aus

Oliver J. Bott, Peter Fricke, Uta Priss, Michael Striewe (Hrsg.)

Automatisierte Bewertung in der Programmierausbildung

Digitale Medien in der Hochschullehre Band 6
2017, 420 Seiten, br., 42,90 €, ISBN 978-3-8309-3606-0

Waxmann Verlag GmbH
www.waxmann.com info@waxmann.com

21 Integration automatisierter Programmbewertung in ILIAS

Thomas Richter

Zusammenfassung
Elektronische Programmierübungen werden oft in dezidierten Entwicklungsumgebungen realisiert und sind daher selten in ein eLearning Gesamtkonzept integriert, welches Programmieraufgaben mit traditionellen Elementen von elektronischen Prüfungen wie Lückentexten, Multiple Choice oder Freitextaufgaben kombinieren kann. In diesem Kapitel wird eine ILIAS-Erweiterung diskutiert, welche die Durchführung einfacher Programmieraufgaben in mannigfaltigen Sprachen erlaubt, neben textuellen auch graphische Ausgabemöglichkeiten bietet und ebenso automatische Auswertung von Lösungen erlaubt. Neben der Softwarearchitektur wird ebenso auf die historische Entwicklung des Systems eingegangen und über die bisherigen Erfahrungen und geplante Entwicklungen berichtet.

21.1 Einführung und Hintergrund

Die Vermittlung elementarer Kenntnisse numerischer Mathematik sind schon seit geraumer Zeit Teil der Ingenieurausbildung. Je nach Studienfach erlernen hier die Studierenden numerische Algorithmen mit Hilfe von Computer-Algebra-Systemen (CAS) wie MATLAB oder implementieren die Algorithmen in höheren Programmiersprachen wie C++. Anders als im Studium der Informatik ist dabei die Kenntnis von Softwareentwicklungssystemen oder das Management größerer Softwareprojekte wenig relevant; ein niedrigschwelliger Einstieg in elementare Programmstrukturen und Algorithmen steht hier im Zentrum der Bedürfnisse.

Ein an der Universität Stuttgart nicht unübliches Vorgehen bestand nun darin, die Studierenden dazu aufzufordern, die notwendige Software – also etwa MATLAB oder ein C++ Compiler – herunterzuladen und auf dem eigenen Laptop oder dem heimischen PC zwecks der Durchführung der Übungsaufgaben zu installie-

ren. Obwohl die Universität Stuttgart über Rechnerpools verfügt, sind die Kapazitäten hier zu begrenzt, um allen Studierenden genügend Rechenzeit einräumen zu können; Übungsaufgaben wurden also typischerweise am eigenen Rechner durchgeführt und Lösungen manuell per E-Mail an das Lehrpersonal verschickt und dort korrigiert. Obwohl die Installation von Software auf dem heimischen Rechner eigentlich keine technische Hürde darstellen sollte, entstanden doch immer wieder erhebliche Verzögerungen durch Installationsprobleme, deren Lösung durch das Lehrpersonal viel Zeit in Anspruch nahm und zu Verzögerungen im Lehrplan führten.

Klausuren wurden und werden immer noch in Papierform durchgeführt, d. h. Programmbeispiele müssen mit Bleistift und Papier entwickelt werden, ohne dass die Möglichkeit besteht, diese ausprobieren zu können. Triviale Fehler wie fehlende Semikolons oder falsche Klammerung von Ausdrücken lassen sich so nur schwerlich verhindern, und die Klausurkorrektur ist zeitaufwendig und fehleranfällig.

Nicht zuletzt durch Studierende entstand eine Initiative, die Prozesse der manuellen Installation und manuellen Korrektur durch eine webbasierte Lösung zu ersetzen, die keinen Installationsaufwand erfordert und die eine sofortige Rückmeldung des Compilers oder CAS erlaubt. Dies war der Start des ViPLab-Projektes – kurz für Virtuelles Programmierlabor.

Anders als viele andere in diesem Buch besprochene Projekte sollte ViPLab von Anfang an ein von der Wahl der Programmiersprache unabhängiges System werden, die Anwenderwünsche reichten von MATLAB über C++ und C und Java bis zu universitären Eigenentwicklungen wie dem Numerikwerkzeug DuMux.

Die Entwicklungsziele von ViPLab beinhalteten dabei jedoch nicht, gängige Werkzeuge zur Softwareentwicklung – wie etwa Eclipse oder Visual Studio – zu ersetzen oder Kenntnisse im Management großer Softwareprojekte zu vermitteln. Die von Studierenden der Ingenieurwissenschaften zu erstellenden Programme sind oft kurz und umfassen selten mehr als eine Quelldatei. ViPLab ist darum nicht als Ersatz für ein graphisches Entwicklungssystem (IDE) gedacht.

Stattdessen soll ViPLab vor allen Dingen von Studierenden sofort vollumfänglich benutzbar sein und nur durch die Installation eines Webbrowsers nutzbar sein. Installation von zusätzlicher Software erzeugt eine Barriere, die es zu vermeiden galt. Zu guter Letzt sollte ViPLab sich in das eLearning-System der Universität eingliedern, welches weiterhin die zentrale Anlaufstelle für Übungsaufgaben, Übungen und Tests bleiben sollte.

