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

3 Automatisierte Bewertung in der objektorientierten Programmierausbildung am Beispiel von Java

Michael Striewe und Robert Garmann

Zusammenfassung
Das folgende Kapitel stellt verschiedene Aufgabentypen sowie ein Veranstaltungskonzept für die Einführung in die objektorientierte Programmierung in Java vor. Es basiert auf Erfahrungen an der Universität Duisburg-Essen sowie der Hochschule Hannover, an denen die Einführung in die objektorientierte Programmierung seit mehreren Jahren durch automatische Bewertungssysteme unterstützt wird.

3.1 Einleitung

Das Erlernen der objektorientierten Programmierung stellt Lernende vor mehrere Herausforderungen: Zum einen müssen die verschiedenen Aspekte der Objektorientierung verstanden und verinnerlicht werden und zum anderen muss eine konkrete Programmiersprache erlernt werden, in der diese Konzepte auf eine bestimmte Weise umgesetzt werden. Programmieranfänger, die zuvor keine andere Sprache erlernt haben, müssen zudem grundlegende Programmierfähigkeiten unabhängig von der Objektorientierung erlernen.

Aufgrund dieser Herausforderungen ist es sinnvoll, in der Lehre zur objektorientierten Programmierung im großen Umfang Übungsaufgaben einzusetzen, in denen verschiedene Fähigkeiten in unterschiedlichen Situationen angewandt werden müssen. Die Aufgaben sollten detailliertes Feedback vorsehen, um Fehler auf den verschiedenen Ebenen der Umsetzung zielgerichtet kommentieren zu können: Eine funktional korrekte Lösung kann im Sinne der Aufgabenstellung falsch sein, wenn sie beispielsweise nicht die gewünschte Vererbungsstruktur umsetzt.

Umgekehrt ist eine strukturell korrekte Lösung nicht zwangsläufig auch fehlerfrei lauffähig.

Daraus ergeben sich sowohl Chancen als auch Herausforderungen, denn große Mengen an Lösungen lassen sich in der Regel nur automatisiert zeitnah bewerten und mit Feedback versehen [Iha+10]. Der didaktische Wert hängt jedoch stark von der Qualität des Feedbacks ab. Im folgenden wird auf Basis des Einsatzes von automatischen Systemen zur Programmbewertung an der Universität Duisburg-Essen sowie der Hochschule Hannover ein umfangreiches Beispielszenario beschrieben. Es basiert auf dem Einsatz der Systeme JACK an der Universität Duisburg-Essen sowie GRAJA an der Hochschule Hannover; jeweils in einer Einführungsvorlesung zur objektorientierten Programmierung mit Java.

Java wird ungeachtet anhaltender Diskussionen über dessen didaktische Eignung (s. bspw. [Bös98], [Pea+07]) in vielen Hochschulen als erste Programmiersprache gelehrt. Java vereint Elemente verschiedener Programmierparadigmen, u. a. die der strukturierten, prozeduralen und objektorientierten Programmierung. Das in diesem Kapitel beschriebene didaktische Szenario geht davon aus, dass Studierende Konzepte der strukturierten und der prozeduralen vor denen der objektorientierten Programmierung erlernen (s. bspw. [Reg06]), setzt diese Reihenfolge jedoch nicht voraus. Die von uns vorgestellten Aufgabenformen und Einsatzszenarien sind ebenso bei einer Einführung nach einem Objects-first-Ansatz oder gar einem Components-first-Ansatz anwendbar (s. bspw. [Gri08], [HTW04]).

Obwohl sich selbstverständlich in allen Schritten des Szenarios eine Lösung manuell prüfen und einzeln mit verschiedenen Testfällen ausführen lässt, werden primär spezifische Vorteile der automatischen Bewertung genannt. Mit „Bewertung“ ist dabei jede Art von Analyse und Generierung von Feedback gemeint, die durch ein System durchgeführt wird, unabhängig davon, ob dies im Kontext formativen oder summativen Assessments erfolgt.

3.2 Aufgabenformen

Dieser Abschnitt stellt verschiedene Aufgabenformen vor, die im Kontext der objektorientierten Programmierung mit Java anwendbar sind. Es werden die jeweiligen Vorteile der automatischen Bewertung genannt, ohne auf die Einbettung der Aufgaben in den organisatorischen Kontext einer Veranstaltung einzugehen. Diese erfolgt im darauf folgenden Abschnitt.

Grundsätzlich gilt für alle Aufgabenformen, dass ein automatisches Bewertungssystem einige grundlegende Eigenschaften der Lösung wie Klassenbezeichner oder Methodensignaturen kennen muss, um ein Programm automatisch bewer-

ten zu können. Aufgabenautoren müssen daher bei der Aufgabenerstellung einige Entwurfsentscheidungen vorweg nehmen, um Funktionstests an internen Schnittstellen des zu erstellenden Programms zu ermöglichen. Studierende müssen sich daher bei der Lösung üblicherweise an einige Vorgaben halten. Diese verringern die Freiheitsgrade bei der Programmierung, was eher untypisch für die professionelle Programmierung ist. Hier müssen also inhaltliche Lernziele gegen die Vorteile einer automatischen Bewertung abgewogen werden. Bei Aufgaben, die auf einzelne, isoliert zu lernende Programmierkonzepte bzw. Algorithmen fokussieren, nimmt man Einschränkungen in der Entwurfsfreiheit in der Regel zugunsten umfangreicherer Bewertungsmöglichkeiten in Kauf.

