J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

Integration agiler Prozesse in die Softwaretechnik-Ausbildung im Informatik-Grundstudium

Petra Becker-Pechau, Wolf-Gideon Bleek, Axel Schmolitzky, Heinz Züllighoven

Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik,
{becker, bleek, schmolitzky, zuellighoven}@informatik.uni-hamburg.de

Zusammenfassung

Objektorientierte Programmierung zu lehren erfordert einiges an Didaktik, da die Objektorientierung stark in die Praxis der Softwareentwicklung eingebunden ist. Das Paradigma muss aber nicht nur praktisch, sondern auch konzeptionell in den Software-Entwicklungsprozessen eingeordnet werden. Die Bezüge reichen von der Analyse über den Entwurf und die Implementierung bis hin zur Integration in bestehende Softwarelandschaften. Nach unseren Erfahrungen ist für eine solche Einordnung die klassische Veranstaltungsform mit Vorlesung und begleitenden, kleinen, bindungslosen Übungen nicht mehr angemessen. Wir planen deshalb für das Sommersemester 2003, die in die Objektorientierung einführende Grundstudiumsveranstaltung neu zu strukturieren. Dabei sollen Prinzipien agiler Entwicklungsprozesse in den Übungsteil eingehen. Agile Methoden sind durch ihre kurzen Projektzyklen, die sich iterativ entwickelnden Architekturen und die vielschichtigen Rückkopplungsmechanismen aus unserer Sicht besonders geeignet. Die Studierenden lernen dadurch relevante Fragenstellungen der objektorientierten Softwareentwicklung in kurzer Zeit aus eigener Erfahrung kennen und können darüber in Rücksprachen diskutieren. So können dann Konzepte in der Vorlesung auf wissenschaftlichem Niveau reflektiert werden.

1 Einführung

Objektorientierte Programmierung (OOP) hat in der Informatik-Lehre der Universität Hamburg eine lange Tradition. Bereits Anfang der 90er Jahre wurde eine zweisemestrige Vorlesung im Hauptstudium mit begleitenden Übungen angeboten. Daran nahmen pro Jahr 50 bis 60 Studierende teil. Sie lösten in den Übungen kleine

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

abgeschlossene Aufgaben mit geringem Bezug zu realen Problemen. Deshalb änderten wir die Lehrveranstaltung so, dass die Studierenden nach einer Phase der Stoffvermittlung eine größere Projektaufgabe bearbeiten sollten [5]. Diese Projektarbeit wurde wiederholt durch Stoffvermittlung unterbrochen. Die neue Veranstaltungsform wurde jeweils von ca. 30 Studierenden besucht, die in Gruppen von fünf bis sieben an einer selbstgewählten Aufgabe bis zu einem Jahr gearbeitet haben.

Inzwischen ist OOP ein Teil des Grundstudiums geworden. Dieses Hineinwachsen in das Grundstudium hat in der neuen Informatik-Studienordnung dazu geführt, dass imperative und objektorientierte Konzepte in einem Semester vermittelt werden müssen. Die entsprechende Veranstaltung *P2* findet im Sommersemester statt und hat um die 350 Teilnehmende. Als Programmierkenntnisse der Studierenden kann lediglich der Stoff von *P1* vorausgesetzt werden, der die funktionale und logische Programmierung umfasst.

In diesem Artikel reflektieren wir die bisherigen Erfahrungen aus *P2*, benennen Probleme und schlagen ein neues Konzept für diese einsemestrige Veranstaltung vor, das erstmals im Sommersemester 2003 an der Universität Hamburg umgesetzt werden wird. Abschließend werfen wir die Frage auf, ob die grundlegende Organisation der Lehrveranstaltung mit den gewünschten Lernzielen zu vereinbaren ist.

Der folgende zweite Abschnitt beschreibt zuerst die Ziele der Veranstaltung. Dann erläutern wir die bisherige Struktur der Lehrveranstaltung in Abschnitt 3 und zeigen in Abschnitt 4 die hierbei beobachteten Probleme auf. Im Abschnitt 5 legen wir das neue Konzept für die Lehrveranstaltung dar. Abschließend diskutieren wir eine grundsätzliche Umgestaltung der Lehre objektorientierter Konzepte.

2 Ziele der Veranstaltung „Objektorientierte Programmierung“

Um OOP angemessen zu lehren, müssen zum einen etliche inhaltliche Themen berücksichtigt werden:

- Grundelemente der imperativen Programmierung
- Ablaufsteuerung, Prozedurbegriff, Datentypen
- Wert und Objekt
- Objekt und Klasse
- Schnittstelle, Kapselung und Sichtbarkeit
- Vererbung
- Objektorientierte Typsysteme
- Modularisierung, Zugriffsrechte
- Referenzen und dynamische Datenstrukturen
- Abstrakte Datentypen und ihre Umsetzung
- Vertragsmodell
- Systematisches Testen

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

OOP als Tätigkeit kann mit Blick auf elementare softwaretechnische Anforderungen aber nicht ohne die folgenden Aspekte vermittelt werden:

