Detlef Krömker und Ulrik Schroeder (Hrsg.): Die 16. E-Learning Fachtagung Informatik,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2018 153

Automatische Generierung von Aufgaben zum Codeverständnis

Michael Striewe[1] und Michael Goedicke[1]

Abstract: Um Codeverständnis zu üben oder zu prüfen, können Multiple-Choice-Aufgaben gestellt werden, in denen zu einem gegebenen Code angegeben werden soll, welche der vorgeschlagenen Codeänderungen auch das Programmverhalten ändert. Das Erstellen solcher Aufgaben ist aufwändig, kann jedoch automatisiert werden. Der Beitrag stellt am Beispiel von Aufgaben zur for-Schleife in Java einen entsprechenden Generator vor und evaluiert die erzeugten Aufgaben anhand von Daten aus einer Erstsemestervorlesung zur Programmierung.

Keywords: Programmierausbildung, Multiple Choice, Aufgabengenerator, Distraktoranalyse

1 Einleitung

In einer gängigen Form von Übungs- und Prüfungsaufgaben zum Codeverständnis soll zu einem gegebenen Stück Programmcode und gegebenen Eingabedaten die zugehörige Ausgabe bestimmt werden. Die Antwort kann dabei als freie Eingabe oder als Auswahl (Single Choice) erfolgen. Als Multiple-Choice-Variante können die Aufgaben so gestellt werden, dass als Antwortoptionen nicht die Programmausgabe angeboten wird, sondern mögliche Änderungen des Codes und die Frage dazu lautet, welche der Änderungen zu einer veränderten Ausgabe führt. Eine Beispielaufgabe zeigt Abbildung 1. Auch wenn Multiple-Choice-Aufgaben häufig als zu leicht angesehen werden, können gute Aufgaben einen hohen Beitrag zur (Selbst-)Überprüfung der Lernenden leisten [LL03a]. Übungen mit derartigen Aufgaben erscheinen zudem sinnvoll, da das Verständnis von Programmstrukturen eine bekannte Schwierigkeit für Programmieranfänger ist [LAJ05].

Das Erstellen solcher Aufgaben ist allerdings verhältnismäßig aufwendig, da Antwortoptionen erzeugt und ihre Auswirkungen auf die Ausgabe überprüft werden müssen [Li01]. Für beide Schritte ist das Vorgehen jedoch weitgehend mechanisch, da Antwortoptionen durch syntaktische Modifikationen (Vertauschungen, Ersetzungen, usw.) einer Programmzeile erzeugt werden können. Die Überprüfung der Auswirkungen kann durch Ausführen des modifizierten Codes erfolgen. Es ist daher naheliegend, diese beiden Schritte zu automatisieren, um schnell eine größere Menge von Aufgaben zu erzeugen. Dabei muss jedoch sichergestellt sein, dass die so erzeugten Aufgaben sinnvoll und die Antwortoptionen von angemessener Schwierigkeit sind.

Dieser Beitrag gibt in Abschnitt 2 einen kurzen Überblick über verschiedene Ausprägungen des geschilderten Aufgabentyps. In Abschnitt 3 erfolgt eine Beschreibung und Diskussion eines Generators, der Aufgaben automatisch erzeugen kann. Die Evaluation in Abschnitt 4 beantwortet drei Forschungsfragen: (1) Sind die Aufgaben motivierend und werden sie von den Studierendenden ernsthaft bearbeitet? (2) Sind die erzeugten

[1] Universität Duisburg-Essen, paluno - The Ruhr Institute for Software Technology, Gerlingstraße 16, 45127 Essen, {vorname.nachname}@paluno.uni-due.de

Antwortoptionen von angemessener Schwierigkeit? (3) Sind inhaltlich vergleichbare Aufgaben unabhängig von Details der Aufgabenstellung von gleicher Schwierigkeit?