3.2.1 Einführende Übungsaufgaben

Ein klassischer Einstieg in die objektorientierte Programmierung mit Java ist es, schrittweise nur ausgewählte Teile des Sprachumfangs detailliert zu betrachten. So kann beispielsweise der Fokus einer ersten Lektion auf primitiven Datentypen und einfachen Operationen liegen, während das zwingend notwendige Erstellen einer `main`-Methode ohne weitere Erläuterung als gegeben hingenommen wird und möglicherweise auftretende Exceptions erst einmal ignoriert werden.

Für diesen Einstieg kann die automatische Bewertung verschiedene didaktische Mehrwerte erzeugen: Erstens können Lösungen durch ein Bewertungssystem ausgeführt werden, auch wenn sie keinen vollständigen Programmcode darstellen, da das System die notwendigen fehlenden Teile automatisch ergänzen kann. Erste Programmieraufgaben können somit als Lückentexte mit Lücken beliebiger Größe gestellt werden, in denen wahlweise einzelne Ausdrücke und Statements oder ganze Methodenkörper ergänzt werden müssen. Die Lernenden können dazu detailliertes Feedback erhalten, wie sich ihr Code für verschiedene Eingaben verhält, ohne jedoch tatsächlich vollständigen Code schreiben zu müssen. Zweitens können auftretende Probleme wie etwa Exceptions automatisch behandelt und in Rückmeldungen umgewandelt werden, die dem Niveau der Lernenden entsprechen. Gleiches gilt insbesondere auch für Meldungen des Compilers, die in ihrer Rohform nicht immer für Anfänger geeignet sind. Mit etwas mehr Aufwand können Aufgabenautoren zudem vorgeben, wie exakt die Ausgabe einer Lösung mit der Vorgabe übereinstimmen muss. So kann zum Beispiel zwischen grundsätzlich fehlerhaften Ausgaben und einem vergessenen Leerzeichen unterschieden und die Rückmeldung an das Niveau bzw. die Ansprüche der Lernenden angepasst werden. Abbildung 3.1 zeigt eine Aufgabe mit vorgegebenem Coderahmen. Die ebenfalls dargestellte Musterlösung wird selbstverständlich nicht an Studierende

ausgegeben. Die von der Musterlösung abweichende Lösung zeigt, dass exakte Division und Ausgabeformatierung korrekt eingesetzt werden, auch wenn die ausgegebenen Zeichenketten aufgrund der Abkürzung von „Stunden“ zu „Std.“ voneinander abweichen. Mit einem geeigneten Bewertungssystem kann auch die abweichende Lösung als korrekt erkannt werden.

Alternativ oder konsekutiv folgend zum klassischen Einstieg muss die Lehre der objektorientierten Programmierung ferner die Konzeption von Klassen und Objekten vermitteln. Hierbei steht weniger die funktionale Korrektheit eines Programms im Vordergrund, sondern dessen struktureller Aufbau.

Trotz dieser scheinbar kleinen Änderung des Fokus eröffnen automatische Bewertungssysteme hier eine sehr große Bandbreite an didaktischen Möglichkeiten: Die reine Testautomatisierung kann alleine bereits große und komplexe Objektstrukturen mit einer geforderten Lösung vergleichen sowie systematisch Getter und Setter von Objekten auf funktionale Korrektheit überprüfen. Mit geeigneten Systemen können eventuelle Fehler aber nicht nur textuell, sondern auch grafisch ausgegeben werden, indem den Lernenden jeweils individuell die von ihrer Lösung erzeugte Objektstruktur angezeigt wird. Auf diese Weise wird vom Bewertungssystem eine Brücke geschlagen zurück zur objektorientierten Modellierung, die häufig parallel zur Programmierung gelehrt wird. Die Lernenden erhalten so die Möglichkeit zum unmittelbaren Vergleich zwischen dem geplanten Systemdesign und ihrer tatsächlichen Umsetzung. Ein Beispiel für eine solche Visualisierung wird in Kapitel 9 diskutiert.

Ferner können statische Prüfverfahren angewandt werden, mit denen unabhängig von der Ausführung der Lösungen geprüft werden kann, ob beispielsweise Vorgaben bezüglich der Vererbungsstruktur eingehalten wurden. So kann insbesondere auch die strukturelle Korrektheit einer Lösung mit (positivem) Feedback

```
public class Beispiel {
  public static void main(String[] args) {
    int sekunden= Integer.parseInt(args[0]);
    // Aufgabe: rechnen Sie die Sekunden in Stunden um
    // und geben Sie diese auf der Console aus.
    // Etwa so: "2,5 Stunden"
    // ...
  }
}
```

Musterlösung:

```
System.out.format("%.1f Stunden%n", sekunden/3600.0);
```

Abweichende, dennoch korrekte Lösung:

```
System.out.format("%.1f Std.%n", (double)sekunden/3600);
```

Abbildung 3.1: Beispiel einer einführenden Übungsaufgabe

versehen werden, selbst wenn die Lösung insgesamt funktional fehlerhaft oder sogar aufgrund von syntaktischen Fehlern gar nicht lauffähig ist. In diesem Fall ist es mit geeigneten Systemen zudem möglich, für die Durchführung von Tests Teile der Lösungen der Lernenden – beispielsweise eine fehlerhaft oder unvollständig implementierte Klasse – automatisiert durch eine gültige Musterlösung zu ersetzen, um so den Rest der Lösung bewerten zu können.