- *Objektorientierte Konstruktion ist Konzeptarbeit gepaart mit „handwerklicher Tätigkeit“*
  Programmieren ist eine Tätigkeit, die eingeübt und praktisch erfahren werden muss. Erst auf dieser handwerklichen Grundlage lassen sich nachhaltig die dahinter stehenden Konzepte vermitteln und gewinnbringend einsetzen.
- *Zusammenhang von Analyse, Entwurf und Konstruktion*
  In einer Grundstudiumsveranstaltung sollte wenigstens in Ansätzen klar werden, dass (objektorientierte) Programmierung die konstruktive Lösung eines wenig strukturierten Anwendungsproblems liefern kann. Programmierung sollte als eine Form der Modellierung verstanden werden.
- *Professionelle objektorientierte Programmierung ist Teamarbeit*
  In Projekten arbeiten Informatiker fast immer in einem Team. Dadurch unterscheidet sich professionelle Programmierung von individueller Freizeitprogrammierung. Arbeitsteilung und Kommunikation gehören deshalb ebenso zur Teamarbeit [6] wie Pünktlichkeit, Verlässlichkeit, Planung und Kontrolle, soziales Verhalten und Sensibilität gegenüber anderen.

Objektorientierte Programmierung umfasst sehr viele, teilweise recht anspruchsvolle programmiersprachliche Konzepte. Diese Konzepte können nach unserer Sicht nur auf der Basis eigener praktischer Programmiererfahrungen eingeordnet und genutzt werden.

3 Die bisherige Veranstaltung

Zunächst halten wir die formalen Randbedingungen für die Lehrveranstaltung im Grundstudium der Hamburger Informatik fest:

- Die Veranstaltung umfasst ein Semester.
- Den Studierenden wurden zuvor noch keinerlei Kenntnisse über imperative oder objektorientierte Konzepte vermittelt.
- Das Format ist mit 2 SWS Vorlesung und 2 SWS Übung vorgegeben. Diese Aufteilung war nicht unumstritten. Von einigen Lehrenden wird auch die Ansicht vertreten, dass eine 3-stündige Vorlesung mit einer 1-stündigen Großübung aufgrund der Kapazitätsengpässe sinnvoll sei.
- Es stehen für die ca. 350 Teilnehmer nur sehr begrenzte Betreuungskapazitäten zur Verfügung. Zurzeit werden 13 Übungsgruppen mit einer Gruppenstärke von 15 bis 30 Personen angeboten. 4 bis 5 dieser Übungsgruppen werden von Studierenden betreut, die anderen von wissenschaftlichen Mitarbeitern.
- In den Übungsgruppen muss ein Leistungsnachweis erbracht werden. Dazu gibt es ein Punktesystem für die wöchentlich ausgegebenen Übungsblätter.

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

- Die gesamte Lehrveranstaltung soll im Semester in einem festen wöchentlichen Zeitraster stattfinden.

Die Lehrveranstaltung wird somit klassisch in Vorlesungen und Übungen aufgeteilt. Während die wöchentliche Vorlesung die bereits aufgeführten Themen und Aspekte vermittelt, dienen die Übungen der Wiederholung, Vertiefung und praktischen Erfahrung. Wöchentlichen Übungsblätter enthalten kleine Programmieraufgaben, mit denen elementare Programmierung erlernt und Themen der Vorlesung vertieft werden sollen. Daneben wird immer eine Textaufgabe gestellt, an der die Studierenden den vermittelten Stoff reflektieren sollen.

Die zweistündigen Übungen finden in rechnerlosen Seminarräumen statt. Lösungen können über einen Overheadprojektor oder einen Beamer mit Laptop präsentiert werden. Die Termine sind so strukturiert:

- Inhalte der Vorlesung wiederholen und zusammenfassen.
- Notwendige Vorkenntnisse für das nächste Übungsblatt vermitteln / wiederholen.
- Präsentation und Diskussion einer oder mehrerer studentischer Lösungen des zwei Wochen zuvor verteilten Übungsblattes. Dieses Übungsblatt wurde durch den Übungsgruppenleiter bereits korrigiert und bewertet; die Ergebnisse werden am Ende des Übungstermins verteilt. Die Präsentation der Lösungen ist ein Teil der Kriterien für den Leistungsnachweis.

Die Studierenden sollen die Übungen in Arbeitsgruppen von zwei bis vier Personen bearbeiten. So sollen sie Teamarbeit lernen. Gleichzeitig reduziert diese Gruppenarbeit den Korrekturaufwand der Übungsgruppenleiter auf ein realistisches Maß. Jede Übungsgruppe hat fünf bis zehn Arbeitsgruppen.

Jedes Jahr werden in der Vorbereitung die Aufgabenblätter samt Musterlösungen (bestehend aus kommentierten Programmen und Erklärungstexten) neu gestaltet und in einem Autor-Kritiker-Zyklus überarbeitet. Jedes Aufgabenblatt kostet mit seiner Musterlösung etwa eine Woche Arbeit, die sich durch den Überarbeitungsprozess über einen längeren Zeitraum verteilt.

