Teambildung in Programmierung und Software-Engineering Kursen

Debora Weber-Wulff
Technische Fachhochschule Berlin

Die Software-Industrie fordert von unseren Absolventinnen und Absolventen die Fähigkeit, im Team arbeiten zu können. Unser Software-Curriculum ist aber nicht darauf eingerichtet, solche nicht-technischen Fähigkeiten zu vermitteln, und wir fragen uns, wie es am einfachsten in unser bestehendes Curriculum zu integrieren ist. Dieser Aufsatz will versuchen aufzuzeigen, wie Methoden aus dem allgemeinen Bildungsbereich für unsere Zwecke anwendbar wären, und gibt einige konkrete Beispiele für den Einsatz.

1 Was ist ein Team?

Wir verstehen unter einem Team eine Gruppe von zwei oder mehr Menschen, die gemeinsam eine Aufgabe lösen wollen. Die Gruppe muß miteinander kommunizieren, um die Aufgabe vernünftig lösen zu können. Darin liegt ein großes Problem: Kommunikation. Die an dem Team Beteiligten müssen schriftlich, mündlich und non-verbal zusammenarbeiten.

Die Fähigkeit zum Kommunizieren ist bei vielen leider nur schwach herausgebildet. Sie sind es eher gewöhnt, Information zu empfangen (durch Fernsehen oder Video, von Lehrpersonen, etc.) bzw. eine eigene Ansicht notfalls mit Gewalt durchzusetzen. Daher ist es unerläßlich, daß wir in unseren Informatik-Curricula nicht nur das Fachwissen vermitteln, sondern unseren Studentinnen und Studenten Gelegenheiten bieten, sich im Schreiben, Vortragen, Diskutieren, Kompromisse schließen und gemeinsames Arbeiten zu üben.

Zum Glück müssen wir Methoden zur Vermittlung solcher Fähigkeiten nicht neu erfinden. Wir können vieles von unseren Kolleginnen und Kollegen aus den erziehungswissenschaftlichen Fächern verwenden. Viele von den Übungen und Gedanken lassen sich auf unser Fach übertragen und mit Erfolg einsetzen. Dieser “Blick über den Zaun” fällt vielen sicherlich schwer, es wird häufig auf Erziehungswissenschaft und Didaktik mit gerümpfter Nase[1] herabgesehen. Ich werde in den folgenden Seiten einige Vorschläge unterbreiten, die im Hochschulunterricht angebracht wären und werde von eigenen Erfahrungen darin berichten.

2 Vorschläge

Dieser Abschnitt bietet einige Organisationsformen und Übungen für die Förderung von Teambildung.

2.1 Schreiben

Es herrscht seltsamerweise oft die Meinung, daß es in der Informatik nicht notwendig ist, "richtig" zu schreiben. Schon in den ersten Semestern gebe ich meinen Studierenden Aufgaben, bei denen sie etwas schreiben sollen, z.B. eine kleine Anforderungsdefinition, ein Vergleich von Struktogrammen mit Pseudocode, einen Testplan oder einen Testbericht erstellen. Rechtschreibung und Ausdrucksfähigkeit werden dabei bewertet.

Auch Programmcode muß ausreichend kommentiert werden, gravierende Grammatikfehler (mehr als drei Schreibfehler[2] oder ein nicht verständlicher Satz) verursachen Punktabzug. Ich biete an, Fehler in auf Englisch geschriebenen Kommentaren zu korrigieren, aber nicht zu bewerten, als Förderung der Englischkenntnisse. Leider machen nur ca. 10% der Studierenden von diesem Angebot Gebrauch.

2.2 Lesen

Programme sollen gelesen werden können wie Artikel im *Spiegel*. In einer Aufgabe im ersten Programmier-Kurs werden die Studierenden gebeten, Programme miteinander zu tauschen, und nur an Hand des Quellcodes zu entscheiden, ob er korrekt ist oder nicht. Außerdem soll ein kurzer Bericht erstellt werden – was ist gut, was ist schlecht am Programmierstil. Damit die Studierenden nicht das Gefühl haben, sich gegenseitig "in die Pfanne zu hauen" wenn Fehler gefunden werden, gebe ich allen die volle Punktzahl für die eigentliche Aufgabe (wird nicht vorher angekündigt!), ziehe aber Punkte ab, wenn sie Fehler in ihrem Bericht übersehen haben. Das macht viel Korrekturarbeit für mich, ist aber eine gute Übung sowohl im Lesen als auch im Schreiben.

