Das GATE-System: Qualitätssteigerung durch Selbsttests für Studenten bei der Onlineabgabe von Übungsaufgaben?

Sven Strickroth, Hannes Olivier, Niels Pinkwart

Institut für Informatik
Technische Universität Clausthal
Julius-Albert-Str. 4
38678 Clausthal-Zellerfeld
{sven.strickroth, hannes.olivier, niels.pinkwart}@tu-clausthal.de

Abstract: Das selbstständige Bearbeiten von Übungsaufgaben ist ein essentieller Teil der universitären Lehre. Ohne praktische Übungen lassen sich in vielen Fachgebieten keine Kompetenzen erwerben. Das in diesem Artikel beschriebene GATE-System kombiniert verschiedene Ansätze, um sowohl Studenten als auch Tutoren beim Übungsbetrieb zu unterstützen und somit letztlich den Lernerfolg zu verbessern. Das webbasierte System wurde begleitend zum Übungsbetrieb bei einer Grundlagen-Programmiervorlesung für Wirtschaftswissenschaftler eingesetzt und evaluiert.

1 Einleitung

Viele Universitätskurse in mathematischen, technischen oder auch naturwissenschaftlichen Fächern wie auch in der Informatik setzen sich traditionell aus zwei Blöcken zusammen: Frontalunterricht bzw. eine Vorlesung, in der das Wissen in Form eines Vortrags vermittelt wird, und ein Übungsbetrieb, um dieses (oft eher theoretische) Wissen praktisch anzuwenden und zu festigen. Die Übungen sind meist essentiell, um praktische Fertigkeiten bzw. Handlungskompetenzen zu erwerben. In einführenden Programmierkursen werden Studenten häufig zu Übungszwecken aufgefordert, kleinere Programme selbst zu schreiben oder existierende Programme zu erweitern. Da Probleme oft mit verschiedenen Programmieransätzen gelöst werden können, werden Übungsleiter benötigt, um studentische Lösungen zu bewerten.

Während kleine Kurse noch von einem Übungsleiter alleine begleitet werden können, skaliert der Ansatz „Vorlesung und Übungen“ bei großen Mengen von Studenten nur, indem man eine ebenfalls steigende Anzahl von Tutoren einsetzt. Erfahrungen aus der Praxis zeigen, dass meist schon eine Gruppengröße jenseits der 30 Studenten nicht mehr von einem einzigen Tutor bewältigt werden kann.

Online-Systeme für die Abgabe und Bewertung von Lösungen sparen Zeit für alle Beteiligten. Es wurde in [PP97] gezeigt, dass dies die Qualität der Lehre nicht negativ beeinflussen muss. Problematisch ist es jedoch, wenn die Online-Abgaben der Studenten in irgendeiner Form mit Prüfleistungen verbunden sind. In diesem Fall muss, z. B. mit einer Plagiat-Erkennung, sichergestellt werden, dass Studenten die Leistung(en) tatsächlich selbständig erbracht haben.

In diesem Paper wird das System GATE (Generic Assessment & Testing Environment) für Programmierübungen präsentiert, das verschiedene Ansätze und Funktionalitäten verbindet: Es ist primär ein Onlineabgabesystem für Übungsabgaben (mit Funktionen zur Punktevergabe, Aufteilung der Studenten in Gruppen etc.) und integriert eine automatische Plagiat-Erkennung sowie Syntax- und Funktionstests, um den Tutoren die Arbeit zu erleichtern. Gleichzeitig soll die Qualität der studentischen Lösungen verbessert werden (bzw. die Studenten in deren Lernprozessen unterstützt werden), indem das System den Studenten limitiertes Feedback zu ihren Abgaben liefert.

Im Folgenden wird ein kurzer Überblick über bereits existierende vergleichbare Ansätze gegeben. Darauf folgen die Beschreibung von GATE und eine Vorstellung des Szenarios, in dem das System evaluiert wurde. Abschließend werden die Ergebnisse der Evaluierung präsentiert und diskutiert.

2 Aktuelle Ansätze

Heutzutage werden im universitären Umfeld verschiedene Online-Systeme eingesetzt, um die Lehre zu unterstützen. Hier wird insbesondere zwischen verwaltungsunterstützenden und lern- bzw. lehrunterstützenden Systemen unterschieden.

Systeme wie z. B. HIS-LSF[1] oder Stud.IP[2] sind auf die Verwaltung von Räumen und Vorlesungen ausgelegt. Diese Systeme ermöglichen unter anderem das Bereitstellen von Dateien zum Download und das Organisieren von Übungsgruppen.

Systeme zur Unterstützung der Lehre gibt es einige: Ein populäres Beispiel ist das Lon-Capa Projekt[3], welches Unterstützung in den Übungen und sogar webbasierte Prüfungen bietet. Innerhalb dieses Systems gibt es für Studenten die Möglichkeit, Aufgaben online zu lösen bzw. Dokumente abzugeben. In Fächern wie der Physik oder der Mathematik gibt es Aufgaben, die automatisch (für unterschiedliche Studenten teils mit verschiedenen Werten) generiert und bei denen studentische Lösungen auch teilweise automatisch evaluiert werden können. Bei Aufgaben, die keine eindeutige Lösung besitzen (wie z. B. textuelle Aufgabenstellungen oder auch anspruchsvollere Modellierungs- oder Programmieraufgaben), versagen die automatisierten Evaluierungsansätze allerdings häufig, da einige (insbesondere kreative) korrekte Lösungen nicht in den systeminternen Schemata enthalten sein können.

