Transparente Bewertung von Softwaretechnik-Projekten in der Hochschullehre

Oliver Hummel, Karlsruher Institut für Technologie (KIT)

hummel@kit.edu

Zusammenfassung

Die Informatik-Curricula an Hochschulen und Universitäten sehen neben Softwaretechnik-Vorlesungen häufig auch praktische Projekte vor, in denen Studierende die Herausforderungen bei der Erstellung eines nicht-trivialen Softwaresystems in einem Team erfahren sollen. Spätestens seit der Bologna-Reform sind auch solche praktischen Studienleistungen zu benoten, was eine nicht unerhebliche Herausforderung für die Betreuer darstellt, da hochkomplexe und entsprechend unterschiedliche Arbeitsergebnisse transparent und nachvollziehbar bewertet werden müssen. Im vorliegenden Beitrag wird aus Kolloquien, einem Programmiertestat und der direkten Bewertung eingereichter Artefakte ein transparentes und verhältnismäßig einfach anwendbares System zur Bewertung von Softwareentwicklungsprojekten an Hochschulen und Universitäten entwickelt und von seiner praktischen Erprobung an der Universität Mannheim berichtet.

Einleitung

Teambasierte Softwareentwicklungsprojekte sind seit Jahren ein fester Bestandteil des Curriculums von Informatikstudiengängen an vielen Hochschulen. Der vorliegende Beitrag zielt primär auf praktische Entwicklungsprojekte im Fach Softwaretechnik (bzw. engl. Software Engineering) an Hochschulen und Universitäten, kann aber, mit entsprechenden Abwandlungen, auch für andere Fachgebiete der Informatik von Nutzen sein. Aus technischer Sicht geht es in entsprechenden Lehrveranstaltungen hauptsächlich um das Erlernen von strukturierten Entwurfs- und Modellierungstechniken für Software sowie deren Umsetzung in lauffähige und getestete Programme. Dazu notwendig ist das Erlernen grundlegender Techniken, Methoden und Werkzeuge zur Entwicklung komplexer Softwaresystemen (vgl. ACM, 2004 oder Ludewig, 1999), so dass Studierende wichtige Kompetenzen zur konstruktiven Mitarbeit in entsprechenden Industrieprojekten erwerben können. Anders ausgedrückt sollen Absolventen möglichst direkt in der Industrie „beschäftigungsfähig" sein (vgl. KMK, 2010 oder auch Coldewey, 2009). Primäres Lernziel der in diesem Artikel diskutierten Veranstaltungen an der Universität Mannheim war die Vermittlung eines strukturierten und artefaktgetriebenen („ingenieurmäßigen") Vorgehens in der Softwareentwicklung und die Einübung dazu erforderlicher Modellierungs- und Implementierungstechniken (UML, Continuous Integration u.ä., vgl. Larman, 2004).

Bekanntlich existiert für die Entwicklung von Softwaresystemen in verschiedensten Domänen kein allgemein gültiges Patentrezept („No Silver Bullet", vgl. Brooks, 1995), so dass erlernte Verfahren von den Ausführenden immer mit Augenmaß an die jeweiligen Umstände angepasst werden müssen (sog. „Tailoring" vgl. z.B. Rausch et al., 2008). Folglich gibt es auch in praktischen Projekten an Universitäten oder Hochschulen nicht eine korrekte Lösung, sondern vielmehr eine Reihe von möglichen und sinnvollen Lösungsstrategien. Darüber hinaus zielen entsprechende Teamprojekte darauf ab, auch sogenannte Softskills, wie Team-, Präsentations- und Kommunikationsfähigkeit der Teilnehmer zu schulen (Böttcher & Thurner, 2011), weshalb sie zumeist in Kleingruppen von etwa drei bis sechs Personen durchgeführt werden. Gleichzeitig sehen einschlägige Prüfungsordnungen aber regelmäßig eine individuelle Benotung aller Teilnehmer vor. Weiter erschwerend kommt bei Softwaretechnik-Projekten hinzu, dass es praktisch unmöglich ist, die Studierenden beständig unter Aufsicht arbeiten zu lassen, wodurch ihre individuellen Beiträge zur Gruppenarbeit nicht ohne weiteres einzuordnen und ggf. auch Täuschungsversuche nur schwer zu entdecken sind (vgl. z.B. Kehrer et al., 2005). Eine objektive Bewertung der studentischen Leistungen ist unter diesen Gegebenheiten offensichtlich nicht trivial (vgl. z.B. Hayes et al., 2007), aber nichtsdestotrotz kritisch, da sie maßgeblich über das Fortkommen der Studierenden entscheidet und letztlich auch ihre Moti-

vation stark beeinflusst. Dabei schränken zahlreiche äußere Zwänge, wie Zeitvorgaben oder Budgetrestriktionen die Möglichkeiten zur Bewertung ein, so dass es nicht verwunderlich ist, wenn in der Lehrpraxis für große Kohorten häufig auf „Ersatzprüfungsverfahren" wie Kolloquien, Lerntagebücher oder gar Klausuren zurückgegriffen wird (vgl. Kehrer et al., 2005), um praktische Studienleistungen zu bewerten. Je nach Ausgestaltung dieser Bewertungsarten ergibt dies einen nicht unerheblichen Mehraufwand für alle Beteiligten, der auf Seiten der Studierenden zudem leicht zu einem Motivationsverlust bei der Entwicklungsarbeit führen kann, da diese überhaupt nicht oder nur in sehr begrenztem Umfang in die Benotung einfließt. Soweit letzteres überhaupt der Fall ist, existiert in der Literatur (und nach Kenntnisstand des Autors auch in der Praxis) bisher selten ein differenziertes und transparentes Bewertungssystem, wie es in diesem Beitrag vorgestellt wird.

Eigene Erfahrungen des Autors, der im Verlauf seines Studiums zwei Mal selbst an einem entsprechenden Projekt teilgenommen und einmal als studentische Hilfskraft an der Betreuung mitgewirkt hat, waren ähnlicher Art: In den ersten beiden Fällen war das Zustandekommen der Note für die Teilnehmer vollkommen intransparent (sie setzte sich „irgendwie" aus der Leistung im Projekt und dem Abschlusskolloquium zusammen); im letztgenannten Fall haben gar die studentischen Hilfskräfte eigene Maßstäbe zur Bewertung von Anforderungs- und Designdokumenten, die sich hauptsächlich auf die Zählung von offensichtlichen Fehlern beschränkten, entwickelt und eingesetzt.

Ziel dieses Beitrags ist es daher, ein nachvollziehbares und dennoch einfach handhabbares Bewertungsverfahren für praktische Softwareentwicklungsprojekte, das eine individuelle und transparente Bewertung der Studierenden ermöglicht, vorzustellen und von Erfahrungen aus seiner praktischen Erprobung an der Universität Mannheim zu berichten. Dazu werden im folgenden Abschnitt zunächst wichtige didaktische und organisatorische Rahmenbedingungen vorgestellt, bevor in aller gebotenen Kürze einige benötigte Hintergründe der Softwaretechnik dargelegt werden. Danach werden gängige Bewertungsverfahren auf ihre Verwendbarkeit in einem Softwaretechnik-Projekt analysiert. In den darauf folgenden Kapiteln werden das in diesem Beitrag thematisierte Bewertungsverfahren im Detail vorgestellt und aus ersten Erprobungen gewonnene Erkenntnisse diskutiert. Ein Vergleich mit anderen in der Literatur veröffentlichten Verfahren und eine Zusammenfassung runden diesen Beitrag schließlich ab.

Grundlagen

Um die wichtigsten Herausforderungen eines Softwaretechnik-Projekts aus verschiedenen Blickwinkeln zu beleuchten, sollen im Folgenden weiterführende Einblicke gegeben werden, die nochmals die Notwendigkeit eines an der Aufgabenstellung orientierten Bewertungssystems unterstreichen sowie wichtige Hintergründe näher erläutern. Anschließend werden zentrale Anforderungen an ein solches Bewertungssystem zusammengetragen.

