Ein Softwaretechnik-Praktikum als Sommerkurs

Christian Lindig, Andreas Zeller
Lehrstuhl für Softwaretechnik, Universität des Saarlandes,
Postfach 15 11 50
66041 Saarbrücken
{lindig,zeller}@cs.uni-sb.de

Zusammenfassung

Ein semesterbegleitendes Softwaretechnik-Praktikum verleitet die Teilnehmer dazu, vor lauter Begeisterung und Gruppendruck andere Veranstaltungen zu vernachlässigen. An der Universität des Saarlandes wird das Praktikum daher in der vorlesungsfreien Zeit absolviert, und zwar als 6-wöchiger Vollzeitkurs mit begleitender Vorlesung. Diese Form wirkt studienverkürzend und vereinfacht die Teamarbeit durch Ganztags-Präsenz. Risikomindernde Maßnahmen wie ein einheitliches, vorgegebenes Pflichtenheft, spielerische Elemente und automatisch testbare Erfolgskriterien sorgen für hohe Motivation bei reibungslosem Ablauf.

1 Stand der Dinge

Die Universität des Saarlandes hat 2001 im Rahmen der Umstellung auf ein Bachelor/Master-System ein neues Softwaretechnik-Praktikum eingeführt, das mit 14 von 30 ECTS-Punkten etwa die Hälfte des 3. Semesters einnahm. Das Ziel des Praktikums ist das systematische Programmieren im Grossen im Team. Dazu durchlief im Praktikum jedes Team aus fünf Studenten fünf Phasen:

- Anforderungsanalyse, abgeschlossen mit Pflichtenheft,
- Grobentwurf, abgeschlossen mit Objektmodell und Sequenzdiagrammen,
- Feinentwurf, abgeschlossen mit Prosa-Beschreibung und Unit-Tests,
- Implementierung, abgeschlossen mit fertigem Produkt,
- Test, abgeschlossen mit Testbericht.

Die Gliederung und die Aufgabenstellung waren betont realistisch. Dazu gehörte, dass die Aufgabe von lehrstuhlfremden *Kunden* gestellt wurden – dies waren Mitarbeiter der Informatik, die im Rahmen von Forschungsprojekten geeignete Programmieraufgaben zu vergeben hatten, von der inhaltlichen Betreuung des Praktikums aber entbunden waren. Im Laufe des Semesters gab es Meilensteine, zu denen die entsprechenden

Phasendokumente vorliegen mussten und bewertet wurden. Die Softwaretechnik-Vorlesung fand zeitgleich statt; sie war im Inhalt auf den Fortschritt im Praktikum abgestimmt.

Obwohl das Praktikum zu den beliebtesten Veranstaltungen des Grundstudiums gehörte, gab es eine Reihe von Problemen:

- **Zeitaufwand.** Unter Druck von Gruppe und Kunden neigten Studierende dazu, ihre Arbeit besonders gut zu machen und darüber andere Vorlesungen zu vernachlässigen [DHZS99]. Dies wirkte tendenziell studienverlängernd.
- **Gruppen aus schlechten Studenten**. Die Leistung von Informatikern schwankt bekanntlich in großem Maße [DHZS99]. Wir haben festgestellt, dass gute Studenten gerne mit anderen guten Studenten eine Gruppe bilden, was die Leistungsunterschiede der Gruppen potenziert – und schlechtere sowie unkommunikative Studenten alleine zurücklässt.
- **Gruppen aus guten Studenten.** Gruppen nur aus guten Studenten handeln sich Probleme ein, wenn sie glauben, dank ihrer Programmierkompetenz Entwurf und Gruppenkommunikation vernachlässigen zu können – was sich oft als Fehleinschätzung herausstellt.
- **Schlechte Abstimmung von Vorlesung und Praktikum.** Die das Praktikum begleitende Vorlesung ist in den allgemeinen Vorlesungsplan des Semesters eingebunden. Dieser favorisiert eine *lineare* Vermittlung von Stoff, die aber nicht dem Bedarf in einem Praktikum entspricht: Themen wie Entwurf und Gruppenarbeit müssen vermehrt am Anfang des Praktikums vermittelt werden, während in späteren Phasen nur noch eine praktische und individuelle Betreuung nötig ist.
- **Mangelnde Vergleichbarkeit**. Die Aufgabenstellungen der Kunden unterschieden sich in Schwierigkeit, Zeitaufwand und didaktischer Eignung beträchtlich [DHZS99]. Als Folge war die von den Teilnehmern geforderte Leistung kaum objektiv zu vergleichen oder zu bewerten.
- **Schwierige Betreuung**. Die Aufgaben der Kunden erforderten tendenziell bereichsspezifische Architekturen und Algorithmen (Netzwerke, Computer-Grafik, Numerik, Information Retrieval), die die betreuenden Mitarbeiter und Studenten sich erst spontan aneignen mussten.
- **Schlechte Skalierbarkeit**. Die Anzahl der Teilnehmer von Praktikum zu Praktikum schwankte stark und war oft erst kurzfristig bekannt. Eine entsprechende Anzahl von Kunden und damit Aufgabenstellungen zu finden war damit ein wiederkehrendes Problem. Eine große Anzahl verschiedenen Aufgabenstellungen verschärft die Probleme der Betreuung und Vergleichbarkeit.