[1] http://www.his.de/abt1/ab10
[2] http://www.studip.de
[3] http://www.lon-capa.org

Das Online Judge System [CKLO03, KLC01] wurde entwickelt, um automatisiert Programme zu testen und Lösungen auf Plagiate zu überprüfen. Diese Software nutzt eine formalisierte Version von E-Mails als Abgabesystem. Lösungen werden in einer sicheren virtuellen Laufzeitumgebung ausgeführt und basierend auf Ein- und Ausgaben sowie Laufzeit bewertet.

Douce et. al. präsentierten in [DLO05] einen Überblick über verschiedene Ansätze des automatisierten Testens. Wenn die Aufgaben ausführlich beschrieben sowie Ein- und Ausgabemengen klar definiert sind, können Systeme Rückschlüsse auf die Korrektheit liefern. Jedoch sind Tutoren nicht zu ersetzen, wenn es darum geht, Faktoren wie Wartbarkeit (u.a. in Form von sinnvollen Kommentaren und Variablenbenennungen oder Code-Formatierung), Modularisierung oder Kreativität von Programmen zu bewerten.

Das BOSS System [JGB05] wurde entwickelt, um Programmierkurse und besonders den Übungsbetrieb bei diesen Kursen online zu unterstützen. Das System erlaubt die Onlineabgabe von studentischen Lösungen (nur Einzelabgaben). Ferner können für einzelne Aufgaben JUnit- oder einfache Eingabe-/Ausgabe-Tests definiert werden, die zur Evaluierung von Java-Programmen von Studenten und Tutoren beliebig häufig ausgeführt werden können. Diese Art von Tests wird im Abschnitt 3 genauer beschrieben. Über die Korrektheit der Lösungen hinaus wird mit Softwaremetriken versucht, Aussagen über die Programme zu treffen. Diese Metriken bilden Quellcode-Eigenschaften (wie z. B. die Anzahl der abstrakten Methoden oder das Verhältnis von Programmcode und Kommentaren) auf Zahlenwerte ab, die dann als Qualitätsmerkmal bzw. Vergleichswert herangezogen werden können. Bei der frei verfügbaren Version von BOSS ist jedoch keine Plagiat-Erkennung integriert.

Das Course Maker System [HHST03] versucht, Studenten direkt zu unterstützen, indem es automatische Rückmeldungen zu hochgeladenen Lösungen bietet. Die Programme der Studenten werden verschiedenen Tests unterzogen und die Ergebnisse an die Studenten zurückgemeldet. Dies erlaubt es den Studenten (je nach Vorgabe des Kurses), mehrere Bearbeitungsiterationen durchzuführen. Jedoch ist das Course Maker System nicht webbasiert und daher nur von Rechnern zu benutzen, die eine spezielle Client-Software installiert haben. Neben dem erhöhten Aufwand, den Client extra zu installieren, gibt es bei solcher Software oft Probleme mit Sicherheitssoftware wie z. B. Firewalls. Dies kann insbesondere bei fachfremden Studenten zu größeren Problemen führen.

Die „Environment for Learning to Program“ (ELP, [Tr07]) legt den Fokus nicht auf die Abgaben von Lösungen zu Programmieraufgaben, sondern es handelt sich hier um eine webbasierte IDE für Java/C#, die speziell für Programmieranfänger entwickelt wurde. Kern des Systems ist es, Studenten einzelne „fill-in-the-gap“ Aufgaben lösen zu lassen und sie dabei auch mit Syntax- und Funktionstests beim Lösen der Aufgaben zu unterstützen. Weitere Systeme dieser Art werden in [Tr07] und [HQW08] beschrieben.

Für Programmiervorlesungen sollte ein System verschiedene Anforderungen erfüllen: Es sollte zunächst eine einfache Abgabestruktur von Lösungen verschiedenster Art (insbesondere Quellcode, UML-Diagramme und Text) bieten und Plagiate erkennen. Des Weiteren sollte es Studenten erlauben, Aufgaben in kleinen Gruppen zu bearbeiten

und die Tutoren bei ihren Aufgaben unterstützen. Es sollte dabei so modelliert sein, dass einfach neue Elemente, wie weitere Tests, hinzugefügt werden können. Die beschriebenen Systeme erfüllen nur Teile dieser Anforderungsliste, jedoch nicht alle.

3 Beschreibung des GATE-Systems

Angestrebt wurde ein System, das von vielen Benutzern gleichzeitig, ohne die Installation von spezieller Software und ohne großen Aufwand genutzt werden kann. Daher wurde das System als moderne Web-Applikation konzipiert. Architektur und Anforderungen ähneln dem parallel entwickelten DUESIE [HQW08]. Es gibt verschiedene Rollen: Administratoren, Betreuer/Kursleiter, Tutoren und Studenten: Der Betreuer (Kursleiter) erzeugt für eine Veranstaltung neue Aufgaben und legt dabei die Aufgabenbeschreibung, den Abgabeschluss sowie Plagiat-Erkennung und automatisierte Testmöglichkeiten der abgegebenen Lösungen (sowohl für die Studenten als auch für die Tutoren) fest. Nachdem Aufgaben angelegt sind, können Studenten (alleine oder optional in kleinen Gruppen) innerhalb der Abgabefrist ihre Lösungen einreichen. Direkt nach dem Upload der Lösung können Studenten ihre Abgaben einsehen und es stehen (ggf. limitierte) Studenten-Tests zur Verfügung, die einzeln angefordert werden können. Sofern es ein Student möchte, kann er innerhalb der Abgabefrist korrigierte bzw. veränderte Lösungen einreichen.