Didaktische Sicht

Aus didaktischer Sicht springen zunächst die hohen Anforderungen, die sich an die Studierenden in einem Softwaretechnik-Projekt auf Grund der zuvor formulierten Lern- und Kompetenzziele ergeben, ins Auge. Beispielsweise Böttcher et al. (2011) halten eine detailliertere Aufschlüsselung dazu notwendiger Kompetenzanforderungen aus den vier Bereichen Sach-, Methoden-, Selbst- und Sozialkompetenzen bereit. Grundsätzlich bleibt festzuhalten, dass die hohe Komplexität von Softwareprojekten von den Teilnehmern ohne Zweifel ein tiefgehendes Verständnis zahlreicher Abläufe und Techniken verlangt und Prüfer damit regelmäßig vor die Herausforderung stellt, etwas zu prüfen, was sich nicht direkt lehren lässt (vgl. Ludewig, 2011), sondern nur durch praktische Anwendung erfahren werden kann.

In der Softwaretechnik wurde (und wird) für eine solche Lehrform traditionell häufig der Begriff Praktikum verwendet, der damit allerdings nicht korrekt eingesetzt ist. Ein Praktikum an einer Hochschule oder Universität ist zwar ebenfalls eine praktische Unterrichtsform, in dieser setzen sich Studierende allerdings mit vorgegebenen und klar umrissenen Aufgaben, die innerhalb weniger Stunden zu bearbeiten sind, auseinander (vgl. Iller & Wick, 2009). Da diese Lehrform hauptsächlich in klassischen Naturwissenschaften (wie Chemie oder Biologie) Verwendung findet, bedeutet das zumeist, dass Aufgaben nur vor Ort in einem entsprechenden Labor bearbeitet werden können. Als „Laborumgebung" in der Informatik und speziell in der Softwaretechnik ist heute allerdings in der Regel ein handelsüblicher Laptop ausreichend, mit dem bequem an beinahe jedem beliebigen Ort gearbeitet werden kann. Auch bei den Aufgabenstellungen in der Softwaretechnik, die zumeist das selbstständige Durchlaufen des kompletten Entwicklungszyklus einer Software vorsehen, passt obige Begriffsdefinition nicht, vielmehr ist hier aus Sicht der Didaktik von einem praktischen Projekt mit entsprechend höheren Anforderungen an die Eigenverantwortung der Teilnehmer auszugehen, weshalb im Folgenden dieser Begriff Verwendung finden soll.

Studentische Perspektive

Aus studentischer Sicht erfahren Softwareentwicklungsprojekte häufig eine komplett konträre Einschätzung. Zum einen gibt es die leistungsstarken „Vollblut-Informatiker", die gerne, gut und viel programmieren und oft privat bereits eigene kleine Softwareprojekte vorangetrieben oder gar als Praktikant Industrieerfahrung gesammelt haben. Für diese ist ein solches Projekt nicht selten einer der Höhepunkte ihres Studiums, den sie mit großer Motivation und Begeisterung angehen und oft mit beeindruckenden Leistungen abschließen, gerade wenn die Aufgabenstellung eventuell in Kooperation mit einem Industriepartner erfolgt ist. Zum anderen gibt es aber – gerade in interdisziplinären Studiengängen wie der Wirtschaftsinformatik – auch viele Studierende, die Softwareentwicklung und insbesondere Programmieren nur als notwendiges Übel ansehen und daher entsprechende Projekte nur als eine mit einem hohem Zeitaufwand verbundene Pflichtübung betrachten.

Dies führt in der Lehrpraxis häufig dazu, dass studentische Teams sehr unterschiedliche Leistungsniveaus aufweisen und schwächere Studierende sich sehr leicht „abgehängt" fühlen, während die stärkeren diese vor allem als Belastung wahrnehmen (vgl. auch die Erfahrungen von Lindig & Zeller, 2005). Somit überrascht es nicht, wenn in solchen Veranstaltungen häufig auf das Phänomen von „Trittbrettfahrern" zu treffen ist, die sich ohne nennenswerte eigene Beiträge von ihren Kommilitonen durch die Veranstaltung schleppen lassen wollen (vgl. Stoyan & Glinz, 2005, van der Duim et al., 2007 oder Stangl, 2002). Die Leistungsträger erdulden das aus falsch verstandener Solidarität heraus häufig zumindest so lange, wie es die eigene Arbeitskraft nicht überlastet, sobald aber ihre Belastungsgrenze erreicht wird, sind entsprechende Konflikte, bis hin zu auseinanderbrechenden Teams, vorprogrammiert.

Organisatorische Sicht

Die im Folgenden beschriebenen Softwaretechnik-Lehrveranstaltungen an der Universität Mannheim waren nach einem verbreiteten Muster organisiert: ein Professor trug als Prüfer die Hauptverantwortung für die Veranstaltung und vermittelte in Vorlesungen die theoretischen Hintergründe. Wissenschaftliche Mitarbeiter übernahmen die Aufgabe den Studierenden den Lehrstoff an Hand von einzelnen Übungsaufgaben mit direktem Projektbezug auch praktisch näher zu bringen. Die wissenschaftlichen Mitarbeiter waren ferner für die Steuerung des eigentlichen Projekts ebenso zuständig wie für die Klärung von offenen Fragen (z.B. bzgl. der Anforderungen). Die direkte Betreuung der Entwicklungsteams oblag auf Grund der recht hohen Teilnehmerzahlen von anfangs teilweise deutlich über fünfzig Studierenden einer Reihe von studentischen Tutoren. Diese hatten in der Regel bei mindestens wöchentlichen Treffen der Teams direkten Kontakt mit den Studierenden und trafen sich ebenso regelmäßig mit den Mitarbeitern, um über die Fortschritte ihrer Teams zu berichten. Von anderen Universitäten sind ähnliche Konstellationen mit teilweise noch weit höheren Teilnehmerzahlen bekannt (z.B. Kehrer et al., 2005). An Hochschulen sind Kohorten hingegen oft um einiges kleiner, so dass dort auch Projekte möglich sind, in denen Professoren die Studierenden persönlich betreuen und sich einen direkten Eindruck über ihre Leistungen verschaffen können.

Eine weitere Schwierigkeit bei der Koordination und auch Bewertung von Projekten stellt die meist geringe Erfahrung der beteiligten Mitarbeiter und Tutoren dar. Bedingt durch die naturgemäß sehr hohe Fluktuation innerhalb dieser Personengruppen kommt es häufig vor, dass Mitarbeiter oder Tutoren, die eine entsprechende Veranstaltung noch vor wenigen Monaten selbst besucht haben, sich in einer betreuenden Rolle wieder finden, das Wissen aus vorangegangenen Durchläufen aber nicht schriftlich dokumentiert worden ist. Sobald neue Tutoren bzw. Mitarbeiter entsprechende Erfahrungen angesammelt haben, schließen sie ihr Studium oder ihre Promotion ab und wenden sich neuen Aufgaben zu. Dieser Erfahrungsverlust gilt natürlich auch für den Bereich der Bewertung studentischer Abgaben, so dass ein klar definiertes Bewertungsschema fraglos allen Lehrenden die Betreuung erleichtern und dadurch zu einer deutlichen Erhöhung der Lehrqualität führen kann.