Im letzten Semester wurde die Lehrumgebung BlueJ (siehe www.bluej.org) zum erstenmal eingesetzt. BlueJ ist eine einfach zu bedienende Programmierumgebung für objektorientierte Programmierung mit Java, die speziell für die Ausbildung von Programmieranfängern entwickelt wurde. BlueJ ermöglicht, den "Objects first"-Ansatz 1 zu lehren, d.h. die Studierenden können schon in den allerersten Aufgaben vorgegebene Klassen und Objekte „anfassen". Wenn diese Kapsel schrittweise geöffnet wird, lernen die Studierenden das Inenleben von Objekten (Exemplarvariablen, Methoden mit Variablen, Ausdrücken und Anweisungen) und damit die imperativen Grundkonzepte kennen. Dieser Ansatz vermeidet unter anderem die didaktischen Klimmzüge bei der erstmaligen Erklärung des Java-Konstrukts "public static void main(String[] args)", das mehr aus Ausnahmen als aus Basiskonzepten besteht.

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

4 Beobachtete Probleme

Vergleichen wir die Ziele der Veranstaltung mit ihrer Realisierung, so zeigen sich deutliche Diskrepanzen. Insbesondere die in Abschnitt 2 aufgeführten Lernziele, die über technische Inhalte hinausgehen, lassen sich in dieser Form nur schwer vermitteln.

- *Zusammenhang von Analyse, Entwurf und Konstruktion*
  Übungsaufgaben für diese drei Aspekte lassen sich nicht in einer Woche lösen. „Folgefehler“ wären nicht zu vermeiden, so würde z.B. eine schlechte Analyse oder ein unangemessener Entwurf zu einer mangelhaften Konstruktion führen. Unmittelbares persönliches Feedback zu Zwischenlösungen ist hier essentiell. Die Verteilung über mehrere Übungsblätter löst dieses Problem auch nicht, da vor Ausgabe der Folgeaufgabe die vorangegangene persönlich besprochen werden müsste. Dazu müsste die abgegebene Lösung aber auch schon korrigiert worden sein.
- *Objektorientierte Konstruktion ist Konzeptarbeit gepaart mit „handwerklicher Tätigkeit“*
  Erst auf Basis von handwerklicher Programmiererfahrung lassen sich dahinter stehende Konzepte vermitteln und in ihrer Tragweite begreifen. Studierende benötigen meist mehrere Wochen, um die ersten programmiersprachlichen Fähigkeiten zu lernen. Durch parallele Übungen und Vorlesungen werden aber abstrakte Konzepte vermittelt, deren Erfahrung in den Übungen noch aussteht. Da in den Übungsstunden nicht programmiert wird, kann der Übungsgruppenleiter keine handwerklichen Fähigkeiten vermitteln. Viele Probleme oder Fortschritte lassen sich aber nach unserer Erfahrung nur direkt bei der Programmierung erkennen.
- *Professionelle objektorientierte Programmierung ist Teamarbeit*
  Obgleich die Studierenden ihre Aufgaben in Arbeitsgruppen von zwei bis vier Personen bearbeiten sollen, ist dies nicht notwendigerweise Teamarbeit. Oft werden die Aufgaben unter den beteiligten Personen aufgeteilt, um den Aufwand für den Leistungsnachweis zu optimieren. Die Übungsgruppenleiter kontrollieren die Leistung und geben Rückkopplung durch die Korrektur der Übungsaufgaben und anhand der Präsentation der Übungslösungen durch die Studierenden während der Übungstermine. Die Teamarbeit bleibt den Studierenden überlassen. Einwöchige Übungsaufgaben sind aber im Umfang zu gering, um die Koordination und Planung von Teamarbeit einzuüben.

Während der Veranstaltungen der letzten Jahren sind diese Konflikte zwischen Zielen und Durchführung immer deutlicher geworden. Es zeigten sich verschiedene Probleme:

- *Hoher Vorbereitungsaufwand*
  Jedes Jahr neue Übungsblättern zu erstellen ist aufwändig und teilweise frust-

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

rierend. Es ist schwierig möglichst kontext-unabhängige Aufgaben zu finden, weil die Themen aufeinander aufbauen und Trennlinien oft künstlich gezogen werden müssen. Für eine interessante und nachprüfbare Veranstaltung müssen aber jedes Jahr überarbeitete Übungen gefunden werden.