Über das Feedback zu funktionalen und strukturellen Eigenschaften der Lösung hinaus ist es möglich, Feedback zur Programm-*Qualität* zu geben. Während Qualitätseigenschaften wie Benutzbarkeit oder Effizienz in der Regel erst in fortgeschrittenen Übungsaufgaben oder Projektaufgaben (s. u.) relevant werden, sollten bereits Programmieranfänger auf gut lesbaren und konventionsgemäß erstellten Programmcode achten. Statische Prüfverfahren können Qualitätsmängel bzgl. der Wartbarkeit der Lösung feststellen und beispielsweise fehlende Kommentare oder Verstöße gegen Bezeichnerkonventionen monieren (s. Beispiel in Abbildung 3.2). Zeitnahes automatisches Feedback kann dafür sorgen, dass sich nachteilige Codierpraktiken gar nicht erst einschleifen. Weiche Kriterien wie beispielsweise die *sprechende* Benennung von Bezeichnern oder die *sinnvolle* Kommentierung von Methoden können mit heutigen Werkzeugen jedoch nicht automatisch geprüft werden. Solche von automatischen Bewertungssystemen offen gelassene Lücken im Feedback können und müssen derzeit noch von Menschen geschlossen werden.

Lösung:
```
double Stunden= sekunden/3600.0;
System.out.format("%.1f Stunden%n", Stunden);
```

Feedback:
Variables should start with a lowercase character, 'Stunden' starts with uppercase character.
```
double Stunden= sekunden/3600.0;
```

Abbildung 3.2: Lösung mit Qualitätsmängeln

3.2.2 Fortgeschrittene Übungsaufgaben

Wenn die Grundkonzepte der Objektorientierung und die Grundlagen der Programmierung bekannt sind, spielen in der Lehre oft algorithmische Details des korrekten Umgangs mit objektorientierten Datenstrukturen eine wichtige Rolle. Typische Aufgaben sind dazu beispielsweise das korrekte Iterieren über alle Elemente einer Liste oder ein rekursiver Durchlauf durch eine Baumstruktur.

Auch hier können Bewertungssysteme durch die Kombination statischer und dynamischer Prüfungen helfen. Zunächst einmal kann durch statische Verfahren geprüft werden, ob Vorgaben bezüglich des Kontrollflusses tatsächlich erfüllt sind. So kann beispielsweise darauf hingewiesen werden, dass eine iterative Lösung Schleifen enthalten muss, während eine rekursive Lösung genau dies nicht sollte. Es können aber auch detaillierte Hinweise gegeben werden, wenn die falsche Methode rekursiv gestaltet wird oder eine verschachtelte Schleife zum Einsatz kommt, obwohl auch eine einfache Schleife reicht. Durch eine reine Testautomatisierung, die lediglich die Ergebnisse einer Lösung vergleicht, werden solche Fehler nicht erkannt.

Trotzdem hat eine dynamische Prüfung weitere Vorteile. Neben Aussagen zur funktionalen Korrektheit können durch die Ausführung einer Lösung mit verschiedenen Eingabewerten insbesondere auch Aussagen zur Performanz gewonnen werden, so dass auch dieser Qualitätsaspekt berücksichtigt werden kann. Zudem können bei der Ausführung der Lösung Traces des Programmablaufs aufgezeichnet werden, mit denen der Kontrollfluss visualisiert werden kann. Wie im vorherigen Szenario bietet dies den Lernenden die Möglichkeit, das tatsächliche Programmverhalten unmittelbar mit der Planung zu vergleichen. Bewertungssysteme können so die Rolle eines Debuggers übernehmen, ohne dass sich die Lernenden tatsächlich im Detail in die Bedienung eines solchen Werkzeugs einarbeiten müssen. Ein Beispiel für einen aufgezeichneten Trace wird in Kapitel 9.3.2 gezeigt.

Nicht unüblich ist es, in einführenden Übungsaufgaben einige Testfälle im Aufgabentext zu verraten, während die Studierenden mit weiteren Testfällen nur durch das daraus resultierende Feedback konfrontiert werden. Je fortgeschrittener die Übungsaufgaben werden, desto mehr sollten die Studierenden gefordert sein, sich selbst möglichst umfassende Testfälle auszudenken. Fehlschlagende Testfälle der automatischen Bewertung geben dabei Hinweise, welche Sonderfälle die Lösung nicht abdeckt. Manche Bewertungssysteme erlauben es sogar, von Lernenden vorbereitete Testfälle als Teil der dynamischen Prüfung auszuführen und dabei den erreichten Abdeckungsgrad zu bewerten [Edw03].

3.2.3 Projektaufgaben

In den bisherigen Übungsaufgaben spielten die Interaktion eines Programms mit dem Benutzer sowie allgemein die Frage von Ein- und Ausgabe keine Rolle. Üblicherweise werden solche Aufgaben daher über Methodenrückgaben oder Konsolenausgaben abgewickelt. In der fortgeschrittenen Lehre ist es aber relevant und

für die Lernenden oft motivierend, kleine Anwendungen zu entwickeln, die über eine grafische Oberfläche (GUI) verfügen und größere Datenmengen verarbeiten können.

Dies stellt Bewertungssysteme vor die Herausforderung, Nutzerinteraktion zu simulieren, um eine Lösung zu prüfen. Als mögliche Fehlerquelle neben strukturellen oder funktionalen Fehlern kommen insbesondere Fehler in der Ausgabe hinzu. Diese können eine Testdurchführung verhindern, wenn beispielsweise ein Button nicht so platziert oder beschriftet ist, wie ihn die Testautomatisierung erwartet. Obwohl dies zu unerwartet schlechten Testergebnissen führen kann, kann es auch genutzt werden um zu motivieren, warum eine exakte Erfüllung der Spezifikation einer (Benutzer-)Schnittstelle notwendig ist.