Anforderungen an die Benotung eines Softwareprojekts

An dieser Stelle soll zunächst kurz auf die allgemeinen Güteanforderungen an Prüfungen eingegangen werden. Die Literatur (beispielsweise Roloff, 2002 oder Müller & Bayer, 2007) hält dazu die folgenden Testgütekriterien bereit: eine Prüfung bzw. auch eine einzelne Prüfungsleistung sei *objektiv*, also unabhängig, sowohl von der Person des Prüfers, als auch von der des Prüflings bewertbar. Insbesondere gilt dabei, dass beispielsweise Aussehen, oder Herkunft eines Prüflings die Note nicht beeinflussen dürfen. Ferner soll ein Prüfungsverfahren *zuverlässig* die tatsächliche Leistungsfähigkeit eines Prüflings erfassen und somit bei vergleichbaren Prüfungsleistungen eine vergleichbare Benotung ergeben. Eine *valide* Prüfungsform misst genau die Leistung, die sie vorgibt zu messen und genau die, die auch gemessen werden soll. *Nützlich* ist eine Prüfung, sobald sie eine sinnvolle Funktion erfüllt, im Allgemeinen werden in diesem Zusam-

menhang beispielsweise eine Rückmeldefunktion für Dozenten bzw. Studierende oder auch Anreiz- bzw. Selektierungsfunktionen genannt. Eine Prüfung gilt als *ökonomisch*, wenn Nutzen und Aufwand in einem vernünftigen Verhältnis zueinander stehen.

Aus den oben diskutierten Hintergründen und Erfahrungen sowie den eben genannten Testgütekriterien ergeben sich folgende grundlegende Anforderungen an die Benotung einer praktischen Softwaretechnik-Veranstaltung, die weitestgehend auf ähnliche Veranstaltungen in anderen Teilgebieten der praktischen Informatik übertragbar sind:

1. Die Benotung muss sowohl für Lehrende als auch für Studierende nachvollziehbar sein und sollte unabhängig von der Person des Bewerters bzw. des Bewerteten zum gleichen Ergebnis kommen. Auf Grund des relativ großen Aufwands, den Studierende in entsprechende Veranstaltungen investieren, werden idealerweise auch die erstellten Artefakte (ggf. in mehreren Teilschritten) benotet; generell erfasst das Bewertungsschema die in den Lernzielen formulierten Fertigkeiten.
2. Gemäß der Bologna-Kriterien (KMK, 2010) muss die Benotung für jeden Studierenden individuell erfolgen, es kann daher bei Teamarbeiten nicht automatisch eine einheitliche Note für alle Teammitglieder vergeben werden. Ein geeignetes Bewertungsverfahren muss die Zuordnung individueller Leistungen erlauben, dabei aber gleichzeitig Teamarbeit zulassen und fördern bzw. im Idealfall sogar in der Note abbilden. Dabei sollte auf Grund entsprechender Erfahrungen die Möglichkeit bestehen, „Trittbrettfahrer" frühzeitig zu erkennen, um die übrigen Mitglieder eines Teams vor einer Überlastung und deren negativen Folgen für die eigenen Noten zu schützen.

Softwaretechnik-Hintergründe

Zum besseren Verständnis des später erklärten Bewertungsschemas sollen im Folgenden in aller Kürze einige Aspekte der Softwaretechnik und daraus abgeleitete Lernziele hervorgehoben werden. Ein zentraler Punkt ist dabei die Definition einer Software selbst. Laut Brooks (1995) umfasst „Software" nicht nur den ablauffähigen Programmcode, den ein Endbenutzer einsetzt, sondern auch die zu seiner Erstellung notwendigen Anforderungen, daraus abgeleitete Analyse- und Entwurfsmodelle sowie Testfälle und weitere Dokumentation (wie z.B. auch das Benutzerhandbuch). Es ist daher bereits in mittelgroßen Projekten notwendig, eine ganze Reihe von Mitarbeitern mit verschiedenen Rollen miteinander zu koordinieren, um ein System mit möglichst wenigen Reibungsverlusten entwickeln zu können.

Dazu stehen in der Softwaretechnik verschiedene Vorgehensmodelle zur Verfügung (vgl. Bunse & v. Knethen, 2008), von denen das Wasserfallmodell (s. z.B. Sommerville, 2010 oder Larman & Basili, 2003) nach wie vor das bekannteste ist. Modernere, modellbasierte Ansätze, wie das aus dem Unified Process entstandene Agile Modelling (vgl. Larman, 2004), arbeiten zumeist iterativ und schlagen die Erstellung verschiedener, voneinander abgeleiteter (UML-)Modelle für die Systementwicklung vor. Das Erlernen einer solchen modellbasierten und möglichst systematischen Vorgehensweise war das primäre Lernziel der vom Autor an der Universität Mannheim angebotenen Softwaretechnik-Projekte. Der Bewertung der von den Studierenden erstellten Artefakten kommt daher im Folgenden eine zentrale Bedeutung zu.

Mögliche Bewertungsverfahren

Dieses Kapitel geht auf kurz gängige Bewertungsverfahren an Schulen bzw. Hochschulen ein und diskutiert, ebenfalls in aller Kürze, mögliche positive und negative Aspekte bezüglich der Anwendbarkeit innerhalb von praktischen Softwaretechnik-Lehrveranstaltungen. Dazu fasst die folgende Tabelle neben den Bewertungsverfahren auch die jeweils wichtigsten positiven und negativen Aspekte zusammen:

Bewertungsverfahren	**Kurzbeschreibung**	**Stärken**	**Gefahren**
Prakt. Leistungsnachweise (Übungsaufgaben)	Die Studierenden bearbeiten vorgegebene Übungsaufgaben	Kleinere Aufgaben zu einer spez. Fragestellung mit meist überschaubarem Arbeitsaufwand	Meist kein unmittelbarer Bezug zum Projekt; erhöhte Gefahr des Abschreibens gleicher Aufgaben
Vorträge	Die Studierenden präsentieren ihr Vorgehen oder	Stärkung der Präsentationskompetenz; Vertiefung	Evtl. überdeckt der Vortragsstil inhaltliche Stärken

	ihre Ergebnisse	des Fachwissens im entspr. Themenbereich; ggf. Selbstreflektion durch Vergleich mit anderen Studierenden	oder Schwächen; hoher Zeitaufwand; mögl. Spezialisierung auf das Themengebiet
Kolloquien	Die Studierenden werden von den Betreuern zu ihrem System befragt (oft in Verbindung mit Vorträgen durchgef.)	Nachhaken und Klären von Verständnisfragen möglich	Bei großen Gruppen entsprechend hoher Zeitaufwand
Bewertung des Systems	Das abgegebene System und seine Artefakte bzw. auch Zwischenabgaben (Meilensteine) werden bewertet	Direkter Bezug zum Lernziel und damit direktes Feedback zum Lernerfolg	Kriterienkatalog notwendig, hoher Zeitaufwand in der Bewertung, Zuordnung der Leistung nicht immer einfach
Mündliche Prüfung(en) (Testate)	Prüfungsgespräch mit einem oder mehreren Studierenden	Planbarer Zeitaufwand für den Prüfer, etablierte Prüfungsform, individuelle Benotung und Nachfragen möglich	Bewertung spiegelt nicht unbedingt den Beitrag zum Projekt wieder, hoher Zeitaufwand, verwendete Fragen verbreiten sich schnell
Schriftliche Prüfung	Die klassische Klausur	Bekanntes Verfahren, Leistung ist klar zuzuordnen und gut vergleichbar	Erfordert mitunter einen hohen Vorbereitungsaufwand; kein direkter Bezug zur Leistung im Projekt
Programmierprüfung	„Programmierklausur“, idealerweise mit einer Entwicklungsumgebung direkt am Rechner ausgeführt	Gute Überprüfung der Lernziele in Bezug auf Programmierkenntnisse, am Rechner über Testfälle automatisch und damit objektiv bewertbar	Ausreichend große Rechnerräume müssen verfügbar sein; bei autom. Bewertung kein Bewertungsspielraum, minimale Programmierfehler können große Auswirkungen haben
Laborbuch	Studierende dokumentieren Aufgabenverteilung und Vorgehen regelmäßig	Erleichtert Nachvollziehbarkeit des Vorgehens	Overhead für Studierende und Betreuer; Benotung schwierig; Gefahr unreflektierter Aufzählungen groß
Reflektion	Studierende reflektieren über ihre Erfahrungen	Bei korrekter Ausführung relativ hoher Lerneffekt	Weiterer Overhead für die Studierenden bei ohnehin schon starker Belastung; dedizierte Benotung ebenfalls schwierig
Mitarbeitsnoten	Mitarbeit der Studierenden in ihrer Gruppe wird bewertet	Motiviert zu regelmäßiger Mitarbeit, geringer Aufwand	Generell nur schwer objektiv einschätzbar, insbesondere bei Heimarbeit
Benotung der Studierenden untereinander (Peer Assessment)	Studierende geben sich gegenseitig Noten oder verteilen eine begrenzte Anzahl von Punkten an ihre Teamkollegen	Geringer Aufwand für den Betreuer; Studierende reflektieren und lernen durch Bewertung ihrer Kommilitonen	Validität & Legalität sehr fraglich, mögl. negative Auswirkungen auf die Gruppendynamik

Tabelle 1: Mögliche Bewertungsverfahren für ein Softwaretechnik-Projekt.

