Einschränkung des investierten Aufwandes erlebbar.

Selbstkompetenz: Im Rahmen der verschiedenen Phasen der Projekt- und Teamarbeit erhalten die Studierenden Feedback von Kommilitonen und Dozenten, sowohl über ihren Arbeitsprozess als auch über die erzielten Ergebnisse. Beispielsweise stellen einzelne Projektteams Teile ihrer Arbeiten den anderen Gruppen vor.

Sozialkompetenz: Durch den relativ hohen Anteil an Teamarbeit kommt dem Bereich der Sozialkompetenz eine fortlaufend wichtige Rolle zu. Bereits beim Bilden der Teams und bei der Aufteilung der zu erledigenden Arbeiten treffen unterschiedliche Interessen und Vorlieben aufeinander.

Der beschrittene Weg, für ein signifikant großes Projekt einzelne Teile in einer ausgearbeiteten Form vorzugeben, die auch kritischen Blicken aus der Praxis Stand hält, hat sich im Ansatz bewährt. An manchen Stellen haben wir einen Teil unserer Studierenden mit der Komplexität des Beispiels aber überfordert. Im Detail müssen wir also noch nachbessern.

Vier Jahre Software-Engineering-Projekte im Bachelor – ein Statusbericht

Stephan Kleuker, Hochschule Osnabrück
Frank M. Thiesing, Hochschule Osnabrück
{s.kleuker,f.thiesing}@hs-osnabrueck.de

Zusammenfassung

Software-Engineering-Projekte wurden als berufspraktische Ausbildungsinhalte als Lehrveranstaltung in viele Informatik-Bachelor-Studiengänge aufgenommen. Der Artikel fasst die Erfahrungen der Autoren bei der Organisation, der Durchführung und mit den Ergebnissen der Projekte aus den Anfangsjahren der Bachelor-Einführung an den Hochschulen RheinMain und Osnabrück zusammen.

Mit der Umstellung der Informatik-Studiengänge auf Bachelor- und Masterabschlüsse wurden auch die zugehörigen Studieninhalte neu strukturiert. Generell enthalten alle Bachelor-Informatikstudiengänge einen wesentlichen Kernanteil aus dem Bereich Software-Engineering, beginnend mit der Programmierausbildung, meist weitergeführt mit objektorientierter Modellierung und Design. In einige Studiengänge wurden neben Vorlesungen zum Software-Engineering auch Software-Engineering-Projekte mit in den Studienplan aufgenommen. Dies war u. a. an der Hochschule RheinMain und der Hochschule Osnabrück der Fall, die jeweils eine Veranstaltung mit 10 Kreditpunkten (KP), aufgeteilt in eine begleitende Vorlesung (3 KP) und ein von 6-10 Studierenden zu bearbeitendes Projekt in ihren Lehrplänen verankerten. Die Software-Engineering-Projektveranstaltung findet an beiden Standorten jeweils nach Abschluss der Programmierausbildung und Vorlesungen zum Thema objektorientierte Analyse und Design im vierten (HS RheinMain) und fünften (HS Osnabrück) Semester statt. Innerhalb der begleitenden Vorlesung werden ausgewählte Themen des Software Engineering (u. a. Qualitätssicherung, Versionsmanagement), eine Einführung in das Projektmanagement und Grundlagen von Soft-Skills behandelt.

Die Findung von Projektthemen findet durch wiederholte Aufrufe per E-Mail und persönlicher Ansprache durch die Veranstalter der Vorlesung statt.

Das Projekt und die Vorlesung werden formal möglichst voneinander getrennt, damit möglichst viele Projektvorschläge eingehen, auch wenn die Projektdurchführenden die Vorlesung nicht kennen. Damit zwischen den Projekten eine Vergleichbarkeit entsteht, wurden für die Veranstalter Rahmenbedingungen definiert, die auch für die Studierenden einsehbar sind.

Als Abschluss der Veranstaltung wird an der HS Osnabrück eine Projektmesse veranstaltet, auf der alle Projekte kurz über ihre Arbeiten vortragen, die Ergebnisse vorführen und auf einem Poster zusammenfassen. Diese Veranstaltung wird auch für jüngere Semester zur Information durchgeführt, da teilweise die Möglichkeit von Anschlussprojekten besteht.

Insgesamt wurde ein relativ breites Spektrum von Projekten angeboten und durchgeführt. Es reichte von einer komponentenbasierten Software-Entwicklung in .NET in Zusammenarbeit mit einem Kooperationsunternehmen, weiter über das Redesign eines Unternehmensplanspiels, das von einem Kollegen der Wirtschaftsinformatik betreut wurde, bis zu verteilten graphischen Editoren für ER-Diagramme und Aktivitätsdiagramme.

Als Einleitung zu Aspekten des Umgangs der Projektteilnehmer miteinander wurde an beiden Hochschulen eine Persönlichkeitsanalyse angelehnt an Meredith Belbin (Belbin, 2010) gemacht.

Das Ergebnis zeigt, dass es sich bei Studierenden in der Selbsteinschätzung um typische Mitarbeiter eines Teams handelt, die ein Team auch zusammenhalten können. Typische Führungspersonen sind seltener, Vorantreiber durchaus vorhanden. Personen mit Tendenz zu Marketing und Vertrieb (RI) sind durchgehend selten.

Die aufgrund der Projektgröße von ca. einem halben Personenjahr notwendige Aufgabenteilung mit der Zuteilung verschiedener Rollen im SWE-Prozess führte für viele Beteiligte zu dem gewünschten Lerneffekt, die Notwendigkeit von Softskills praktisch zu erfahren. Kommunikation, Organisation, Präsentation und Führung wurden eingeübt. Kritisch wurde von Studierenden und Lehrenden angemerkt, dass die eigentliche Zeit zur Umsetzung der Projekte relativ knapp ist.

Literatur

Belbin, M. (2010), Management Teams, 3. Auflage, Elsevier, Oxford, UK