Flexibles E-Assessment auf Basis einer Service-orientierten Architektur
Konzepte, Implementierung und Praxiserfahrungen

Mario Amelung, Katrin Krieger, Dietmar Rösner
Otto-von-Guericke-Universität Magdeburg
Institut für Wissens- und Sprachverarbeitung
Postfach 4120, 39016 Magdeburg
{amelung,kkrieger,roesner}@iws.cs.uni-magdeburg.de

Abstract:
Mit den eduComponents liegt seit mehreren Semestern eine Sammlung von Komponenten vor, die ein quelloffenes allgemeines Content Management System um Funktionen zur Unterstützung von Lernprozessen erweitern. In diesem Beitrag stellen wir insbesondere die Komponenten zur automatischen Überprüfung von studentischen Einreichungen zu (Programmier-)Aufgaben vor und beschreiben die zugrunde liegende Systemarchitektur. Weiterhin wird eine Vorgehensweise zur Spezifikation neuer Testkomponenten (z. B. für bisher nicht unterstützte Programmiersprachen) erläutert, mit denen das Systems einfach und flexibel erweitert werden kann. Darüber hinaus gehen wir auf den praktischen Einsatz des Systems ein und präsentieren die in unseren Lehrveranstaltungen des Wintersemesters 2008/2009 und Sommersemesters 2009 gesammelten Erfahrungen.

1 Motivation

Für das Verständnis des Lernstoffs einer Vorlesung ist es unverzichtbar, das erworbene Wissen anzuwenden und sich durch das selbständige Lösen von Problemen mit den Theorien auseinanderzusetzen. So lässt sich beispielsweise Kompetenz im Programmieren nur durch das eigenständige Lösen von Programmieraufgaben und die Umsetzung von Algorithmen in lauffähige Programme erlangen. Die Ergänzung der Präsenzlehre durch E-Learning-Elemente, insbesondere die automatische Überprüfung von studentischen Lösungen zu (Programmier-)Aufgaben (Computer Aided Assessment, CAA), kann helfen, die universitäre Ausbildung zu intensivieren und effizienter zu gestalten.

Aus technischer Sicht bieten Content Management Systeme (CMS) bereits eine Vielzahl von Funktionen, die für die Organisation des Lehrens und Lernens benötigt werden. Daher basiert unsere E-Learning-Umgebung auf dem quelloffenen allgemeinen CMS *Plone*[1], das wir um Komponenten zur Unterstützung von Lernprozessen erweitert haben (vgl. [RPA07], [Pio09]). Diese Komponenten – die eduComponents – umfassen Funktionen zur Verwaltung von Lehrveranstaltungen, zur Unterstützung von Multiple-Choice-Tests

[1] http://plone.org/

und zur Einreichung von manuell bewerteten Übungsaufgaben. Die entstandenen Software-Komponenten können sowohl für reines E-Learning als auch für verschiedene Formen des *blended learning* genutzt werden.

Zusätzlich zu manuell bewerteten Aufgaben sollten auch Programmieraufgaben in unsere Lernumgebung integriert und Einreichungen möglichst automatisch überprüft werden können. Unser Ziel war es, den Studierenden möglichst zeitnahe Rückmeldungen zu geben, die Aufgaben und Lösungen ausführlicher zu besprechen, und ihnen mehr Möglichkeiten zu bieten, ihr Wissen anzuwenden und ihre Fähigkeiten zu trainieren. Darüber hinaus wollten wir die Lehrenden von vermeidbaren (administrativen) Tätigkeiten befreien und ihnen einen besseren Überblick über den Leistungstand und den Lernfortschritt der Studierenden geben.

Aus diesen Gründen suchten wir ein E-Assessment-System zur Überprüfung von Programmieraufgaben, das folgende Anforderungen erfüllt:

- automatische Auswertung von Programmierlösungen in verschiedenen Programmiersprachen und mit unterschiedlichen Testmethoden;
- automatische Auswertung von Aufgaben in anderen formalen Systemen, wie z. B. reguläre Ausdrücke, XSL-Transformationen (XSLT), UIMA-Analysis-Engines[2];
- Bewertung und Benotung von Aufgaben, die kurze Textantworten in natürlicher Sprache erfordern;
- einfache Erweiterbarkeit um zusätzliche Aufgabenformen, Programmiersprachen und Testmethoden;
- flexible Integration der Test- und Bewertungsfunktionalität in bestehende Lernumgebungen bei gleichzeitiger Vermeidung redundanter Nutzerverwaltung und Datenhaltung.

In diesem Beitrag gehen wir daher zunächst auf den Stand der Technik bei E-Assessment-Systemen ein und beleuchten verwandte Projekte. Anschließend stellen wir einen Service-orientierten Ansatz für Systeme zum automatisierten Testen vor. Danach gehen wir auf die notwendigen Schritte zur Spezifikation neuer Services für das Testen von Programmieraufgaben ein. Zum Abschluss berichten wir von unseren Erfahrungen beim Einsatz der vorgestellten Komponenten, gehen auf mögliche Limitierungen ein und geben einen Ausblick auf zukünftige Entwicklungen.