21.2 Historische Entwicklung

ViPLab entstand als ein durch Studiengebühren finanziertes Projekt mit Zustimmung der Vertretung der Studierenden und wurde getragen vom Rechenzentrum der Universität Stuttgart (damals RUS, heute TIK), dem Institut für Wasser- und Umweltsystemmodellierung, dem Institut für angewandte Analysis und Numerische Simulation und dem Institut für Aero- und Gasdynamik als Projektleiter. Als externer Projektpartner liefert FreeIT eine Middleware zur Kommunikation zwischen den technischen Systemkomponenten, siehe auch Abschnitt 21.5 für weitere Details zum technischen Aufbau des Gesamtsystems.

Die erste Version von ViPLab basierte noch auf einem Java-basierten Plugin, welches als Teil des ViPLab-Benutzerinterfaces vom eLearning-System ausgeliefert wurde. Dieser Ansatz erlaubte zwar, die ViPLab-Benutzerschnittstelle als SCORM-Modul [Lea] zu realisieren und somit eine vom Learning-Management-System (LMS) unabhängige Lösung zu ermöglichen, die Betriebserfahrung zeigte jedoch, dass die Architektur diverse Einschränkungen mit sich brachte. Einerseits musste seitens der Studierenden eine Systemkomponente installiert werden – nämlich das Java-Plugin für den verwendeten Browser. Nicht zuletzt aufgrund mangelhafter Pflege seitens des Herstellers Oracle bei der Behebung auftretender Sicherheitslücken wurden Java-Applets unpopulär, und die Installation eines Plugins provoziert erneut Fehler seitens der Anwender, deren Vermeidung eines der Ziele des Projektes war.

Andererseits sind die technischen Möglichkeiten, über ein SCORM-Modul Einfluss auf ein Learning-Management-System zu nehmen, recht begrenzt. So ist in SCORM beispielsweise nicht vorgesehen, Berechnungsergebnisse zwecks manueller Nachkorrektur im Lernsystem zu hinterlegen; stattdessen beschränkt sich der Datensatz auf die Übermittlung einer bereits vom Plugin zu ermittelnden Punktzahl. Auch die Aufgabenerstellung durch das Lehrpersonal wurde häufig als zu komplex und unhandlich kritisiert, da Aufgaben außerhalb des eigentlichen Lernmanagementsystems erstellt und dann manuell in dieses hochgeladen werden mussten.

Viele dieser Kritikpunkte wurden in der zweiten Version von ViPLab beseitigt. Statt eines generischen SCORM-Plugins basiert die Version 2.0 auf einem proprietären Plugin für das ILIAS-System [Ili], des an der Universität Stuttgart verwendeten Lernmanagementsystems, welches sich jedoch tiefer und damit ohne weitere Barrieren in das System integriert.

Die Benutzerschnittstelle wird vollständig als Javascript an die Studierenden ausgeliefert und erfordert somit kein Browserplugin mehr. Durch die tiefere Integration in das Gesamtsystem werden die studentischen Lösungen im System

hinterlegt und können dort von dem Lehrpersonal zur Nachkorrektur eingesehen werden.

21.3 Die Benutzeroberfläche

Bild 21.1 zeigt eine ViPLab-Aufgabe aus Sicht eines Studierenden. Der zu bearbeitende Quellcode ist dabei zentral in der Mitte angeordnet und kann mit einem Editor bearbeitet werden. Der grau hinterlegte obere Teil ist dabei vom Dozenten als Teil der Aufgabenstellung vorgegeben und kann nicht verändert werden.

Über den „Berechnung starten“-Knopf unten wird der Code compiliert und ausgeführt, wobei dies auf dezidierten Servern des Rechenzentrums und nicht auf dem studentischen PC erfolgt. Eine Ausgabe in Textform oder als Plot erscheint rechts vom Quellcode.