Alle aufgeführten Bewertungsansätze sind im Kontext eines Softwareprojekts sicherlich sinnvoll anwendbar. Wie beispielsweise von Hayes et al. (2007) vorgeschlagen, erscheint in der Praxis aber

eine Kombination verschiedener Verfahren erforderlich, um alle zuvor genannten Anforderungen an die Benotung abdecken zu können und dabei möglichst wenig Mehraufwand für Studierende und Betreuer zu generieren. So eignet sich beispielsweise die Bewertung der abgegebenen Systemartefakte allein sehr gut als Feedback über die im Projekt erbrachte Leistung, stößt aber an Grenzen, sobald diese in Heimarbeit erstellt werden können und sollte daher durch Präsenzverfahren (wie Kolloquien o.ä.) ergänzt werden. Bewertungsansätze mit hoher Eigenverantwortung (wie Reflektionen oder Lerntagebücher) erzielen bei entsprechend motivierten Studierenden meist gute Ergebnisse, sind jedoch in einem ohnehin stressigen Projekt offensichtlich mit zusätzlichem Mehraufwand verbunden und ferner in Zweifelsfällen nur schwer als nicht ausreichend zu benoten. Sie erfordern daher, wie die meisten anderen Verfahren, ein klares und idealerweise im Voraus kommuniziertes Bewertungsschema, was den Aufwand für die Betreuenden weiter erhöht, insbesondere, wenn die Zusammenstellung der Inhalte individuell gestaltbar ist.

Bewertungsschema

Aus den zuvor angestellten Überlegungen lässt sich das folgende Bewertungsschema ableiten, das sich im Wesentlichen aus fünf einzelnen Bewertungsverfahren zusammensetzt:

1. Einfache Übungsaufgaben mit direktem Projektbezug
2. Ein praktisches Programmiertestat (gegen vorgegebene Testfälle am Rechner)
3. Führen von Stundenzetteln („Timesheets“)
4. Bewertung der abgegebenen Modellierungsartefakte, Quellcodes und Binaries
5. Mehrere Kolloquien

Der zeitliche Ablauf einer entsprechenden organisierten Veranstaltung gestaltet sich folgendermaßen: zeitnah zu Semesterbeginn werden den Studierenden, grobe Anforderungen an das zu erstellende System mitgeteilt und sie erhalten erste Aufgaben bzgl. der Erfassung und Modellierung der Anforderungen mit direktem Projektbezug. Diese sind einzeln oder in Zweiergruppen zu bearbeiten und abzugeben. Somit ist gewährleistet, dass jeder Teilnehmer sich individuell einen grundlegenden Überblick über die Anforderungen an das zu erstellende System erarbeiten muss. Idealerweise werden die Lösungen der Studierenden kontrolliert oder zumindest in einem Tutorium besprochen, aber nicht benotet. Gleichzeitig werden erste (einfache) Programmieraufgaben gestellt, die auch dazu dienen, die Studierenden mit der verwendeten Entwicklungsumgebung (also z.B. Eclipse, s. z.B. Burnette & Staudemeyer, 2009) vertraut zu machen, falls dies nicht bereits in früheren Veranstaltungen geschehen ist.

Programmiertestat & Kolloquien

Nach etwa drei bis vier Wochen wird ein (ebenfalls einfaches) Programmiertestat mit großzügig bemessener Bearbeitungszeit als erster harter Leistungsnachweis zum Einstieg in das Projekt verlangt. Es bietet sich an, dieses als Programmierübung direkt am Rechner (mit Eclipse und den Javadocs, aber ohne Internet-Zugang) durchzuführen und den Studierenden Interfaces und JUnit-Tests vorzugeben, gegen die sie entwickeln müssen (vgl. testgetriebene Entwicklung nach Beck, 1999). Somit wird Beschwerden, dass Programmieren in Papierklausuren weitaus schwieriger als in einer Entwicklungsumgebung (mit „Code-Completion“) sei, von vorne herein der Wind aus den Segeln genommen. Gleichzeitig entfällt das äußerst zeitaufwändige Bewerten von zumeist unübersichtlichem Programmcode für die Betreuer, da nur das automatisierbare Ausführen der JUnit-Tests notwendig wird. Idealerweise werden mehrere in etwa gleichschwere Teilaufgaben gestellt, so dass die Programme der Studierenden nicht alle Testfälle erfüllen müssen, sondern beispielsweise nur drei aus fünf, o.ä. Auf Grund von in Praxis und Literatur bekannten extremen Leistungsunterschieden bei Softwareentwicklern (vgl. Endres & Rombach, 2003), sollte den Studierenden ausreichend Zeit zum Lösen dieser Aufgaben gegeben werden. In der Mannheimer Praxis hat sich etwa die zehnfache (!) Zeit, die wissenschaftliche Mitarbeiter zum „Proberechnen“ benötigen, bewährt (z.B. ca. 18 zu 180 Minuten).

Die Ergebnisse dieses Testats können im Anschluss zur Einteilung der eigentlichen Projektgruppen genutzt werden. Es bietet sich an, die Studierenden gemäß ihrer Leistungen (also z.B. der Zeit, die bis zur Erfüllung der Testkriterien benötigt wurde) zusammen zu gruppieren, um zu große Leistungsgefälle innerhalb der Teams und eine daraus resultierende Demotivation aller Teilnehmer zu vermeiden (vgl. z.B. Stangl, 2002). Gruppen mit einer großen Leistungsheterogenität haben sich in der Erfahrung des Autors nicht bewährt, da sie das Auftreten von „Trittbrettfahrern“ zu fördern scheint. Wer zu diesem Zeitpunkt nicht über minimale Programmierkenntnisse verfügt (also das Testat nicht bestanden hat), muss vom weiteren Verlauf des Projekts ausgeschlossen werden, um die übrigen Studierenden vor einer Überlastung zu schützen. Dies bringt natürlich eine gleichzeitige Reduktion der verlangten Features für den Rest der Gruppe mit sich. Eine Wiederholungsmöglichkeit für das Testat würde den weiteren Ablauf des Projekts

offensichtlich stark beeinträchtigen und ist daher nicht vorgesehen, zumal fehlende Programmierkenntnisse ohnehin nicht binnen weniger Tage aufzuholen sind. Auf Grund der harten Konsequenzen dieses Kriteriums, liegt es aber nahe, das Testat bereits zu oder gar vor Beginn des Semesters als Freiversuch anzubieten, um den Studierenden ein frühes Feedback zu geben und ihnen ggf. eine mehrwöchige Auffrischung ihrer Vorkenntnisse zu ermöglichen.

Nach erfolgter Einteilung der Projektteams sind diese gehalten, sich vertieft in die Projektanforderungen einzuarbeiten und diese entsprechend zu analysieren und zu dokumentieren (durch Use Cases, System-Sequenzdiagramme oder Operation Contracts). Um die abschließende Benotung zu erleichtern, sind alle abzugebenden Artefakte mit dem Namen des Bearbeiters zu versehen und unter dem eigenen Benutzernamen in das verwendete Versionskontrollsystem hochzuladen. Ferner sollten alle Aktivitäten wie das Erstellen von Dokumenten, Modellen oder Quellcode, aber auch Team-Meetings o.ä. in sogenannten Timesheets gelistet werden, ähnlich, wie es auch zu Abrechnungszwecken in Industrieprojekten praktiziert wird.