- *Hoher Durchführungsaufwand*
  Das schriftliche Korrigieren der Aufgabenlösungen ist sehr zeitaufwändig (ca. 4 bis 6 Zeitstunden pro Woche).
- *Geringer Praxisbezug*
  Aufgrund des (notwendigerweise) begrenzten Umfangs der Aufgaben ist ihr Praxisbezug minimal. Dies gilt insbesondere für Aufgaben zu fortgeschrittenen Konzepten wie Modularisierung oder Vererbung. Die Studierenden bekommen leicht den Eindruck, dass es bei der Software-Entwicklung eher um kleine, zusammenhangslose Aufgaben geht als um die langfristige Arbeit an einem größeren Projekt.
- *Keine Berücksichtigung von Vorkenntnissen*
  Die Aufgaben sind für einige Studierende zu einfach, für andere zu schwierig. Dies langweilt Studierende mit Vorkenntnissen in der ersten Hälfte der Veranstaltung, während schwächere Studierende in der zweiten Hälfte überfordert sind. In beiden Fällen besteht die Gefahr, dass die Teilnahme wegen Misserfolg abgebrochen wird, denn Studierende mit vielen Vorkenntnissen verpassen häufig den Moment, an dem ihre Vorkenntnisse nicht mehr ausreichen.
- *Zu kurze Übungstermine*
  Die zweistündigen Übungstermine sind zu kurz für den mit ihnen verbundenen Anspruch. Allein die Vertiefung des Vorlesungsstoffs würde die Zeit ausfüllen; zusätzlich müssen jedoch alte Aufgaben besprochen und neue Aufgaben vorgestellt werden.
- *Zu wenig individuelles Feedback*
  Die individuelle Rückkopplung kommt für die Studierenden zu kurz. Gute Lösungen können nicht ausreichend gewürdigt werden, bei schlechten Lösungen wären persönliche Gespräche hilfreich. So lassen sich z.B. Programmierrichtlinien am besten direkt am Bildschirm diskutieren.
- *Arbeitsteilung statt Gruppenarbeit*
  Studierende können sich für eine Gruppenlösung eintragen, ohne an der Lösung mitgearbeitet zu haben. Die Tendenz steigt, die Veranstaltung lediglich als „Punktesammeltreffen“ aufzufassen, um den notwendigen Leistungsnachweis zu bekommen. Die Gruppenarbeit wird durch die Studierenden oft lediglich zur Effizienzsteigerung benutzt.
- *Zu viel Raum für Täuschungsversuche*
  Die aktuelle Gestaltung der Übungen erleichtert den Austausch von Lösungen unter den Studierenden sehr, da dies über Übungsgruppen hinweg nur schwer festzustellen ist. Täuschungsversuche frustrieren sowohl die ehrlichen Studierenden als auch die Übungsgruppenleitungen.

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

Neben den organisatorischen Problemen und den in einem prüfungsorientierten Lehrbetrieb üblichen Reibungsverlusten fällt die Abstimmung zwischen Lehrinhalt und den didaktisch-organisatorischen Formen besonders ins Gewicht. Die klassische Trennung zwischen Vorlesung und Übung passt offensichtlich schlecht zu den vermittelten Inhalten. Durch die strengen Rahmenbedingungen (Punktevergabe, Anwesenheit, Leistungsnachweis) verkehren sich die begrenzten didaktischen Mittel (Gruppenarbeit, Versuch von Praxisbezug) ins Negative.

5 Der neue Ansatz: Agile Prozesse in der Softwaretechnik-Ausbildung

Der neue Ansatz soll im Grundstudium einen realistischen Eindruck von objektorientierter Softwareentwicklung vermitteln und dabei die vorgenannten Probleme vermeiden. Als Nebeneffekt soll die Arbeitsbelastung der Mitarbeiter in einem guten Verhältnis zum Lernerfolg der Studierenden stehen.

In einem projektartigen Übungsbetrieb [2, 8] werden die Studierenden durch eigene Erfahrung an die typischen Aufgaben und Fragestellungen der objektorientierten Softwareentwicklung herangeführt. Die Übungsgruppenleiter können bei der Projektarbeit viel unmittelbarer und effektiver Rückkopplung geben und den Lernerfolg verfolgen, als dies mit Korrekturen von Übungszetteln der Fall ist.

Projekte nach agiler Vorgehensweise werden in kurzen Iterationszyklen durchgeführt, in denen alle typischen Entwicklungsaufgaben von der Analyse bis zur Konstruktion enthalten sind. Ein solcher Übungsbetrieb vermittelt den Studierenden innerhalb kürzester Zeit relevante Erfahrungen. Vielfältige Rückkopplungsmechanismen, von der Arbeit in Paaren [10] bis zur Abnahme einer Softwareversion am Ende einer Iteration, können den Lernerfolg beschleunigen.

Die ursprüngliche Idee

Das neue Konzept sollte ursprünglich eine größere Projektaufgabe umfassen, an der die Studierenden während der gesamten 14 Wochen des Semesters nach Aspekten des Extreme Programming [4] arbeiten. Diese Aufgabe sollte genügend Raum bieten, um die unterschiedlichen Aspekte von OOP zu erlernen. Eine absichtlich unscharfe Aufgabenstellung sollte als roter Faden durch den Stoff der Veranstaltung führen. Diese Aufgabenstellung sollte in kleinen Gruppen projektartig bearbeitet werden.

Es waren regelmäßige Rückkopplungstreffen zwischen dem Übungsgruppenleiter, nun besser als Tutor bezeichnet, und den ihm zugeordneten Gruppen geplant. Dabei sollten Entwurfsentscheidungen und Implementierungen diskutiert und bewertet werden.

Der Tutor sollte über die gesamte Zeit einen kontinuierlichen Entwicklungs- und Rückkopplungsprozess anleiten, der wiederholt durch neue Teilaufgaben angerei-

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

chert wird. Einige wenige Projektthemen sollten zur Verfügung stehen (z.B. „Pizza Lieferservice“, „Autovermietung“, „Taxizentrale“, „Kinokartenverkauf“, „Bibliotheksausleihe“) [3].