2 Stand der Technik

Es existiert heute eine Vielzahl von – zumeist Web-basierten – Systemen zur automatischen Überprüfung von Programmieraufgaben, die zur Unterstützung in der Informatik-Ausbildung eingesetzt werden. Einige dieser Systeme sind spezialisiert auf eine Programmiersprache oder eine bestimmte Testart. Dazu zählen beispielsweise *TRAKLA2* [LSKM04],

[2] `http://incubator.apache.org/uima/`

Scheme-robo [SMK01] (Scheme) oder *AutoGrader* [Hel07] (Java). Mit anderen Systemen lassen sich mehrere Programmiersprachen testen, wie z. B. mit *CodeLab* [3] (Java, C/C++, Python) oder mit dem Portal *MyCodeMate* [4] (Java, C/C++) des Verlags Addison-Wesley. Einige wenige Systeme unterstützen prinzipiell beliebige Programmiersprachen und Testarten, da das eigentliche Testen in Modulen gekapselt ist (z. B. *CourseMarker* [HGST05]).

Das Projekt *Praktomat* [KSZ02] der Universität Passau, das sich einer besseren Qualitätskontrolle von Programmieraufgaben verschrieben hat, bietet neben dem Übersetzen und Testen von Programmcode auch die Möglichkeit, Einreichungen auf die Einhaltung der Java Code Conventions zu überprüfen. Projekte wie *WeBWorK* [GSW08] oder *Web-CAT* [EP08] legen den Fokus auf das Erlernen von testgetriebener Software-Entwicklung. Mit dem System *DUESIE* [HBN+08] der Universität Siegen können sogar Aufgaben zu UML computergestützt ausgewertet werden.

Gemeinsam ist allerdings allen Systemen, dass sie neben der eigentlichen Überprüfung immer auch Funktionen zur Verwaltung von Nutzern, Lehrveranstaltungen, Aufgaben und Einreichungen bereitstellen. Dies führt dazu, dass die Funktionalität des Testens und Bewertens nicht allein und unabhängig genutzt werden kann. Auch ist eine Integration in bestehende Learning Management Systeme (LMS) nicht möglich. Der Einsatz führt folglich zwangsläufig dazu, dass zwei oder mehr Systeme administriert und Daten redundant gehalten werden müssen. Darüber hinaus können diese Systeme nur schwer erweitert und an die eigenen Bedürfnisse angepasst werden.

3 Service-orientierter Ansatz

Unser Ansatz basiert – im Gegensatz zu den im Abschnitt 2 vorgestellten Systemen – auf der klaren Trennung aller Aspekte, die die Verwaltung von Studierenden, Aufgaben und Einreichungen betreffen von der tatsächlichen Durchführung der Tests.

Wie genau das eigentliche Testen von Einreichungen abläuft, hängt stark von der verwendeten Testmethode sowie von der Programmiersprache bzw. der formalen Notation ab. Zum Beispiel können Programmieraufgaben sowohl statisch als auch dynamisch getestet werden. Bei dynamischen Tests wird der Programmcode zur Ausführung gebracht und auf Testdaten angewendet. Die Ausgabe kann mit der Ausgabe einer Musterlösung verglichen oder hinsichtlich bestimmter Eigenschaften überprüft werden. Wird auf diese Weise unbekannter – und somit potenziell gefährlicher – Programmcode ausgeführt, sind auch Fragen der Sicherheit zu betrachten. Daher sollten alle Aspekte, die den genauen Testsablauf betreffen, möglichst gekapselt sein.

Diese Kapselung lässt sich sehr gut mittels *Services* in einer *Service-orientierten Architektur* (SOA) [5] erreichen. Ein Service ist eine Softwarekomponente, deren Funktionalität

[3] `http://www.turingscraft.com`

[4] `http://www.mycodemate.com`

[5] Eine SOA ist ein Framework für die Integration von (Geschäfts-)Prozessen in Form von sicheren, standardisierten Komponenten – Services –, die sich wiederverwenden und kombinieren lassen, um wechselnde Anforderungen abzubilden (vgl. [BBF+05]).

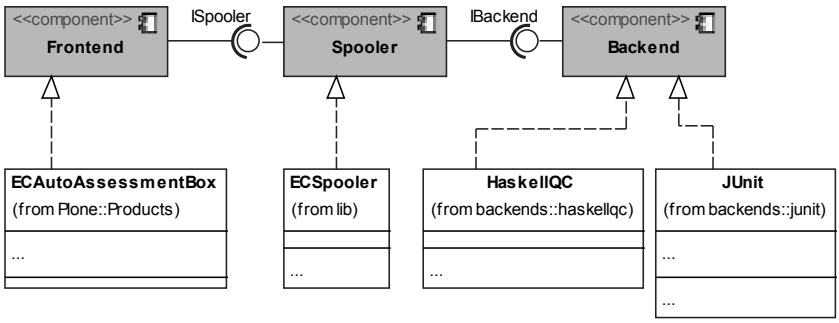

Abbildung 1: Komponenten des Service-orientierten Ansatzes und mögliche Realisierungen

plattformunabhängig über eine durch das Netzwerk erreichbare Schnittstelle angeboten wird. In unserem Modell bezeichnen wir solche Services auch als **Backends**. Backends bilden somit den ersten wichtigen Baustein für ein verteiltes System zum automatischen Testen.