Nach etwa weiteren zwei Wochen wird ein erstes Kolloquium durchgeführt, um sicherzustellen, dass jedes Teammitglied zum Projektziel beiträgt und alle Anforderungen vollständig verstanden wurden. Um den Studierenden etwaige Berührungsängste zu nehmen, bietet es sich an, dieses Kolloquium nicht zu benoten, sondern nur eine rudimentäre Mindestleistung zum Bestehen zu verlangen, die ggf. in einem Folgegespräch nochmals nachgefordert werden kann. Der Autor hat gute Erfahrungen damit gemacht, bis zum Ende des Projekts noch zwei weitere, benotete Kolloquien durchzuführen, eines nach etwa zwei Dritteln der Dauer, eines in Form einer Systempräsentation bzw. -abnahme ganz am Ende des Projekts.

Systembewertung

Die Benotung des Softwaresystems und der mitgelieferten Artefakte erfolgt auf Basis des im Folgenden vorgestellten Bewertungsrasters. Dabei sind **fett** gedruckte Elemente zwingend zum Bestehen erforderlich (d.h. eine nicht ausreichende Leistung in einem entsprechenden Bereich führt direkt zum Nicht-Bestehen des gesamten Projekts). Ohne Auszeichnung gedruckte Kategorien werden vom Team verantwortet und für das gesamte Team einheitlich benotet, für *kursiv* gedruckte ist jeder Teilnehmer individuell verantwortlich und bekommt eine eigene Bewertung. Eine entsprechende Kennzeichnung der Dokumente ist dafür wie bereits beschrieben obligatorisch.

Im Sinne der agilen Softwareentwicklung (Pichler, 2007 oder Beck, 1999) sollen alle Teilnehmer indviduell Verantwortung für vollständige Features (also meist komplette Use Cases, vgl. Larman, 2004) übernehmen, so dass die Anforderungen vergleichbar bleiben und eine technologieorientierte Aufteilung (z.B. ein Teammitglied programmiert, eines entwickelt die UI, eines testet und eines erstellt nur Diagramme oder Anforderungen) vermieden wird. Details zu den im Folgenden verwendeten Modellen finden sich bei Bedarf bei Larman (2004). Je nach genauer Ausrichtung der Veranstaltung ist natürlich auch eine geänderte Zusammenstellung mit anderen Artefakten (wie z.B. Projektplänen o.ä.) oder veränderten Gewichtungen denkbar:

Kontextmodellierung

1. **Domänenmodell * 2**
2. Geschäftsprozessmodelle * 1
   (bei Wirtschaftsinformatik-Bezug)

Anforderungsmodellierung

3. ***Use Cases* *5**
4. Use Case Diagram *1

Analysemodelle

5. ***System-Sequenzdiagramme* *3**
6. *Contracts der Systemoperationen *3*

Entwurfsmodelle

7. **Schichtenarchitektur und Architekturdiagramm * 2**
8. ***Interaktionsdiagramme (GRASP und Design Patterns)* *5**
9. **Klassendiagramme *3**
10. *UI Storyboard *1*

Spezifikationsdokument

11. Lesbarkeit und Aufbau *1
12. Konsistenz der enthaltenen Modelle *2

Source Code

13. **Ausführbarkeit (lauffähiges JAR-File) *1**
14. *Korrektheit *2*
15. Einhaltung von Coding Guidelines *2
16. Verwendung von Javadoc und Kommentierung *1
17. Konsistenz mit der Spezifikation *2

Testfälle

19. ***Überdeckung* *4**

Benutzerschnittstelle

20. *Ergonomie/Bedienbarkeit *3*
21. ***Reaktivität* *1**

Insgesamt besteht das vorgestellte Bewertungsschema aus zwanzig Einzelleistungen n_1 bis n_{20}, die alle auf einer vierstufigen Skala mit 0 bis 3 Punkten wie folgt bewertet werden:

- 0 Punkte (und ungenügend) werden vergeben, wenn ein Element komplett fehlt oder grob fehlerhaft bzw. unvollständig ist
- 1 Punkt gibt es für Kategorien, die sinnvolle Inhalte haben, die aber noch deutlich fehlerhaft bzw. lückenhaft sind
- 2 Punkte gibt es für weitgehend vollständige und überwiegend korrekte Elemente
- 3 Punkte werden für vollständige, korrekte und sehr sorgfältig erarbeitete Artefakte vergeben

Der Multiplikator am Ende jeder Bewertungskategorie gibt die vorgesehene Gewichtung w_i innerhalb des gesamten Rasters an, so dass eine Gesamtpunktzahl leicht nach folgender Formel berechnet werden kann:

$$g = \sum_{i=1}^{20} n_i \cdot w_i$$

In der vorgestellten Form werden teambasierte Bewertungsanteile 18-fach gewichtet, individuelle Bewertungsanteile 27-fach, so dass sowohl die Teamarbeit gefördert, als auch individuelle Anstrengungen berücksichtigt werden und somit die geforderte individuelle Benotung sichergestellt ist.

Zusammenfassung des Schemas

Da auch bei Verwendung des gerade vorgestellten Bewertungsschemas Täuschungsversuche der Studierenden durch Abschreiben erfahrungsgemäß nicht ausgeschlossen werden können, empfiehlt es sich, das Bestehen des Projekts bzw. seine Gesamtnote nicht alleine von der Artefakt-Bewertung abhängig zu machen. Die vom Autor eingesetzte Kombination mit anderen bereits angesprochenen und vorgestellten Bewertungsverfahren gestaltet sich im Detail wie folgt:

- Übungsaufgaben mit direktem Projektbezug *(unbenotet, nur abzugeben)*
- Programmiertestat am Rechner *(unbenotet, aber zu bestehen)*
- 1. Kolloquium *(unbenotet, nur zu bestehen)*
- 2. Kolloquium *(15% der Endnote)*
- Abschlusspräsentation und -kolloquium *(25% der Endnote)*
- Bewertung des Systems, wie oben gezeigt *(60% der Endnote)*
- Stundenzettel (*unbenotet, glaubhaft und nachvollziehbar geführt abzugeben*)

Durch dieses Bewertungsmodell wird gewährleistet, dass jeder Teilnehmer zumindest über grundlegende Programmierkenntnisse verfügt, die ihn in die Lage versetzen, konstruktiv zur Arbeit seines Teams beizutragen. Durch die Kolloquien wird ferner das Verständnis der geleisteten Arbeit überprüft und eine soziale Kontrollinstanz eingefügt, wodurch letztlich mit großer Wahrscheinlichkeit davon ausgegangen werden kann, dass die abgelieferten und benoteten Artefakte auch von den entsprechenden Studierenden erstellt worden sind. Natürlich sollte auch die Bewertung Kolloquien auf bekannten Richtlinien und Bewertungskriterien, wie z.B. aus Schubert-Henning (2004), aufbauen.

Ein früheres Feedback über die Leistungen der Teilnehmer sollte zur Vervollständigung des Eindrucks ohnehin über regelmäßige Gespräche mit den Tutoren erhoben werden, so dass bei Verdacht auf mangelhafte Leistungen bereits im Projektverlauf eingegriffen werden kann. Auch eine Bewertung einzelner Artefakte aus für den Projektverlauf definierten Meilensteinen ist möglich und gibt den Teilnehmern eine frühe Rückmeldung über ihren Leistungsstand.