Aus diesen Erfahrungen heraus haben wir beschlossen, das Praktikum rundzuerneuern und komplett neu zu organisieren[1].

[1] http://www.st.cs.uni-sb.de/edu/sopra/2004/

2 Sechs Wochen Vollzeit

Unsere erste (und wesentliche) Entscheidung war, *das Praktikum vom sonstigen Vorlesungsbetrieb zu trennen* und in der vorlesungsfreien Zeit („Semesterferien") zu veranstalten. Hierfür gibt es eine Reihe von Gründen:

- In der vorlesungsfreien Zeit gibt es keine Konflikte mit anderen Vorlesungen oder sonstigen Verpflichtungen.
- Studierende benötigen die vorlesungsfreie Zeit nicht zur Prüfungsvorbereitung, da im Bachelor-Studiengang alle Prüfungen studienbegleitend stattfinden.
- Studierende können in der Vorlesungszeit an Stelle des weggefallenen Praktikums weitere Veranstaltungen besuchen, was die Studiendauer verkürzt.

Die vorlesungsfreie Zeit im Sommer umfasst etwa drei Monate. Wir gönnen den Studierenden sechs Wochen für Urlaub und andere Aktivitäten, womit sechs Wochen für das Praktikum zur Verfügung stehen. Dies bedingt, dass das Praktikum in Vollzeit absolviert werden muss – für Studierende wie Betreuer. Vollzeit hat für die Studierenden viel gutes, da sie den ganzen Tag vor Ort sind und sich so die Teams ganz auf die Aufgabe konzentrieren können. Auch Betreuer und Dozenten kommen zu kaum etwas anderem, was allerdings in der folgenden Vorlesungszeit ausgeglichen wird: Im kommenden Wintersemester bietet unser Lehrstuhl keine Veranstaltungen an und kann sich für sechs Monate der Forschung widmen.

Ein Praktikum in den Semesterferien konkurriert nicht mit anderen Veranstaltungen, ermöglicht volles Engagement der Studierenden und verkürzt die Studienzeit.

3 Risiken vermeiden

Ein von 13 auf 6 Wochen reduziertes Software-Praktikum muss sich auf den wesentlichen Stoff konzentrieren und lässt allen Beteiligten weniger Raum für Fehler. Anders als im semesterbegleitenden Praktikum können Gruppen kaum Fristverlängerungen gewährt werden. Eine *homogene Struktur* minimiert diese Risiken:

- **Zufällig zusammengestellte Gruppen.** Im Gegensatz zu früheren Praktika, wo die Studierenden selbst Gruppen bildeten, haben wir die Gruppen zufällig zusammengestellt. Dies sorgt für eine gleichmäßigere Verteilung der Leistungen und größere Diversität in den Gruppen. Da sich die Gruppen anfangs nicht einschätzen können, werden sorgfältiger Entwurf und gute Teamarbeit als willkommene Mittel zur Risikominimierung empfunden.

- **Wegfall der Anforderungsanalyse.** In einem 6-Wochen-Praktikum fehlt die Zeit, sich in fremde Anwendungsgebiete einzuarbeiten. Daher haben wir auf Kunden und die Analyse ihrer Anforderungen im Pflichtteil verzichtet.
- **Eine Aufgabenstellung.** Alle Gruppen bearbeiten eine identische Aufgabenstellung an Hand eines Pflichtenheftes, für das zuvor eine Referenzimplementierung erstellt wurde.
- **Einheitliche, testbare Erfolgskriterien.** Es muss ein faires (und weitgehend automatisierbares) Verfahren geben, um den Erfolg festzustellen. Dies bestimmt maßgebend die Wahl der Aufgabenstellung.
- **Inhalt statt Oberflächen.** Gut gestaltete graphische Oberflächen (GUIs) können mehr als die Hälfte der Implementierungsarbeit ausmachen, sind aber gleichzeitig schwer zu testen und im Hinblick auf Ergonomie nur aufwändig zu bewerten. Für ein 6-Wochen-Praktikum können GUIs daher nur optionale Teile sein.

Ein einheitlicher, detailliert geplanter Praktikumsablauf minimiert Risiken.