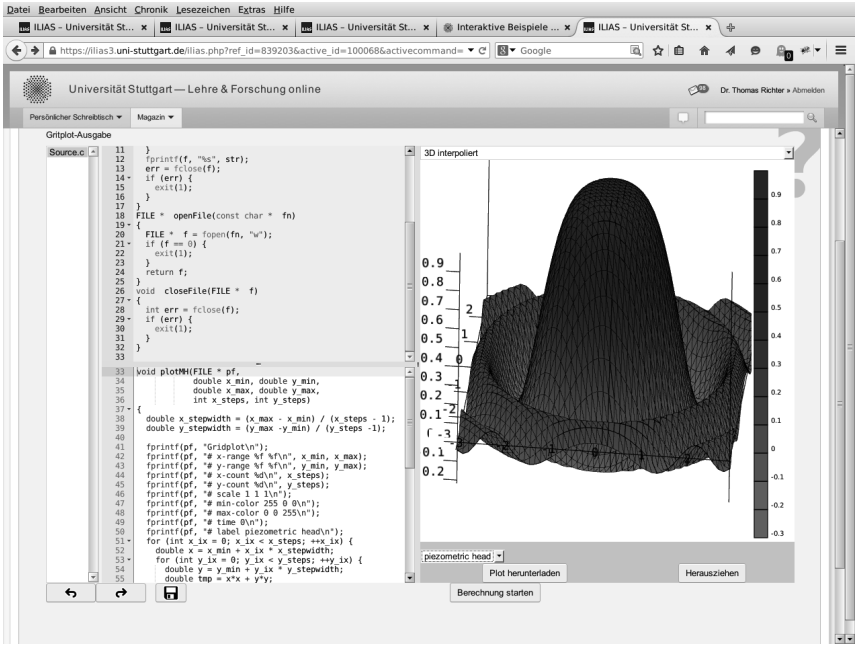

Abbildung 21.1: Eine typische ViPLab-Aufgabe aus Sicht eines Studierenden. Ganz links eine Leiste zur Auswahl des zu bearbeitenden Quellcodes, rechts daneben der Programmcode. Ganz rechts das Resultat der Berechnung, hier als Plot.

Die Visualisierung von Berechnungsergebnissen war dabei ein von den Vertretern der Studierenden schon früh angefragtes Feature von ViPLab; die graphische Ausgabe ist darüber hinaus nicht statisch, sondern erlaubt die Skalierung, Rotation und Ausrichtung des Plots über die Maus, ähnlich den nativen Plotfunktionen von MATLAB. Die graphische Ausgabe steht allen Programmiersprachen und nicht nur MATLAB zur Verfügung.

Das Benutzerfrontend für Lehrende entspricht weitgehend der Schnittstelle für Studierende, siehe Bild 21.2. Eine Aufgabe besteht dabei aus mehreren Quelldateien, die über ein Pop-Up-Menü (nicht im Bild) über der linken Auswahlleiste erzeugt oder gelöscht werden können. Eine Quelldatei wiederum besteht aus einem oder mehreren Abschnitten, hier zentral im Bild, die sichtbar, veränderbar, konstant oder unsichtbar sein können. Mehr zum Aufgabenmodell von ViPLab in Abschnitt 21.6.

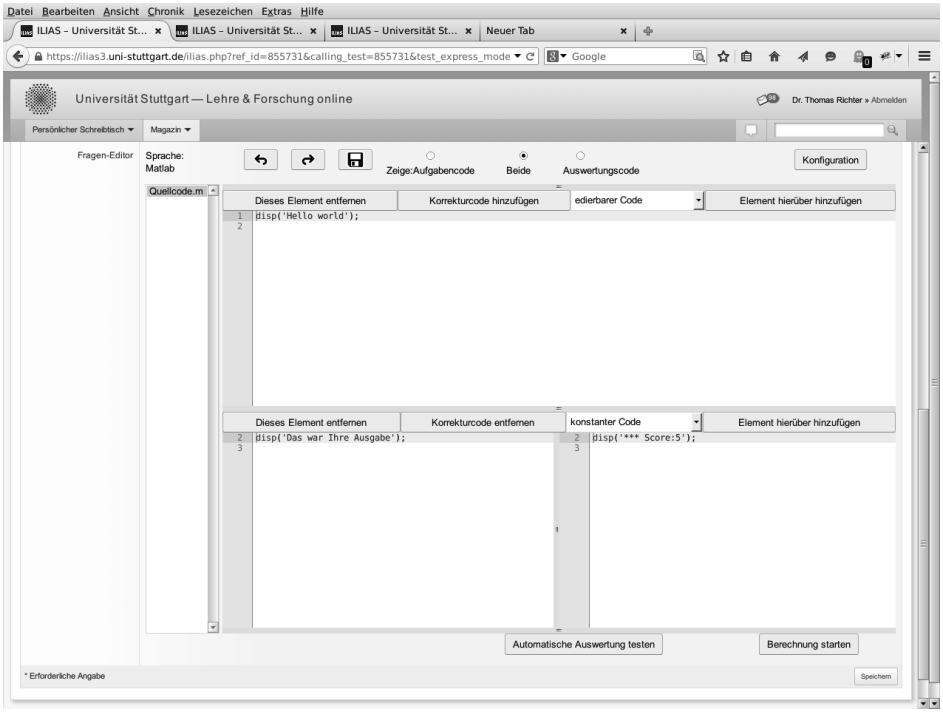

Abbildung 21.2: Die Entwicklung einer ViPLab-Aufgabe aus Dozentensicht. Eine Quellcodedatei ist hierbei in mehrere Abschnitte unterteilt, die veränderbar, konstant oder unsichtbar sein können.

Zur Aufgabenkorrektur können einer oder mehrere der Abschnitte durch einen Korrekturcode ersetzt werden. Details zur automatischen Aufgabenkorrektur sind in Abschnitt 21.7 beschrieben.