Lösungen, die größere Datenmengen verarbeiten sollen, produzieren oder konsumieren diese üblicherweise über Dateischnittstellen. Dies stellt das Bewertungssystem vor die Aufgabe, technische Randbedingungen wie Zeichenkodierungen und begrenzten Speicherplatz in didaktisch geeigneter Form an die Lernenden heran zu tragen. Zeichenkodierungsfehler oder Überschreitungen des zur Verfügung stehenden Platzes auf einem Speichermedium führen zu unverständlichen Fehlermeldungen, die das Bewertungssystem verständlich aufbereiten kann.

Gleichzeitig bieten Bewertungssysteme durch die Kombination vieler verschiedener Verfahren die Möglichkeit, eine Lösung auf allen Ebenen zu testen, so dass über die Gestaltung und Implementierung des GUI bzw. der Datenverarbeitung die zuvor erlernten Aspekte der funktionalen und strukturellen Korrektheit einer Lösung nicht vergessen werden. Dabei können auch wieder Qualitätsaspekte aufgegriffen werden. Neben Codequalität und Performanz sind beispielsweise die gute Analysierbarkeit des Codes oder die sinnvolle Aufteilung der Funktionalität in mehrere Methoden relevante Facetten dieses Themas. Bewertungssysteme können durch die Anwendung klassischer und objektorientierter Codemetriken (s. etwa [SSB10]) beispielsweise unnötig aufgeblähten und damit schlecht analysierbaren Code identifizieren oder einen aus Wartungssicht nachteiligen prozeduralen bzw. objektorientierten Entwurf diagnostizieren. Weiterhin können sie Programmierpraktiken aufdecken, die zu mangelnder Robustheit führen können, z. B. einen leeren `catch`-Block oder ein `return`-Statement in einem `finally`-Block. Anstatt Programmieraufgaben ausschließlich hierzu zu stellen, können Qualitätsaspekte durch eine schrittweise strengere Prüfung im Verlaufe allgemeiner Programmieraufgaben eingeführt werden, wozu sich Projektaufgaben insbesondere eignen.

3.3 Einsatzszenario

Dieser Abschnitt betrachtet nun ein konkretes Einsatzszenario, in dem die bisher vorgestellten Aufgabentypen und ihre automatische Bewertung in den Kontext einer Lehrveranstaltung eingebettet werden. Diese Veranstaltung ist als Einführung in die objektorientierte Programmierung im Kontext einer Hochschule ausgelegt. Der Vorlesungsumfang beträgt 2 oder 4 SWS[1] Vorlesungszeit und 2 SWS Übung, die im Semesterverlauf durch freie Tutorien und verpflichtende Testate ergänzt werden.

3.3.1 Vorlesung

Die Vorlesung ist im Wesentlichen durch den Einsatz automatischer Bewertungssysteme für Programmieraufgaben nicht betroffen, sondern wird im klassischen Format mit der Präsentation des Stoffes durch den Dozenten durchgeführt. Zur Auflockerung der Präsentation des Stoffes können einführende Übungsaufgaben eingestreut werden, die von den Studierenden live in der Vorlesung gelöst und sofort besprochen werden können. Dazu ist es hilfreich, dass ein automatisches Prüfungssystem Zusammenfassungen der häufigsten Fehler erstellen kann, so dass auf diese sofort eingegangen werden kann. Zudem kann das Bewertungssystem dem Dozenten Informationen zur Verfügung stellen, die im Feedback für die Studierenden nicht enthalten sind, da sie über den aktuellen Stoff der Vorlesung hinausgehen oder einer Erläuterung bedürfen. Beispielsweise könnte das Bewertungssystem den Dozenten auf eine unnötig komplexe oder aber eine besonders elegante Lösung aufmerksam machen, um diese sofort in der Vorlesung zu zeigen und zu diskutieren. Die rohen Metrikwerte, auf denen eine solche Auswahl basieren kann, wären dagegen für den Autor der Lösung von weit geringerem Wert oder sogar völlig unverständlich.

3.3.2 Übung und Tutorium

Zur selbstständigen Vertiefung des Stoffs werden wöchentlich mehrere zunächst einführende und später fortgeschrittene Übungsaufgaben zur Verfügung gestellt, die von den Studierenden wahlfrei bearbeitet werden können. Das Bewertungssys-

1 Das hier beschriebene Einsatzszenario ist hypothetisch in dem Sinne, dass es Aspekte mehrerer in Essen und Hannover tatsächlich durchgeführter Lehrveranstaltungen kombiniert.

tem übernimmt dabei zunächst die Rolle eines Tutors, der zu jeder eingereichten Lösung zeitnah Feedback geben kann und den Studierenden somit selbstgesteuertes Arbeiten ermöglicht. Die Präsenzübung und das Tutorium dienen somit lediglich als Ergänzung für diejenigen Studierenden, die eine intensivere, individuelle Betreuung benötigen beziehungsweise Beispiele detaillierter erläutert bekommen wollen, als dies im Rahmen der Vorlesung möglich wäre. Die Bearbeitung der Übungsaufgaben ist daher auch freiwillig und ohne Einfluss auf die spätere Note. Zur Vorbereitung auf die regelmäßigen Testate (s. u.) wird zudem im selben Rhythmus eine Projektaufgabe ausgegeben, die verpflichtend zu bearbeiten ist.