Direkt nach Ablauf der Abgabefrist werden die konfigurierten Tutor-Tests und die Plagiat-Erkennung für die Tutoren automatisiert ausgeführt. Dadurch können die Ergebnisse den Tutoren während des Bewertungsprozesses sofort angezeigt werden, ohne dass Tests einzeln angefordert oder lokal ausgeführt werden müssen. Dem Tutor werden auf der Übersichtsseite einer Aufgabe alle Lösungen tabellarisch mit Kurzinformationen zu möglichen Duplikaten, Ergebnissen der Tests sowie bisher vergebenen Punkte angezeigt. Von dort aus kann ein Tutor zu konkreten Abgaben navigieren, wo er detailliert alle Informationen inklusive der eingesandten Daten abrufen und die Lösung bewerten kann. In der Vergangenheit wurde schon bei verschiedenen Systemen festgestellt, dass Onlinesysteme oft die Qualität der Rückmeldungen reduzieren (siehe [MW99]). Um dieses Problem zu reduzieren, wurde bei GATE den Tutoren eine Freitext-Kommentarfunktion zur Verfügung gestellt. Insbesondere für die Erkennung von Plagiaten gibt es im System eine Möglichkeit, die eingesandten Quelltexte ohne Kommentare anzuzeigen. Die Bepunktung kann entweder komplett frei oder über fest vorgegebene Checkboxen erfolgen. Insbesondere bei Letzterem kann dem Dozenten ein genauer Überblick über Lernerfolge gegeben werden. Wenn alle Lösungen bewertet wurden, können die Studenten schließlich ihre Bewertung online einsehen.

Der Prototyp des Systems benutzt Java 1.6 sowie Apache Tomcat 6.0. Damit ist das ganze System nicht nur im Bezug auf den Client, sondern auch auf der Server-Seite plattformunabhängig. Technisch besteht das Websystem aus drei Schichten: der Persistenzschicht (Datenbankzugriff mittels Hibernate auf MySQL), der Applikationsschicht (Anwendungslogik im Web Container, MVC Model 2, vgl. [LR06]) und der GUI (Browser auf den Clients, CSS und HTML getrennt). Dieses Design soll eine einfache Erweiterbarkeit und Anpassbarkeit für neue Features ermöglichen.

Die Plagiat- und Funktionstests (Studenten- und Tutoren-Tests) wurden innerhalb des Systems als erweiterbares Framework entworfen und implementiert, so dass relativ einfach weitere Tests bzw. Codenormalisierungen eingebunden werden können. Für die Tests existieren im Prototyp zwei Varianten: Syntax- und Funktionstests. Für den Syntaxtest wird versucht, den Quelltext der Abgabe zu kompilieren bzw. zu parsen. Funktionstests lassen sich in Black- und Whitebox-Tests unterteilen: Blackbox-Tests sind einfache I/O-Tests (starten der Abgabe mit einer definierten Eingabe und Vergleichen der Ausgabe, z. B. mit einem regulären Ausdruck). Whitebox-Testing kann im Fall der Programmiersprache Java über JUnit-Tests [GB99] realisiert werden. Grundsätzlich lassen sich beliebige weitere Programmiersprachen oder Anfragesprachen wie SQL in das Test-Framework integrieren. Einzige Voraussetzung ist, dass für diese Sprachen auf der Server-Plattform Compiler bzw. Interpreter existieren müssen. Lediglich für eine sichere Ausführungsumgebung der zu testenden Programme muss gesorgt werden (Sandbox gegen Schadcode, vgl. [Re89], [HQW08]).

Der Prototyp des GATE-Systems beinhaltet drei verschiedene Plagiat-Erkennungsalgorithmen, die auf einer einstellbaren Normalisierungvariante der Einsendung (keine Normalisierung / keine Leerzeichen, Tabulatorzeichen / keine Kommentare / nur Kommentare) operieren. Für kleinere Aufgaben (Programmierung oder freie Text-Aufgaben) kann das System die Editier- bzw. Levenshtein-Distanz [Le66] berechnen; für Programmier- oder Design-Aufgaben (z. B. UML-Diagramme) kann das System die universelle Normalized-Compression-Distance (NCD, basiert auf der Kolmogorow-Komplexität, [LCLMV03]) bestimmen; speziell für (fortgeschrittene) Java-Programmieraufgaben bietet das System den darauf optimierten Plaggie-Algorithmus [ASR06]. Damit ist es dem Kursleiter möglich, einen oder auch mehrere Algorithmen mit unterschiedlichen Konfigurationen auszuwählen, die (wahrscheinlich) am besten für die spezielle Aufgabe geeignet sind.