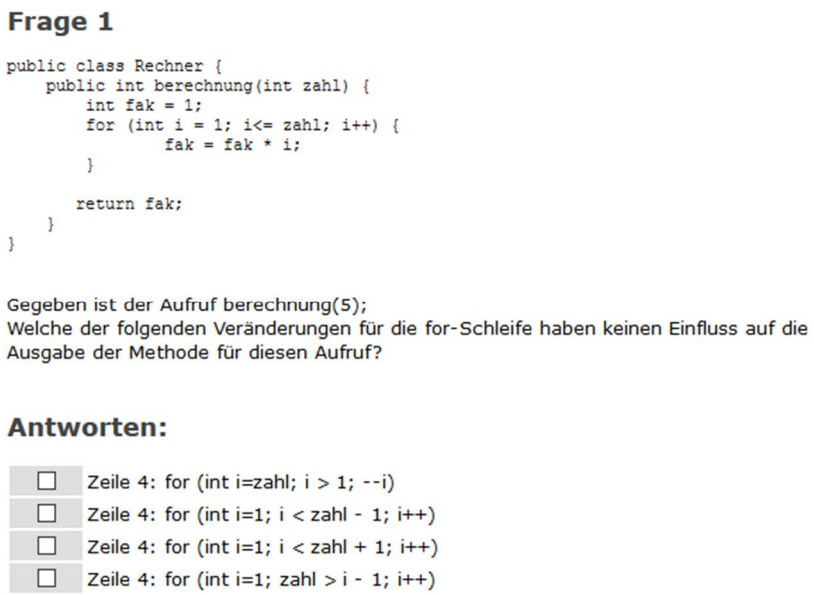

Abb. 1: Eine einfache, automatisch generierte Beispielaufgabe zur Überprüfung des Codeverständnisses als Multiple-Choice-Aufgabe mit vier Antwortoptionen

2 Verwandte Arbeiten

Zwei verschiedene Ansätze zur Generierung von Multiple-Choice-Aufgabe zum Codeverständnis wurden von Traynor und Gibson erprobt und verglichen [TG05]. Dabei stellte sich ein template-basierter Ansatz als geeigneter heraus als ein vollständig zufälliger Generierungsalgorithmus. Die Aufgabenstellung fragt dabei immer nach der Ausgabe des gegebenen Codes. Eine Untersuchung weiterer Fragen sowie eine genauere Evaluation der erzeugten Antwortalternativen erfolgt in der Arbeit von Traynor und Gibson nicht. Der Ansatz des vorliegenden Papers basiert ebenfalls auf template-artigen Ersetzungen, verwendet jedoch andere Fragestellungen und enthält eine genauere Evaluation der erzeugten Antwortoptionen.

Die Qualität der Antwortoptionen in Multiple-Choice-Aufgaben zur Programmierung wird auch von Lister diskutiert [Li01]. Als Beispielaufgabe aus dem Bereich der ÄDebugging Skillsñ wird dort eine Aufgabe genannt, in dem zu einem gegebenen Programmcode angegeben werden soll, welche der als Antwortoption vorgegebenen Ersetzungen keine ArrayIndexOutOfBoundsException erzeugt. Ein weiteres Beispiel ist in [LL03b] zu finden. Beide Paper befassen sich jedoch nicht mit der automatischen Generierung von Aufgaben und nennen den Aufwand zur manuellen Erzeugung guter

Aufgaben explizit als Nachteil. Auch für die ĂCanterbury QuestionBankñ [SAC+13] wurden Aufgaben nicht explizit automatisch erzeugt, aber es sind Aufgaben enthalten, die sich mit dem hier diskutierten Verfahren erzeugen lassen.

Ein template-basierter Ansatz zur Generierung individueller Aufgaben wurde auch von Brusilovsky und Sosnovsky umfassend evaluiert [BS05]. Dieser erzeugt jedoch Aufgaben, bei denen eine freie Antworteingabe erfolgt, so dass eine Evaluation von automatisch erzeugten Antwortoptionen nicht notwendig ist.