4 Ablauf des Praktikums

Um die in Abschnitt 3 aufgeführte Struktur umzusetzen, haben wir das Praktikum in drei Phasen aufgeteilt, dargestellt in Abbildung 1:

- Zu Beginn erhalten die Studierenden ein einheitliches, fertig ausgearbeitetes *Pflichtenheft* mit vollständig automatisch testbaren Anforderungen.
- In den ersten zwei Wochen soll das System entworfen werden. Bereits nach einer Woche muss das *UML-Objektmodell* (Klassen und ihre Beziehungen) stehen. Meilenstein nach einer weiteren Woche ist der vollständige Entwurf, der neben einem überarbeiteten Objektmodell eine Reihe von *Standard-Szenarien* mit Sequenzdiagrammen und Unit-Tests beschreibt. Alle Entwürfe werden in Kolloquien mit den Gruppen durchgesehen.
- Nach abgeschlossenem Entwurf wird ein *überarbeitetes Pflichtenheft* ausgegeben, in dem sich einige Details geändert haben. Die Studierenden wissen zu Beginn, dass sich Details ändern können (aber nicht, welche Details das sind) und streben angesichts dieses Risikos einen möglichst flexiblen Entwurf an.
- In den folgenden zwei Wochen wird das System implementiert. Meilenstein ist hier das Bestehen eines vorgegebenen *automatischen Tests*, der die gesamte Funktionalität des Systems abdeckt. Zum Einreichen und Testen der Programme setzen wir Praktomat [Zel00] ein; Plagiaten spüren wir mit JPlag [PMP00] nach.
- In den letzten zwei Wochen wird das System eingesetzt. Ohne Kunden muss es für die Studierenden eine weitere Motivation geben, das System auf Herz

und Nieren zu prüfen. Meilenstein ist daher die Qualifikation für ein *Turnier*, in dem die besten Systeme gekürt werden.

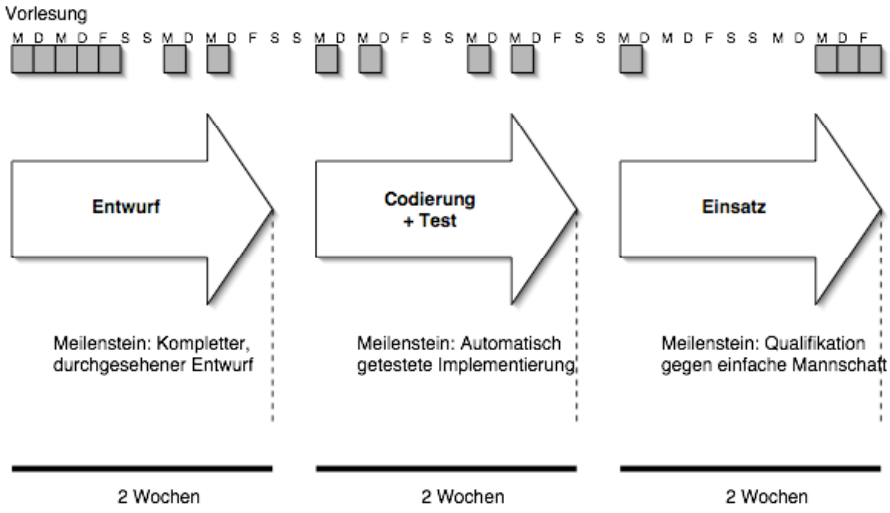

Abb. 1.: Schematischer Ablauf des Praktikums

Die Forderung nach einem objektorientierten Entwurf innerhalb von zwei Wochen bedingt eine große Menge an Stoff, der zu Beginn behandelt werden muss. Konkret haben die Studierenden in der ersten Woche jeden Morgen zwei Vorlesungsstunden absolviert – mit Themen wie Teamarbeit, objektorientierte Modellierung, Entwurfsmuster, Architekturmuster, und Dokumentation. Ab der zweiten Woche fand die begleitende Vorlesung zweimal wöchentlich statt, mit dann implementierungsorientierten Themen wie Unit-Tests, Refactoring, Qualitätssicherung oder Fehlersuche. Zum Ende des Praktikums war keine Vorlesung mehr nötig. Man bemerke, dass solch unregelmäßige Vorlesungstermine in der Vorlesungszeit nur schwer durchzuführen sind; so ist es kaum möglich, den größten Hörsaal für alle Vormittage der ersten Semesterwoche zu requirieren.

Insgesamt haben wir die Vorlesung auf den Fortschritt im Praktikum abgestimmt – einerseits zeitlich, aber auch *inhaltlich:* Wir vermitteln das Material, was die Studierenden für das Praktikum brauchten, und sonst nichts. Anders gesagt: Die Vorlesung richtet sich nach dem Praktikum, und nicht umgekehrt. Anforderungsanalyse, Prozessmodelle, Spezifikation, Zertifizierung nach ISO 9000, Management von Softwareprojekten – all dies sind wichtige Themen der Softwaretechnik, aber in unseren Augen erst dann gut vermittelbar, wenn die Teilnehmer die Probleme des Programmierens im Großen selbst erlebt haben. Daher behandeln und üben wir diese Stoffe in einer Stammvorlesung „Softwaretechnik" ab dem 4. Semester.

In einem kurzen Praktikum muss man genau das unterrichten, was die Studierenden zum Erfüllen der Aufgabe benötigen.