[1] "Das ist doch alles nur Spielkram", sagte eine Kollegin zu mir, die sich sogar für Ingenieursdidaktik interessiert!

[2] Wenn ich als Ausländerin schon einen Fehler finde, gibt es vermutlich noch weitere!

2.3 Paararbeit

- Eine in der Programmier-Grundausbildung gut einsetzbare Organisationsform ist die der Paararbeit mit *Korrektur in Anwesenheit*. Zwei Personen fertigen eine Lösung an, sie wird gemeinsam mit der Lehrkraft besprochen. Kann eine Studentin oder ein Student einen Teil der Lösung nicht erklären, wird eine entsprechende Notendifferenzierung vorgenommen. Da aber ca. 15 Minuten pro Gruppe notwendig sind für diese Art der Korrektur, müssen entweder sehr viele Lehrkräfte eingesetzt werden, oder es wird jede Woche nur eine Auswahl der Gruppen zur Korrektur gebeten.

  Ich habe dieses Verfahren schon zwei Semester in dem zweiten Programmiersemester erfolgreich angewandt. Ich versuche, alle Gruppen zwei- oder dreimal im Semester zu prüfen. Da ich die Arbeiten vorher durchsehe, ist mir oft klar, wer von den beiden die Lösung tatsächlich angefertigt hat, und ich bitte die andere Person, es mir zu erklären. So entstehen schon zum Teil große Differenzierungen bei der Notengebung innerhalb der Gruppe. Es werden auch die ersten Erfahrungen mit dem "arbeiten-teilen" und dem "sich-auf-einander-verlassen-können" gemacht. Eine Studentin berichtete: "Wir sind zwar befreundet, und das immer noch nach diesem Semester, aber ich habe eingesehen, daß wir nicht gut zusammenarbeiten können".

- Ein Problem kann die Partnerwahl sein, besonders in einer Lerngemeinschaft, die sich noch nicht kennt. Eine hervorragende Methode zum Einteilen in Gruppen ist, halb so viele gleichlange Stränge wie es Lernende gibt mittig in die Hand zu halten und alle aufzufordern, ein Ende zu greifen. Wenn die Lehrkraft losläßt, gibt es erst mal eine Gemeinschaftsaufgabe – den Knoten zu lösen. Wer miteinander mit einem Strang verbunden ist, bildet eine Partnerschaft. Oder gleich große Bilder werden mit der Zackenschere in der Mitte durchgeschnitten, gemischt, und ausgeteilt, die passenden Gegenstücke müssen sich gegenseitig suchen. Nach einer anfänglicher Scheu vor dem Spielen ist die Gruppe meistens schnell dabei, den Knoten zu lösen. Dabei kann man gut beobachten: wer führt die Gruppe an, gibt Anweisungen, wer läßt alles mit sich geschehen.

- Es kann sein, daß es "Cracks" gibt, die lieber alleine arbeiten wollen oder "Faule", mit denen niemand ein ganzes Semester gepaart sein will. Es gibt auch die Möglichkeit der wechselnden Partnerschaften – jede Aufgabe wird in neuer Besetzung gelöst, es darf keine "Wiederholungen" geben. Das erfordert einiges an Verwaltung, gibt aber viele Erfahrungen im Umgang mit anderen. Dieses Verfahren lässt sich aber eher an einer Schule – eine Lehrerin berichtete von gutem Erfolg hiermit – oder Hochschule mit

Anwesenheitspflicht durchführen. Viele Studierende fühlen sich nicht dazu verpflichtet, alle Aufgaben zu lösen, sondern nur genug, um die Lehrveranstaltung bestehen zu können, und beteiligen sich nicht an allen Aufgaben.

2.4 Test-Rallye

In der 2.-Semester-Programmierung mit Paararbeit haben wir eine Implementierung eines abstrakten Datentyps für Zeichenreihen variabler Länge erstellt und dafür Testtreiber geschrieben. Für die Testrallye musste jede Gruppe erstmal eine Kopie des ADTs von allen anderen Gruppen besorgen (ziemlich einfach in Ada, das ist nur ein Paket, das in einer Datei steht) und irgendwie bei sich auf dem Verzeichnis ordnen[3]. Dann sollen sie jedes Paket mit ihren eigenem Testtreiber testen und Fehler suchen. Jede Gruppe beginnt mit 10 Punkten. Für jeden im eigenen Paket gefundenen Fehler wird ein Punkt abgezogen und auf alle diejenigen Gruppen verteilt, die diesen Fehler gefunden haben (also wenn vier Gruppen den gleichen Fehler finden, bekommt jede Gruppe 1/4 Punkt dazu). Die Gewinnstrategie war (neben einem möglichst fehlerfreien eigenen Paket) als einzige Gruppe Fehler bei den anderen zu finden, und möglichst viele davon.