3 Generierungsprozess und Beispielaufgaben

In diesem Abschnitt wird der automatische Generierungsprozess am Beispiel von drei Aufgabenstellungen zur for-Schleife in Java näher erläutert. Der verwendete Generator ist grundsätzlich so implementiert, dass ein Aufgabenautor einen kompilierfähigen Codeabschnitt (in Java also insbesondere eine Quellcode-Datei) laden kann und der Generator daraufhin überprüft, welche der von ihm erzeugbaren Aufgaben sinnvoll darauf angewendet werden können. Dazu wird im Quellcode nach dem Auftreten von Mustern gesucht, deren Existenz die Voraussetzung für die Sinnhaftigkeit der jeweiligen Aufgabe ist. Der Generator erzeugt dann mehrere Antwortoptionen, von denen der Aufgabenautor beliebige nach manueller Kontrolle für die endgültige Verwendung ausschließen kann. Anschließend kann der Autor die fertige Aufgabe exportieren und in ein E-Learning-System importieren. Die Aufgaben sind dabei stets so konfiguriert, dass alle vom Generator erzeugten und vom Autor akzeptierten Antwortoptionen enthalten sind, jedoch stets nur vier zufällig gewählte bei der Bearbeitung angezeigt werden.

Als erstes Beispiel dient die Aufgabe, die bereits in Abbildung 1 gezeigt wurde. Die Erzeugung dieser Aufgabe hängt davon ab, dass im Code eine Methode vorhanden ist, die eine for-Schleife enthält. Ferner muss diese Methode entweder - wie im Beispiel gezeigt - eine Rückgabe haben oder eine Ausgabe auf der Konsole erzeugen. Sind diese Bedingungen erfüllt, nimmt der Generator nacheinander eine Reihe von Ersetzungen oder Vertauschungen im Kopf der gefundenen for-Schleife vor, um Antwortoptionen zu erzeugen: (1) Umkehr der Zählrichtung der Schleife; (2) Umkehrung der Schreibweise der Abbruchbedingung; (3) Verändern der Schleifenlänge durch Hinzufügen oder Entfernen eines = in der Abbruchbedingung; (4) Verändern der Schleifenlänge durch Hinzufügen oder Entfernen eines +1 oder -1 in der Abbruchbedingung. Modifikationen können auch kombiniert angewendet werden, so dass sich insgesamt bis zu neun Varianten ergeben. Theoretisch sind noch deutlich mehr Umformungen möglich, doch die sich daraus ergebende Menge an Antwortoptionen ist für eine initiale Studie zu groß und wurde daher beschränkt. Die Erzeugung und Untersuchung weiterer Antwortoptionen ist Teil zukünftiger Arbeiten. Dabei sind insbesondere auch didaktische Aspekte zu berücksichtigen, damit sich die angewandten Umformungen aus typischen Fehlern oder Verständnisproblemen motivieren lassen.

Welche der erzeugten Antwortoptionen die Rückgabe bzw. Ausgabe der Methode ändert, hängt mit Ausnahme von Fall (2) vom restlichen Code ab und wird daher durch Ausführung aller modifizierten Varianten festgestellt. Dazu wird ein für alle Tests identischer Eingabewert für die Methode zufällig gewählt. Im Beispiel aus Abbildung 1

berechnet der Code die Fakultätsfunktion, so dass die Umkehrung der Zählrichtung (erste Antwortoption) beispielsweise keine Auswirkung auf das Ergebnis hat. Von den neun Varianten führen insgesamt fünf zu keiner Änderung. Im Beispiel aus Abbildung 1 führt damit die Wahl der ersten, dritten und vierten Antwortoption zur richtigen Lösung.

Ein etwas umfangreicheres Beispiel ist in Abbildung 2 zu sehen. Der Code erfüllt wieder die oben genannten Bedingungen, so dass dieselbe Frage gestellt werden kann. Der Code