Erfahrungsbericht & Diskussion

Unter der Verantwortung des Autors wurde das vorgestellte, kombinierte Bewertungssystem in drei Softwaretechnik-Lehrveranstaltungen (zwei Mal in einer einsemestrigen Bachelorveranstaltung mit je ca. 50 Teilnehmern in Vierergruppen und einmal in einem zweisemestrigen Masterprojekt mit vier Teilnehmern) erfolgreich eingesetzt und aufbauend auf vorherigen Erfahrungen immer wieder leicht verändert. Insbesondere hat sich gezeigt, dass die Verwendung der vollen Notenskala (also 1,0; 1,3; … 4,0 bzw. 5,0) für die Systembewertung nicht zielführend ist, da die entsprechend feinen Unterschiede zwischen den Notenstufen zu zeitaufwändig in der Ermittlung und im Nachhinein praktisch nicht mehr nachvollziehbar sind. Dies gilt insbesondere wenn mehrere Prüfer in den Bewertungsprozess involviert sein sollten. Basierend auf der recht hohen Zahl von ca. 20 Einzelbewertungen hat es sich als ausreichend erwiesen, nur die zuvor genannten vier Bewertungsstufen zu verwenden. Bei deren Verwendung ergeben sich für obiges Raster mit seinen 45 Gewichtungsfaktoren maximal 135 erreichbare Punkte, die mit Hilfe gängiger (teilweise sogar in Prüfungsordnungen festgelegten) Prozentskalen problemlos in entsprechende-Noten umgerechnet werden können.

Ein berechtigter Einwand gegen das vorgeschlagene Bewertungsraster ist sicher sein hoher Detaillierungsgrad, der den Studierenden deutliche Hinweise auf die verlangte Lösung liefern kann (also z.B. welche Modelle werden erwartet?). Sofern die sinnvolle Wahl von Modellen (d.h. das „Tailoring“ der Vorgehensweise und die Auswahl der erstellten Artefakte) ein Lernziel der Veranstaltung dar-

stellt, ist das Bewertungssystem in seiner vorgestellten Form sicherlich nicht anwendbar. Zu diesem Einwand gibt es allerdings zwei wichtige Erwiderungen: zum einen die Feststellung, dass Anfänger im Software Engineering (die erstmals an einer entsprechenden Veranstaltung teilnehmen) häufig noch genug mit dem Erlernen und Einsetzen der vorgestellten Notationen und Techniken gefordert sind und entsprechend mit deren Auswahl noch weit überfordert wären. Konkretere Vorgaben werden somit ohnehin notwendig, um die Verunsicherung der Studierenden zu reduzieren und auch Tutoren und Mitarbeitern eine Orientierung zur Beantwortung entsprechender Rückfragen zu geben. Das vorgestellte Bewertungsraster transportiert somit die Erwartungen des Prüfers an die Studierenden und ermöglicht ihnen, ihre Arbeitsschwerpunkte entsprechend der gegebenen Gewichtungen zu setzen.

Für fortgeschrittenere Studierende, die bereits ein Anfängerprojekt durchlaufen haben, gibt das Raster in der gezeigten Form andererseits tatsächlich zu viele Hinweise preis. Es besteht natürlich die Möglichkeit, beispielsweise nur fünf Hauptkategorien (wie Kontextmodellierung, Anforderungen, Analyse, Entwurf und Sourcecode) und ihre Gewichtungen zu nennen, nicht aber die konkret darin erwarteten Modelle. Ferner könnte noch eine zusätzliche Bewertungskategorie zur Modellauswahl etabliert werden, die die Sinnhaftigkeit der gewählten Modelle bewertet. Die Gewichtung innerhalb dieser Kategorie kann beispielsweise gleichmäßig auf die gelieferten Modelle verteilt werden oder auch von den Studierenden vorgeschlagen werden, wenn sie mit einem entsprechenden Bewertungsschema bereits in einer Grundlagenveranstaltung vertraut gemacht worden sind. Ferner liegt es für länger laufende Projekte ohne feste Funktionalitätsvorgabe nahe, auch einen entsprechend hoch gewichteten Bewertungsfaktor für die Menge der umgesetzten Funktionalität (also für die Produktivität) einzuführen. Weiterhin besteht natürlich die Möglichkeit, auch die Einhaltung des gewählten Vorgehensmodells, die Erstellung und Befolgung eines Projektplans oder auch Softskills, wie z.B. die Teamfähigkeit zu bewerten bzw. letztere evtl. durch Peer Assessments (Race et al., 2005) bewerten zu lassen. Insgesamt bietet das vorgeschlagene Bewertungsraster genügend Flexibilität, um mit wenig Aufwand an die Lernziele vieler Softwaretechnik-Lehrveranstaltungen angepasst werden zu können.

Betrachtung der Testgütekriterien

Die in diesem Beitrag vorgestellte Kombination von Bewertungsverfahren erreicht eine gute Überdeckung der eingangs genannten Testgütekriterien. Da die Bewertung des vorgeschlagenen Programmiertestats automatisiert über die zu durchlaufenden Testfälle erfolgen kann, ist es ein sehr objektives Verfahren. Es misst zudem genau die Leistung, die auch gemessen werden soll, nämlich die Fähigkeit, ein gegebenes Problem innerhalb eines beschränkten Zeitraums in ein passendes Programm umzusetzen. Es gibt sowohl Studierenden als auch Lehrenden ein hilfreiches Feedback bei der Einschätzung der Programmierkenntnisse und ist zu dem noch günstig (da automatisiert) durchführbar. Auch eine gewisse Skalierbarkeit ist gegeben, solange genügend gleichzeitig verfügbare Arbeitsplätze in einem oder mehreren Rechnerpools zur Verfügung stehen.

Da es sich bei Kolloquien im Wesentlichen um Präsentationen und Prüfungsgespräche handelt, ist die Bewertung naturgemäß zu einem gewissen Grad subjektiv (s. z.B. Müller & Bayer, 2007). Ferner erfassen Kolloquien nicht notwendigerweise die tatsächliche Leistungsfähigkeit eines Prüflings, und noch weniger garantieren sie, seine Fähigkeiten in der Softwareentwicklung zu messen, da auch Präsentations- und Gesprächskompetenz eine nicht zu unterschätzende Rolle spielen. Der praktische Nutzen ist nichtsdestotrotz für beide Parteien hoch, da sich alle Beteiligten einen Eindruck über den Wissensstand und das Verständnis eines Studierenden machen und im Zweifelsfall direkt nachfragen können. Für kleine Gruppen sind solche Gespräche auch ein durchaus ökonomisches Prüfungsverfahren, sobald allerdings viele Studierende geprüft werden müssen, steigt die Belastung für den Prüfer sehr stark an.

Das vorgestellte Raster zur Systembewertung ermöglicht eine hohe Objektivität, zumindest dann, wenn, wie oben beschrieben, nicht die volle Notenskala zur Bewertung der Teilleistungen verwendet wird. Die Unterscheidung zwischen beispielsweise einer 1,7 und einer 2,0 für ein Klassendiagramm ist nach Erfahrung des Autors objektiv oft nicht mehr möglich. Mit der vereinfachten, vierstufigen Benotungsvariante hingegen ist das Verfahren sehr zuverlässig und valide, da es wiederholbar genau die Leistung eines Studierenden oder eines Teams bei der Erstellung des Systems bewertet, die bewertet werden soll. Somit ergibt sich auch ein hoher Nutzen für Prüfer und Studierende, da beide ein gutes Feedback über den tatsächlichen Leistungsstand der Studierenden erhalten. Einzig signifikante Schwäche des Verfahrens ist der recht hohe Bewertungsaufwand, der sich bei einem einsemestrigen Projekt mit ca. vier Studierenden pro Team schnell auf vier und mehr Stunden pro Team belaufen kann. Legt man ca. fünf ECTS und damit insgesamt rund 600 Stunden Arbeitslast für eine Viergruppe von Studierenden zugrunde, relativiert

sich diese Zahl allerdings wieder; wie auch im Vergleich zu einer Klausur, die oft einen noch höheren Korrekturaufwand benötigt. Zudem lässt sich durch klare Vorgaben für die geforderten Abgaben (z.B. direkt ausführbares System, Quellcode mit klarer Struktur in einem Versionierungssystem, Dokumente mit einer gewissen Struktur und klare Benennung der Urheber) signifikant Zeit bei der Bewertung einsparen.