Der Fokus der automatischen Bewertung liegt auf der Erzeugung von möglichst umfangreichem Feedback. Eine detaillierte Gewichtung der verschiedenen Fehler für eine differenzierte Punktevergabe ist daher nicht notwendig, solange die Punktzahlen in etwa den Fortschritt bei der Bearbeitung einer Aufgabe widerspiegeln und somit eine motivierende Rolle einnehmen. Allerdings kann auch ein Übermaß an Feedback nachteilige Effekte haben, wenn es zu umfangreich ist, um von Lernenden noch als hilfreich wahrgenommen zu werden. Insbesondere könnte es Lernenden selbst bei Einsatz einer unterschiedlichen Gewichtung verschiedener Bewertungsaspekte schwer fallen, ihre zentralen Fehler zu erkennen, um diese vorrangig zu beheben. Zudem kann eine große Menge negativen Feedbacks einen demotivierenden Effekt haben. Daher kann es sinnvoll sein, wenn ein Bewertungssystem eine solche Priorisierung intern vornimmt und nur die wichtigsten Rückmeldungen ausgibt. Die weiteren Meldungen können dann beispielsweise nur für Tutoren sichtbar sein, die diese im Rahmen ihrer individuellen Betreuung für zusätzliches Feedback nutzen können. Eine vom Bewertungssystem vorgenommene Gruppierung der Meldungen nach zu lernenden Kompetenzen bzw. nach Bewertungsaspekten ist hierbei häufig hilfreicher, als eine sequenzielle Ordnung nach Programmzeilen.

Auch wenn der Einsatz von Bewertungssystemen eine Einsparung beim Korrekturaufwand bedeutet, der die Korrektur massenhafter Übungsaufgaben für große Studierendengruppen überhaupt erst ermöglicht, darf der Aufwand für die Erstellung guter Aufgaben nicht unterschätzt werden. Insbesondere für detailliertes, aufgabenspezifisches Feedback müssen entsprechende Prüfregeln, Testfälle usw. individuell erstellt werden. Daher muss mit einem Zeitaufwand von 1-3 Stunden für eine einfache Übungsaufgabe und noch einmal erheblich mehr Zeit für eine Projektaufgabe mit umfangreichem Feedback gerechnet werden. Ein Erstellungsaufwand von 16 Stunden ist für eine komplexe Projektaufgabe nicht ungewöhnlich. Der Aufwand lohnt sich, da einmal erstellte Aufgaben mehrere Jahre oder in verschiedenen Studiengängen wiederverwendet werden können. Nötige An-

passungen können dabei in der Regel leicht durchgeführt werden, beispielsweise durch das Ändern der Punktegewichtung oder den Austausch von Textbausteinen.

Will man die Anzahl der verpflichtend zu bearbeitenden Übungsaufgaben erhöhen, kann man von Studierenden als Teil der Prüfungsleistung fordern, dass alle oder ein großer Anteil aller Übungsaufgaben semesterbegleitend bearbeitet und automatisiert bewertet werden müssen. Der menschliche Überprüfungsaufwand steigt in diesem Fall, denn in der Regel müssen Tutoren die Einreichungen nach Abgabefrist zumindest auf grobe Fehler des Bewertungssystems hin prüfen, ggf. die erreichte Punktzahl anpassen und Kommentare ergänzen sowie eine geeignete Überprüfung auf Plagiatsfälle und deren Behandlung sicherstellen.

3.3.3 Testat

Zur regelmäßigen, verpflichtenden Kontrolle des Lernfortschritts werden alle zwei Wochen Testate durchgeführt, in denen eine fortgeschrittene Übungsaufgabe unter Prüfungsbedingungen innerhalb von 45 Minuten gelöst werden muss. Die Studierenden bekommen dazu Arbeitsplätze im Rechnerpool zugewiesen, die über eine Entwicklungsumgebung verfügen, über die die Aufgabe vom Bewertungssystem herunter- und hochgeladen werden kann. Weiterer Netzwerkzugriff ist jedoch nicht zugelassen, so dass die Studierenden insbesondere keine Ressourcen aus dem Lernmanagementsystem der Vorlesung oder dem Internet nutzen dürfen und auch während der Bearbeitung keinen Zugriff auf die Ergebnisse der automatischen Bewertung haben.

Die Testate müssen von den Studierenden nicht alle bestanden werden, aber es muss eine Mindestpunktzahl erreicht werden, um die Zulassung zur abschließenden Klausur zu erhalten. Das Bewertungssystem muss daher bei den Testaten sehr viel differenzierter bei der Vergabe von Punktzahlen vorgehen. Eine der ersten Entscheidungen ist dabei, ob die Lauffähigkeit einer Lösung als K.O.-Kriterium verwendet wird. Wie in Abschnitt 3.2 bereits erwähnt, sind Bewertungssysteme grundsätzlich in der Lage, verschiedene Aspekte unabhängig voneinander zu bewerten. Es besteht also keine technische Notwendigkeit, nicht lauffähige Lösungen mit 0 Punkten zu bewerten. Da es üblich ist, in Klausuren Teilpunkte zu vergeben und die Testate der Klausurzulassung dienen, sollte daher nach Möglichkeit ein Bewertungssystem verwendet werden, mit dem Teilpunkte vergeben werden können.