Verwaltungsfunktionen, wie beispielsweise das Speichern von Aufgaben und Einreichungen, die Beachtung von Einreichungsfristen, die Rückmeldung der Überprüfung/Bewertung oder die Erstellung von Statistiken, werden typischerweise bereits von LMS abgedeckt. Diese Systeme, in unserem Modell als **Frontends** bezeichnet, können auf der Basis einheitlicher Anwendungs- und Transportprotokolle auf die Funktionen der o. g. Backends (Services) zugreifen und auf diese Weise um die Funktion des automatischen Testens erweitert werden.

Für den einheitlichen Zugriff auf die Backends und eine möglichst lose Kopplung von Frontends und Backends haben wir eine dritte Komponente, den sog. **Spooler**, eingeführt. Beim Spooler handelt es sich ebenfalls um einen Service, dessen Aufgabe es ist, neue Einreichungen zum Testen entgegen zu nehmen und diese an das entsprechende Backend weiterzuleiten. Alle Komponenten – Frontends, Spooler und jedes einzelne Backend – können auf verschiedene Rechner in einem Netzwerk verteilt sein. Daraus ergeben sich ein hoher Grad an Interoperabilität und Flexibilität sowie die Möglichkeit, eine Vielzahl von Frontends und Backends miteinander zu kombinieren. Die in den Backends gekapselte Testfunktionalität lässt sich wiederverwenden und erweitern.

Abbildung 1 zeigt die drei wesentlichen Komponenten unseres Modells und deren mögliche Realisierungen. Diese werden nachfolgend näher beschrieben.

3.1 Frontend: ECAutoAssessmentBox

Das Modul ECAutoAssessmentBox aus den eduComponents bietet Lehrenden die Möglichkeit, Aufgaben zu stellen, deren Lösungen online eingereicht und automatisch getestet werden können. Erstellt ein Lehrender eine neue Aufgabe, muss diese zunächst mit einem bestimmten Backend assoziiert werden. ECAutoAssessmentBox kommuniziert dazu direkt

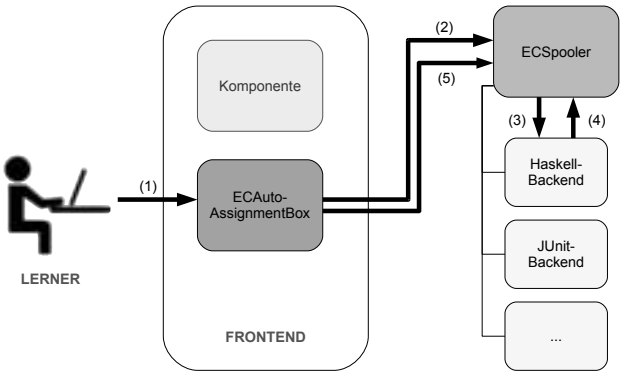

Abbildung 2: Entgegennahme und Verarbeitung einer Einreichung: (1) der Lernende trägt seine Lösung (Programmcode) in die ECAutoAssessmentBox ein. (2) ECAutoAssessmentBox schickt diese Einreichung an den Spooler und teilt diesem mit, welches Backend die Einreichung überprüfen soll. (3) Der Spooler gibt die Einreichung an das entsprechende Backend weiter. Dieses führt den Compiler und/oder Interpreter mit dem eingereichten Code aus und (4) gibt Statusmeldungen und eventuelle Fehlerbeschreibungen an den Spooler zurück. (5) ECAutoAssessmentBox fragt beim Spooler an, ob das Ergebnis der Überprüfung vorliegt, holt es ab und zeigt es als Feedback an.

mit dem ECSpooler, der eine Liste der verfügbaren Backends und pro Backend die für das Testen benötigten Eingabefelder liefert. Die Eingabefelder variieren je nach gewähltem Backend, so dass der Lehrende für ein Backend z. B. Testdaten und eine Musterlösung, für ein anderes Backend dagegen einen Unit-Tests angeben muss. Zusätzlich kann der Lehrende verschiedene Parameter festlegen, wie beispielsweise die Einreichungsfrist oder die Anzahl der Versuche. Darüber hinaus besteht die Möglichkeit, eine Antwortvorlage in Form eines Programmskeletts zu hinterlegen.