Dieser Ansatz erfordert allerdings, dass die Studierenden bereits elementare imperative und objektorientierte Programmierkenntnisse besitzen. Da diese im Rahmen der Veranstaltung erst vermittelt werden, haben wir uns für einen Kompromiss entschieden.

Der geplante pragmatische Ansatz

Die zur Verfügung stehende Zeit wird aufgeteilt in eine *Laborphase* und eine *Projektphase*. Die Laborphase soll das „Programmieren im Kleinen“ abdecken, während in der Projektphase fortgeschrittene Konzepte (ADTs, Modularisierung, Sichtbarkeitsregeln etc.) anhand einer größeren Projektaufgabe vermittelt werden. Die Laborphase soll acht Wochen, die Projektphase sechs umfassen. Die Vorlesung soll weiterhin die Konzepte der objektorientierten Programmierung vermitteln. Idealerweise werden besonders in der Projektphase die Themen der Vorlesung den unmittelbaren Anforderungen aus den anstehenden Projektaufgaben angepasst.

Laborphase in Zweiergruppen

In der Laborphase werden wie bisher Aufgabenblätter verteilt und wöchentlich bearbeitet. Statt einer schriftlichen Lösung werden die Aufgaben in betreuten Laborsituationen bearbeitet und testiert. Auf diese Weise entfallen der 4- bis 6-stündige Korrekturaufwand für die Übungsgruppenleiter und das Warten der Studierenden auf Rückkopplung. Diese Zeit verbringen sie stattdessen in den Labors, während sich die Studierenden jeweils zu Paaren zusammenfinden und gemeinsam am Rechner die Aufgaben bearbeiten. Die persönliche Betreuung ermöglicht dem Tutor, die individuellen Vorkenntnisse sowie Probleme der einzelnen Studierenden während der Programmierung wahrzunehmen. So kann der Tutor sehr viel unmittelbarer und persönlicher Feedback geben und erhalten. Wir planen Laborblöcke von jeweils drei Zeitstunden. Da innerhalb dieser Blöcke auch die Anteile der Übungen stattfinden sollen, die bisher in den zweistündigen Übungen positioniert waren, kann ein Tutor bei gleichem Zeitaufwand zwei dieser Blöcke im Labor betreuen (siehe die Aufstellung am Ende dieses Abschnitts).

Projektphase mit Gruppen von 6-8 Studierenden

In dieser Phase werden wir die bisherige Veranstaltungsform am deutlichsten verändern. Lediglich Studierende, die an der Laborphase erfolgreich teilgenommen haben, werden für die Projektphase zugelassen, da elementare Programmierkenntnisse eine notwendige Voraussetzung für die Programmierarbeit im Projekt darstellen.

Die Laborzeiten werden in der Projektphase fortgeführt, jedoch mit anderem Schwerpunkt. Die Studierenden sollen sich erneut zu Gruppen zusammenfinden,

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

diesmal zu Gruppen von sechs oder acht (drei oder vier Paare). Jede dieser Gruppen bildet ein Projektteam. Für die Projektorganisation werden Konzepte agiler Entwicklungsmethoden genutzt [7]. Dies umfasst u.a. die Arbeit in Paaren, die Organisation in Iterationen sowie das Prinzip des einfachen Entwurfs und Anpassungen der Konstruktion mit Hilfe von Refactorings [9].

Die Organisation als Projekt ermöglicht, die in Abschnitt 2 aufgeführten Aspekte in der Lehre direkt zu adressieren:

- Der Zusammenhang von Analyse, Entwurf und Konstruktion wird im Projekt erfahren. Zwar werden Analyse und Entwurf durch den Übungsgruppenleiter angeleitet und teilweise vorgegeben, da im Grundstudium die Konstruktion im Vordergrund steht. Dennoch muss im Projekt analysiert und entworfen werden.
- Relevante Fragestellungen ergeben sich auch aus der Größe der Aufgabenstellung und der projektartigen Organisation und werden von den Tutoren aufgegriffen und in Zusammenhang mit der Vorlesung gestellt. So kann das „handwerkliche Begreifen“ konzeptionelle Überlegungen anregen.
- Die Arbeit in Teams an einer gemeinsamen, längeren Aufgabe ermöglicht, typische Herausforderungen der Teamarbeit in objektorientierten Projekten zu erfahren. Die Studierenden werden sensibilisiert für Fragen der Aufgabenplanung und -teilung. Übungsblätter mit kleinen, genau spezifizierten Aufgaben legen fälschlicherweise nahe, dass es für Aufgaben korrekte, eindeutige Lösungen gibt, die direkt zu Beginn der Konstruktion geplant werden können. Diese Gefahr ist in der Projektarbeit geringer. Hier soll vorrangig über mehr oder weniger angemessene Entwurfs- und Konstruktionsalternativen vor dem Hintergrund der unscharfen Anforderungen diskutiert werden. Es soll deutlich werden, dass „richtig“ und „falsch“ keine objektiven Kriterien für Softwareentwürfe sind. Neue Anforderungen im Verlauf des Projekts regen zu Umstrukturierungen (Refactorings [9]) an und unterstreichen den vorherigen Punkt.