Eine zweite Entscheidung ist, ob nur zwischen verschiedenen Kategorien von Fehlern gewichtet wird, oder ob auch innerhalb der Kategorien Abstufungen vorgenommen werden können. Da ein einzelner tatsächlicher Fehler (z. B. eine falsch

gesetzte Klammer) zu mehreren Fehlermeldungen des Compilers führen kann, kann es sinnvoll sein, das Vorhandensein von Compilerfehlern pauschal zu bewerten. Gleichzeitig kann es sinnvoll sein, verschiedenen Testfällen ein unterschiedliches Gewicht zu geben, um zentrale Fälle stärker in die Bewertung einfließen zu lassen als Randfälle. Auch die Art des Fehlers kann mit in die Bewertung aufgenommen werden, um zwischen einer fehlenden oder völlig falschen Ausgabe und einem leichten Rundungsfehler zu unterscheiden. Auf diese Weise wird sichergestellt, dass eine im Kern richtige Lösung eine hinreichend hohe Punktzahl erhält, auch wenn es zahlreiche Sonderfälle gibt, die nicht korrekt beachtet wurden.

Zu beachten ist außerdem, dass nicht alle Funktionen eines Bewertungssystems für den Einsatz bei der Testat- oder Klausurbewertung geeignet sind. So kann beispielsweise randomisiertes Testen bei Übungsaufgaben sehr nützlich sein, weil Lösungen so bei jedem Versuch mit neuen Eingaben konfrontiert werden. Dies spornt dazu an, eine allgemeingültige Lösung zu gestalten, die alle Randfälle abdeckt, ohne dass der Aufgabenautor für jeden dieser Fälle manuell einen Testfall erzeugen muss. Das gleiche Verfahren kann jedoch in einem Testat oder einer Klausur rechtlich problematisch sein, wenn zwei identische Lösungen unterschiedlich bewertet werden, weil durch die Randomisierung der Testfälle ein Fehler nur in einem der beiden Fälle entdeckt wird.

Gegenüber den Übungsaufgaben steigt bei Testataufgaben noch einmal der Aufwand für die Erstellung, da insbesondere mehr Sorgfalt bei der Punktevergabe nötig ist. Idealerweise werden dazu nicht nur eine, sondern mehrere Musterlösungen oder auch prototypische fehlerhafte Lösungen angefertigt, um später das Systemverhalten für diese Fälle überprüfen zu können. Anschließend müssen wie bei Übungsaufgaben für alle gewählten Prüfverfahren die nötigen Testfälle, Prüfregeln, usw. sowie ihre Gewichtung gemäß des gewählten Punkteschemas und die zugehörigen Rückmeldungen erstellt werden. Erfahrungsgemäß gelingt eine für alle denkbaren Lösungen zutreffende Aufteilung von Teilpunkten auf Bewertungsaspekte erst, nachdem viele Lösungen bewertet wurden und die Ergebnisse nachjustiert werden konnten. Dies macht einen großen Teil des Zeitaufwandes für die Vorbereitung aus. Insgesamt kann daher im Schnitt ein Aufwand von etwa 8 Stunden für die Einrichtung einer Testataufgabe angenommen werden.

Wie bereits weiter oben erwähnt wurde, ist es erforderlich und sinnvoll, die automatisch erzeugten Bewertungen durch Menschen zu überprüfen, sobald sie direkt oder indirekt in die Prüfungsnote eingehen. Die entstehenden Überprüfungsaufwände liegen allerdings erheblich unter den Aufwänden einer vollständig manuellen Korrektur, da korrigierende Eingriffe in das Bewertungsresultat bei sorgfältigem Aufgabendesign erfahrungsgemäß eher selten sind. Der Erstellungsaufwand einer Testataufgabe kann gegen den zu erwartenden menschlichen

Überprüfungsaufwand abgewogen werden. Je häufiger eine Testataufgabe genutzt wird, desto eher lohnt sich die Entwicklung eines alle denkbaren Lösungen abdeckenden, ausgefeilten Punkteschemas.

3.3.4 Klausur

Da durch die regelmäßigen Testate bzw. durch regelmäßige Übungseinreichungen bereits im Verlauf des Semesters sichergestellt ist, dass die Studierenden ausreichende Programmierfähigkeiten nachweisen, um überhaupt zur Teilnahme an der Klausur zugelassen zu werden, kann der Schwerpunkt der Klausur auch auf konzeptionelle Fragen der Programmierausbildung gelegt werden. Die Klausur wird daher idealerweise als Hybridklausur durchgeführt, in der Programmieraufgaben am Rechner gelöst werden, während weitere Aufgaben schriftlich auf Papier oder in einem allgemeinen elektronischen Prüfungssystem bearbeitet werden. Fehlt die entsprechende Infrastruktur für eine E-Prüfung, gehen wir in diesem Szenario von vollständig handschriftlich angefertigten Klausuren aus, die sich einer automatisierten Bewertung entziehen.

3.4 Erfahrungen

3.4.1 Erfahrungen der Lehrenden

Die Entwicklung und der Einsatz des oben beschriebenen Konzepts mit wöchentlichen Übungs- und Testataufgaben war in einer großen Einführungsveranstaltung überhaupt erst möglich, weil automatische Bewertungssysteme zu Hilfe genommen werden konnten. Die einzige Alternative ist der keineswegs kostengünstige Einsatz von Tutoren als Bewertungspersonal. Zwar können Tutoren prinzipiell ein differenziertes Feedback geben, zu bedenken ist dabei allerdings, dass die Qualität der von Tutoren abgegebenen Bewertungen erheblich schwanken kann und die Wartezeiten für die Studierenden bis zum Erhalt von Feedback auf jeden Fall höher liegen als bei einer automatisierten Lösung.