21.4 Automatische Korrektur studentischer Lösungen

In der VipLab-Version 1.0 erfolgte die Korrektur von Übungsaufgaben vollständig manuell: Studierende erarbeiteten ihre Lösungen am eigenen Laptop oder dem heimischen PC und verschickten die Lösungen am LMS vorbei an das Lehrpersonal, welches dann die Abgaben manuell bewerteten und die Punktzahl in das LMS eintrugen. Eine derartige Lösung skaliert nicht gut auf große Teilnehmerzahlen, so dass schon von Anfang an eines der Entwicklungsziele von ViPLab die automatische Korrektur von Übungsaufgaben war.

Mit der Version 2.0 entstand zunächst eine halbautomatische Lösung: Das System kann hierbei selbstständig eine Punktzahl ermitteln, jedoch muss diese Ermittlung manuell von einem Dozenten angestoßen werden. Dem Dozenten obliegt es, die Bewertung des Systems zu überprüfen und ggf. eine Nachbewertung durchzuführen. Dieses Vorgehen war einerseits durch die Systemarchitektur bedingt (siehe Abschnitt 21.5), andererseits herrschte seitens der ViPLab-Entwickler auch eine gewisse Skepsis gegenüber vollständig automatischen Bewertern und deren Einschätzungen von menschlichen Fehlern. Ein Syntaxfehler mag aus menschlicher Sicht zwar trivial sein, verhindert aber die maschinelle Ausführung eines Lösungsansatzes und somit die maschinelle Bewertung einer Abgabe. Andererseits sind derartige Fehler auch vom Studierenden durch Testen einer Lösung trivial zu finden, und so mehrten sich Stimmen von Anwendern, eine vollständig automatische Auswertung zuzulassen, die mit ViPLab 2.1 umgesetzt wurde.

Der in ViPLab verwirklichte Ansatz zur automatischen oder halbautomatischen Aufgabenbewertung unterscheidet sich dabei radikal von den typischerweise verfolgten Strategien, die auf sprachspezifischen Mechanismen aufbauen. Ein oft verfolgter Ansatz ist dabei die Anwendung von Unit-Tests, etwa von JUnit für Java. Aufgrund der angestrebten Sprachunabhängigkeit von ViPLab war diese Möglichkeit nicht gegeben. Genaueres über die Technik zur automatischen Auswertung findet sich in Abschnitt 21.7.

Das Fernziel von ViPLab ist auch, elektronische Klausuren durchführen zu können, die neben den üblichen Multiple- oder Single-Choice-Aufgaben ebenso Programmieraufgaben enthalten. Näheres zu den Konzepten für elektronische Klausuren ist in Abschnitt 21.9 beschrieben.

21.5 Softwarearchitektur

Die ViPLab-Softwarearchitektur besteht aus vier Schichten, siehe Bild 21.3: Zuoberst die Benutzerschnittstelle (siehe auch Bild 21.1), welche über Javascript im Browser des Nutzers abläuft. Die Benutzerschnittstelle wird dabei über `http` vom ILIAS System (dem LMS) ausgeliefert. Benutzercode wird auf den Backends (rechts in Bild 21.3) compiliert und zur Ausführung gebracht. Hierbei wird pro Aufgabe eine „Sandbox“ erzeugt, in der der studentische Code vom restlichen Betriebssystem abgeschirmt ablaufen kann.

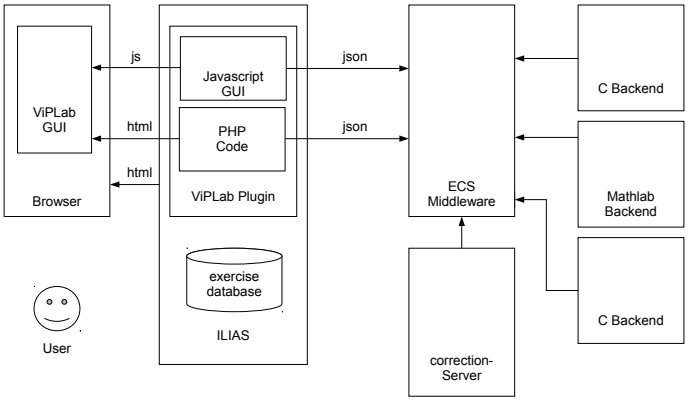

Abbildung 21.3: Gesamtarchitektur von ViPLab. Links die Benutzeroberfläche im Browser des Lernenden, in der Mitte ILIAS und die ECS-Middleware. Rechts die Backend-Server, die die eigentlichen Berechnungen ausführen.

Vermittelnd zwischen den beiden steht die ECS-Middleware, die Warteschlangen für Backends und Frontends zur Verfügung stellt und die Last zwischen den Backends verteilt. Eine Skalierung des Systems ist somit immer leicht durch den Anschluss weiterer Backends möglich.

Ebenfalls an der Middleware dockt der Korrekturserver an, der die Bewertung von studentischen Lösungen koordiniert und hierfür über die Middleware mit dem ECS und dem ILIAS kommuniziert.

