Web-basierte Programmierpraktika mit Praktomat

Jens Krinke* Maximilian Störzer* Andreas Zeller**

Abstract: Das System *Praktomat* unterstützt die webbasierte Durchführung von Programmierpraktika durch automatisiertes Praktikums-Management und Qualitätskontrolle der eingereichten Lösungen. Die Kernfunktionalität des Systems wird in diesem Artikel knapp umrissen. Die vierjährigen Erfahrungen haben gezeigt, dass der Einsatz zu reduziertem Betreuungsaufwand führt, die Qualität der studentischen Programme deutlich steigt und softwaretechnische Methoden wie Code-Review gut vermittelt werden können.

1 Erfahrungen aus der Lehre

Um Studenten Programmieren zu lehren, reicht es nicht, studentische Programme nur nach „funktioniert“ und „funktioniert nicht“ zu beurteilen. Gerade bei größeren Programmen von Programmierpraktika ist eine angemessene *Qualitätskontrolle* eingereichter Lösungen unerlässlich. Diese ist jedoch *von Hand* sehr aufwendig. Es muss also nach Wegen gesucht werden, um den Aufwand zu minimieren und gleichzeitig optimalen Lernerfolg zu gewährleisten.

Zusätzlich ergibt sich das Problem, dass es durch die Finanzlage der Universitäten nicht mehr möglich ist, genug Rechnerarbeitsplätze zur Verfügung zu stellen und es wünschenswert ist, dass Studenten Praktika auf dem eigenen Rechner durchführen können. Daher gibt es bereits eine Reihe von web-basierten multimedialen Lehr- und Lernsystemen (eine Übersicht bietet [1]), die das Einreichen von studentischen Programmen und das anschließende elektronische Testieren erlauben. Leider geht die Unterstützung beim Testieren kaum über einfache automatische Tests hinaus („funktioniert“ oder „funktioniert nicht“).

Um die Qualitätskontrolle in Programmierpraktika zu unterstützen, haben wir das System *Praktomat* entwickelt, das mittlerweile seit vier Jahren in Passau (und nun auch in Saarbrücken) erfolgreich eingesetzt wird.

In der Praxis wird die Qualität eines Programms durch *Testen* und *Software-Inspektion*, also das Lesen eingereichter Programme, ermittelt. Diese Techniken werden durch Praktomat automatisiert bzw. stark unterstützt, so dass in unseren Praktika *bei gleichem Aufwand* die Qualität der Programme deutlich verbessert wurde und Studenten neben dem Schreiben auch das *Lesen* fremder Programme lernen.

*Universität Passau, Lehrstuhl für Software-Systeme, 94032 Passau, {stoerzer, krinke}@fmi.uni-passau.de

**Universität des Saarlandes, Lehrstuhl für Softwaretechnik, 66041 Saarbrücken, zeller@cs.uni-sb.de

2 Einsatz von Praktomat

In Passau ist für Studenten im dritten Semester ein individuelles Programmierpraktikum mit vier Aufgaben unterschiedlichen Schwierigkeitsgrads verpflichtend. Diese Praktika werden mit Hilfe von Praktomat durchgeführt. Praktomat ist im Wesentlichen eine webgestützte Datenbankanwendung, die Aufgaben, eingereichte Lösungen und deren Bewertungen verwaltet. Praktomat bietet darüber hinaus:

Ortsungebundenen Zugriff Als *Webanwendung* ist der Einsatz von Praktomat nicht ortsgebunden. Von jedem internetfähigen Arbeitsplatz weltweit können sowohl Aufgaben bearbeitet als auch administrative Tätigkeiten oder das Bewerten von Lösungen durchgeführt werden.

Funktionale Tests Eingereichte Lösungen werden schon beim Einreichen übersetzt und müssen eine Reihe von *Tests* bestehen. Schlägt ein Test fehl, wird das fehlschlagende Testelement präsentiert.

Automatische Stilprüfung Neben funktionalen Tests sind auch Prüfungen auf Konformität mit *Programmier-Richtlinien* möglich. Diese Prüfungen waren bisher mit sehr großem Aufwand verbunden. Durch die Integration externer Tools in Praktomat konnten auch diese Prüfungen weitgehend automatisiert werden.

Gegenseitiges Kommentieren Praktomat erlaubt *gegenseitige Software-Inspektion* durch Studenten, wodurch sowohl Autor als auch Kommentierer einer Lösung profitieren: man lernt nicht nur beim aktiven Programmieren, sondern auch beim Lesen fremder Programme.