4 Einsatz des GATE-Systems

Der von uns entwickelte Systemprototyp findet Anwendung in der Betreuung von Lehrveranstaltungen (Programmierung für BWL- bzw. Informatik-/Mathematik-Studenten) an der TU Clausthal, die sich alle mit der Programmiersprache Java beschäftigten. Das System wurde bei einer Vorlesung „Grundlagen der Programmierung“ für BWL-Studenten mit ca. 350 Teilnehmern und 13 Tutoren evaluiert. In dieser Veranstaltung gab es zwei Typen von Aufgaben: Theoretische Aufgaben, die in Form von Dokumenten oder Bildern abgegeben wurden, und Programmieraufgaben, bei denen die Studenten Java-Dateien hochladen mussten. Bei allen Aufgaben mussten die Studenten zusätzlich in den Übungen ein kleines mündliches Testat bestehen, um zu verifizieren, dass sie die Aufgaben selbst gelöst hatten (erreichte Punkte wurden erst dadurch „freigeschaltet“). Studenten, die 50 % der Gesamtpunkte erarbeitet und fünf von sechs Aufgabenzetteln (mit jeweils zwei Aufgaben) bearbeitet hatten, erhielten einen unbenoteten Schein.

Für die Evaluation wurden alle Systemfeatures genutzt: Abgabe alleine oder in Zweiergruppen, verschiedene Tutor-Tests und alle drei Plagiat-Algorithmen. Für die

Studenten gab es pro Programmieraufgabe drei verschiedene Tests. Jeder Test konnte nur ein einziges Mal durchgeführt werden und meldete ausschließlich Erfolg oder Misserfolg zurück (hierdurch sollte eine reine „Try-and-Error"-Strategie verhindert werden). Aus Forschungssicht waren die Autoren daran interessiert, ob dies bei der erfolgreichen Bearbeitung von Hausaufgaben behilflich ist und die Qualität der Abgaben steigert.

5 Auswertung und Ergebnisse

Im Folgenden werden drei zentrale Fragestellungen untersucht:

- Hilft GATE den Tutoren bei der Bewertung von eingesandten Lösungen sowie bei der Erkennung von Plagiaten?
- Wie werden die Tests von den Studenten genutzt?
- Steigern die für die Studenten verfügbaren Tests die Qualität der abgegebenen Lösungen?

Dazu wurden für die Analyse des Systems sowohl qualitative als auch quantitative Daten erhoben: Während des Einsatzes wurden durchgeführte Aktionen von GATE mitgeloggt. Insbesondere beinhalten die Logs detaillierte Informationen über Tests (wann und wie oft wurde ein Test mit welchem Ergebnis durchgeführt?) und Uploads, um deren Nutzen bzw. Anwendung auswerten zu können. Nach Semesterende wurde den Tutoren ein Fragebogen (basierend auf bisherigen Erfahrungen) ausgehändigt, auf dem sie einige Features bewerten (in Werten von 1 total nutzlos, bis 5 sehr nützlich) und Probleme sowie Verbesserungen aufzeigen sollten, auf die sie bei der Benutzung des Systems gestoßen sind. Anschließend wurde zusammen mit den Tutoren eine allgemeine Nachbesprechung der Vorlesung durchgeführt. In diesem Rahmen wurde auch GATE angesprochen. Dabei kamen unter anderem auch Themen und Probleme zur Sprache, die nicht bzw. nur indirekt in den Fragebögen erwähnt wurden.

5.1 Auswertung der Tutorenreaktionen

Die Möglichkeit, Programme ohne Kommentare anzuzeigen, wurde im Durchschnitt mit 4,7 bewertet (sd = 0,6). Diese Funktion wurde zum einen von den Tutoren in erster Linie bei den Präsenztestaten genutzt, um zu überprüfen, ob die Studenten ihre Programme wirklich selbst verstanden haben (diese konnten so nicht einfach die Kommentare nutzen, um die Programme zu erklären). Zum anderen hat diese Funktion das Finden von Plagiaten erleichtert. Meist sahen die Programme ohne Kommentare auf den ersten Blick gleich aus (teilweise inklusive identischer Formatierung). Die Tutoren bewerteten bei Programmieraufgaben Plaggie im Schnitt mit 4,3 (sd = 0,8), die NCD und Levenshtein im Mittel mit 2,8 (sd = 1,1). Bei Aufgaben, die nicht mit Programmieren zu tun hatten, haben die Tutoren die Plagiatsalgorithmen schlechter bewertet. Hier erhielten der Levenshtein und der NCD nur eine durchschnittliche Bewertung von 2. Dies lag in erster Linie daran, dass die Algorithmen schlechte Ergebnisse liefern, wenn die verglichenen

Lösungen als Dateien mit unterschiedlichen Dateitypen (.txt, .pdf, .doc) vorliegen, da der Overhead in den Datei-Formaten bereits genug war, um selbst ähnliche Lösungen als „nicht identisch“ zu erkennen.

Die Hauptprobleme der Tutoren mit dem System waren die Unzuverlässigkeit der Funktionstests in bestimmten Situationen: Da diese Tests in erster Linie Eingaben an das Programm senden und Ausgaben vergleichen, können schon kleine Tippfehler dazu führen, dass ein korrektes Programm als fehlerhaft markiert wird. Falls der Syntax-Test schon fehlschlug, können Funktionstests auch keine Lösung liefern. Gerade ersteres passierte häufiger, wenn z. B. die Paket-Informationen in Programmen nicht korrekt gesetzt wurden (welches zwar ein Fehler ist, allerdings kein sehr gravierender).