Die Kommunikation zwischen den Systemkomponenten erfolgt dabei in JSON [Bra], der nativen Systemsprache von Javascript. Die Wahl fiel hierbei auf JSON, weil es deutlich einfacher zu verarbeiten ist als XML und für die Zwecke von ViPLab komplett ausreicht. In JSON werden sowohl Aufgaben, als auch studentische Lösungen und Berechnungsresultate codiert.

Möchte ein Studierender eine ViPLab Aufgabe bearbeiten, so legt im ersten Schritt das ILIAS-Plugin den Quellcode der Aufgabe auf der ECS-Middleware ab. Daraufhin liefert ILIAS die graphische Oberfläche an den Browser des Studierenden aus und übergibt ihr als Parameter die URL der Aufgabe auf dem ECS. Der im Browser ablaufende Javascript-Code zieht seinerseits wiederum die zu bearbeitende Aufgabe vom ECS; diese Aufgabe enthält den Korrekturcode nicht, es besteht also keine Gefahr, dass der Nutzer durch Beobachtung des Netzwerkverkehrs Hinweise auf eine korrekte Lösung bekommt.

Nach Bearbeitung der Aufgabe codiert das Javascript-Frontend im Browser den veränderten Codeabschnitt zurück an den ECS, der hierdurch wiederum ein Ereignis auf den Backends auflöst. Diese studentische Lösung enthält ihrerseits die URL der Aufgabe auf dem ECS. Ein Studierender kann also nicht durch Manipulation des Netzwerkverkehrs nachträglich den Aufgabentext verändern und so Punkte erschleichen.

Das Backend zieht die Lösung – und über den Verweis auf die Aufgabe auch den Quelltext – des Code-Templates aus dem Aufgabentext, kombiniert beides, und compiliert den hieraus erzeugten Quelltext. Der studentische Code wird dann innerhalb einer Sandbox, d. h. eines vom restlichen System getrennten Bereiches, zur Ausführung gebracht. Die Ausgaben der Lösung werden in JSON codiert und zurück auf den ECS geschickt, der nun seinerseits ein Ereignis auf dem Frontend auslöst. Das Frontend holt sich die Lösung, dekodiert sie und zeigt sie an.

Ähnlich funktioniert die Erstellung von Aufgaben, wobei das ILIAS Rechtemanagement die Aufgabendatenbank vor unberechtigtem Zugriff absichert. Die automatische Korrektur wird im Abschnitt 21.7 separat beschrieben.

21.6 Aufbau von ViPLab-Aufgaben

Eine ViPLab-Aufgabe besteht aus einer oder mehreren Übersetzungseinheiten, von der jede wiederum aus einem oder mehreren Abschnitten besteht, siehe hierzu Bild 21.4.

Eine Übersetzungseinheit kann man grob mit einer Quelldatei einer lokalen Compiler-Installation vergleichen; abhängig von der Wahl der Programmiersprache klassifiziert ViPLab diese Quelldateien auch nach Typ und unterscheidet Quellcode, Header und Datendatei. Letztere enthalten dabei vom studentischen Code zu bearbeitende Rohdaten.

Die Strukturierung einer Übersetzungseinheit in mehrere Abschnitte erlaubt, invariante Codeteile aus didaktischen Gründen vor dem Überschreiben durch Stu-

dierende zu schützen. Dies könnte etwa das Hauptprogramm sein, welches die Ansteuerung eines vom Studierenden zu erzeugenden Codeteils übernimmt.

Des Weiteren können ganze Codeteile verborgen werden; sie werden dann vom Frontend nicht dargestellt. Typischerweise werden hier Bibliotheksfunktionen ausgeblendet, die als solches nur verwirren würden und keinen didaktischen Wert haben. So enthält etwa die Beispielaufgabe aus Bild 21.1 einen solchen Bibliothekscode, der die graphische Ausgabe statt auf den (nicht existenten) Bildschirm des Backends in eine Datei umlenkt und durch das Frontend visualisieren lässt.

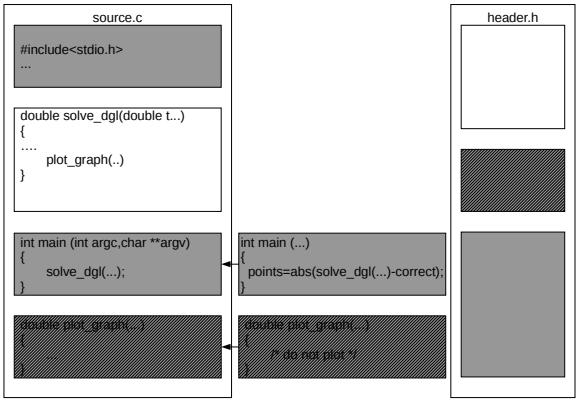

Abbildung 21.4: Der Aufbau einer ViPLab-Aufgabe: Sie besteht aus mehreren Übersetzungseinheiten, die sich wiederum in mehrere Abschnitte gliedern.