Individuelle Aufgaben Praktomat erlaubt die Definition *unterschiedlicher Varianten* einer Aufgabe, um einerseits das gegenseitige Kommentieren zu ermöglichen und andererseits das Abschreiben fremder Lösungen zu erschweren.

2.1 Ortsungebundener Zugriff

Das Internet erlaubt es, Programmierpraktika vollständig ortsunabhängig durchzuführen. Praktomat ist als Webanwendung ein Mittel, um dies zu erreichen: es erlaubt die vollständige Administration des Praktikums als auch die Bearbeitung und Bewertung der zu lösenden Aufgaben nur mit Hilfe eines Browsers.

Dies bedeutet eine Entlastung der Universitäts-Rechnerpools, da die meisten Studenten zu Hause über einen Rechner mit Netzanschluss verfügen. Dadurch ist aber eine persönliche Betreuung vor Ort nicht mehr möglich. Statt in Praktomat Board- oder Chat-Systeme zu integrieren, wurden in Passau sehr positive Erfahrungen durch den Einsatz von *moderierten Newsgroups* gemacht - Studenten konnten hier Fragen zu Aufgaben stellen, deren Beantwortung in der Newsgroup oft vielen Studenten half. Durch die Moderierung wird verhindert, dass Lösungen in der Newsgroup ausgetauscht werden. Die zusätzlich angebotene Präsenzberatung der Studenten konnte damit deutlich reduziert werden, da sie kaum

noch gebraucht wurde. In Saarbrücken wurden ähnliche Erfahrungen mit einer offenen Mailingliste gemacht.

Durch die *Fernwart-* und *Fernnutzbarkeit* des Praktomats bietet sich der Einsatz des Systems auch für Fern- oder virtuelle Universitäten an, wie z. B. die *Virtuelle Hochschule Bayern* (VHB), die die Weiterentwicklung des Praktomats im Rahmen des JOP-Projektes (*Java Online Praktikum*) fördert. Praktomat wird damit in eine vollwertige Lernumgebung integriert.

2.2 Automatische Funktionalitätstests

Die studentischen Lösungen werden noch beim Einreichen übersetzt und müssen eine Reihe von zuvor definierten Testfällen durchlaufen. Die Testfälle können dabei in zwei Gruppen aufgeteilt werden: in verpflichtende und nicht-verpflichtende Testfälle. Lösungen werden nur dann entgegengenommen, wenn die verpflichtenden Testfälle bestanden wurden. Da die Testfälle mit Hilfe von DEJAGNU[1] definiert und geprüft werden, kann der Student die Ursache eines Fehlschlags genau feststellen. Hat der Student die Fehler seines Programms korrigiert, so dass es die verpflichtenden Testfälle besteht, kann er seine Lösung auf dem Praktomat als zu bewertende Lösung belassen. Schlagen nicht verpflichtende Testfälle fehl, so kann der Student dies ignorieren und trotzdem seine Lösung zum Bewerten freigeben. Er kann aber auch weiterhin sein Programm verbessern, bis es auch diese Testfälle besteht. Diese Unterteilung ermöglicht es, schon vorab deutlich härtere Tests durchzuführen, dabei aber leistungsschwächere Studenten nicht zu überfordern.

Neben den für die Studenten bei der Abgabe sichtbaren öffentlichen Testfällen gibt es noch *erweiterte* (bei der Abgabe sichtbare aber nicht verpflichtende) und für die Studenten nicht sichtbare *geheime* Testfälle. Damit sollen Studenten motiviert werden, Ihre Programme ausführlich zu testen und sich nicht nur auf die öffentlichen Testfälle zu verlassen.

Als sinnvoll hat sich auch die Nutzung von Testabdeckungs-Werkzeugen herausgestellt. Ein Beispiel ist JCOV (integriert im SUN Java Compiler) mit dem Methoden in den Programmen der Studenten aufdeckt werden, die während der Testfälle nicht aufgerufen werden. Dabei kann es sich um im Code verbliebene Testtreiber oder auch bewusst oder unbewusst nicht benutzte optimierende Algorithmenteile handeln, die aber in der Aufgabenstellung gefordert werden.[2]

Insgesamt sind die automatischen Tests viel breiter als manuelle Inspektion durch das Lehrpersonal, da jedes Detail getestet werden kann. Diese Anforderungen werden von Studenten als deutlich härter empfunden. Praktomat ist aber nicht nur hart, sondern auch gerecht: die automatischen Funktionalitätstest sind immer fair – im Gegensatz zu manueller Inspektion. Im Vergleich zu früheren Praktika hat sich die Durchfallquote nicht erhöht – durchfallende Studenten geben größtenteils frühzeitig auf und bearbeiten nicht alle Aufgaben.