5 Eine spielerische Aufgabenstellung

In jedem Softwaretechnik-Praktikum muss man eine Aufgabe finden, die nur im Team bewältigt werden kann. Bei uns lösen alle Gruppen die gleiche Aufgabe: In einer zweidimensionalen Welt konkurrieren zwei Schwärme von *Bugs* um Futter. Es gilt einen Schwarm zu entwickeln, der mehr Futter in sein Nest trägt als ein vorgegebener Schwarm.

Alle Bugs eines Schwarms werden durch denselben endlichen Automaten kontrolliert, der in einem sehr einfachen Maschinencode spezifiziert wird (Beispiel in Abbildung 2) – Bugs können sich bewegen, ihre Umwelt erfühlen und einfache Markierungen hinterlassen, etwa um Wege zu kennzeichnen.

Dieser Code wird von einem *Simulator* eingelesen, der die Automaten zweier Bug-Schwärme in einer gegebenen Welt ausführt und so bestimmt, welcher Schwarm erfolgreicher ist (Abbildung 3).

Zustand	Code	Kommentar
0	`sense ahead 1 3 food`	Wenn Futter voraus, weiter bei 1; sonst bei 3
1	`move 2 0`	Gehe auf Futter, weiter bei 2; sonst bei 0
2	`pickup 8 0`	Nimm Futter auf und weiter bei 8
3	`flip 3 4 5`	Suchen: Münzwurf; weiter bei 4 oder 5
4	`turn left 0`	Drehe nach links; weiter bei 0
5	`flip 2 6 7`	Münzwurf; weiter bei 6 oder 7
6	`turn right 0`	Drehe nach rechts; weiter bei 0
7	`move 0 3`	Nach vorne gehen; weiter bei 0
8	`sense ahead 9 11 home`	Rückweg: Wenn Nest voraus, weiter bei 9
9	`move 10 8`	Gehe auf Nest...
10	`drop 0`	... und lege Futter ab; Neubeginn bei 0
11	`flip 3 12 13`	Wieder mit Münzwurf suchen (wie 3-7)
12	`turn left 8`	
13	`flip 2 14 15`	
14	`turn right 8`	
15	`move 8 11`	

Abb. 2.: Ein (sehr einfacher) Bug sucht erst Futter und dann sein Nest

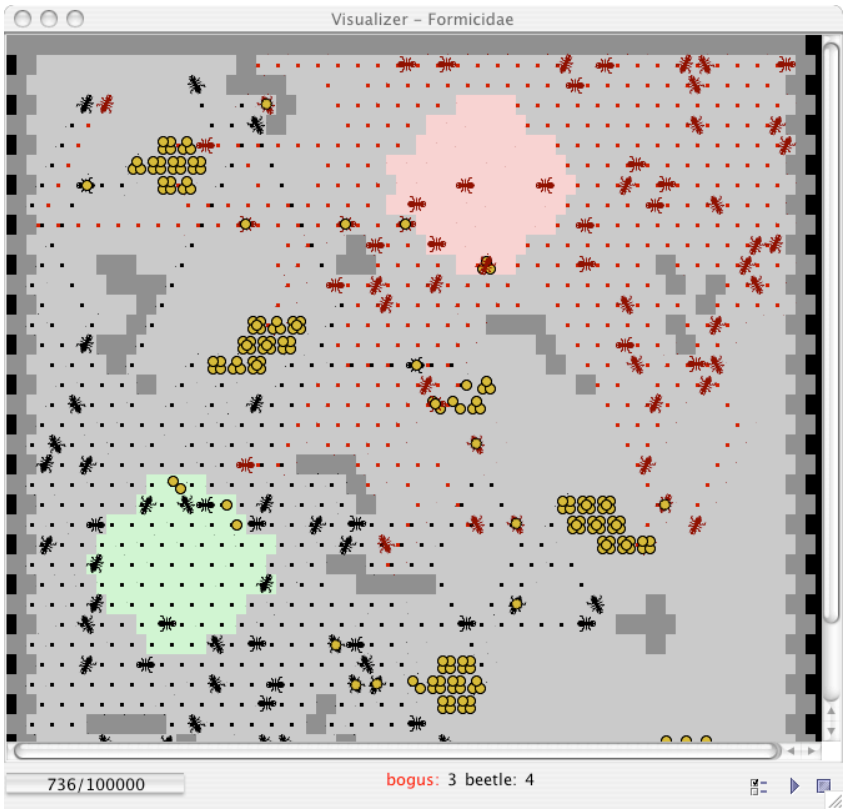

Abb. 3.: Simulator mit schwarzen und roten Bugs, Futter und zwei Nestern

Die Studierenden müssen einen solchen Simulator und einen erfolgreichen Schwarm entwickeln. Die Aufgabe vereinigt Aspekte, die unterschiedlichen Kenntnisse und Talente in einer Gruppe ansprechen:

- Der *Simulator* ist eine klassische Aufgabe für einen objektorientierten Entwurf [GR83, DM91]. Er lässt sich präzise spezifizieren und validieren, was ihn für ein Praktikum gut geeignet macht.
- Die Entwicklung der *Strategie* und ihre Kodierung als endlicher Automat hingegen bieten Raum für Originalität. Die endlichen Automaten für das Verhalten werden in einer maschinennahen Sprache formuliert.
- In einem *Turnier* am Ende des Praktikums stehen die Gruppen im sportlichen Wettstreit um die besten Schwarm-Strategien. Dies stachelt Studenten zu Spitzenleistungen an und verhindert den allzu freien Austausch von Ideen und Code [H95].
- Um eine gute Strategie zu entwickeln, benötigt man *Werkzeuge* - zunächst einen korrekten und leistungsfähigen Simulator (was Testbarkeit und Qualitätssicherung motiviert). Darüber hinaus lassen sich intelligente Bugs

leichter in einer selbst entworfenen Hoch- oder Assemblersprache formulieren, die von den absoluten Adressen und dem expliziten Kontrollfluss der Maschinensprache abstrahieren. Kenntnisse der theoretischen Informatik können zur Minimierung von Automaten verwandt werden. Als Folge hängt der Turniererfolg wesentlich von der Qualität der selbst entwickelten Werkzeuge ab.

Trotz ihrer weiterführenden Merkmale ist die unmittelbar zu lösende Aufgabe überschaubar. Während ein ausdifferenzierter Entwurf etwa 30 Klassen enthält, kann ein monolithischer Entwurf mit 10 Klassen auskommen; Abbildung 4 vergleicht die Größe verschiedener Implementierungen. Eine moderarte Komplexität garantiert allen Gruppen ein Erfolgserlebnis, selbst wenn die softwaretechnischen Ziele nur ausreichend erreicht werden.

Die Aufgabenstellung spezifiziert die Syntax und Semantik der Sprache für endliche Automaten und das Datei-Format der Umwelt. Die Spezifikation ist eine Mischung aus informaler Beschreibung, Grammatiken zur Definition der Syntax und Programmfragmenten in der funktionalen Sprache ML zur Spezifikation der operationalen Semantik des Simulators.

Implementierung	**Code-Zeilen**	**Module**	**Methoden**
Referenz. ML	1635	10	141
Referenz, C++	2223	32	134
22 Lösungen, C++, Durchschnitt	2432	24	
22 Lösungen, C++, Minimum	1146	12	
22 Lösungen, C++, Maximum	4629	45	

Abb. 4.: Größe von Implementierungen im Vergleich

In der überarbeiteten Aufgabenstellung, die die Studenten nach erfolgreichem Entwurf erhielten, wurde die Geometrie des Spielfeldes von vier- auf sechseckige Zellen geändert, ein Zustand im Automaten verboten und verschiedene Wertebereiche verändert.

Die Korrektheit jedes Simulators wird durch einen Vergleich mit einer Referenzimplementierung geprüft. Jeder Simulator erzeugt Log-Dateien, die den vollständigen Zustand der simulierten Welt nach jedem Simulationszyklus enthalten. Die Log-Dateien der Referenzimplementierung stehen den Studierenden während der Entwicklung zur Fehlersuche zur Verfügung. Eine lange Simulation von zahlreichen Bugs in einer kleinen Welt kann auch unwahrscheinliche Situationen provozieren, die so zu einer guten Abdeckung führen und mögliche Fehler aufdecken.

Eine solch vielseitige Aufgabenstellung ist zu schade, um sie nur für ein Praktikum einzusetzen. Tatsächlich ist die Aufgabe eine isomorphe Kopie der Aufgabe des ICFP Programming Contest [ICFP04]. Dort hatten internationale Gruppen 72 Stunden Zeit, einen möglichst leistungsfähigen Schwarm zu entwickeln, der gegen die Schwärme

anderer Gruppen antreten soll. Während der Schwerpunkt des Praktikums auf der Implementierung des Simulators liegt, lag er im Wettbewerb auf der Entwicklung einer Hochsprache, um leistungsfähige Strategien ausdrücken zu können. Die Tatsache, dass es im Contest 230 Gruppen schafften, innerhalb von 72 Stunden einen Simulator und einen Schwarm zu implementieren, gab uns die Sicherheit, eine trotz ihrer Vielseitigkeit nicht zu große Aufgabenstellung gefunden zu haben.

Ein Wettbewerb am Ende des Praktikums steigert das Teamgefühl und stachelt die Studierenden zu Spitzenleistungen an.

6 Betreuung

Am Praktikum nahmen 117 Studierende teil, die zufällig in 22 Gruppen zu 5 Studierenden aufgeteilt wurden [M04]. Jeweils drei dieser Gruppen waren einem studentischen Tutor zugeordnet, der das Software-Praktikum bereits erfolgreich absolviert hatte und Hilfestellung geben konnte. Aufgabe eines Tutors war, die Erfolgschancen der Gruppe zu maximieren, dabei aber selbst nicht mitzuarbeiten – etwa wie ein Trainer im Sport. Entsprechend dieser Rolle waren Tutoren auch nicht mit der Bewertung von Leistungen befasst.