21.7 Automatische und halbautomatische Korrektur von Aufgaben

Anders als viele andere Zugänge zur automatischen Korrektur von Aufgaben ist der ViPLab-Ansatz vergleichsweise einfach; das Entwicklungsziel war eine vollständige Sprachunabhängigkeit, wie man sie durch die Nutzung von Frameworks wie JUnit nicht erreichen kann. Die Komplexität dieser Frameworks gefährdete in der Sicht des Entwicklungsteams auch die Akzeptanz des Gesamtsystems seitens der Lehrenden, so dass stattdessen ein möglichst einfacher und verständlicher Ansatz gewählt wurde. Die Korrektur von Aufgaben erfolgt in ViPLab einzig durch *Substitution von Codeabschnitten*.

Hierbei können konstante oder unsichtbare Codeabschnitte (siehe Kapitel 21.6, Bild 21.4) zum Zwecke der Korrektur durch anderen Code ersetzt werden, der über den Aufruf der studentischen Lösung eine Punktzahl ermittelt. Ein solcher Korrekturcode würde etwa ein vom Dozenten erstelltes Hauptprogramm ersetzen und einen studentischen Lösungsansatz mit Testdaten füttern, und die Rückgabewerte der Lösung auf Gleichheit mit einer Musterlösung überprüfen. Eine ausgefeiltere Strategie würde zufällige Eingabeparameter für die studentische Lösung erzeugen und parallel mit der abgegebenen Lösung und einem Musteralgorithmus Berechnungen durchführen. Stimmt die abgegebene Lösung mit der Musterlösung innerhalb einer gewissen Toleranz überein, so ist die Lösung korrekt. Schlägt die Auswertung in bestimmten Ausnahmesituationen fehl, so kann die Bepunktung entsprechend geringer ausfallen.

Im Falle einer halbautomatischen Korrektur erfolgt die Ersetzung der Codeteile durch das Frontend des Dozenten innerhalb von ILIAS, muss dann aber manuell angestoßen werden. Der Korrekturserver (siehe Bild 21.3) wird dann nicht verwendet.

Für den Einsatz in großen Vorlesungen oder elektronischen Klausuren (siehe Abschnitt 21.9) ist die halbautomatische Lösung allerdings immer noch zu aufwendig, da das Lehrpersonal jede Lösung einzeln einsehen muss. Deshalb erreichten uns nach der Veröffentlichung von ViPLab 2.0 sehr rasch Anwenderwünsche für eine vollautomatische Korrektur.

Wird die automatische Korrektur eingeschaltet, so wird bei der Abgabe einer studentischen Lösung diese genauso wie bei einer manuellen Auswertung zunächst auf dem ECS abgelegt; vergleiche hierzu den Vorgang bei der Berechnung einer Aufgabe in Abschnitt 21.5. Gemeinsam mit der Aufgabe und der studentischen Lösung legt das ILIAS-Plugin den Korrekturcode auf den ECS. Hierdurch wird am Korrekturserver (unten Mitte in Bild 21.3) ausgelöst, der nun statt des Backends die Lösung, die Aufgabe und den Korrekturcode vom ECS abholt. Dort erfolgt nun – statt im Dozenten-Frontend – die Ersetzung von Codeteilen der Aufgabe durch den Korrekturcode. Der Korrekturserver spielt über den ECS den Rechenjob an die Backends, die die Ergebnisse der Berechnung an den Korrekturserver zurück liefern und der nun seinerseits die Punktzahl aus dem Ergebnis extrahiert. Die Punktzahl wird über ein Ereignis an den ILIAS zurückgeliefert, der diese in seine Datenbank einträgt.

Eine detailliertere Analyse des studentischen Codes findet momentan nicht statt; so wäre durchaus der Einsatz von Codemetriken sinnvoll, oder von statischen Codeanalysewerkzeugen wie Lint zur Bewertung des Programmierstils. Derartige Funktionalitäten ließen sich als Teil des Korrekturservers implementieren. Ebenso würde die Systemarchitektur prinzipiell ermöglichen, durch den Korrekturcode

Aufrufe auf Unit-Test-Frameworks in die abgegebene Lösung zu injizieren und somit auch eine Prüfung des Codes durch Unit-Tests durchzuführen.

Alle diese Ansätze sind jedoch im Augenblick noch nicht im Einsatz; momentan beschränkt sich die Fähigkeit des Systems auf die oben eingeführte einfache Codesubstitution zur Bewertung einer Lösung.

21.8 Erfahrungen und Evaluation von ViPLab

ViPLab ist seit ca. 2010 an der Universität Stuttgart im Einsatz; neben einigen anfänglichen technischen Schwierigkeiten läuft der Betrieb aus der Sicht des Rechenzentrums reibungslos. Um nun auch Daten über die Nützlichkeit von ViPLab aus Anwendersicht zu sammeln, führte das Rechenzentrum im Jahr 2013 eine Vergleichsstudie [Van+13] durch, bei der Übungen mit ViPLab klassischen computergestützten Übungen gegenüber gestellt wurden; hierbei bearbeiteten Studierende an einem klassischen Editor mit vom Lehrpersonal bereitgestellten Codeabschnitten und einem lokal installierten Compiler oder mit ViPLab ihre Übungsaufgaben. Der Inhalt der Übungsaufgaben bestand aus dem Erstellen eines kurzen C++ Programms aus vorgegebenen Codeabschnitten. Die Übungen fanden bei ansonsten identischer Ausstattung im gleichen Computerpool der Universität statt. Das verwendete Betriebssystem im Pool war Debian-Linux, das Lehrpersonal war in beiden Gruppen ebenfalls identisch. Die Gruppengröße der ViPLab-Nutzer war 22, die der Kontrollgruppe mit einer klassischen Installation bestand aus 35 Studierenden.