Verwandte Arbeiten

In der Literatur ist zwar eine erkleckliche Menge von Berichten über Softwareentwicklungsprojekte an Hochschulen verfügbar, die dort gemachten Aussagen über das verwendete Bewertungsverfahren beschränken sich aber meist auf wenige Sätze. Systematische Vergleiche verschiedener Ansätze finden sich daher leider ebenso wenig, wie auch empirische Studien zur Zufriedenheit der Beteiligten mit den folgenden Bewertungsansätzen bisher nicht verfügbar sind.

Kerer et al. (2005) verwendeten für ihr Entwicklungsprojekt mit ca. 600 Studierenden eine Kombination aus kleineren Übungsaufgaben (sog. Assignments) und einer Klausur mit projektbezogenen Fragen und Programmieraufgaben. Interessant ist bei dieser Veröffentlichung speziell die offenbar hohe, aber nicht näher spezifizierte Zahl von aufgedeckten Plagiatsfällen, sowohl bei den Assignments, als auch bei den abzugebenden Programmen, die nochmals die Notwendigkeit einer individuellen Bewertung unterstreicht.

Ludewig (2011) hingegen kritisiert das bloße Abfragen von Fakten (z.B. mit Hilfe einer Klausur) zur Bewertung einer praktischen Lehrveranstaltung als nicht zielführend und benennt die, aus seiner Sicht recht einfache Bewertung der Gruppenleistung (also des erstellten Systems) als Alternative. Kriterien für deren Bewertung werden aber leider nicht genannt. Von der Prüfungsordnung verlangte individuelle Noten wurden in Ludewigs Veranstaltungen „für Teilnehmer, die [positiv oder negativ] auffielen, durch ein angemessenes Delta“ aus der Gruppennote abgeleitet. Dieses Verfahren wurde nach Kenntnisstand des Autors erstmals von Forbrig (1997) vorgestellt und beispielsweise auch von Stoyan & Glinz (2005) angewendet. Lindig und Zeller (2005) gehen in ihrem Bericht über ein sechswöchiges Vollzeitprojekt ebenfalls nicht näher auf die Benotungskriterien ein, es werden nur Kolloquien und automatisierte Systemtests als Prüfkriterien genannt.

Ein wenig detaillierter berichtet Metzger (2003), der in seinem Projekt zu 50% die Individualleistung und zu 50% die Gruppenleistung bewertet hat. „Zur Gruppenleistung tragen die Abgaben zu den einzelnen Aufgaben und die Abschlusspräsentation bei. In die Individualbewertung fließt die Leistung während der Präsenzzeiten und die Qualität des Einzelvortrags ein.“ Konkrete Bewertungskriterien für die Leistungen der Studierenden werden allerdings wiederum nicht genannt. Predoiu (2010) hat jüngst ein Konzept vorgestellt, in dem Artefakt-Bewertungen durch einen Betreuer mit Peer Assessments der Studierenden kombiniert werden, um eine individuelle Benotung zu ermöglichen. Dazu werden zu Beginn eines Projekts Artefakte von Seiten des Betreuers den Meilensteinen zugeordnet: jeder Studierende muss pro Meilenstein mindestens ein Artefakt bearbeiten und wird zeitnah nach der Abgabe in einer Assessmentsitzung benotet. Vorteile dieses Vorgehens sind sicher die kontinuierliche Bewertung der Teilnehmer und der enge Kontakt zwischen Studierenden und Betreuer; letzterer kann sich aber z.B. hinsichtlich Skalierbarkeit bei vielen Studierenden auch als nachteilig erweisen. Erprobt wurde das System in einer Kleingruppe mit weniger als zehn Teilnehmern. Zudem gibt es in Predoius Vorschlag keine teambasierten Benotungsanteile, so dass allein die individuelle Leistung zählt und nicht die notwendigen Beiträge für das Gesamtsystem. Ungeklärt bleibt auch die Effektivität eines Peer Assessments zur Erkennung bzw. Benennung von Trittbrettfahrern.

Nach dem Kenntnisstand des Autors am nächsten an das in diesem Beitrag vorgestellte Bewertungsraster kommt der Vorschlag von Wikstrand und Börstler (2006) heran. In ihrem Bewertungsansatz erwirbt ein Team gemäß seiner Leistung eine Anzahl von Credits für verschiedenste Aufgaben im Projekt, dazu gehören die Erstellung von Artefakten und Projektplänen ebenso wie gehaltene Präsentationen. Am Ende des Projekts sind die Teammitglieder gefordert, diese Credits eigenständig untereinander zu verteilen, so dass sich gegebenenfalls individuelle Abstufungen in der Benotung ergeben. Für die Betreuer besteht bei offensichtlichen Ungerechtigkeiten eine Eingriffsmöglichkeit. Eine effektive Erkennung von Studierenden, die auf Grund mangelnder Vorkenntnisse nicht hinreichend zum Fortschritt des Teams beitragen konnten, gibt es bei diesem Vorschlag allerdings wiederum nicht, es wird allein auf gegenseitige soziale Kontrolle durch die Studierenden selbst gesetzt.

Verschiedene Empfehlungen zur Verminderung dieses Problems, wie auch zur organisatorischen Durchführung von Softwaretechnik-Projekten insgesamt, geben beispielsweise van der Duim et al. (2007). Sie schlagen vor „Free Riders“, also Trittbrettfahrer, dadurch zu erkennen, dass die Studierende Stundenzettel zu führen und einen vordefinierten Prozess zu verwenden haben; ferner sind wöchentliche Feedback-Gespräche mit den Betreu-

ern vorgesehen, konkrete Nachweise der Programmierfähigkeiten werden aber wiederum nicht verlangt.

ZUSAMMENFASSUNG

Der vorliegende Beitrag beschreibt eine neuartige Zusammenstellung von Bewertungsverfahren zur Benotung von Softwareentwicklungsprojekten an Hochschulen und Universitäten. Durch die Kombination eines automatisierten Programmiertestats mit mehreren Kolloquien und einem gewichteten Bewertungsraster für im Verlauf des Projekts erstellte Artefakte ermöglicht es die Komposition von teambasierten und individuellen Benotungsanteilen sowie eine effektive Erkennung von sogenannten Trittbrettfahrern. Somit wird im Vergleich zu bisherigen Bewertungsansätzen eine wesentlich objektivere Bewertung möglich, zudem wird die Nachvollziehbarkeit der Benotung durch die Studierenden deutlich verbessert.

Das vorgestellte Bewertungsraster für in der Softwareentwicklung erstellte Artefakte ist sicherlich die Kerninnovation dieses Beitrags, die zuvor nur in der Arbeit von Wikstrand und Börstler (2006) in annähernd ähnlicher Form vorgeschlagen wurde. Es bewertet im Gegensatz zu zahlreichen anderen Verfahren wie Klausuren, Übungsaufgaben oder auch Kolloquien direkt die Ergebnisse der Projektarbeit und damit die Erreichung der Lernziele und versucht nicht, eine Benotung über „Umwege“ sicherzustellen. Somit wird die Motivation und Konzentration der Studierenden auf die wesentlichen Ziele des Projekts gerichtet, und sie werden in einem ohnehin schon komplexen Lernumfeld nicht zusätzlich mit weiteren Aufgaben wie der Vorbereitung auf eine Klausur belastet. Durch die Bewertung teambasierter und individueller Leistungsanteile wird sowohl die Zusammenarbeit im Team gefördert, als auch den Ansprüchen von Prüfungsordnungen und Akkreditierungsagenturen nach einer individuellen Benotung genüge getan. Ferner wird durch die Kombination mit einem automatisiert auswertbaren Programmiertestat und zwei Kolloquien sichergestellt, dass Trittbrettfahrer frühzeitig entdeckt werden und die Leistungen ihrer Teamkollegen nicht nachhaltig negativ beeinflussen können. Insgesamt ist die Bewertung der abgelieferten Systemartefakte allerdings mit einem recht hohen Bewertungsaufwand verbunden (nach bisherigen Erfahrungen ca. 4-5 h pro Team), dieser liegt allerdings in der Regel unter einem Prozent des auf Seiten der Studierenden geleisteten Gesamtaufwands und ist somit der Komplexität der Aufgabenstellung angemessen und auch mit dem Aufwand beim Einsatz einer Klausur vergleichbar. Ein weiterer Vorteil ist die hohe Flexibilität des vorgestellten Bewertungsrasters, da abweichende Schwerpunkte in der Lehre durch veränderte Gewichtungen oder eine veränderte Zusammensetzung relativ leicht abgebildet werden können. So sind beispielsweise Aspekte des Projektmanagements, wie ein Projekt- oder Iterationsplan und dessen Einhaltung, in fortgeschrittenen Lehrveranstaltungen problemlos bewertbar. Insgesamt erscheint das vorgestellte Bewertungsschema damit so flexibel, dass es prinzipiell auch für Projekte in anderen Fachgebieten der Informatik anwendbar sein sollte.