Der typische Tagesablauf eines Studierenden sah in der zwei Wochen dauernden Entwurfsphase so aus:

09:00-11:00 Uhr: Vorlesung
11:00-12:30 Uhr: Erstes Treffen mit Tutor (alle drei Gruppen gemeinsam)
13:00-17:30 Uhr: Arbeit in der Gruppe
17:30-18:00 Uhr: Zweites Treffen mit Tutor (jede Gruppe einzeln)

In den Vormittags-Treffen wurden die Ziele für den Tag festgesetzt, meist aufgrund des gerade behandelten Stoffs der Vorlesung. In den Nachmittags-Treffen wurden die Ergebnisse besprochen, und zwar für jede Gruppe einzeln. Die lange Pause zwischen den Treffen bewirkt, dass alle Gruppenmitglieder präsent sind und sich tatsächlich voll und ganz der Gruppenarbeit widmen können.

Da alle Gruppen vor den gleichen Entwurfsproblemen standen, stellten sich schnell Entwurfsmuster heraus, die zunächst von einzelnen Tutoren an andere von ihnen betreute Gruppen weitergegeben wurden, aber auch später beim Treffen der Tutoren diskutiert wurden und so ihren Weg in Vorlesung und andere Gruppen fanden. Tatsächlich wiesen alle Entwürfe ähnliche Grundkonzepte auf.

In der Implementierungsphase trafen sich die Gruppen nur noch einmal täglich mit ihrem Tutor. Zu diesem Zeitpunkt war das Gruppengefühl schon so weit gefestigt, dass weitergehende disziplinarische Regeln nicht mehr notwendig waren. Hier gaben die

Tutoren Hilfestellungen zum Programmieren und Fehlersuche; auch hier wurden auftretende Probleme weitergemeldet, so dass sie schnell behoben werden konnten.

In den zwei Wochen der Einsatzphase waren unsere Tutoren im Wesentlichen arbeitslos, da die Gruppen an ihren Strategien arbeiteten und die Tutoren ihnen nicht mehr folgen konnten. Stattdessen wurden die Tutoren zur Durchführung des Turniers eingesetzt.

Ständige Präsenz vereinfacht die Teamarbeit und verringert Risiken.

7 Prüfung

Ein Software-Praktikum ist eine Prüfungsleistung und soll deswegen so objektiv und fair wie möglich geprüft werden. Die Prüfung teilt sich in Teilprüfungen, die zum Abschluss jeder Phase stattfanden.

- In der Entwurfsphase beurteilte ein Prüfer den Entwurf in zwei Kolloquien mit Gruppe und Tutor. Hierbei ging der Prüfer den Entwurf mit der Gruppe durch, prüfte, ob die gesamte Gruppe mit dem Entwurf vertraut war, und gab eine Reihe von Auflagen mit auf den Weg, um den Entwurf zu verbessern. Die Prüfer sind die Dozenten und wissenschaftlichen Mitarbeiter des Lehrstuhls für Softwaretechnik; sie bieten die höchste Qualifikation für die Diskussion von Entwurfs- und Konstruktionsproblemen.
- In der Implementierungsphase musste der Simulator einen Systemtest bestehen. Einreichung und Test wurden durch Praktomat [Zel00] automatisiert und basierte auf der Referenzimplementierung, so dass hier keine manuelle Prüfung stattfand.
- In der Einsatzphase musste der selbst programmierte Schwarm ein Spiel gegen eine gegebene (schwache) Mannschaft bestehen. Dies wurde durch einen Lauf des offiziellen Simulators der Referenzimplementierung entschieden.
- Am Ende des Praktikums fand ein Turnier statt, in dem die Schwärme gegeneinander antraten. Die Teilnahme war freiwillig; es gab Buchpreise für die Gewinner sowie Sonderpreise für den schönsten Simulator und das beste Programmierwerkzeug.
- Das Praktikum wurde formal durch ein Kolloquium abgeschlossen, in dem die Gruppen ihrem Prüfer noch einmal Details des Simulators und der Strategie vorstellten; Hauptzweck dieses Kolloquiums war, sicherzustellen, dass alle Gruppenmitglieder sich ausreichend beteiligt hatten und den Studierenden Raum für Lob und Kritik zu bieten.

Auch wenn wir einen großen Teil der Arbeit automatisieren konnten, bleibt die Organisation eines solchen Praktikums nicht-trivial. Neben den Tutoren waren Dozent und drei wissenschaftliche Mitarbeiter über zehn Wochen Vollzeit eingespannt – und

das bei weitgehend vorgegebener Aufgabenstellung. Die sechswöchige Durchführung benötigt eine fast ebenso lange Vorbereitungsphase, in der die Aufgabenstellung, geschrieben, eine oder zwei Referenzimplementierungen erstellt und die technische organisatorische Infrastruktur (Praktomat, Tutorials, Installation von Software) vorbereitet wird.