Das Ziel dieser Vergleichsstudie war, festzustellen, ob zwischen den beiden Arbeitsumgebungen – lokale Installation, Compiler und Editor oder ViPLab – ein statistisch signifikanter Unterschied in Bezug auf die Zufriedenheit der Nutzer besteht, und inwieweit derartige Unterschiede mit der Computeraffinität der Studierenden korreliert.

Das Ziel der Untersuchung war es herauszufinden, wie sich der Einsatz von ViPLab auf die Zufriedenheit der Studierenden auswirkt. Dabei sollten zum einen das Vorwissen und die Vorkenntnisse der Studierenden erhoben werden um auf diesen Faktor zu kontrollieren. Die eigenen Selbtwirksamkeitsüberzeugungen der Studierenden im Bezug auf die Computerbenutzung sollten ebenfalls ermittelt werden.

Die Vorkenntnisse und die Selbtwirksamkeitsüberzeugungen wurden mit einer angepassten INCOBI-R Skala [RNG01; RNH10] erhoben. Die Zufriedenheit in Bezug auf das verwendete System mit einer Skala vermessen, die auf Gruber [Gru+10] basiert. INCOBI-R untersucht folgende Aspekte mittels eines Fra-

genkatalogs auf drei Skalen: Die praktische Computererfahrung (PRAECOWI), theoretisches Wissen (TECOWI), und das Selbstvertrauen der Nutzer im Umgang mit Rechnern (COMA). Zusätzlich wurden die Studierenden nach ihrem bislang verwendeten Betriebssystem auf ihren eigenen Rechnern befragt.

Das Resultat der Studie zeigte, dass zwischen der Zufriedenheit mit der lokalen Compilerinstallation und der Nutzung von ViPLab kein statistisch signifikanter Unterschied besteht. Ferner sollte die Studie klären, ob zwischen der Technikaffinität der Studierenden und der Zufriedenheit mit den zwei Lösungen ein Unterschied besteht. Hier konnte nur in einem Fall eine statistisch signifikante negative Korrelation nachgewiesen werden, nämlich zwischen Nutzern des Windows-Betriebssystems und der Zufriedenheit mit der lokalen Installation. Es zeigte sich, dass Windows-Nutzer mit der Unix Oberfläche im Computerpool weniger zufrieden waren als mit ViPLab. Bei Nutzern anderer Betriebssysteme (Apple und Linux) waren derartige Korrelationen statistisch nicht signifikant.

Das Resultat zeigt, dass ViPLab genauso gut angenommen wird wie eine klassische gut vorbereitete Installation auf lokalen Rechnern, wobei die Vorbereitungszeit für das Erstellen der Arbeitsumgebung durch die Dozenten und das Lehrpersonal für eine lokale Installation erheblich höher ist. Zusammenfassend kann man sagen, dass ViPLab von den Studierenden ebenso gut wie eine lokale Installation angenommen wird, aber die Arbeitsleistung beim Lehrpersonal erheblich reduziert.

21.9 Ausblick: Elektronische Klausuren

Ein möglicher zukünftiger Einsatz von ViPLab liegt in der Durchführung elektronischer Klausuren. ViPLab würde dabei ein Hauptproblem der bisherigen Papierklausuren lösen, nämlich dass Studierenden die Möglichkeit gegeben wird, Programmcode und Zwischenergebnisse direkt am Rechner zu validieren. ViPLab-Aufgaben würden hierbei mit klassischen Elementen elektronischer Prüfungen verknüpft werden, nämlich Multiple- und Single-Choice-Aufgaben, Freitext- und Formelfragen.

Leider verfügt die Universität Stuttgart über keinen geeignet großen Computerpool, um typische Bachelorstudiengänge aufnehmen zu können; hierbei sind Teilnehmerzahlen von bis zu 2000 Studierenden keine Seltenheit. Der größte verfügbare Pool verfügt hingegen nur über knapp 70 Plätze.

Zur Durchführung elektronischer Klausuren müssen stattdessen gewöhnliche Hörsäle mit mobilen Rechnern ausgestattet werden. Es wurden hierbei zwei mögliche Ansätze durchgespielt: Einerseits könnte man Studierende verpflichten, ei-

gene Geräte zur Klausur mitzubringen und die Klausur in einer geeignet abgesicherten Umgebung auf diesen Geräten durchzuführen. Andererseits könnte eine elektronische Klausur auch auf Rechnern der Universität durchgeführt werden, wobei hier auch je nach Wahl der Lösung ebenso eine abgesicherte Arbeitsumgebung erstellt werden müsste und die Geräte von Personal der Universität gewartet werden muss.