Lernende reichen ihre Lösungen über die Web-Oberfläche von ECAutoAssessmentBox ein, wobei sie wahlweise den Quellcode in eine Textbox kopieren oder den Dateiupload nutzen können. Die eingereichten Programme werden dann über den ECSpooler zum angegebenen Backend weitergeleitet – inklusive der notwendigen Testdaten. Das Ergebnis der Überprüfung wird dem Lernenden sofort nach der Überprüfung angezeigt, so dass zeitnah ein Feedback zu den Lösungen verfügbar ist. Liefert eine Einreichung auf den hinterlegten Testdaten nicht die erwarteten Ergebnisse, hat der Lernende während der Einreichungsfrist die Möglichkeit, eine verbesserte Lösung einzureichen. Dabei bleibt die ursprüngliche Einreichung erhalten, so dass auf diese bei Bedarf zurückgegriffen werden kann.

ECAutoAssessmentBox ist ein mögliches Frontend, das wie Plone in *Python* implementiert ist und via XML-RPC [6] auf die Funktionen des Spoolers und der Backends zugreift. Die gewählte Service-basierte Architektur erlaubt die Integration beliebiger Backends – selbst in heterogenen Systemlandschaften.

[6] XML-RPC ist ein Standard zum Methodenaufruf in verteilten Systemen, der zum Transport HTTP und als Codierung XML benutzt. Komplexe Datenstrukturen können übermittelt, verarbeitet und zurückgegeben werden. (`http://www.xmlrpc.com/`)

Neben ECAutoAssessmentBox können aber auch beliebige andere Frontends eingesetzt werden. Zur Veranschaulichung haben wir einen auf SOAP basierenden Web-Service und Client in *Java* entwickelt, mit dessen Hilfe Lehrende ohne den Überbau des CMS oder LMS ihre Aufgaben allein mit dem Spooler und den Backends entwickeln und testen können.

3.2 Spooler

Der Spooler, von uns als XML-RPC-Server in Python implementiert, übernimmt in unserem verteilten System die Aufgaben, die von der Programmiersprache unabhängig sind: Verwalten einer Liste der verfügbaren Backends, Entgegennahme von Einreichungen, Anstoßen der Tests durch ein Backend und Weiterleitung der Ergebnisse.

Wird eine Lösung zu einer Aufgabe eingereicht (z. B. mittels ECAutoAssessmentBox, s. Abschnitt 3.1), wird die Einreichung zusammen mit allen nötigen Testdaten zunächst an den Spooler geschickt, der sie in Form eines sog. *Test-Job* wiederum an dasjenige Backend weiterleitet, das vom Lehrenden für diese Aufgabe ausgewählt wurde. Ist das Backend gerade belegt, bleibt der Test-Job so lange in der Warteschlange, bis er bearbeitet werden kann. Die Ergebnisse der Überprüfung durch das Backend werden temporär gespeichert und vom Frontend abgeholt.

3.3 Backends

Die von uns entwickelten Backends sind ebenfalls als XML-RPC-Server implementiert. Verschiedene Backends können auf verschiedenen Rechnern laufen und sich bei unterschiedlichen Spoolern anmelden.

Jedes Backend definiert ein Schema, das alle Eingabefelder beschreibt, die für die vollständige Spezifikation des Testablaufs notwendig sind. Für eine Testdaten-basierte Auswertung sind das beispielsweise die Musterlösung und eine Reihe von Funktionsaufrufen. Das Schema definiert darüber hinaus mindestens eine sog. *test method option*, die es ermöglicht, zwischen verschiedenen Compilern oder Interpretern für eine Programmiersprache oder verschiedenen Vergleichsfunktionen (z. B. exakte Übereinstimmung im Gegensatz zum Vergleich von Gleitkommazahlen mit Toleranz) zu wählen. Auch wenn in unserer Betrachtung Backends zum Überprüfen von Programmieraufgaben im Mittelpunkt stehen, kann das System auch dazu benutzt werden, Aufgabenlösungen in anderen formalen Notationen oder in natürlicher Sprache auszuwerten (vgl. [Feu06]).

Alle Backends werden von allgemeinen Backend-Klassen abgeleitet. Diese stellen den Großteil der Standardfunktionalität bereit, wie beispielsweise das Starten und Stoppen eines Backends oder das Anmelden des Backends an einem Spooler. Somit reduziert sich die Erstellung eines Backends darauf, ein neues Eingabe- und Testschema zu definieren sowie die Methode(n) zur Ausführung der konkreten Tests zu implementieren.

4 Entwicklung eigener Backends

4.1 Vorgehen

(Programmier-)Sprache und Testmethode Sollen neue Backends entwickelt werden, gilt es im ersten Schritt, die zwei grundlegenden Fragen „Was soll getestet werden?“ und „Wie soll getestet werden?“ zu beantworten. Zur Beantwortung der ersten Frage muss zunächst entschieden werden, welche *Programmiersprache* oder andere formale Notation das Backend unterstützen soll. Diese Festlegung hat Auswirkungen auf alle nachfolgenden Schritte.