[1] `http://www.gnu.org/software/dejagnu/`

[2] Als Beispiel mag TSP mit Branch&Bound dienen, wo eine Methode zur Ermittlung einer unteren Schranke gefordert wird. Der Algorithmus funktioniert auch ohne diese Methode, aber viel ineffizienter.

2.3 Automatische Stilprüfung

Während die Funktionalitätstests schon mit früheren Versionen von Praktomat automatisiert wurden, ist es nun mit der Einbindung externer Tools gelungen, auch *Reviews des Programmierstils* zu automatisieren.

Für die Java-Ausbildung der Studenten verwenden wir CHECKSTYLE[3], das die Konformität der Programme bzgl. der Java Code Conventions prüft. Interessanterweise beachten Studenten die Ausgaben dieser Tools eher als Hinweise, die ihnen das Lehrpersonal gibt (und die gerne ignoriert werden). Auch die automatischen Stilprüfungen können aufgeteilt werden in verpflichtende, nicht verpflichtende und geheime Prüfungen. Als geeignetes Vorgehen hat sich herausgestellt, in den ersten Aufgaben die Stilprüfungen noch geheim zu belassen und den Studenten durch die Bewertung ausführliches Feedback zu geben. Bei späteren Aufgaben sind die automatischen Stilprüfungen zuerst öffentlich, aber nicht verpflichtend und am Ende schließlich verpflichtend. Das Ergebnis sind deutlich lesbarere Programme in einem einheitlichen Programmierstil, die auch leichter zu bewerten sind.

An der Anbindung kommerzieller Tools wie Together und JTest wird derzeit gearbeitet. Prinzipiell ist jedes Tool mit textueller Ausgabe in Praktomat integrierbar. Dabei hat sich aber gezeigt, dass nicht alle Arten von Tools sinnvoll sind – die Berechnung von *Metriken* hat sich weder für Studenten noch für Bewerter als hilfreich herausgestellt.

2.4 Individuelle Aufgaben und Feedback

HTML als Beschreibungssprache für die Aufgabenstellung ermöglicht die Benutzung verschiedenster multimedialer Inhalte. Die notwendigen ergänzenden Lehrinhalte können durch externe Lehrsysteme integriert werden. Eine enge Verzahnung ist möglich: Aufgabenstellungen im Praktomat benutzen die externen Inhalte, die wiederum Aufgaben im Praktomat zur Vertiefung des Gelernten anbieten.

Zusätzlich erlaubt es Praktomat, Aufgaben durch einen einfachen Makromechanismus in unterschiedlichen Varianten zu stellen, beispielsweise Sortieren mit Mergesort bzw. Quicksort oder Baumtraversierung in Pre-, Post- bzw. Inorder. Hierbei werden der Aufgabentext und die zugehörigen Testfälle mit M4 Makroanweisungen versehen (Bsp.: „Implementieren Sie den ifdef(VARIANTE1, ‘Quicksort’, ‘Mergesort’)-Algorithmus“). Praktomat wertet diese Informationen automatisch bei der Seitengenerierung entsprechend der dem aktuellen Teilnehmer zugewiesenen Variante aus. Dies erschwert das direkte Übernehmen von Lösungen Anderer. Da Studenten bei unerlaubtem *Code Reuse* leider überaus kreativ sind, setzen wir in Passau und Saarbrücken JPLAG [2] ein, mit dem Quelltexte automatisch auf Duplikate untersucht werden können – Plagiarismus wird so schnell entlarvt.

In einer optionalen *Kommentierphase* können Studenten gegenseitig Programme lesen und Kommentare dazu abgeben. Bei der Kommentierung bewerten Studenten die Programme ihrer Kommilitonen nach exakt denselben Kriterien, nach denen später auch Bewertun-

[3] `http://checkstyle.sourceforge.net/`

gen durch das Lehrpersonal (Testate) erstellt werden, Maßstab sind Funktionalität und Verständlichkeit.

Kommentierungen haben keinen direkten Einfluss auf die Bewertung durch das Lehrpersonal, da die abgegebenen Kommentare weder bewertet werden, noch für das Lehrpersonal sichtbar sind. Manchmal wäre aber eine Korrektur der abgegebenen Kommentare sinnvoll, da Studenten von ihrem eigenen *schlechten* Programmierstil so überzeugt sein können, dass Sie den *guten* Programmierstil ihrer Kommilitonen als *schlecht* bewerten.