Um die Praktikabilität der ersten Lösung abzuwägen, führte das Rechenzentrum gemeinsam mit dem Institut für Wasser- und Umweltsystemmodellierung eine Probeklausur durch, wobei als gesicherte Umgebung ein selbstbootendes Linux auf Knoppix-Basis [Kno] eingesetzt wurde. Dieses wurde nur insofern verändert, als nach dem Systemstart automatisch ein Browser gestartet wurde, der sich selbsttätig mit dem LMS (ILIAS) der Universität verband. Alle weiteren Programme wurden aus der Installation entfernt.

Die Erfahrungen zeigten, dass dieser Ansatz wenig praktikabel ist: Neben typischen Hardwareproblemen – Knoppix unterstützt die WLAN-Chipsätze der neuesten Notebookgeneration nur unzureichend – gab es auch eine Reihe unerwarteter Probleme mit dem Nutzerinterface von Knoppix. Beispielsweise bootet das System grundsätzlich mit einer deutschen Tastaturbelegung, aber aufgrund einer Vielzahl ausländischer Kursteilnehmer entspricht dies nicht immer der am Gerät vorhandenen Tastatur. Da die problemlose Funktion von Knoppix nicht für jedes Laptopmodell garantiert werden konnte und somit die vom Gesetzgeber geforderte Chancengleichheit aller Prüfungsteilnehmer so nicht sichergestellt werden kann, wurde dieses Modell verworfen.

Die Alternative, Geräte für die Prüfung zu stellen, erfordert jedoch deren regelmäßige Wartung, wie etwa das Einspielen von Updates, durch Personal der Universität und ist deshalb mit wiederkehrenden Folgekosten verbunden. Ein in diese Richtung interessanten Ansatz verfolgt die Universität Utrecht: Statt gewöhnlicher Notebooks werden hier Chromebooks verwendet. Diese Geräte sind nicht nur relativ preisgünstig in der Anschaffung, sondern verfügen von Haus aus über eine abgesicherte Ausführungsumgebung – d. h. der Anwender kann keinerlei Software installieren – und sie können zentral administriert werden. Anders als bei typischen Laptops oder Netbooks läuft hier nur ein vorinstallierter Browser, der aber für die Zwecke elektronischer Prüfungen vollkommen ausreicht. Die Einstellungen aller Geräte, auch der Netzzugang und die für Nutzer erreichbaren Webseiten können zentral, also einmal für alle Geräte administriert werden. Bei der Verwendung des Kiosk-Modus verlassen Nutzerdaten auch nicht das universitäre Netz und die recht strengen Datenschutzrichtlinien in Deutschland werden eingehalten.

Im Zuge dieser Überlegungen beschaffte das Rechenzentrum der Universität im Sommer 2016 einen ersten Satz von 40 Chromebooks; die bisherigen Tests mit Studierenden im Rahmen einer Probeklausur verliefen positiv. Als eine der wenigen gewünschten Verbesserungen wird das Rechenzentrum noch Sichtschutzfilter auf den Chromebooks montieren, um die Gefahr des Abschreibens zu minimieren, und Mäuse als zusätzliche Eingabegeräte anschaffen.

Literatur für dieses Kapitel

[Bra] T. Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. https://tools.ietf.org/html/rfc7159. (besucht am 23.05.2015).

[Gru+10] Thorsten Gruber u. a. „Examining student satisfaction with higher education services: Using a new measurement tool“. In: *International Journal of Public Sector Management* 23.2 (2010), S. 105–123. DOI: 10.1108/09513551011022474.

[Ili] *ILIAS Open Source e-Learning*. http://www.ilias.de. (besucht am 23.05.2016).

[Kno] Klaus Knopper. *KNOPPIX Website*. http://www.knoppix.org/. (besucht am: 23.05.2016).

[Lea] Advanced Distributed Learning. *SCORM Technical Specification*. https://adlnet.gov/wp-content/uploads/2011/07/SCORM_2004_4ED_v1_1_Doc_Suite.zip. (besucht am 23.05.2016).

[RNG01] Tobias Richter, Johannes Naumann und Norbert Groeben. „Das Inventar zur Computerbildung (INCOB): Ein Instrument zur Erfassung von Computer Literacy und computerbezogenen Einstellungen bei Studierenden der Geistes- und Sozialwissenschaften.“ In: *Psychologie in Erziehung und Unterricht* 48.1 (2001), S. 1–13.

[RNH10] Tobias Richter, Johannes Naumann und Holger Horz. „Eine revidierte Fassung des Inventars zur Computerbildung (INCOBI-R).“ In: *Zeitschrift für pädagogische Psychologie* 24.1 (2010), S. 23–37.

[Van+13] J. Vanvinkenroye u. a. „A Quantitative Analysis of a Virtual Programming Lab“. In: *Proc. of Multimedia (ISM), 2013 Intl. Symposium on* (2013), S. 457–461.