Auf die Frage, ob sie lieber mit oder ohne GATE arbeiten wollen, antworteten alle Tutoren, dass sie das System bevorzugen. Sie nannten verschiedene Gründe:

- Umweltschutz (keine Papierstapel), da die Abgaben von theoretischen Aufgaben nicht wie früher auf Papier erfolgten und auch Ausdrucke nicht mehr nötig sind.
- Zeitersparnis und keine Lauferei zum Einsammeln von Abgabenblättern
- Möglichkeit, die Korrekturen überall zu korrigieren (sofern man Internetzugang hat); Heimarbeit ist möglich.
- Einfacher Überblick über die aktuellen Punktstände
- Die Plaggie-Tests waren sehr hilfreich: Viele (auch gruppenübergreifende) Plagiate wurden gefunden, die normalerweise nicht aufgefallen wären.
- Syntax- und Funktionstests waren sehr hilfreich

5.2 Auswertung der von Studenten durchgeführten Tests

Für den Übungsbetrieb wurden 13 Übungsaufgaben herausgegeben, die entweder allein oder in Zweiergruppen bearbeitet werden konnten. Vier Aufgaben waren keine Programmieraufgaben, sondern Text-Aufgaben theoretischer Natur, und eine Aufgabe war eine Bonus-Aufgabe, bei der keine Tests zur Verfügung standen. Die acht verbleibenden Programmieraufgaben werden im Folgenden als Grundlage der Betrachtung verwendet. Bei jeder Aufgabe standen den Studenten ein Syntax-Test sowie zwei unterschiedliche Funktionstests zur Verfügung, die von einer Abgabegruppe jeweils (maximal) einmal ausgeführt werden konnten (in beliebiger Reihenfolge).

Insgesamt wurden 1031 Abgaben/Lösungen eingesandt. Davon wurden 617 Abgaben alleine und 414 in Zweiergruppen bearbeitet. In Tabelle 1 sind die Anzahl der eingesandten Lösungen und die Anzahl der durchgeführten Tests nach Aufgaben aufgeschlüsselt dargestellt. Der Übersichtlichkeit halber werden in der Tabelle beide Funktionstests zusammengefasst und als ein einziger betrachtet. Der Funktionstest wurde als „durchgeführt“ aufgeführt, sofern mindestens einer ausgeführt wurde.

Die sinkenden Abgabezahlen lassen sich damit erklären, dass die Studenten zum Ende des Übungszeitraumes vermehrt die Prüfungsleistungen bereits erfüllt hatten und somit

insbesondere die besseren Studenten keine weiteren Lösungen mehr einreichen mussten (es wurde nur ein unbenoteter Schein vergeben; daher gab es keinen Anreiz, mehr als 50 % der Gesamtpunkte zu erzielen).

Es ist in Tabelle 1 gut zu erkennen, dass die Tests bereits ab der ersten Aufgabe sehr häufig genutzt wurden und dass es keinen Anstieg der Nutzungsraten über die Zeit zu verzeichnen gab. Daraus lässt sich schließen, dass für die Benutzung der Tests keine Eingewöhnungs- bzw. Lernphase nötig war. Wider Erwarten lässt sich für die Benutzung der Tests kein Trend ablesen: Vermutet wurde, dass die Syntax-Tests am Anfang deutlich häufiger als zum Ende der Veranstaltung hin ausgeführt werden. Diese Vermutung basierte auf der Annahme, dass dieser Test durch neu erlernte Kompetenzen, wie dem Umgang mit einer Entwicklungsumgebung, in den Hintergrund treten würde. Erkennbar ist aber, dass die Syntax-Tests sehr häufig ausgeführt wurden und sich die Test-Rate hier durchgängig zwischen 70 und 93 Prozent bewegt.

Steigerung der Abgabequalität durch Tests
Als erstes wird die Hypothese, dass von Studenten ausführbare Programmtests vor Abgabeschluss die Qualität der abgegebenen Lösungen steigern, betrachtet. Zur Analyse wurden hier die Anzahl korrekter Abgaben von Studenten, die den Syntax-Test ausgeführt hatten, der Anzahl korrekter Abgaben von Studenten, die den Syntax-Test nicht ausgeführt hatten, gegenübergestellt (Tabelle 2). Die Angabe „Finale Syntax korrekt“ bezieht sich hierbei auf die Einstufung/Bewertung der finalen Abgabe durch einen der Tutoren (in dem Sinne, dass dieser einer Lösung die Punkte für „korrekte Syntax“ gegeben hat). Die Spalte „Diff“ enthält die Differenz der Korrektheitsraten mit und ohne vorherigen Syntax-Test in Prozentpunkten. Aufgabe 8 kann auf Grund der geringen Anzahl von Abgaben (15) als nicht repräsentativ angesehen werden.

Über alle Aufgaben ist hier erkennbar, dass bei Abgaben, die durch die Studenten mit dem Syntax-Test geprüft wurden, die finalen Versionen zu durchschnittlich 90 % syntaktisch korrekt waren; bei Abgaben, die nicht auf korrekte Syntax geprüft wurden, ist dies hingegen nur zu 65 % der Fall – ein Unterschied von 25 Prozentpunkten. In beiden Fällen ist aber ein Sinken der Korrektheitsraten über den Übungszeitraum hinweg auszumachen. Dies kann zum Teil damit begründet werden, dass die Schwierigkeit der Aufgaben mit der Zeit zunahm, zusätzlich wurden gegen Ende der Veranstaltung weniger Lösungen abgegeben – und diese vermehrt von den eher leistungsschwächeren Studenten stammten, die die Punkte unbedingt benötigt haben, um noch auf insgesamt 50 % der Punkte für den Scheinerwerb zu kommen.