Die Frage nach dem „Wie“ wird durch die Entscheidung für eine bestimmte *Testmethode* (z. B. statisch vs. dynamisch; vgl. [Lig02]) beantwortet. Diese Entscheidung ist allerdings abhängig von der gewählten Programmiersprache. Sollen z. B. Einreichungen dynamisch mittels Unit-Tests überprüft werden, setzt dies die Verfügbarkeit eines Unit-Test-Frameworks für die gewählte Sprache voraus. Weiterhin muss festgelegt werden, unter welchen Bedingungen der Testlauf abgebrochen wird. So sollte bei dynamischen Tests keine semantische Überprüfung stattfinden, wenn der syntaktische Test bereits fehlgeschlagen ist. Eine andere Bedingung könnte sein, dass grundsätzlich der komplette Testlauf abgebrochen wird, sobald ein einzelner Test fehlschlägt.

Eingabe und Ausgabe Ausgehend von der Testmethode ergeben sich die Anforderungen an die *Eingabedaten*. Hier muss unterschieden werden zwischen den Informationen, die der Lehrende liefert, und denen, die die Einreichung des Lernenden enthalten muss. In den meisten Fällen beinhaltet die Einreichung des Lernenden die Lösung zu einer Aufgabe in Form einer oder mehrerer Text-/Quellcode-Dateien.

Der Lehrende muss die zum Testen notwendigen Informationen zur Verfügung stellen. Für einen Vergleich mit einer Musterlösung sind beispielsweise Testdaten und eine Implementierung der Musterlösung notwendig. In diesem Fall muss der Lehrende auch festlegen, ob die Ergebnisse der Einreichung identisch mit der der Musterlösung sein sollen oder ob z. B. Permutationen im Falle eines listenwertigen Ergebnisses erlaubt sind.

Neben den Eingabedaten ist die *Ausgabe* des Backends genau zu definieren, da dieser Wert als Rückmeldung an den Lernenden gegeben wird. Wurden alle Tests erfolgreich durchlaufen, sollte dies eine positive Meldung sein. Im Fehlerfall muss es eine detaillierte Rückmeldung über die Art des Fehlers geben, so dass der Einreichende auch Hinweise darüber erhält, was an seiner Lösung falsch ist. So sollten Meldungen des Compilers/Interpreters über Syntax- oder Laufzeitfehler ebenso wie Informationen über fehlgeschlagene Tests, Testdaten und erwartete Ergebnisse an den Lernenden weitergegeben werden.

Compiler und Interpreter Abhängig von der Wahl des konkreten Compilers und/oder Interpreters für eine Programmiersprache ergeben sich bestimmte Voraussetzungen und Limitierungen. Für die Rückmeldungen des Backends sind insbesondere Art und Informationsgehalt der Rückgabewerte eines Compiler/Interpreter zu untersuchen. Die Existenz

von zuverlässig auswertbaren Fehlermeldungen sind für den Testlauf ebenso wichtig wie Informationen über mögliche vordefinierte Funktionen des Interpreters.

Sicherheit Insbesondere bei dynamischen Tests müssen aufgrund der Tatsache, dass unbekannter und somit potentiell gefährlicher Quellcode ausgeführt wird, verschiedene Sicherheitsaspekte berücksichtigt werden. Die Einreichungen können mit den Rechten des Backend-Nutzers grundsätzlich beliebige Funktionen oder Programme ausführen, Denial-of-Service-Attacken fahren oder Wissen über andere Nutzer, das System oder Aufgaben (Ausspähen der Musterlösung) erlangen.

Daraus ergeben sich Sicherheitsanforderungen, die beim späteren Einsatz des Backends und in Abhängigkeit von Programmiersprache und Plattform zu beachten sind. Beispielsweise kann ein eingeschränkter Interpreter benutzt werden, die Ausführung innerhalb einer Sandbox-Umgebung erfolgen oder eine Zusatz-Software wie *Systrace* [Pro03] verwendet werden, die es erlaubt den Zugriff auf Systemfunktionen einzuschränken. Desweiteren kann sich das Setzen eines Zeitlimits für die Überprüfung des eingereichten Programmcodes als sinnvoll erweisen. Wird dieses Zeitlimit überschritten, wird die Ausführung der Einreichung wegen des Verdachts auf eine Endlosrekursion beendet.

4.2 Beispiel:JUnit

Am Beispiel des von uns entwickelten JUnit-Backends soll nachfolgend das in Abschnitt 4.1 beschriebene Vorgehen demonstriert werden.

Programmiersprache und Testmethode Zu einer gegebenen Programmieraufgabe sollen von den Lernenden Lösungen in Java implementiert werden. Die Lösungen müssen zunächst syntaktisch korrekt sein. Die semantische Korrektheit soll automatisch mittels geeigneter Unit-Tests überprüft werden. Ein Unit-Test bezeichnet „ein Stück Quelltext, das ein Entwickler nur zum Testen eines sehr kleinen und klar umrissenen Teiles der Funktionalität geschrieben hat. Üblicherweise führt jeder Unit-Test eine ganz bestimmte Methode in einem ganz bestimmten Kontext aus.“ ([HT04]). Das verwendete Framework ist *JUnit*[7]. Eine Lösung gilt dann als korrekt, wenn alle Tests ohne Fehler durchlaufen wurden. Schlägt ein Test fehl, wird der komplette Testlauf abgebrochen.