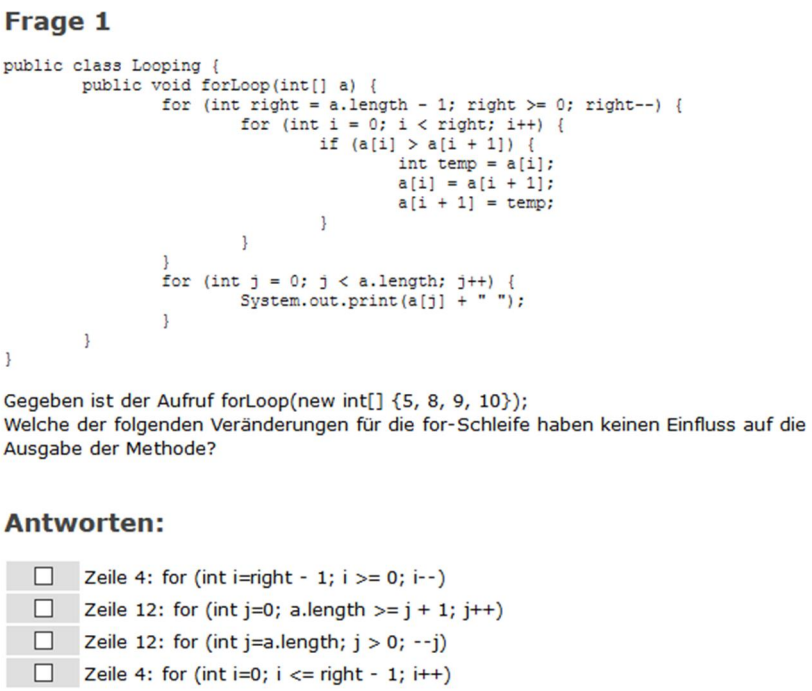

Abb. 2: Eine weitere Aufgabe zum Codeverständnis, die dieselbe Aufgabenstellung, aber einen längeren Programmcode als in Abbildung 1 verwendet

enthält jedoch insgesamt drei for-Schleifen, so dass mehr Antwortoptionen erzeugt werden. Für die zweite und dritte Schleife können wieder neun Varianten erzeugt werden. Der erste Schleifenkopf ist etwas anders formuliert, so dass der Generator hier weniger Änderungsmöglichkeiten findet. Für diese Schleife werden daher nur fünf Alternativen erzeugt, so dass zu dieser Aufgabe in Summe 23 Antwortoptionen erzeugt werden, von denen 12 keine Auswirkung auf die Ausgabe der Methode haben. Um dies festzustellen werden erneut alle Ersetzungen ausgeführt. Dazu wird ein zufälliges Array als Eingabe erzeugt. Für den konkreten Code aus Abbildung 2 ist diese Auswahl möglicherweise ungünstig, da das Array bereits aufsteigend sortiert ist. Einige Antwortoptionen, die die durch die verschachtelte Schleife realisierte Sortierung verkürzen, führen daher trotzdem zu keiner Änderung der Ausgabe. Andererseits bedarf es bereits eines umfassenden Verständnisses über die Arbeitsweise des Codes, um dies zu erkennen, so dass die

Aufgabe durch diese Schwäche nicht zwangsläufig leichter wird. Im Beispiel aus Abbildung 2 führt die Wahl der ersten, zweiten und vierten Antwortoption zur richtigen Lösung. Zu beachten ist, dass einige Antwortoptionen erzeugt werden können, für die die Ausführung zu einer ArrayIndexOutOfBoundException führt. Dies zählt im Sinne der Aufgabenstellung auch zu einer Beeinflussung der Ausgabe der Methode.

Der in diesem Beispiel verwendete Programmcode erfüllt auch noch die Bedingungen für eine weitere Aufgabe, die dem Paper von Lister [Li01] entnommen ist. Diese Aufgabe ist in Abbildung 3 zu sehen und befasst sich noch einmal explizit mit dem oben bereits genannten Auftreten von Exceptions. Voraussetzung ist hier, dass innerhalb einer for-Schleife über einen Index auf ein eindimensionales Array zugegriffen wird. Beide Schleifen erfüllen diese Bedingung. Der Generator versucht hier nun wieder eine Reihe von Ersetzungen, die auf typischen Fehlern oder Verständnisproblemen basieren: (1) Verändern der Schleifenlänge durch Hinzufügen oder Entfernen eines = in der Abbruchbedingung; (2) Verändern der Schleifenlänge durch Ersetzung von length-1 durch length; (3) Verschieben des Vergleichs von a[i] und a[i+1] auf a[i-1] und a[i]. Weitere Änderungen wie die aus den bisherigen Beispielen sind zwar auch möglich, lassen aber keine Auslösung einer ArrayIndexOutOfBoundException erwarten und werden daher vom Generator für diese Aufgabe nicht angewandt. Insgesamt ergeben sich daher für diesen Code lediglich fünf Antwortoptionen. Die Überprüfung durch Ausführung ergibt, dass davon nur eine keine Exception provoziert. Diese Antwortoption ist in Abbildung 3 an vierter Stelle angezeigt. Anders als bei der vorherigen Aufgabe ist diesmal ein Array als Eingabe gewählt worden, das nicht aufsteigend sortiert ist. Die Ausgabe der Methode wird durch diese Antwortoption also verändert, was jedoch diesmal gemäß Aufgabenstellung nicht relevant ist.