Die Angabe „Finaler Code korrekt“ in Tabelle 2 bezieht sich hier auf die Bewertung eines Tutors, dass die studentische Abgabe die funktionalen Anforderungen für ihn hinreichend gut erfüllt (Tutoren konnten bzw. haben bei kleineren Fehlern oftmals noch alle Punkte vergeben). Aufgabe 3 musste bei den Funktionstests auf Grund eines Fehlers im Test selbst aus der Auswertung entfernt werden. Auch bei diesen Daten ist ein deutlicher Unterschied zwischen den funktionsgetesteten und ungetesteten Abgaben erkennbar: Unter den Abgaben, die mit einem Funktionstest geprüft wurden, waren die finalen Versionen zu durchschnittlich 52 % korrekt; unter den Abgaben, die nicht geprüft wurden, war dies hingegen nur zu 18 % der Fall - ein Unterschied von 34 Prozentpunkten.

Auf den Quelldaten wurde ein t-Test für unabhängige Stichproben durchgeführt, um die Bedingungen „Test durchgeführt“ und „Test nicht durchgeführte“ hinsichtlich der syntaktischen bzw. funktionalen Korrektheit des Programms nach Abgabeschluss zu vergleichen. Der Unterschied war in beiden Fällen statistisch signifikant ($p < 0{,}0001$). Dies lässt also vermuten, dass die studentischen Tests in der Tat eine Auswirkung auf die Qualität (Korrektheit von Syntax bzw. Funktionalität) der finalen Lösung haben.

Aufgabe	Anzahl Abgaben	Syntaxtest durchgeführt	Funktionstest durchgeführt	Negative Syntax-Tests	Änderungen nach Test	Syntax der geänderten Version OK
1	209	90 % (n = 188)	90 % (n = 187)	19	17	13
2	190	93 % (n = 176)	92 % (n = 174)	14	10	5
3	161	77 % (n = 124)	75 % (n = 121)	28	14	8
4	156	81 % (n = 126)	80 % (n = 125)	13	8	2
5	92	77 % (n = 71)	70 % (n = 64)	15	4	0
6	159	81 % (n = 129)	81 % (n = 128)	20	9	5
7	49	71 % (n = 35)	73 % (n = 36)	6	1	0
8	15	93 % (n = 14)	93 % (n = 14)	4	2	0
	1031	**84 % (n = 863)**	**82 % (n = 849)**	**119**	**65**	**33**

Tabelle 1: Nutzung der studentischen Tests

Aufgabe	Finale Syntax korrekt ohne stud. Test	Finale Syntax korrekt mit stud. Test	Diff	Finaler Code korrekt ohne stud. Test	Finaler Code korrekt mit stud. Test	Diff
1	86 % (18 / 21)	97 % (182 / 188)	11	27 % (6 / 22)	56 % (104 / 187)	29
2	79 % (11 / 14)	93 % (164 / 176)	14	38 % (6 / 16)	66 % (114 / 174)	28
3	68 % (25 / 37)	84 % (104 / 124)	16	(Fehlerhafter Test)		
4	57 % (17 / 30)	91 % (115 / 126)	34	13 % (4 / 31)	59 % (74 / 125)	44
5	33 % (7 / 21)	79 % (56 / 71)	44	21 % (6 / 28)	61 % (39 / 64)	40
6	70 % (21 / 30)	88 % (114 / 129)	18	6 % (2 / 31)	27 % (35 / 128)	21
7	64 % (9 / 14)	80 % (28 / 35)	14	8 % (1 / 13)	22 % (8 / 36)	14
8	100 % (1 / 1)	71 % (10 / 14)/	-29	0 % (0 / 1)	43 % (6 / 14)	43
	65 % (109 / 168)	**90 % (773 / 863)**	**25**	**18 % (25 / 142)**	**52 % (380 / 728)**	**34**

Tabelle 2: Auswertung der studentischen Tests

Auswirkungen negativer Syntax-Tests

In diesem Abschnitt sollen die Auswirkungen bzw. die Reaktionen der Studenten auf einen negativen Syntax-Test beleuchtet werden. Tabelle 1 beinhaltet die von Studenten durchgeführten Syntax-Tests mit negativem Resultat und zeigt, wie viele Abgaben danach verändert wurden und wie viele davon schließlich von einem Tutor als syntaktisch korrekt bewertet wurden. Ersichtlich ist hier, dass über alle Aufgaben gut die Hälfte der Studenten als Reaktion auf einen erkannten Syntaxfehler eine neue Version hochgeladen hat. Ungefähr die Hälfte dieser aktualisierten Lösungen konnten das Problem so lösen, dass die Abgaben letztlich von einem Tutor als korrekt bewertet wurden: Es konnten durch den negativen Syntax-Test also in ca. 25 % der Fälle eine Verbesserung der Lösung erreicht werden.