Ich habe diese Rallye in zwei Übungen durchgeführt mit interessanten Ergebnissen. Eine Gruppe ging einfach Kaffeetrinken – und bekam 6 Punkte, weil 4 Fehler gefunden wurden. Eine Gruppe mit einem ziemlich fehlerhaften Paket hat sich durch gutes Fehlerfinden auf 8 Punkte halten können. Eine Gruppe hatte einen interaktiven Testtreiber mit schicker Menusteuerung konstruiert, der sich bald als sehr mühsam herausstellte, als alle 20 Funktionen mit 81 Zeichen langen Zeichenketten getestet werden sollten. Eine Gruppe hatte viel Arbeit in einen vollautomatischen Testtreiber gesteckt, der Ergebnisse mit Erwartungswerten verglich und Testprotokolle druckte – sie war nach einer halben Stunde fertig und hatte 18 Punkte geschafft! Die Gruppen haben in der nächste Woche bei der Preisverleihung mitbekommen, welche Strategien am erfolgreichsten waren, ich glaube, daß es eine gute Lernerfahrung für alle war.

2.5 Gruppenbildung

Die bisherigen Vorschläge sind eher bei Programmierkursen einsetzbar, wo es wöchentliche Aufgaben zu lösen gibt. Wie ist es aber bei Software-Engineering-Projekten, wo es grössere Gruppen gibt, die über ein Semester oder ein Jahr ein System erstellen sollen?

[3] Das bedeutet etwa fünf Minuten Chaos, da es sich meistens herausstellt, daß viele ihre Pakete gleich genannt haben, und eine neue Namenssystematik für die Dateien muß aufgestellt werden.

2.5.1 Gruppenauswahl

Die erste Frage betrifft die Gruppenauswahl. Wie findet sich eine Gruppe, mehr oder weniger zwangsweise, zusammen? Wenn im vorherigen Semester feste Paararbeit verlangt war, haben sich oft zwei Paargruppen als Projektgruppe zusammengetan. Sonst kann die Lehrkraft kleine bunte Zettel vorbereiten, eine Farbe für jede Gruppe und so viele von jeder Farbe, wie die Anzahl von Gruppenmitgliedern (vier ist die beste Größe). Die Zettel gut mischen und in z.B. zwei Haufen aufstellen, jede(r) Studierende nimmt sich einen. Leute, die unbedingt zusammen sein wollen, werden sich schnell verständigen und ggfs. tauschen, um in der gleichen Gruppe zu sein. Dieses Verfahren eignet sich gut für alle Angelegenheiten, wo eine große Gruppe irgendwie aufzuteilen ist, z.B. auf Übungstermine oder auf Klausurtermine. Es gibt dann erstaunlicherweise selten Streit über die Aufteilung, es ist einsichtig und wird als fair empfunden.

Die Gruppen sollen aufgefordert werden, sich Namen zu geben. Das fördert die Gruppenzugehörigkeit besser als mit “Gruppe 1” oder “Blue Team” angeredet zu werden. Vielleicht lassen sich ganz Kreative ein Firmenlogo einfallen und drucken T-Shirts damit! Eine Gruppe hat sogar ihre Zeiterfassungsformulare, Briefpapier und Sitzungsprotokollbögen erstellt, alles mit ihrem Firmenlogo.

2.5.2 Sich kennenlernen

Um als Team gut zu funktionieren, müssen die Mitglieder sich als Menschen kennenlernen, und sich gegenseitig respektieren. Das ist besonders wichtig, wenn zu einer bestehenden Gruppe neue Mitglieder dazukommen. Es gibt viele Möglichkeiten hierzu – DeMarco und Lister empfehlen in [deMarco/Lister 87], um ein Team zu bilden, sollte die Gruppe gemeinsam ein Spaghettiessen kochen und einen geselligen Abend organisieren.