4 Evaluation

Die drei oben gezeigten Beispielaufgaben sowie drei weitere Aufgaben zu den Themen ÄRekursionñ, Ästatische Felderñ und ÄPolymorphieñ wurden in einer Erstsemester-vorlesung zur Programmierung in Java zwei Wochen lang zur Klausurvorbereitung in einem E-Learning-System zur Verfügung gestellt.

Frage 1

```
public class MoreLoops {
        public void forLoop(int[] a) {
                for (int right = a.length - 1; right >= 0; right--) {
                        for (int i = 0; i < right; i++) {
                                if (a[i] > a[i + 1]) {
                                        int temp = a[i];
                                        a[i] = a[i + 1];
                                        a[i + 1] = temp;
                                }
                        }
                }
                for (int j = 0; j < a.length; j++) {
                        System.out.print(a[j] + " ");
                }
        }
}
```

Welche der folgenden Veränderungen lösen für den Aufruf forLoop(new int[] {4, 5, 3, 7, 3}); keine ArrayIndexOutOfBoundsException aus?

Antworten:

- ☐ Zeile 4: i <= right anstelle von i < right
- ☐ Zeile 3: right=a.length anstelle von right=a.length - 1
- ☐ Zeile 5: a[i - 1] > a[i] anstelle von a[i] > a[i + 1]
- ☐ Zeile 3: right > 0 anstelle von right >= 0

Abb. 3: Eine weitere Aufgabe zum Codeverständnis, in der nicht nach der Ausgabe des Codes, sondern nach dem Auftreten von Exceptions gefragt wird

Die Aufgaben deckten damit sowohl Themen ab, die tendenziell als leicht empfunden werden als auch solche, die als schwer gelten [LAJ05]. Die Bearbeitung war für die Studierenden freiwillig und nicht an Voraussetzungen oder Bonuspunkte geknüpft. Auf Basis der so gewonnenen Daten sollen nun die drei in der Einleitung genannten Forschungsfragen genauer untersucht und beantwortet werden.

4.1 Sind die Aufgaben motivierend und werden sie von den Studierendenden ernsthaft bearbeitet?

Diese Frage zielt darauf ab, ob die Aufgaben grundsätzlich sinnvoll als Übungsaufgaben eingesetzt werden können. Wenn die Studierenden die Aufgaben nicht motiviert bearbeiten, könnte zwar ein Einsatz als Prüfungsaufgabe möglich sein, jedoch nicht als Übungsangebot im Selbstlernformat. Wenn die Studierenden die Aufgaben nicht ernsthaft bearbeiten, sondern die Lösung durch Raten oder Ausprobieren erreichen wollen, kann durch die Aufgaben kein Lerneffekt festgestellt bzw. erreicht werden. Dann ist weder ein Einsatz als Übungsaufgabe noch ein Einsatz als Prüfungsaufgabe sinnvoll.