Bereits bei der ersten Aufgabe wurde bei fast 90 Prozent der inkorrekten Abgaben mindestens eine überarbeitete Fassung eingereicht, die dann zu etwa 75 Prozent korrekt war. Scheinbar war auch hier keine Einarbeitungsphase erforderlich und die Studenten konnten ihre Fehler in fast drei Viertel der Fälle korrigieren. Der Trend sinkender Werte über das Semester ist auch hier erkennbar. Allerdings scheint es, als ob zum Ende prozentual weniger Änderungen nach einem negativen Syntax-Test vorgenommen wurden. Neben den bereits erwähnten Ursachen kommen hier weitere mögliche hinzu: Studenten, denen zum Ende hin noch Punkte fehlten, fielen die letzten (komplexeren) Aufgaben vermutlich schwerer und könnten daher eventuell nicht in der Lage gewesen sein, den bzw. die Fehler zu korrigieren. Eventuell könnten die Studenten auch bewusst auf eine Fehlerkorrektur verzichtet haben: Wenn die Studenten nur noch wenige Punkte für die Prüfungsleistung benötigten, war es ihnen eventuell ein zu großer Aufwand, alle Fehler zu beheben. Stattdessen haben sie evtl. darauf gesetzt, ausreichend viele Teilpunkte zu erhalten. Weiterhin wäre möglich, dass das Testergebnis von den Studenten angezweifelt wurde. Dieser Aspekt wird im nächsten Abschnitt weiter beleuchtet. Eine genaue Klärung der Ursachen ist hier ohne weitere Untersuchungen nicht möglich.

Allgemeine Evaluation der Tests
Eine detaillierte Analyse der Testergebnisse (manueller Vergleich von Testergebnissen mit Tutor-Bewertungen) ergab, dass es sowohl bei den Syntaxtests als auch bei den Funktionstests keine falsch-positiven Klassifizierungen von eingesandten Lösungen gab. Die Syntax-Tests haben an sich keine falsch-negativen Klassifizierungen hervorgerufen, jedoch haben die Tutoren in 13 Fällen (< 1,5 %) auch auf nicht ganz korrekte Lösungen Punkte vergeben (Fehler waren nur Kleinigkeiten wie z. B. das Fehlen einer Klammer).

Den Tutoren war es bei ihrer Bewertung vor allem wichtig, dass man anhand der Lösung erkennen konnte, dass sich der Student mit der Aufgabe beschäftigt und die Grundlagen verstanden hat. Die Funktionstests weisen eine höhere Rate an falsch-negativen Klassifizierungen auf: Von 1031 Lösungen wurden 759 Lösungen (73,6 %) als nicht korrekt getestet, aber die Tutoren haben davon 201 Lösungen (26,5 %) trotzdem als korrekt bewertet. Als Gründe konnten folgende Fälle identifiziert werden: Das Package war falsch oder nicht gesetzt, die erwartete Ausgabe stimmte nicht mit der tatsächlichen Ausgabe überein (z. B. Tippfehler) oder es traten kleinere Fehler auf, für die von den Tutoren keine Punkte abgezogen wurden. Daraus ist ersichtlich, dass die Tutoren nicht blind den Tests vertraut haben, sondern sich die Lösung trotzdem genau angesehen und – wie intendiert – die Tests lediglich als Orientierungshilfe für ihre Tätigkeit angesehen haben.

6 Diskussion

In nur 12 von 1033 Fällen (verteilt über den gesamten Übungszeitraum) haben es Studenten im Rahmen der Vorlesung (für BWL-Studenten) nicht geschafft, ihre Lösung selbständig hochzuladen. In diesen Fällen haben die betroffenen Studenten die Lösungen per E-Mail an den Kursleiter gesandt und die Lösungen mussten manuell ins System integriert werden. Die Probleme waren darin begründet, dass GATE einen Filter für gültige Dateinamen beinhaltete und Studenten die Dateien nicht korrekt benannt hatten (z. B. waren die Dateiendung oder der Dateiname von Java-Klassen inkorrekt).

Bei Betrachtung der Test-Upload-Reihenfolgen stellte sich heraus, dass in 37 Fällen ein Funktionstest ohne vorherigen Syntax-Test durchgeführt wurde. Bei Programmier-Anfängern hätte man eigentlich ein anderes Vorgehen erwartet, wobei keine Informationen darüber vorliegen, ob die betreffenden Studenten bereits lokal einen Syntax-Test durchgeführt hatten. Viel überraschender ist, dass bei 53 Abgaben (5 %) ein Funktionstest direkt, also ohne Änderungen an der Lösung vorzunehmen, nach einem fehlgeschlagenen Syntax-Test durchgeführt wurde. Diese 53 Fälle verteilen sich auf alle Aufgaben. Interessant war auch, dass es vier Abgaben gab, die den studentischen Syntax-Test bestanden, aber danach durch veränderte Versionen ersetzt wurden, die nicht mehr syntaktisch korrekt waren.

Auf die Plagiat-Erkennung kann in diesem Artikel nicht ausgiebig eingegangen werden, jedoch sollen einige interessante Auffälligkeiten erwähnt werden: Im Interview haben die Tutoren angemerkt, dass bei der Plagiat-Erkennung nur Abgaben mit einer Ähnlichkeitsheuristik von mindestens 90 Prozent weiterverfolgt wurden. Die Plagiat-Tests waren hingegen so eingestellt, dass sie alle möglichen Abgaben mit einer Mindestähnlichkeit von 50 Prozent in der Tutoren-Sicht angezeigt haben. Basis der Tests war immer die vollständige Normalisierung (keine Kommentare, Konvertierung in Kleinbuchstaben und Entfernung doppelter Leerzeichen/Tabulatoren). Dies führte dazu, dass teilweise bis zu 45 Abgaben als mögliche Plagiate angezeigt wurden. Weiter berichten die Tutoren, dass sie sich nach den ersten zwei bzw. drei Programmier-aufgaben fast ausschließlich an der Kombination von Plaggie und Levenshtein-Distanz orientiert haben – bei einer hohen errechneten Ähnlichkeit (> 90 %) lag immer ein Plagiat vor. Die vermeintlichen Plagiate bestätigten sich in der Regel bei den Testaten, in denen die Studenten ihre Abgaben nicht erklären konnten bzw. das Abschreiben direkt zugaben.