Gleichwohl sollen automatische Bewertungssysteme die menschliche Betreuung nicht gänzlich abschaffen, sondern ihren Einsatz gewinnbringender gestalten. In Tutorien können Lehrende ihre Zeit erfahrungsgemäß insbesondere für sinnvollere Aufgaben als die Fehlersuche in studentischen Programmen investieren. Individuelle Beratung von schwächeren Studierenden, die Wiederholung von in der Vorlesung vermittelten Konzepten an weiteren Beispielen, aber auch die ein-

gehende Beantwortung und manchmal Recherche zu über den Prüfungsstoff hinausgehenden Fragen von ambitionierten Studierenden wird so ermöglicht. Die Arbeitsweise in Tutorien und Übungsgruppen hat sich unserer Erfahrung nach entsprechend geändert und führt insbesondere auch auf der Seite der Tutoren und Lehrenden zu höherer Motivation.

Die Erfahrung zeigt allerdings auch, dass man gute, automatisiert bewertbare Java-Programmieraufgaben erst nach vielen Monaten Erfahrung zügig erstellen kann. Wie jede Spezifikation oder Software können auch Testtreiber oder Prüfregeln Lücken oder Fehler enthalten, die erst dann offenbar werden, wenn die Aufgabe zum ersten Mal vielen studentischen Lösungen ausgesetzt war. Um Programmieraufgaben hoher Qualität nachhaltig zu entwickeln, müssen diese wie ein langfristig eingesetztes Softwareprodukt über die üblichen Entwicklungsstadien von der Beta-Fassung über ein erstes Release und dann folgende Hotfixes und Patches gepflegt werden.

Die Aufgaben, die im hier beschriebenen Einsatzszenario verwendet wurden, sind zunächst vollständig an der Universität Duisburg-Essen bzw. der Hochschule Hannover neu entwickelt worden. Die Aufgaben wurden dann in mehreren aufeinander folgenden Semestern wiederverwendet, teilweise nach Korrekturen oder einfachen Anpassungen. Ursprünglich für Studierende der Informatik bzw. der Angewandten Informatik konzipiert, wurden einige Aufgaben auch in anderen, informatiknahen Studiengängen (z. B. Medizinisches Informationsmanagement) genutzt. Unsere Erfahrungen mit der Wiederverwendung von Aufgaben sind gut, weil sie das aus einer größeren Zahl von Lernenden stammende Feedback und damit die Robustheit der Aufgaben erhöht. Ferner wurden in Essen entwickelte Aufgaben auch an Lehrende anderer Hochschulen weitergegeben, die ebenfalls das Bewertungssystem JACK einsetzen und so mit weniger Vorbereitungsaufwand in den Produktivbetrieb starten konnten.

Eine weitere interessante Erfahrung ist, dass sich mit dem Einsatz automatischer Bewertungssysteme auch die grundsätzliche Art der Bewertung anpassen muss. Beispielsweise kann es sinnvoll sein, Studierende zu belohnen, die sich eher durch Nachdenken an die richtige Lösung herantasten, als durch inflationär viele unzureichende Einreichungsversuche [Iha+10]. Automatisch vergebene Strafpunkte ab einer gewissen Zahl von Versuchen sind eine mögliche Maßnahme. Eine andere Maßnahme sind dynamische Prüfungen mit zwei Testfallmengen, von denen eine vor Abgabefrist aktiv ist, während die zweite erst nach Abgabefrist verwendet wird. Als spielerisches Element könnte man Studierenden sogar erlauben, einzelne Testfälle der zweiten Menge gegen Punktabzug „hinzuzukaufen“. Derartige Verfahren werden allerdings nicht von allen Bewertungssystemen

unterstützt. Ferner sind sie nicht spezifisch für die objektorientierte Programmierung in Java und sollen daher hier nicht weiter vertieft werden.

3.4.2 Erfahrungen der Studierenden

Evaluationen aus studentischer Sicht zum Einsatz automatischer Bewertungssysteme in der Java-Lehre sind z. B. in [Stö+13; SG09; SG11b] verfügbar. Für Evaluationen zu den von uns eingesetzten Systemen JACK und GRAJA wird auf Kapitel 9 und 11 verwiesen. In diesem Abschnitt sollen beispielhaft Details aus weiteren Studien erörtert werden, die in direkterem Bezug zu den oben beschriebenen Aufgabentypen und Einsatzszenarien stehen.

Im Wintersemester 2012/13 wurde dazu eine Umfrage unter 56 Studierenden eines 1. Semesters im Studiengang „Angewandte Informatik“ der Hochschule Hannover durchgeführt. Im Einsatzszenario „Übung und Tutorium“ (siehe Abschnitt 3.3.2) mussten die Studierenden 60% aller in Übungsaufgaben erreichbaren Punkte als verpflichtenden Teil der Prüfungsleistung erlangen. Die Aufgabenform entsprach weitestgehend einfachen und fortgeschrittenen Übungsaufgaben (siehe Abschnitt 3.2.1 und 3.2.2). Wenige Aufgaben hatten den Charakter von Projektaufgaben (siehe Abschnitt 3.2.3). Zusammenfassend beurteilten 78% der Studierenden das Bewertungssystem GRAJA als hilfreich. Verglichen mit menschlichen Tutoren wurde GRAJA als schneller, vergleichbar gut beim Entdecken von Fehlern, jedoch nicht als gerechter eingeschätzt. Dem Einsatz in einer Klausur (siehe Abschnitt 3.3.4) ohne flankierende, durch menschliche Bewerter durchgeführte Überprüfungen, standen die Befragten dieser Studie mit 73% mehrheitlich ablehnend gegenüber.