Eingabe und Ausgabe Der Lehrende hinterlegt Testdaten in Form von Unit-Tests. Zudem können Importe und Hilfsfunktionen eingegeben werden. Die Signatur der auszuführenden Methode (Name, Typ der Argumente und des Rückgabewerts) wird in der Aufgabenstellung vorgegeben. Der Lernende reicht seine Lösungen in Form von Quellcode ein.

Die Rückmeldung an den Lernenden enthält Informationen darüber, ob die eingereichte Lösung syntaktisch korrekt ist und ob *alle* Unit-Tests fehlerfrei ausgeführt wurden. Sobald ein Test fehlschlägt, wird ein negatives Feedback zurückgegeben. Dabei wird bei einem

[7] `http://www.junit.org/`

Syntaxfehler die Meldung des Java-Compilers und bei einem semantischen Fehler der Rückgabewert des entsprechenden JUnit-Tests ausgegeben. Bei semantischen Fehlern werden dem Lernenden die erwarteten und die tatsächlich erhaltenen Ausgaben angezeigt. Somit hat dieser die Möglichkeit, Fehler in seiner Lösung zu erkennen und diese entsprechend zu überarbeiten.

Compiler und Interpreter Die Syntaxüberprüfung wird durch den Java-Compiler durchgeführt. Mögliche Fehlermeldungen werden gesammelt und dienen als direkte Rückmeldung an den Lernenden. Die hinterlegten Unit-Tests werden durch den Java-Interpreter ausgeführt und auf die eingereichte Lösung angewendet.

Sicherheit Die Sicherheit wird zum einen dadurch gewährleistet, dass das Backend auf einem separaten Server mit eingeschränkten Nutzerrechten ausgeführt wird. Zum anderen werden bestimmte Funktionsaufrufe durch das schon erwähnte *Systrace* unterbunden und die Ausführung eines Programms nach Überschreitung eines Zeitlimits beendet.

5 Einsatz und Erfahrungen in der Lehre

Seit mehreren Semestern stehen Backends für die Programmiersprachen Haskell, Scheme, Erlang, CommonLisp, Prolog, Python und Java zur Verfügung. Zu den Anwendern zählt neben der Otto-von-Guericke-Universtität Magdeburg (OvGU) beispielsweise das Institut für Informatik der Universität Rostock, wo die Backends für Java und Haskell eingesetzt werden (vgl. [AFR08]). Darüber hinaus wurde am Institut für Informatik der LMU München ein Backend für Standard ML (SML)[8] entwickelt und in der Vorlesung „Programmierung und Modellierung“ eingesetzt.

Zur Vorlesung „Algorithmen und Datenstrukturen“ (AuD I und II), die im Wintersemester 2008/2009 und im Sommersemester 2009 jeweils wöchentlich an der OvGU gehalten wurde, fanden obligatorische begleitende Übungen statt. Diese Lehrveranstaltung ist ein Pflichtfach für alle Studierenden in informatischen Studiengängen (Informatik, Wirtschaftsinformatik, Computervisualistik, Ingenieurinformatik).

Die Zulassung zur Prüfung ist abhängig von der Lösung von Übungsaufgaben – unter anderem Programmieraufgaben in Haskell und Java. Jede Woche wurden in unserem LMS Aufgaben bereitgestellt, die die Studierenden lösen und online einreichen mussten. Für die Einreichung wurde das Modul ECAutoAssessmentBox verwendet und so konfiguriert, dass es ein festgelegtes Zeitfenster sowie eine begrenzte Anzahl an Einreichungsversuchen gab. Alle Einreichungen zu den Programmieraufgaben wurden automatisch überprüft.

Eine Befragung am Ende des Wintersemesters 2008/2009, in der 189 Fragebögen erfasst wurden, hat ergeben, dass 60% der Studierenden diese Art der Einreichung als komfortabel oder sehr komfortabel einschätzen. Die Beobachtung, dass durch die elektronisch vorliegenden Lösungen eine Zeitersparnis in den Übungen erreicht wird, wurde von 68% der

[8] SML ist eine von ML abstammende funktionale Programmiersprache; `http://www.smlnj.org/`

Studierenden als sehr zutreffend (31%) bzw. als zutreffend(37%) bezeichnet. Weiterhin empfanden 71% der Studierenden die Verfügbarkeit aller Übungsaufgaben und Lösungen online und an zentraler Stelle – im Gegensatz zur herkömmlichen Papierlösung – als sehr hilfreich (35%) bzw. hilfreich (36%).