Ich versuche es mit einer Kennenlernrunde in der ersten Woche – das funktioniert aber nur in kleinen ($\leq$ 20) Übungsgruppen. Die Gruppen sollen sich intern durchnummerieren: 1, 2, 3, 4. Dann soll jedes Teammitglied ein anderes interviewen und der Gruppe kurz vorstellen, 1 stellt 2 vor, $2 \rightarrow 3$, $3 \rightarrow 4$ und $4 \rightarrow 1$. Damit spricht jede(r) erst mal genauer mit 2 Teammitgliedern. Die Vorstellung soll vor der gesamten Übungsgruppe erfolgen und sich auf ein paar Fragen beschränken, z.B. wie sie heißen, wo sie herkommen, warum sie beschlossen haben, Informatik zu studieren. Mit 10 Minuten Diskussionszeit und 20 mal einer Minute Vorstellung sind schon eine halbe Stunde vergangen.

Auch in Übungen ohne Gruppenarbeit lasse ich erst mal die Runde sich vorstellen – die Studierenden sind eher dazu bereit einander zu helfen, wenn sie ein wenig voneinander wissen.

2.5.3 Gemeinsam Erfolg haben

In [Höper et al. 74], einem Buch mit vielen Vorschlägen für Gruppenübungen für Jugendliche und Erwachsene, werden mehrere Übungen im Bereich Kommunikation und Gruppenbildung vorgestellt, die ein ähnliches Muster haben. Gruppen von 3 bis 5 Menschen werden eine Sammlung von Materialien und eine Aufgabe gegeben, die sie in einer Stunde lösen müssen. Beispiele sind ein Haus, ein möglichst hoher Turm oder eine ganze Stadt zu bauen. Die Gruppe muß sich auf eine Vorgehensweise einigen – oder auch nicht –, Aufgaben verteilen und durchführen, und immer ein Auge auf der Uhr haben. Die Lehrkraft beobachtet nur die Gruppen, gibt keine Hilfe. Es ist zu beobachten, wie die Zusammenarbeit klappt, ob die Gruppe sich an die Anweisungen hält, wie sie Widersprüche und Konflikte löst, etc. Bei einigen der Übungen, z.B. den Turmbau, prämiert eine Jury den höchsten, schönsten, stabilsten Turm.

Ich verwende eine eigene Variante dieses Spiels, in der außerdem eine Fähigkeit geübt werden soll, die in der Software-Entwicklung häufig nötig ist – Improvisationsgeschick. Ich verteile an jede Gruppe (am liebsten vier Leute) einen großen Bogen Papier (alte Plakate sind dafür besonders gut), eine Schere, einen Bleistift und ein Lineal. Keine weiteren Hilfsmittel sind zulässig. Die Aufgabe ist es, ein Haus zu konstruieren mit den Vorgaben, daß es mindestens eine Tür, zwei Fenster, drei Dimensionen (also keine Zeichnung!) und ein Dach haben muß. Durch das Fehlen von Tesa oder Kleber muß ein Weg gefunden werden, die Standfestigkeit des Hauses herzustellen. Die Gruppen haben 60 Minuten Zeit.

Nach der ersten Ratlosigkeit entsteht in einigen Gruppen recht bald Streit über den Entwurf – z.B. ob ein Spitzdach oder ein Flachdach besser ist –, der aber meist schnell gelöst werden kann. Wenn dies nicht der Fall ist, sollte eine Änderung der Gruppenzusammensetzung für das Software-Projekt vorgenommen werden! Bei dieser Aufgabe werden die meisten Software-Erstellungs-Phasen durchlaufen: Spezifikationsklärung (sind z.B. runde Häuser zugelassen?), Entwurf, Ausführung und Endkontrolle. Am Schluß der Übung werden die Häuser aller Gruppen kurz vorgestellt. Es ist erstaunlich, welche Vielfalt der Haustypen und Befestigungsmöglichkeiten zu Tage treten!

Die Lehrkraft muß sich ernsthaft bemühen, etwas Tolles an jeder Lösung zu finden, und diesen Erfolg vor der ganzen Gruppe loben. Dies ist auch ein Erfolg für die Gruppe, der ihnen Freude an der Zusammenarbeit geben wird. Es ist aber von äußerster Wichtigkeit, daß das Lob ernst gemeint ist, denn Scheinlob wird sofort wahrgenommen und schlägt in das Gegenteil um, die einzelnen Gruppenmitglieder ärgern sich, es kann so weit gehen, daß sie sich gegenseitig Vorwürfe machen.