Zu Beginn der Projektphase wird den Studierenden in einer konstituierenden Sitzung das Projektthema vorgestellt (etwa "eine Verwaltungssoftware für ein Kino erstellen"). Sie erhalten ein Kernsystem, das minimale Anforderungen bereits erfüllt. Diese Kernsystem gibt eine Grundarchitektur der zu erstellenden Anwendung vor; es wird im Laufe des Projekts von den Studierenden erweitert und überarbeitet.

Während des 6-wöchigen Projekts werden den Studierenden drei Impulse für 2-wöchige Iterationen gegeben, indem sie weitere Anforderungen erhalten. Dies kann z.B. sein „Implementieren Sie eine Kundenverwaltung!“ Während sie versuchen, die domänenspezifischen Inhalte zu erarbeiten, müssen sie gleichzeitig die korrespondierenden objektorientierten Konzepte finden. Wöchentliche Treffen mit dem Tutor begleiten diesen Prozess. Der Tutor erläutert die Aufgaben, gibt Hilfestellung beim Entwurf und der Aufgabenverteilung und kommentiert die Ergebnisse. Er leitet den

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

Prozess an und sorgt für eine tragfähige Gesamtarchitektur, die er implizit in den Design-Diskussionen motiviert. Unterstützt wird er hierbei durch das vorgegebene Kernsystem. Auf diese Weise erhalten die Studierenden eine Modellarchitektur, ohne dass dieses Thema explizit aufgegriffen werden muss. Modellarchitekturen sind Thema weiterführender Veranstaltungen im Hauptstudium. In den wöchentlichen Treffen präsentiert die Projektgruppe ihre Arbeit der vergangenen Woche. Der Tutor stellt Fragen. Die Vorstellung wird vom Tutor protokolliert; Fragen und offene Probleme werden für den nächsten Termin festgehalten. Der Tutor wird der Gruppe spezifische Anforderungen stellen, die dann umgesetzt werden sollen. Da die neuen Anforderungen auf der Basis bereits geleisteter Arbeit umgesetzt werden, wird das bereits Gelernte wiederholt.

Der Lernerfolg wird in der Projektphase nicht mehr durch gelöste Übungsaufgaben definiert, sondern über die Fähigkeit, neu angeeignete Konzepte für eine reale Aufgabenstellung anzuwenden. Dies ist kein formales Kriterium, sondern muss den Umständen entsprechend bewertet werden. Auch in der Projektphase betreut der Tutor die Studierenden während der Programmierung. Dadurch kann er die Fortschritte, das Verständnis und die individuellen Probleme der einzelnen Projektmitglieder weiterhin verfolgen und ummittelbares Feedback geben.

Randbedingungen des neuen Ansatzes

Betreuungsaufwand: An der Veranstaltung werden voraussichtlich 320 Studierende teilnehmen. Die Studierenden werden in Gruppen von 20 Personen jeweils 3 Zeitstunden in der Universität an ihren Aufgaben arbeiten sowie in dieser Zeit Rücksprache mit dem Tutor halten. Es werden immer zwei Gruppen parallel in benachbarten Räumen stattfinden; diesen 40 Studierenden werden zwei Tutoren und eine studentische Hilfskraft für technische Fragen zur Verfügung stehen. Ein wissenschaftlicher Mitarbeiter mit einer Lehrverpflichtung von 2 SWS wird zwei solcher Gruppen betreuen, also insgesamt 6 Stunden anwesend sein. Dies entspricht dem Zeitaufwand der klassischen, übungsorientierten Organisation (siehe Abschnitt 4), da in der neuen Organisationsform die zusätzlichen Korrekturen der Übungsaufgaben wegfallen. Diese Tätigkeit wird in Anwesenheit der Studierenden ausgeführt.

Die folgende Gegenüberstellung der Aufwände in Zeitstunden/Woche für eine Lehrverpflichtung von 2 SWS verdeutlicht, dass sich die Aufwände in der Betreuung nicht erhöhen.

Bisheriger übungsorientierter Lehrbetrieb:

Koordinationstreffen der Übungsgruppenleiter	1
Übungstermin (mit 26 Personen):	2
Korrekturen und Vorbereitung des Übungstermins:	5
Gesamtsumme	8

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

Neues Konzept (in beiden Phasen):

Koordinationstreffen der Übungsgruppenleiter	1
Betreuung von zwei Gruppen á 20 Personen in Anwesenheit	6
Vor- und Nachbereitung	1
Gesamtsumme	8

Bisher wurden durchschnittlich 26 Studierende von einem Tutor betreut, bei ca. 320 Studierenden wurden also 13 Betreuer benötigt. Jetzt werden zwei Gruppen mit jeweils 20 Studierenden von einem Tutor und zusätzlich in Halbzeit von einer studentischen Hilfskraft betreut. Dies ergibt einen Bedarf von 9 Tutoren und 4 studentischen Hilfskräften.

Wir gehen also davon aus, dass der zeitliche Aufwand sich nicht erhöhen wird. Hinzu kommt, dass während der Projektphase die Erstellung von detailliert ausgearbeiteten Aufgabenzetteln und Musterlösungen wegfällt. Lediglich die Zeiteinteilung der Tutoren wird unflexibler.