Das System sammelt alle Einreichungen eines Studierenden und bietet in Form einer Statistik einen Überblick darüber, wie viele Aufgaben im laufenden Semester bereits erfolgreich gelöst wurden. Dies bietet zum einen den Lehrenden die Möglichkeit, frühzeitig zu erkennen, welche Übungsteilnehmer Probleme mit den Lerninhalten haben. Zum anderen erlaubt es den Studierenden ihren eigenen Leistungsstand besser einzuschätzen – ein Aspekt, der von den Studierenden als sehr hilfreich angesehen wird. Wie hilfreich zeigte sich zum Wintersemester, als die eduComponents auf eine neue Plone-Version umgestellt wurden und die Statistikfunktion vorübergehend nicht nutzbar war: In der Befragung zu AuD gaben die Studierenden an, dass sie genau diese Möglichkeit zur Selbsteinschätzung vermissten.

Die automatische Überprüfung von Lösungen zu Programmieraufgaben und die zeitnahe Rückmeldung mit Hinweisen auf – im Fehlerfalle – noch nicht korrekt verarbeitete Testdaten wurde von den Studierenden fast einhellig begrüßt und als sehr positiv bewertet. Der mögliche Nutzen im Sinne eines Lernerfolgs (und auch eines motivierenden Erfolgserlebnisses) wird natürlich unterlaufen, wenn Studierende sich nicht selbst mit den Aufgaben auseinandersetzen, sondern lediglich fremde Lösungen einreichen – seien diese nun aus dem reichhaltigen Angebot im Internet verfügbarer Lösungen oder von „hilfsbereiten“ KommilitonInnen übernommen. Wir haben uns allerdings bewusst dazu entschlossen, dem nicht mit systematischer Plagiatserkennung und entsprechenden Sanktionen entgegegenzuwirken. Vielmehr verweisen wir stets auf die selbstschädigenden Effekte von Plagiaten im obigen Sinne und erlauben auch ausdrücklich die Einreichung unvollständiger Lösungen. Für die positive Bewertung einer Einreichung ist es lediglich erforderlich, dass die Studierenden durch ihre eingereichten Ansätze und ggf. Teilergebnisse sowie durch mündliche Erläuterungen in den Übungen belegen konnten, dass sie sich „ausreichend“ intensiv mit der jeweiligen Problemstellung beschäftigt haben.

6 Limitierungen durch das System

Mit Hilfe der eduComponents ist eine effizientere Gestaltung des Übungsbetriebs möglich, jedoch müssen mitunter organisatorische und pädagogische Anpassungen vorgenommen werden. Einreichungen zu Programmieraufgaben, die automatisch testbar sein sollen, müssen bestimmte Bedingungen erfüllen ([Fen08]). So muss der Code einer Lösung beispielsweise eine Funktion mit einem festgelegten Namen, einer bestimmten Signatur sowie den Typ des Rückgabewerts enthalten. Die Vorgaben dazu müssen in der Aufgabenstellung spezifiziert werden. Dieses Vorgehen kann aber in einigen Fällen zu restriktiv sein, da Repräsentationen für das Problem und seine Lösung nicht mehr vom Lernenden gefunden werden müssen, sondern bereits vorgegeben sind. Sinnvoller hinsichtlich des Lösungsvorgangs als Lernprozess wäre hier beispielsweise, den Typ des Rückgabewerts offen zu lassen. Die automatische Prüfung dieser Art von Einreichungen ist jedoch weitaus komplexer. Gleiches gilt für das Testen von interaktiven Programmen.

Nach unseren Erfahrungen gewöhnen sich Lehrende sehr schnell an das automatische Testen von Programmieraufgaben und beginnen damit, den Inhalt ihrer Aufgaben danach auszurichten. Dies birgt jedoch die Gefahr, dass bestimmte Aufgabentypen nicht mehr angewendet werden, da sie nicht automatisch getestet werden können. Deshalb ist es notwendig, die Lernziele nicht aus den Augen zu verlieren und die Eignung bestimmter Aufgaben für diese Lernziele kritisch zu untersuchen. Die Herausforderung für den Lehrenden ist es, solche Aufgabenstellungen zu wählen, die mehr als auswendig gelernte Informationen abfragen und an Stelle dessen die anspruchsvolleren kognitiven Dimensionen in der Bloomschen Taxonomie [BEF+56] wie Anwendung, Analyse und Beurteilung ansprechen.

7 Ausblick

Der erfolgreiche Einsatz von ECAutoAssessmentBox, ECSpooler und Backends in verschiedenen Lehrveranstaltungen führte zu weiteren Wünschen seitens der Lehrenden hinsichtlich der Funktionalität des Systems. Derzeit wird der Einatz der eduComponents für das Wintersemester 2009/2010 vorbereitet. Neben „Algorithmen und Datenstrukturen I“ (AuD I) soll auch die Lehrveranstaltung „ Funktionale Programmierung - fortgeschrittene Konzepte und Anwendungen“ (FP II) mit den eduComponents unterstützt werden. Für FP II ist besonders hilfreich, dass ECAutoAssessmentBox automatisches Testen für unterschiedliche *funktionale* Programmiersprachen bietet. Dies ist besonders wichtig, da nur so die Lernziele der Lehrveranstaltung erreicht werden können, nämlich die Studierenden mit unterschiedlichen Ausprägungen des funktionalen Programmierparadigmas und den dadurch bedingten Programmiermustern – insbesondere auch durch eigenständiges Problemlösen – wirklich vertraut zu machen.