Eine Gruppe hatte ein Haus erstellt, das erst im letzten Moment fertig wurde, und sofort unter der Last des Dachs in sich zusammenfiel. Auf einmal legten die Mitglieder der Gruppe los mit Schuldzuweisungen. Ich habe Mühe gehabt, die Diskussion zu unterbrechen und zum Nachdenken über Teamarbeit anzuregen! D.h., nicht nachgeben und hinterher sagen: "habe ich doch gesagt!", oder zu begreifen, daß es keine Einzelschuldfrage gibt, sondern nur Erfolg oder Mißerfolg der Gruppe. Die Gruppe ist sehr nachdenklich auseinander gegangen.

2.5.4 Verantwortlichkeiten

Tomayko vom Software Engineering Institute an der Carnegie-Mellon Universität schlägt in [Tomayko 87] eine Aufteilung von Verantwortlichkeiten innerhalb der Gruppe vor. Es werden verschiedene Rollen definiert wie

- Hauptarchitekt(in),
- Projektverwalter(in),
- Konfigurationsverwalter(in),
- Qualitätssicherungsingenieur(in),
- Entwurfs-Verantwortliche(r),
- Implementierungs-Verantwortliche(r),
- Dokumentations-Spezialist(in), oder
- Verifikations- und Validierungs-Ingenieur(in).

So hat die Implementierungs-Verantwortliche z.B. die Aufsicht über die Einhaltung der Programmierrichtlinien, die Projektverwalterin erstellt wöchentliche Kostenberichte (nur Stundenaufwand).

Wer eine Rolle innehat, muß nicht die Arbeit der Rolle alleine machen, sondern dafür verantwortlich sein, daß die Aufgaben korrekt und pünktlich abgeliefert werden. Wer dieser Verantwortung nicht nachkommt, wird mit einer schlechteren Note als der Gruppendurchschnitt "bestraft". Da jedes Gruppenmitglied mindestens ein Amt verwaltet, gibt es ein gegenseitiges Interesse an der Zusammenarbeit. Ich habe es häufig erlebt, daß die "Mitläufer" plotzlich vom Arbeitseifer gepackt werden, als der Abgabetermin, z.B. für die Dokumentation naht – es war eben kein weiteres Mitglied dazu bereit, mitzuschreiben, weil die Mitläufer sich bisher an nichts beteiligt hatten. Für die Lehrkraft gibt es einen weiteren Vorteil: klare Zuordnung der Verantwortlichkeiten zu Personen, da wir in einem Vertrag die Aufteilung festhalten.

Zwei Nachteile dieser Methode haben sich ergeben. Zum einen kennt sich die Gruppe noch nicht sehr gut am Anfang, um die Ämter optimal zu verteilen. Es sollte also ein Verfahren geben, um Verantwortlichkeiten auszutauschen. Zum anderen ist es problematisch, wenn jemand mitten im Semester aus dem Projekt aussteigt. Da müssen die Verantwortlichkeiten umverteilt werden, das kann die nun kleiner gewordene Gruppe sehr belasten. Aber im großen und ganzen hilft die Rollenverteilung dem Zusammenhalt des Teams.

3 Zusammenfassung

Es ist außerordentlich wichtig, daß neben den rein inhaltlichen Bestandteilen von Übungen in Programmierung und Software-Engineering die Bildung eines Teams praktisch durchgeführt wird. Dazu können Abwandlungen von in der Sozialpädagogik bekannten Methoden der Gruppenbildungen erfolgreich angewandt werden. Das fordert von den Lehrkräften, von der Vermittlung von reinem Faktenwissen abzurücken und sich mit Teambildung und Menschenführung zu beschäftigen.

Literatur

[deMarco/Lister 87]
Tom de Marco and Timothy Lister. *Peopleware - Productive Projects and Teams.* Dorset House, 1987.

[Höper et al. 74]
Claus-Jürger Höper, Ulrike Kurzleb, Alke Stobbe und Bertram Weber. *Die spielende Gruppe. 115 Vorschläge für soziales Lernen in Gruppen.* Jugenddienst-Verlag, Wuppertal, 1974.

[Tomayko 87]
J. Tomayko. Teaching a project-intensive introduction to software engineering. Technical Report CMU/SEI-87-TR-20, AD-A200603, SEI, Carnegie-Mellon University, 1987.

Prof. Debora Weber-Wulff
Technische Fachhochschule Berlin
FB Informatik
Luxemburger Str. 10
13353 Berlin
Tel: (030) 4504-2529
email: *weberwu@tfh-berlin.de*