Das neue Konzept verlangt deutlich höhere Kompetenzen der Tutoren durch die Bewertung und Hilfestellung in Anwesenheit sowie die implizite Anleitung der Gesamtarchitektur eines wachsenden Systems bei unscharfer Aufgabenstellung. Als Tutoren werden deshalb ausschließlich wissenschaftliche Mitarbeiter eingesetzt. Studentische Hilfskräfte bewerten die Lösungen nicht, wie es in der vorherigen Organisationsform der Fall war, sondern unterstützen die Tutoren bei der Gruppenbetreuung.

6 Kritische Überlegungen

Das neue Modell für die Grundstudiums-Lehrveranstaltung ist für uns ein Kompromiss, der unsere Wunschvorstellungen mit den aktuellen Möglichkeiten im Hamburger Grundstudium verbindet. Allerdings fragen wir uns, ob nicht eine radikalere Revision des Lehrekonzepts notwendig ist.

Universitäre Informatik umfasst zahlreiche Themen – die anwendungs-, d.h. praxisorientierte, Softwareentwicklung nach dem objektorientierten Paradigma ist nur eines. Doch scheint uns ein Informatikstudium ohne solide handwerkliche Fähigkeiten in der Softwarekonstruktion verbunden mit den dahinter stehenden Konzepten schwer vorstellbar. Dies weist einer Lehrveranstaltung „objektorientierte Softwareentwicklung“ im Grundstudium ihren Platz zu. Hier sollen, aus unserer Sicht, dieses Handwerkszeug und die dazu passenden Konzepte elementar vermittelt werden.

Wenn wir uns von den aktuellen organisatorischen und vor allem strukturellen Randbedingungen befreien könnten, würden wir eine andere Vorgehensweise wählen. Denn wir haben umfangreiche Erfahrungen in der beruflichen Aus- und Weiterbildung von konventionell arbeitenden Programmierern zu objektorientierten Anwendungsentwicklern. Dort gehen wir, kurz gefasst, so vor:

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

- Wir vermitteln zunächst ganz elementare Begriffe und Konzepte der Programmierung (z.B. Variable, Algorithmus, Ablaufsteuerung) im Seminarstil. Dabei wechseln kurze Vortragseinheiten mit kleinen „Fingerübungen“ in betreuter Gruppenarbeit ab.
- Nach ca. einer Woche lernen die Kursteilnehmer intensiv (d.h. ganztägig) die Basiskonstrukte einer objektorientierten Programmiersprache. Dabei werden vom Dozenten jeweils einzelne Konstrukte und Sprachmerkmale (teils interaktiv am Rechner) vorgestellt, die dann unmittelbar in kleinen Übungen ausprobiert werden. Dabei arbeiten die Teilnehmer in Programmierpaaren (nach den Prinzipien des Extreme Programming [4]) unter Anleitung eines erfahrenen Trainers.
- Nach zwei bis drei Wochen beherrschen die Teilnehmer die Basiskonstrukte. Dann werden weitergehende Techniken und Konstruktionen vorgestellt und in „Miniprojekten“ umgesetzt. Neben den Konstruktionsaufgaben steht die Projektarbeit im Team an erster Stelle. Konzepte werden dann vom Projekt-Coach vorgestellt, wenn sie ein aufkommendes Problem im Projekt lösen helfen.

Wir haben diese Ausbildungsform in mehreren innerbetrieblichen Schulungszyklen erprobt. Dass sie deutlich erfolgreicher ist, lässt sich durch zwei Feststellungen belegen:

- Die Resonanz ist sehr positiv und das Engagement der Teilnehmer ist sehr viel höher als bei unseren früheren Ausbildungsseminaren, die sich eher an konventionellen Schulungsformen orientierten.
- Die Erfolgsquote ist sehr hoch. Durch Rücksprachen können wir die fachlichen Kenntnisse der Teilnehmer gut einschätzen. Da wir oft auch als Coaches in nachfolgenden Projekten des Unternehmens arbeiten, können wir auch die Eignung der ehemaligen Trainees für die Projektarbeit einschätzen. Zwischen 80 und 90 % der Teilnehmer erreichen das gesteckte Kursziel.

Was bedeutet dies bezogen auf die universitäre Ausbildung?

- Das Verhältnis von Vorlesung, d.h. Vortrag, und Übung muss umgekehrt werden. Die Übungen dürfen nicht mehr nur Erläuterungen dessen sein, was in der Vorlesung an Konzept und Theorie vorgetragen wird. Die praktische Auseinandersetzung mit Aufgaben und Projektthemen wird zum treibenden Motor. Wenn dabei Fragen aufkommen, können konzeptionelle Lösungsansätze direkt aufgegriffen und verarbeitet werden. Damit werden Vortragseinheiten ereignisgetrieben oder situativ in den Lernprozess integriert. Wenn genügend Erfahrungswissen vorliegt, lassen sich in der Reflexion weitergehende Konzepte bruchlos in die Projektarbeit einbringen.
- Wenige elementare Grundkenntnisse der Programmierung reichen aus, um mit der problem- und aufgabenorientierte Projektarbeit zu beginnen. Soweit sie bei

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

Studierenden nicht ohnehin aus der Schulzeit vorhanden sind, können sie leicht in einem einführenden Schnellkurs vermittelt werden.