Aufgabe	Anzahl Studierende	Anzahl Bearbeitungen	Anzahl Versuche	Anzahl erfolgreiche Bearbeitungen	Anteil erfolgreiche Bearbeitungen	Schnitt Versuche
Quiz 1	59	70	390	46	66 %	5,57
Quiz 2	51	56	133	52	93 %	2,38
Quiz 3	37	40	104	38	95 %	2,60
Quiz 4	34	40	200	31	78 %	5,00
Quiz 5	24	26	117	22	85 %	4,50
Quiz 6	28	31	85	29	94 %	2,74

Tab. 1: Bearbeitungszahlen der sechs in der Evaluation verwendeten Aufgaben. Quiz 1, 4 und 5 entsprechen den drei Beispielaufgaben aus den Abbildungen 1 bis 3. Studierende konnten während einer Bearbeitung einer Aufgabe mehrere Versuche unternehmen. Verlassen und erneutes Betreten der Aufgabe zählt als neue Bearbeitung

Tabelle 1 gibt eine Übersicht über die Bearbeitung der sechs Aufgaben. Quiz 1, 4 und 5 entsprechen dabei den Beispielaufgaben aus den Abbildungen 1 bis 3. Folgende Beobachtungen können getroffen werden: Mit Ausnahme von Quiz 1 entspricht die Zahl der erfolgreichen Bearbeitungen im Wesentlichen der Zahl der Studierenden. Auch die Zahl der Bearbeitungen liegt nur leicht darüber. Tatsächlich haben im Wesentlichen alle Studierenden die Aufgaben so lange bearbeitet, bis sie einen Erfolg erzielt haben. Bei allen Aufgaben liegt der Schnitt der Versuche pro Bearbeitung zwischen 2 und 6. Es ist daher anzunehmen, dass die Studierenden ihre Erfolge nicht durch Raten oder systematisches Ausprobieren erzielt haben, sondern eine gezielte Auswahl getroffen haben. Aufgrund dieser beiden Beobachtungen kann geschlossen werden, dass die Aufgaben motiviert und ernsthaft bearbeitet wurden. Diese Schlussfolgerung wird auch dadurch gestützt, dass es insgesamt 5 Bearbeitungen gibt, in der mehr Versuche unternommen wurden, als bei systematischem Durchprobieren aller 16 kombinatorisch möglichen Lösungen nötig sind. Andererseits muss auch festgestellt werden, dass es bei Quiz 1 zwar die höchste Anzahl an Bearbeitungen, aber einen geringeren Anteil erfolgreicher Bearbeitungen gibt und die Zahl der Studierenden in den weiteren Aufgaben absinkt. Es gibt daher wohl auch einige Studierende, auf die die erste Aufgabe nicht motivierend wirkte und die daher keine weiteren Aufgaben bearbeitet haben. Ferner scheint von den Studierenden nicht bemerkt oder erwartet worden zu sein, dass sie beim erneuten Bearbeiten der Aufgabe möglicherweise andere Antwortoptionen angezeigt bekommen. Theoretisch hätte dies die Studierenden dazu motivieren können, die Aufgaben mehrfach zu bearbeiten. Dieser Effekt kann jedoch nicht beobachtet werden.

4.2 Sind die erzeugten Antwortoptionen von angemessener Schwierigkeit?

Diese Frage betrifft die Qualität der Antwortoptionen. Nur wenn diese nicht zu leicht als richtig oder falsch identifiziert werden können, sind die Aufgaben für Übungs- und Prüfungszwecke geeignet. Einen ersten Hinweis dazu liefert bereits die durchschnittliche Anzahl der Versuche, die für die erste Forschungsfrage betrachtet wurde. Daraus kann geschlossen werden, dass die Antwortoptionen nicht zu leicht sind, denn sonst wären

mehrfache Versuche in einer Bearbeitung nicht nötig gewesen. Im Folgenden werden die Antwortoptionen für die drei oben gezeigten Beispielaufgaben analysiert. Eine vollständige Distraktoranalyse (vgl. [LR98]) ist dabei jedoch nicht möglich, da keine vollständigen Tests und damit auch keine Daten zur Trennschärfe vorliegen. Die Analyse ist daher auf die Betrachtung der Wahlhäufigkeiten der Antwortoptionen beschränkt und differenziert zwischen der Wahl im ersten Versuch und der Wahl insgesamt.