Es hat sich gezeigt, dass Teilnahme an gegenseitigen Kommentierungen die Lesbarkeit der eigenen Programme deutlich steigert [3].

2.5 Endgültige Bewertung

Trotz der weitgehenden Automatisierung hat das Lehrpersonal das letzte Wort, was die Bewertung der eingereichten Programme angeht. Die Bewerter erhalten die anonymisierten Quelltexte und Ergebnisse aller automatischen Funktionalitäts- und Stilprüfungen im Browser, in dem sie nicht nur die Bewertung und allgemeine Anmerkungen angeben, sondern auch Anmerkungen direkt in den Quelltext einfügen können.

Zur Bewertung der *Verständlichkeit* eines Programms werden verschiedene Kategorien (Einrückung, Dokumentation, Struktur, Lokalität, ...) herangezogen, aus denen schließlich der Gesamteindruck des Programms abgeleitet wird.

Die fertiggestellte Bewertung wird gespeichert und dem Autor der Lösung sowohl als Mail als auch zum Abruf im Praktomat zur Verfügung gestellt.

3 Fazit

Programmieraufgaben sind nur dann sinnvoll, wenn diese *verpflichtend* sind und mit *Feedback* über Funktionalität *und Programmierstil* an die Studenten zurückfließt. Dies erfordert jedoch einen hohen Aufwand an Lehrpersonal, so dass größere Praktika als individuelle Arbeiten mit herkömmlichen Mitteln bei steigenden Anforderungen immer weniger möglich sind.

Wir haben Praktomat langjährig erfolgreich in Passau und Saarbrücken eingesetzt und damit selbst größte Praktika weitgehend automatisiert. Durch den Einsatz externer Tools zur Funktionalitäts- und Stilprüfung konnte bei gleichzeitig anspruchsvolleren Aufgabenstellungen der Korrekturaufwand des Lehrpersonals reduziert werden (in Passau liest ein Testierer etwa 200 Programme mit insgesamt 150000 Zeilen Code). Der größte Vorteil aber ist die deutlich erhöhte *Qualität* des Praktikums, die mit traditionellen Mitteln nicht erreichbar ist.

Die Lehrevaluation durch Studenten belegt die äußerst positive Bewertung von Praktika mit Praktomat. Unsere Studenten haben durch Praktomat gelernt, sorgfältiger, funktionel-

ler und verständlicher zu programmieren als in herkömmlichen Praktika – und das bei gleichem Betreuungsaufwand.

4 Praktomat selbst einsetzen

Der Praktomat-Quellcode ist frei verfügbar, so dass Sie Praktomat auch für eigene Lehrveranstaltungen einsetzen können. Technische Voraussetzungen sind ein Python-Interpreter, die Datenbank PostgreSQL und das DEJAGNU-Testpaket, die ebenfalls frei erhältlich sind.

Praktomat lässt sich leicht an neue Umgebungen anpassen, indem Konfigurationsvariablen gesetzt werden – so etwa der Name der Veranstaltung, der Name der Hochschule, die verwendete Programmiersprache und einige Prozess-Parameter.

Praktomat wird derzeit erfolgreich mit Java, C++ und Haskell eingesetzt. Eine Anbindung weiterer Programmiersprachen ist ohne größere Probleme möglich, solange eine Bedienung von Compiler/Linker mittels Kommandozeile möglich ist.

Die meisten externen Prüfungen werden mit Hilfe von GPL-Tools implementiert. Eine Anbindung an kommerzielle Tools wie Together oder JTest wird derzeit entwickelt.

Praktomat und weiterführende Literatur finden Sie im WWW unter

http://www.fmi.uni-passau.de/st/praktomat/

Danksagung. Innovationen in Praktomat profitierten von den Anregungen und Rückmeldungen der Anwender und einer Vielzahl studentischer Beiträge.

Literaturverzeichnis

[1] P. Brusilovsky and P. Miller. Course delivery systems for the virtual university. In T. Tschang and T. Della Senta, editors, *Access to Knowledge: New Information Technologies and the Emergence of the Virtual University*. Elsevier Science, Amsterdam, 2000.

[2] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. JPlag: Finding plagiarisms among a set of programs. Technical Report 2000-1, Fakultät für Informatik, Universität Karlsruhe, Germany, 2000.

[3] Andreas Zeller. Making students read and review code. In *Proc. 5th ACM SIGCSE/SIGCUE Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000)*.