- Die Orientierungen an wöchentlichen Aufgaben mit „objektiven“ Punktesystemen drängen die Übungsgruppen in die Richtung konventioneller Schulformen – Übungsaufgaben werden gestellt und Ergebnisse werden mit Blick auf eine Musterlösung bewertet und besprochen. Dies verstellt den Blick für die Probleme und Lösungsstrategien in der praktischen Softwareentwicklung. Statt der schulischen Bewertung von „richtig“ und „falsch“ sollte eine Diskussion über „angemessen“ und „unangemessen“ treten, in der dann die unterschiedlichen Sichtweisen und Wertesysteme von Auftraggebern und Benutzern sowie den Entwicklern deutlich werden.
- Erfahrungen im Programmieren und in der Projektarbeit lassen sich am besten intensiv, d.h. ganztägig, vermitteln. Diese Einsicht verträgt sich nicht mit den konventionellen Lehreschemata deutscher Universitäten. Die Zwänge, die sich aus den festgelegten Veranstaltungsformen und -rastern, der Raumvergabe und der Verknüpfung von Hauptstudium und Nebenfach ergeben, verhindern länger laufende Projektveranstaltungen. Einzelne deutsche und vor allem zahlreiche skandinavische Universitäten haben allerdings erfolgreich diese Umstrukturierung realisiert.
- Die Studierendenzahlen und die personelle Ausstattung der universitären Informatik passen nicht zusammen. Länger laufende Studienprojekte mit intensiver Softwareentwicklung erfordern ein hohes Maß an qualifizierter Betreuung. Die unmittelbare Diskussion von Programmentwürfen und rasche Hilfen bei Konstruktionsproblemen erfordern eine höhere Qualifikation als die Korrektur von schriftlichen Übungsaufgaben. Diese Qualifikation gibt es in der universitären Informatik derzeit nicht in ausreichendem Maße.
- Der Aufwand für Lehre und Betreuung im Studium steht heute in keinem gesunden Verhältnis zu den Lernerfolgen. Unsere eigenen Erfahrungen aus der Übungsbetreuung, den Grundstudiumsprüfungen und der Zusammenarbeit mit Studierenden im Hauptstudium sagen uns, dass nicht annähernd die Erfolgsquoten oder das Teilnehmerengagement unserer Ausbildungsseminare erreicht werden. Wobei wir, in aller Bescheidenheit, trotzdem der Meinung sind, dass wir unseren Studierenden eine vergleichsweise hochwertige Ausbildung mitgeben.

Vielleicht zeigt sich an der Problematik der Grundstudiumslehre im Bereich objektorientierter Programmierung das Dilemma der universitären Informatik in Deutschland: Wie lassen sich die Ansprüche nach einer Ausbildung zum Wissenschaftler mit den Anforderungen der gesellschaftlichen Praxis verbinden? Wir glauben nicht, dass darin ein unüberbrückbarer Widerspruch liegt; vielmehr sollte in einer wesentlich von Informationstechnologie geprägten Gesellschaft eine praxisbezogene Informatik ebenso selbstverständlich sein wie eine forschungsnahe Praxis.

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

Literatur

1. Barnes, D.J., and Kölling, M.: Objects First with Java – A Practical Introduction using BlueJ, Prentice Hall / Pearson Education, New York, 2003.
2. Bastian, J., and Gudjons, H.: Das Projektbuch, Bd. 1, Theorie, Praxis, Erfahrung. Bergmann/Helbig, Hamburg, Germany, 4. Auflage, 1994.
3. Bastian, J., and Gudjons, H.: Das Projektbuch, Bd. 2, Über die Projektwoche hinaus, Projektlernen im Fachunterricht. Bergmann/Helbig, Hamburg, 3. Auflage, 1998.
4. Beck, K.: eXtreme Programming explained – embrace change, Addison-Wesley, Boston, 2000.
5. Bleek, W.-G., Gryczan, G., Lilienthal, C., Lippert, M., Roock, S., Wolf, H., Züllighoven, H.: Von anwendungsorientierter Softwareentwicklung zu anwendungsorientierten Lehrveranstaltungen - der Werkzeug & Material-Ansatz in der Lehre, Software Engineering im Unterricht der Hochschulen (SEUH) 99, Berichte 52, B. Dreher/Ch. Schulz/D. Weber-Wulff (Hrsg.), Workshop des German Chapter of the ACM und der Gesellschaft für Information (GI) am 25. und 26. Februar 1999 in Wiesbaden, S. 9-20, 1999.
6. Clark, H. H., Brennan, S. E.: Grounding in Communication. In Resnick, L., Levine, J. M., Teasley, S. D. (eds.): Perspectives on Socially Shared Cognition. Washington, USA, 1991.
7. Cockburn, A.: Agile Software Development, Addison-Wesley, Boston, 2002.
8. Dewey, J.: How We Think: A Restatement of the Relation of Reflective Thinking to the Educative Process. Chicago, USA, 1933.
9. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston, 1999.
10. Williams, L., Kessler, R.: All I Really Need to Know about Pair Programming I Learned in Kindergarten, *Communicationsof the ACM*, Vol. 43, No. 5, 2000, S. 108