Antwortoption	Bearbeitungen	Versuche	Wahlhäufigkeit insgesamt	Wahlhäufigkeit 1. Versuch
Z4: for (int i=zahl; i >= 1; i--)	28	165	46,1 %	57,1 %
Z4: for (int i=zahl; i > 1; --i)	24	122	40,2 %	25,0 %
Z4: for (int i=1; zahl > i - 1; i++)	31	219	42,5 %	25,8 %
Z4: for (int i=1; i < zahl + 1; i++)	41	221	48,9 %	63,4 %
Z4: for (int i=1; zahl >= i; i++)	31	217	50,2 %	51,6 %
Z4: for (int i=1; i < zahl - 1; i++)	35	160	32,5 %	5,7 %
Z4: for (int i=zahl + 1; i >= 1; i--)	35	193	32,6 %	28,6 %
Z4: for (int i=1; zahl > i + 1; i++)	30	160	33,8 %	20,0 %
Z4: for (int i=zahl - 1; i >= 1; i--)	25	103	22,3 %	24,0 %

Tab. 2: Statistik der Antwortoptionen für Quiz 1 bzw. die Aufgabe aus Abbildung 1. Grau hinterlegte Antwortoptionen sind richtig im Sinne der Aufgabenstellung

Tabelle 2 zeigt die Daten für die erste Schleifenaufgabe. Durch den Generator wurden für die for-Schleife insgesamt neun Antwortoptionen erzeugt, von denen fünf als richtig und vier als falsch im Sinne der Aufgabenstellung zu werten sind. Es ist zu erkennen, dass in der Häufigkeit über alle Versuche alle ñrichtigenñ Antwortoptionen prozentual häufiger gewählt wurden als alle Äfalschenñ Antwortoptionen. Dies entspricht dem erwartbaren Verhalten, nach dem Studierende eine korrekte Wahl über mehrere Versuche hinweg beibehalten, während sie versuchen, eine falsche Wahl zu eliminieren.

Während die Ärichtigenñ Antwortoptionen alle eine leicht unterschiedliche Wahlhäufigkeit aufweisen, liegen drei der vier Äfalschenñ Optionen sehr dicht beieinander und die vierte fällt etwas ab. Ein differenzierteres Bild ergibt sich bei der Betrachtung des ersten Versuchs. Hier liegen drei Ärichtigeñ Antwortoptionen mit Abstand vor einer Gruppe aus zwei Ärichtigenñ und drei Äfalschen Antwortoptionen. Die verbleibende Antwortoption hat eine sehr geringe Wahlhäufigkeit. Insgesamt ergibt sich daraus das Bild einer angemessenen Schwierigkeitsverteilung.

Antwortoption	Bearbeitungen	Versuche	Wahlhäufigkeit insgesamt	Wahlhäufigkeit 1. Versuch
Z3: for (int right=0; right < a.length - 1; ++right)	7	35	51,4 %	57,1 %
Z3: for (int right=0; right <= a.length - 1; right++)	8	41	48,8 %	50,0 %
Z3: for (int right=a.length - 1; 0 < right - 1; right--)	6	39	53,8 %	50,0 %
Z3: for (int right=a.length - 1; 0 < right + 1; right--)	8	49	49,0 %	62,5 %
Z3: for (int right=a.length - 1; 0 <= right; right--)	8	42	50,0 %	75,0 %
Z4: for (int i=0; i <= right - 1; i++)	4	30	60,0 %	25,0 %
Z4: for (int i=0; right > i; i++)	10	23	73,9 %	90,0 %
Z4: for (int i=0; right >= i + 1; i++)	7	48	43,8 %	14,3 %
Z4: for (int i=right - 1; i >= 0; i--)	10	53	43,4 %	20,0 %
Z4: for (int i=0; i <= right + 1; i++)	9	24	41,7 %	33,3 %
Z4: for (int i=0; right >= i - 1; i++)	5	21	38,1 %	40,0 %
Z4: for (int i=right + 1; i >= 0; i--)	9	40	50,0 %	44,4 %
Z4: for (int i=right; i > 0; --i)	7	34	41,2 %	28,6 %
Z4: for (int i=right; i >= 0; i--)	6	31	45,2 %	16,7 %
Z12: for (int j=0; a.length > j; j++)	3	11	63,6 %	100,0 %
Z12: for (int j=0; a.length >= j + 1; j++)	10	60	48,3 %	40,0 %
Z12: for (int j=0; j <= a.length - 1; j++)	7	28	64,3 %	57,1 %
Z12: for (int j=0; j <= a.length + 1; j++)	3	19	36,8 %	0,0 %
Z12: for (int j=0; a.length >= j - 1; j++)	7	66	45,5 %	14,3 %
Z12: for (int j=a.length - 1; j >= 0; j--)	8	45	37,8 %	50,0 %
Z12: for (int j=a.length + 1; j >= 0; j--)	6	24	20,8 %	0,0 %
Z12: for (int j=a.length; j > 0; --j)	5	13	15,4 %	20,0 %
Z12: for (int j=a.length; j >= 0; j--)	7	24	20,8 %	28,6 %