Danksagung

Der auf SOAP basierende Web-Service für ECSpooler (s. Abschnitt 3.1) wurde von den Studierenden Konrad Kügler und Eric Clausing, das in Abschnitt 4.2 vorgestellte JUnit-Backend von den Studierenden Julia Preusse und Christian Baumann jeweils im Rahmen eines Softwarepraktikums implementiert.

Literatur

[AFR08] Mario Amelung, Peter Forbrig und Dietmar Rösner. Towards Generic and Flexible Web Services for E-Assessment. In *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*, Seiten 219–224, New York, NY, USA, 2008. ACM.

[BBF+05] Norbert Bieberstein, Sanjay Bose, Marc Fiammante, Keith Jones und Rawn Shah. *Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap*. IBM Press, 2005.

[BEF+56] Benjamin S. Bloom, Max D. Engelhart, Edward J. Furst, Walker H. Hill und Krathwohl. *Taxonomy Of Educational Objectives: Handbook 1, The Cognitive Domain*. Allyn & Bacon, Boston, 1956.

[EP08] Stephen H. Edwards und Manuel A. Peréz-Quiñones. Web-CAT: Automatically Grading Programming Assignments. In *ITiCSE '08: Proceedings of the 13th annual SIGCSE conference on Innovation and technology in computer science education*, New York, NY. USA, 2008. ACM Press.

[Fen08] Wolfram Fenske. Formen der elektronischen Testaufgabe. Diplomarbeit, Fakultät für Informatik, Otto-von-Guericke-Universität, Magdeburg, 2008.

[Feu06] Thomas Feustel. Analyse von Texteingaben in einem CAA-Werkzeug zur elektronischen Einreichung und Auswertung von Aufgaben. Master's thesis, Fakultät für Informatik, Otto-von-Guericke-Universität, Magdeburg, 2006.

[GSW08] Olly Gotel, Christelle Scharff und Andrew Wildenberg. Teaching software quality assurance by encouraging student contributions to an open source web-based system for the assessment of programming assignments. In *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*, Seiten 214–218, New York, NY, USA, 2008. ACM.

[HBN+08] Andreas Hoffmann, Markus Bode, Marco Nichau, Michael Garbas, Christoph Hellweg, Alexander Quast und Roland Wismüller. DUESIE - Ein Online-Übungssystem zur Informatik Ausbildung. Demo-Session DeLFI, 2008.

[Hel07] Michael T. Helmick. Interface-based programming assignments and automatic grading of java programs. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, Seiten 63–67, New York, NY, USA, 2007. ACM.

[HGST05] Colin A. Higgins, Geoffrey Gray, Pavlos Symeonidis und Athanasios Tsintsifas. Automated Assessment and Experiences of Teaching Programming. *J. Educ. Resour. Comput.*, 5(3):5, 2005.

[HT04] Andrew Hunt und David Thomas. *Pragmatisch programmieren - Unit-Tests mit JUnit*. Hanser Fachbuchverlag, 2004.

[KSZ02] Jens Krinke, Maximilian Störzer und Andreas Zeller. Web-basierte Programmierpraktika mit Praktomat. In *Proceedings des Workshop Neue Medien in der Informatik-Lehre*, Dortmund, Oktober 2002.

[Lig02] Peter Liggesmeyer. *Software-Qualität - Testen, Analysieren und Verifizieren von Software*. Spektrum Akademischer Verlag, Heidelberg/Berlin, 2002.

[LSKM04] Mikko Laakso, Tapio Salakoski, Ari Korhonen und Lauri Malmi. Automatic Assessment of Exercises for Algorithms and Data Structures – A Case Study with TRAKLA2. In *Proceedings of the 4th Finnish/Baltic Sea Conference on Computer Science Education, October 1-3, 2004, Koli, Finland*, Seiten 28–36, 2004.

[Pio09] Michael Piotrowski. *Document-Oriented E-Learning Components*. Dissertation, Otto-von-Guericke-Universität Magdeburg, Magdeburg, 2009.

[Pro03] Niels Provos. Improving Host Security with System Call Policies. In *Proceeding of the 12th USENIX Security Symposium, Washington, DC*, August 2003.

[RPA07] Dietmar Rösner, Michael Piotrowski und Mario Amelung. A Sustainable Learning Environment based on an Open Source Content Management System. In Wilhelm Bühler, Hrsg., *Proceedings of the German e-Science Conference (GES 2007)*. Max-Planck-Gesellschaft, 2007.

[SMK01] Riku Saikkonen, Lauri Malmi und Ari Korhonen. Fully automatic assessment of programming exercises. In *ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education*, Seiten 133–136. ACM Press, 2001.