Positiver bzgl. des letztgenannten Aspekts fallen im Vergleich dazu die Ergebnisse einer ähnlichen Befragung unter den Studierenden der Studiengänge „Angewandte Informatik“ und „Wirtschaftsinformatik“ an der Universität Duisburg-Essen im Wintersemester 2011/12 aus. Von 63 Teilnehmern gaben hier ca. 55% an, ihre Klausur gerne durch JACK bewerten zu lassen. In der gleichen Befragung gaben zudem ca. 50% der Teilnehmer an, dass ihnen das Konzept mit freien Übungsaufgaben und automatischer Bewertung ein selbstständiges und unabhängiges Arbeiten ermöglicht und dass JACK im Rahmen dieses Konzeptes ein sinnvolles E-Learning-Werkzeug darstellt. Folgerichtig würden 40% der Befragten das System auch in anderen Lehrveranstaltungen nutzen wollen.

Unterstützt werden diese einmaligen Befragungsergebnisse durch Beobachtungen des Nutzungsverhaltens in den folgenden Semestern. So wurden in Essen im Wintersemester 2015/16 von knapp 700 Studierenden in der Vorlesung nicht nur

knapp 11.000 Lösungen zu Testaten und etwa 12.000 Lösungen zu verpflichtenden Übungs- und Projektaufgaben eingereicht, sondern darüber hinaus auch gut 9.000 Lösungen zu weiteren freiwilligen Übungsaufgaben. Nicht zuletzt konnte durch das gesamte Konzept auch der Anteil derjenigen Studierenden, die die Lehrveranstaltung erfolgreich abschließen, erheblich gesteigert werden.

3.5 Fazit und Ausblick

Perspektiven für den weiteren zukünftigen Einsatz automatischer Bewertungssysteme ergeben sich vor allem daraus, dass sich noch einige Analyseaufgaben automatisieren lassen sollten. Insbesondere ist dabei zu berücksichtigen, dass in der industriellen Softwareentwicklung zum Teil schon Verfahren existieren, die in E-Learning-Systemen noch nicht zum Einsatz kommen, aber interessante weitere Möglichkeiten für Feedback und Bewertung eröffnen können.

Grenzen hat der Einsatz automatischer Bewertungssysteme immer dort, wo die kreative Leistung eines Studierenden wichtiger ist als die Umsetzung formaler Regeln. So kann eine automatisierte Bewertung zwar sowohl auf syntaktische als auch inhaltliche Fehler hinweisen und dabei durch den Vergleich mit Musterlösungen möglicherweise sehr präzise benennen, an welcher Stelle ein Fehler liegt, aber dies bedeutet nicht zwangsläufig, dass der Fehler nur durch eine Korrektur an dieser Stelle behoben werden kann. Stattdessen ermöglichen Programmieraufgaben im Allgemeinen beliebig viele korrekte Lösungen und damit auch beliebig viele Möglichkeiten, einen gegebenen Fehler zu beheben.

Literatur für dieses Kapitel

[Bös98] László Böszörményi. „Why Java is not my favorite first-course language“. In: *Software-Concepts & Tools* 19.3 (1998), S. 141–145.

[Edw03] Stephen H. Edwards. „Improving Student Performance by Evaluating How Well Students Test Their Own Programs“. In: *J. Educ. Resour. Comput.* 3.3 (Sep. 2003). DOI: 10.1145/1029994.1029995.

[Gri08] David Gries. „A principled approach to teaching OO first“. In: *ACM SIGCSE Bulletin* 40.1 (2008), S. 31–35.

[HTW04] Emily Howe, Matthew Thornton und Bruce W. Weide. „Components-first approaches to CS1/CS2: principles and practice“. In: *ACM SIGCSE Bulletin* 36.1 (2004), S. 291–295.

[Iha+10] Petri Ihantola u. a. „Review of Recent Systems for Automatic Assessment of Programming Assignments“. In: *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. Koli Calling '10. ACM, 2010, S. 86–93. DOI: 10.1145/1930464.1930480.

[Pea+07] Arnold Pears u. a. „A Survey of Literature on the Teaching of Introductory Programming“. In: *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*. ITiCSE-WGR '07. ACM, 2007, S. 204–223. DOI: 10.1145/1345443.1345441.

[Reg06] Stuart Reges. „Back to basics in CS1 and CS2“. In: *ACM SIGCSE Bulletin* 38.1 (2006), S. 293–297.

[SG09] Michael Striewe und Michael Goedicke. „Effekte automatischer Bewertungen für Programmieraufgaben in Übungs- und Prüfungssituationen“. In: *DeLFI 2009 – Die 7. E-Learning Fachtagung Informatik*. Bd. 153. LNI. GI, 2009, S. 223–234.

[SG11b] Michael Striewe und Michael Goedicke. „Studentische Interaktion mit automatischen Prüfungssystemen“. In: *DeLFI 2011 – Die 9. e-Learning Fachtagung Informatik*. Bd. 188. LNI. GI, 2011, S. 209–220.

[SSB10] Harry M. Sneed, Richard Seidl und Manfred Baumgartner. *Software in Zahlen*. Carl Hanser Verlag GmbH & Co. KG, 2010.

[Stö+13] Andreas Stöcker u. a. „Evaluation automatisierter Programmbewertung bei der Vermittlung der Sprachen Java und SQL mit den Gradern *aSQLg* und *Graja* aus studentischer Perspektive.“ In: *DeLFI 2013 – Die 11. E-Learning Fachtagung Informatik*. Bd. 218. LNI. GI, 2013, S. 233–238.