Tutoren sollten keinen Einfluss auf die Benotung haben. So können sie allein mit ihrer Erfahrung Ratschläge geben, die von den Gruppen besser angenommen werden.

8 Evaluation

Trotz der Umstellung lief das Praktikum problemlos. Wir freuen uns, dass wir die Risiken eines 6-Wochen-Praktikums erfolgreich reduzieren konnten – und das bei hoher Erfolgsquote:

- Alle Gruppen haben fristgerecht den Meilenstein der Entwurfsphase erreicht und einen tragfähigen objektorientierten Entwurf abgeliefert.
- Alle Gruppen haben fristgerecht den Meilenstein der Implementierungsphase erreicht und eine Implementierung abgeliefert, die alle Tests bestanden hat.
- Alle Gruppen hatten bereits fünf Tage vor Praktikumsende den Meilenstein der Einsatzphase erreicht und somit das Praktikum erfolgreich abgeschlossen.

„Alle Gruppen“ bedeutet nicht „alle Studenten“: Einige wenige Teilnehmer wurden in den ersten zwei Wochen wegen Nichtbeteiligung oder mangelnder Kenntnisse ausgeschlossen.

Die studentische Evaluation dokumentiert die Wirkung der Verbesserungen – hier im Vergleich mit dem (semesterbegleitenden) Praktikum des Vorjahres:

- Der durchschnittliche Arbeitsaufwand ist kaum verändert – statt semesterbegleitend 22 Stunden/Woche hatten wir nun 40 Stunden/Woche Vollzeit.
- Die Arbeitsbelastung war für 40% „gerade richtig“ (zuvor 0%), für 60% „zu hoch“ (zuvor 55%) und für 0% „viel zu hoch“ (zuvor 45%) – insgesamt also eine leichte Überforderung, die genau so erwünscht ist.
- Die gute Stimmung drückte sich auch in der Bewertung der Dozenten und Materialien aus, die (obwohl weitgehend identisch) in sämtlichen Fragen zur Lehreffektivität mit 0,5–0,7 Notenstufen besser bewertet wurden.
- Die größte Steigerung (1 Notenstufe) gab es in der Bewertung der Tutoren, was für unser Betreuungskonzept spricht.
- 80% wünschen sich eine zufällige oder auf Fähigkeiten basierte Zusammensetzung der Gruppen, wie bei uns eingesetzt.
- Alle Teilnehmer (mit einer Ausnahme) meinen, das Praktikum solle in der vorlesungsfreien Zeit stattfinden.

Ein Softwarepraktikum in den Semesterferien ist bei gleichem Arbeitsaufwand deutlich weniger belastend.

9 Vergleich

Der Unterschied zwischen einem Praktikum in einer semesterbegleitenden Form mit diversen Kunden und Aufgaben und einem 6-wöchigen Kurs ohne Kunden und einheitlicher Aufgabe ist gewaltig. Deswegen ist es nicht immer einfach, die Faktoren für den Erfolg der neuen Form zu benennen. Einen interessanten Vergleichspunkt zwischen diesen Extremen bietet das Softwarepraktikum der TU Kaiserslautern [M04].

Das Praktikum in Kaiserslautern ist semesterbegleitend, hat aber ansonsten viele Merkmale unseres neuen Praktikums: eine einheitliche Aufgabe (Ampelsteuerung), eine Modellierungsphase mit UML, die frühe Erstellung von Testfällen, Gruppen von 6 Studierenden und ein abschließender Wettbewerb. Metzger berichtet in [M04] von einem insgesamt erfolgreichen Verlauf, gegenüber einem früheren Praktikum einem engeren und besseren Notenspektrum und einer positiven Evaluierung. Dies deckt sich mit unserer Erfahrung, dass eine einheitliche Aufgabe die Vergleichbarkeit von Leistungen fördert und der risikoärmere Verlauf zu größeren Erfolgen der Teilnehmer führt.

Wegen der Ähnlichkeit zu unserem Praktikum sind die von Metzger beobachteten Probleme interessant: 45% der Teilnehmer klagten über einen sehr hohen Arbeitsaufwand; außerdem sahen sich die Organisatoren gezwungen, Präsenzzeiten einzuführen, da Studierende dazu neigten, zu Hause zu arbeiten und dort nicht zu betreuen waren. Das Problem der Arbeitsbelastung (durch Konflikt mit anderen Verpflichtungen) deckt sich mit unseren früheren Erfahrungen und war kein Problem in unserem Vollzeit-Praktikum. Gleichzeitig bestand in unserem Praktikum eine Anwesenheitspflicht, die die Betreuung erleichterte und von den Teilnehmern positiv aufgenommen wurde. Dies legt nahe, dass der Wechsel zu einem Vollzeitpraktikum tatsächlich einige spezifische Probleme eines semesterbegleitenden Praktikums löst.

