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

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

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

Zusammenfassung

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

1 Einleitung

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

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

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

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

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

2 Fallbeispiel: Das Software-Praktikum an der Universität Zürich

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

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

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

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

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

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

3 Methoden und Techniken

3.1 Didaktisches Ziel: Praxisnähe

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Weitere Maßnahmen:

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

3.2 Didaktisches Ziel: Motivation der Teilnehmer

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

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

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

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

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

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

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

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

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

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

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

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

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

3.3 Didaktisches Ziel: Motivation der Tutoren

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

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

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

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

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

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

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

3.4 Didaktisches Ziel: Erfolg im Umgang mit unterschiedlichen Vorkenntnissen

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

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

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

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

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

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

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

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

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

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

3.5 Didaktisches Ziel: Individualität und Wirksamkeit der Leistungskontrolle

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4 Diskussion

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

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

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

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

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

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

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

eröffnet sich die Möglichkeit, fehlende Programmiervorkenntnisse gezielt auszugleichen.

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

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

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

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

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

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

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

Danksagung

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

Literatur

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

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

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

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

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

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

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

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

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

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

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

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

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