7 Zusammenfassung und Ausblick

In diesem Artikel wurde ein Ansatz vorgestellt, um Studenten, Tutoren und Dozenten beim Übungsbetrieb zu unterstützen. Eine Prototypumsetzung (das GATE-System) wurde entwickelt und in mehreren Programmiervorlesungen getestet. Ergebnisse zeigen, dass sowohl Tutoren als auch Studenten Vorteile durch den Einsatz des Systems hatten und dass mit dem Konzept studentischer Tests vor Abgabeschluss durchaus Qualitätssteigerungen erreicht werden können.

Tutoren konnten mit GATE unter den eingesandten Lösungen einfacher Plagiate erkennen. Für die Bewertung und Testate standen den Tutoren nach eigenen Angaben weitere nützliche Features zur Seite, die den Aufwand für sie reduzierten, ohne dass sie dabei den Hilfestellungen blind vertrauten, sondern weiterhin mit Augenmaß Bewertungen vornahmen. Zudem konnten die Studenten immer ihre eigenen aktuellen Punktstände online einsehen und sich in die Übungsgruppen eintragen.

Das GATE-System wird kontinuierlich weiter entwickelt: Die modulare Architektur erlaubt es, weitere Tests einzubinden, um z. B. andere Programmiersprachen zu unterstützen. Auch ist geplant, weitere Funktionen zum System hinzuzufügen. Tutoren gaben an, dass bei Aufgaben, in denen vorgegebene Programme erweitert werden

sollten, es für sie relativ mühselig war, die Änderungen der Studenten zu erkennen. Daher wurde vorgeschlagen, eine „Differenz"-Funktion in das System zu integrieren, welche nur die Erweiterungen enthält bzw. diese besonders hervorhebt. Zudem werden in der vorgestellten Version Plagiate nur innerhalb der eingesandten Lösungen gesucht. Da einige Studenten in der Vergangenheit Lösungen aus dem Internet kopiert haben, wird überlegt, die Plagiat-Algorithmen mit einer Google-Funktion bzw. Internet-Recherche zu erweitern, die ähnliche Lösungen im Internet sucht. Eine Erweiterung für UML-Modellierungsaufgaben befindet sich derzeit in der Entwicklung.

Um die in Abschnitt 6 erwähnten „unsinnigen" Tests zu reduzieren, ist auch die Entwicklung einer Empfehlungsfunktion vorgesehen, die dem Nutzer sagt, ob ein Test zu diesem Zeitpunkt aussagekräftig sein kann oder nicht.

Literaturverzeichnis

[ASR06] Ahtiainen, A.; Surakka, S.; Rahikainen, M.: Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. Baltic Sea; Vol. 276, Proceedings of the 6th Baltic Sea conference on Computing education research, 2006.

[CKLO03] Cheang, B.; Kurnia, A.; Lim, A.; Oon; W.-C.: On automated grading of programming assignments in an academic institution, 2003. Computers and Education 41.

[DLO05] Douce, C.; Livingston, D.; Orwell, J.: Automatic Test-Based Assessment of Programming: A Review, ACM JERIC, 5(3), 2005.

[GB99] Gamma, E.; Beck, K.: JUnit: A cook's tour. Java Report, 1999. 4(5).

[HHST03] Higgins, C.; Hegazy, T.; Symeodinis, P.; Tsintsifas, A.: The CourseMarker CBA System: Improvements over Ceilidh. Education and Information Technologies 8(3), 2003.

[HQW08] Hoffmann, A.; Quast, A.; Wismüller, R.: Online-Übungssystem für die Programmierausbildung zur Einführung in die Informatik. DeLFI 2008.

[JGB05] Joy, M.; Griths, N.; Boyatt, R.: The BOSS Online Submission and Assessment System. ACM JERIC, 5(3), 2005.

[KLC01] Kurnia, A.; Lim, A.; Cheang, B.: Online Judge. Computers & Education 36(4), 2001.

[LCLMV03] Li, M.; Chen, X.; Li, X.; Ma, B.; Vita Nyi, P.: The similarity metric. In: SODA 2003, Proc. of the 14th annual ACM-SIAM symposium on Discrete algorithms. Philadelphia, PA, USA : Society for Industrial and Applied Mathematics.

[Le66] Levenshtein, W.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, Vol. 10, 1966.

[LR06] Lahres, B.; Rayman, G.: Praxisbuch Objektorientierung. Professionelle Entwurfsverfahren. Galileo Computing, 2006. Kapitel 8.2

[MW99] Mason, D. V.; Woit, D. M.: Providing Mark-up and Feedback to Students with Online Marking, SIGCSE '99 3/99, New Orleans. LA, USA.

[PP97] Price, B.; Petre, M.: Teaching Programming through Paperless Assignments: An empirical evaluation of instructor feedback. ITiCSE, 1997.

[Re89] Reek, K. A.: The TRY system -or- how to avoid testing student programs. In: SIGCSE 21(1), 1989.

[Tr07] Truong, N.: A Web-Based Programming Environment for Novice Programmers. QUT Thesis, 2007: http://eprints.qut.edu.au/16471/1/Nghi_Truong_Thesis.pdf