10 Fazit

Mit dem Softwaretechnik-Praktikum als 6-Wochen-Sommerkurs können wir nicht nur die vorlesungsfreie Zeit besser ausnutzen, sondern es können sich auch alle Beteiligten voll und ganz dem Praktikum widmen – sei es in der Teamarbeit, in der Betreuung, oder in der Vorlesung. Enthusiasmus geht nicht zu Lasten anderer Lehrveranstaltungen; in den Semesterferien ist Abstimmung von Vorlesung und Praktikum kein Problem; eine einheitliche Aufgabenstellung für alle ist fair, reduziert Risiken und vereinfacht das Finden genereller Lösungen in der Betreuung. Schließlich hat der Wettbewerb die

Motivation der Studierenden noch einmal erheblich gesteigert und so Freude an der Teamarbeit und am Programmieren im Großen vermittelt. Insofern haben wir zahlreiche Probleme früherer Praktika erfolgreich gelöst, was die Evaluation bestätigt.

Offensichtlicher Mangel eines Blockkurses ist, dass sich die Vorlesungsinhalte dem Praktikum unterordnen müssen. Erfahrungen aus früheren Veranstaltungen zeigen, dass Studierende im Praktikum genau an solchen Inhalten interessiert sind, die ihnen einen unmittelbaren Nutzen bringen. Weiterführende Themen der Softwaretechnik sollten in eigenen, spezifischen Übungen vermittelt werden. Ohnehin greift das Verständnis für Programmieren im Großen erst, wenn entsprechende Programmier- und Teamerfahrungen vorhanden sind – und diese werden im Praktikum gemacht. Was uns allerdings schmerzt, ist der Wegfall der Anforderungsanalyse, die wir wieder ins Praktikum einfließen lassen möchten. Dies kann so geschehen, dass ein Wahlpflichtteil wie „Erstellen einer Simulator-GUI“ oder „Model Checking von Schwarm-Automaten“ von einem „Kunden“ grob vorgegeben wird und über einige Iterationen in Abstimmung mit dem „Kunden“ verfeinert wird – ebenfalls innerhalb von sechs Wochen, parallel zur Hauptaufgabe.

In [BBSZ04] lesen wir, dass „Erfahrungen im Programmieren und in der Projektarbeit sich am besten intensiv, d.h. ganztägig vermitteln lassen“. Dem können wir nur zustimmen. Wenn es weiter heißt „Die Zwänge, die sich aus den festgelegten Veranstaltungsformen und –rastern, der Raumvergabe und der Verknüpfung von Hauptstudium und Nebenfach ergeben, verhindern länger laufende Projektveranstaltungen“, so schlagen wir als pragmatische, nunmehr erprobte Lösung vor, Projekte und Praktika in der vorlesungsfreien Zeit stattfinden zu lassen.

Vorlesungsfreie Zeit ist Praktikumszeit!

Bibliographie

[Zel00] Andreas Zeller: Making Students Read and Review Code. Proc. *5th ACM SIGCSE/SIGCUE Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE ‘2000)*, Helsinki, Finnland, Juli 2000, S. 89-92.

[PMP00] Lutz Prechelt, Guido Malpohl, Michael Philippsen: JPlag: Finding plagiarisms among a set of programs. Interner Bericht 2000-1, Universität Karlsruhe, März 2000.

[ICFP04] ICFP 2004 Programming Contest. http://www.cis.upenn.edu/proj/plclub/contest/index.php

[BBSZ04] Petra Becker-Pechau, Wolf-Gideon Bleek, Axel Schmolitzky und Heinz Züllighoven: Integration agiler Prozesse in die Softwaretechnik-Ausbildung im Informatik-Grundstudium. Proc. SEUH 8, dpunkt.verlag, Heidelberg, 2004.

[GR83] Adele Goldberg und David Robson: Smalltalk-80, The Language and ist Implementation, Part Three. Addison Wesely, 1983

[DM91] Jocelyn R. Drolet and Colin L Moodie: Object Oriented Simulation with Smalltalk-80: A Case Study. In Nelson, Kelton Clark (eds.): Proceedings of the 1991 Winter Simulation Conference, 1991.

[M04] Andreas Metzger: Konzeption und Analyse eines Softwarepraktikums im Grundstudium. Proc. SEUH 8, Seiten 41–48, dpunkt verlag, Heidelberg, 2004.

[DHZS99] Birgit Demuth, Heinrich Hußmann, et al.: Erfahrungen mit einem frameworkbasierten Softwarepraktikum. Proc. SEUH 6, Teubner-Verlag, 1999.

[H95] Eva Hornecker: Präsentationstechniken und Teamtraining für das Software-Praktikum. Proc. SEUH ’95, Seiten 69-81, Teubner-Verlag 1995.