Bis dato wurde das Verfahren in jeweils leicht abgewandelter Form in drei Projekten an der Universität Mannheim mit insgesamt gut 100 Studierenden erprobt. Die Benotung und im Extremfall auch ein Durchfallen lassen von Studierenden waren insbesondere durch die angewendete vierstufige Systembewertung gut begründbar und letztlich auch für die Studierenden nachvollziehbar, so dass sich Detaildiskussionen über die Bewertung subjektiv deutlich reduzieren ließen (entsprechende Parameter wurden leider nicht systematisch erfasst). Neben kleineren Verbesserungen, wie einer weiteren Verfeinerung der Abgaberichtlinien oder eventuellen Anpassungen der Gewichtungsfaktoren, soll bei zukünftigen Projekten auch eine systematische Befragung der Projektteilnehmer durchgeführt werden, um Stärken und Schwächen des Verfahrens aus Sicht der Studierenden zu erkennen und gegebenenfalls identifizierte Probleme zu beheben.

DANKSAGUNG

Der Autor möchte Melanie Klinger vom Referat für Hochschuldidaktik sowie Colin Atkinson, Werner Janjic und Marcus Schumacher von der Software-Engineering-Gruppe der Universität Mannheim herzlich für zahlreiche konstruktive Diskussionen und Anregungen im Zusammenhang mit dem vorgestellten Bewertungsschema danken.

Literatur

ACM (2004): Software Engineering Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, in Computing Curricula Series.

Beck, K. (1999): Extreme Programming Explained: Embrace Change. Addison-Wesley.

Böttcher, A. & Thurner, V. (2011): Kompetenorientierte Lehre im Software Engineering, Proceedings des Workshops für Software Engineering im Unterricht der Hochschulen.

Brooks, F. (1995): The Mythical Man-Month. Essays on Software Engineering (2. Aufl.), Addison-Wesley.

Bunse, C. & von Knethen, A. (2008): Vorgehensmodelle kompakt (2. Aufl.), Spektrum Akademischer Verlag.

Burnette, E. & Staudemeyer, J. (2009): Eclipse IDE – kurz & gut (2. Aufl.), O'Reilly.

Coldewey, J. (2009): Schlechte Noten für die Informatik-Ausbildung, in OBJEKTspektrum, Ausgabe 05.

Endres, A. & Rombach, D. (2003): A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories, Addison-Wesley.

Forbrig, P. (1997): Probleme der Themenwahl und der Bewertung bei der Projektarbeit in der Software-Engineering-Ausbildung, Proceedings des Workshops für Software Engineering im Unterricht der Hochschulen.

Hayes, J.H., Lethbridge, T.C. & Port, D. (2007): Evaluating Individual Contribution toward Group Software Engineering Projects, Proceedings of the International Conference on Software Engineering, IEEE.

Iller, C. & Wick, A. (2009): Prüfungen als Evaluation der Kompetenzentwicklung im Studium, Das Hochschulwesen, Vol. 6.

Kerer, C.; Reif, G.; Gschwind, T., Kirda, E.; Kurmanowytsch, R. & Paralic, M.: ShareMe (2005): Running a distributed systems lab for 600 students with three faculty members, IEEE Transactions on Education, Vol. 48, No. 3.

Kultusministerkonferenz: Ländergemeinsame Strukturvorgaben für die Akkreditierung von Bachelor und Masterstudiengängen (2010), verfügbar unter: http://www.kmk.org/fileadmin/veroeffentlichungen_beschluesse/2003/2003_10_10-Laendergemeinsame-Strukturvorgaben.pdf, letzter Zugriff: Juli 2012.

Larman, C. & Basili, V. (2003): Iterative and incremental developments: a brief history, IEEE Computer, Vol. 36, No. 6.

Larman, C. (2004): Applying UML and Pattern (3. ed.), Prentice Hall.

Lindig, C. & Zeller, A. (2005): Ein Softwaretechnik-Praktikum als Sommerkurs, Proceedings des Workshops für Software Engineering im Unterricht der Hochschulen.

Ludewig, J. (1999): Softwaretechnik in Stuttgart – ein konstruktiver Informatik-Studiengang, Softwaretechnik-Spektrum Vol. 22, No. 1.

Ludewig, J. (2011): Erfahrungen bei der Lehre des Software Engineering, Ludewig und Böttcher (Hrsg.): Software Engineering im Unterricht der Hochschulen, CEUR-WS.org.

Metzger, A. (2003): Konzeption und Analyse eines Softwarepraktikums im Grundstudium, Siedersleben & Weber-Wulff (Hrsg.): Software Engin. im Unterricht der Hochschulen, dpunkt.

Müller F. & Bayer C. (2007): Prüfungen: Vorbereitung – Durchführung - Bewertung. In Kröning: Förderung von Kompetenzen in der Hochschullehre, Asanger.

Pichler, R. (2007): Scrum Agiles Projektmanagement erfolgreich einsetzen, dpunkt.

Predoiu, L. (2010): Didaktik und Bewertung in längerfristigen Teamprojekten in der Hochschullehre. Proceedings des 6. Workshops der GI-Fachgruppe "Didaktik der Informatik".

Race, P.; Brown, S. & Smith, B. (2005): 500 Tips on Assessment, Routledge.

Rausch, A.; Hohn, R.; Broy, M.; Bergner, K. & Höppner, S. (2008): Das V-Modell XT: Grundlagen, Methodik und Anwendungen, Springer.

Roloff, S. (2002): Hochschuldidaktisches Seminar: Schriftliche Prüfungen, verfügbar unter: http://www.lehrbeauftragte.net/documents_public/SchriftlPruef_Roloff.pdf, letzter Zugriff: Juli 2012.

Schubert-Henning (2004), S.: Empfehlungen zur Vorbereitung und zum Vortrag eines Referates, Studienwerkstatt der Universität Bremen, verfügbar unter: http://www.philosophie.uni-bremen.de/fileadmin/mediapool/philosophie/Formulare-Informationen/empfehlungen_referate.pdf, letzter Zugriff: Juli 2012.

Sommerville, I. (2010): Software Engineering (9th ed.), Addison-Wesley.

Stangl, W. (2002): Lernen in Gruppen, Werner Stangls Arbeitsblätter, verfügbar unter: http://www.stangl-taller.at /ARBEITSBLAETTER/LERNEN/Gruppenlernen.shtml, letzter Zugriff: Juli 2012.

Stoyan, R. & Glinz, M. (2005): Methoden und Techniken zum Erreichen didaktischer Ziele in Software-Engineering-Praktika, Löhr & Lichter (Hrsg.): Software Engineering im Unterricht der Hochschulen, dpunkt.

van der Duim, L.; Andersson, J. & Sinnema, M. (2007): Good practices for Educational Software Engineering, Proceedings of the International Conference on Software Engineering, IEEE.

Wikstrand, G. & Börstler, J. (2006): Success Factors for Team Project Courses. Proceedings of International Conference on Software Engineering Education and Training.