Tab. 3: Statistik der Antwortoptionen für Quiz 4 bzw. die Aufgabe aus Abbildung 2. Grau hinterlegte Antwortoptionen sind richtig im Sinne der Aufgabenstellung

Ein gemischteres, aber in Summe ähnliches Bild zeigt die zweite Aufgabe mit ihren 23 Antwortoptionen (Tabelle 3). Die beiden schwierigsten Ărichtigenñ Antwortoptionen liegen hier mit einer Wahlhäufigkeit über alle Versuche von gut 43 % unterhalb der Wahlhäufigkeiten der drei schwierigsten Äfalschenñ Optionen, die Häufigkeiten von 45-50 % zeigen. Auch die Extremwerte liegen weiter auseinander als bei der ersten Aufgabe. Die Werte für den ersten Antwortversuch verhalten sich entsprechend. Es ist jedoch zu

beachten, dass durch die große Zahl an Antwortoptionen die Anzeigehäufigkeit der einzelnen Optionen insgesamt geringer ist und die statistische Genauigkeit damit sinkt. Es kann trotzdem geschlossen werden, dass auch hier eine im Wesentlichen angemessene Schwierigkeitsverteilung vorliegt, in der jedoch einzelne Antwortoptionen als zu leicht identifiziert werden können. Besonders auffällig sind dabei die beiden Distraktoren für Zeile 12, die im ersten Versuch nie gewählt wurden. Diese enthalten das Fragment a.length + 1, welches für Schleifen über Arrays tatsächlich ungewöhnlich ist. Vermutlich wurden die Distraktoren schon alleine deshalb schnell ausgeschlossen.

Antwortoption	Bearbeitungen	Versuche	Wahlhäufigkeit insgesamt	Wahlhäufigkeit 1. Versuch
Z3: right > 0 anstelle von right >= 0	23	106	54,7 %	65,2 %
Z3: right=a.length anstelle von right=a.length - 1	23	109	49,5 %	34,8 %
Z4: i <= right anstelle von i < right	18	47	34,0 %	27,8 %
Z5: a[i - 1] > a[i] anstelle von a[i] > a[i + 1]	21	107	43,0 %	33,3 %
Z12: j <= a.length anstelle von j < a.length	19	99	40,4 %	26,3 %

Tab. 4: Statistik der Antwortoptionen für Quiz 5 bzw. die Aufgabe aus Abbildung 3. Die grau hinterlegte Antwortoption ist als einzige richtig im Sinne der Aufgabenstellung

Die dritte Aufgabe hat nur fünf Antwortoptionen, von denen eine richtig ist (Tabelle 4). Die Werte entsprechen hier sowohl für die Summe über alle Versuche als auch für den ersten Versuch den bisherigen Beobachtungen. Insgesamt ist daher festzustellen, dass die automatisch erzeugten Antwortoptionen von angemessener Schwierigkeit sind.

4.3 Sind inhaltlich vergleichbare Aufgaben unabhängig von Details der Aufgabenstellung von gleicher Schwierigkeit?

Mit dieser Frage soll untersucht werden, ob es neben den unterschiedlich schwierig zu beurteilenden Antwortoptionen noch weitere Faktoren in den Aufgaben gibt, die einen Einfluss auf die Schwierigkeit haben. Wäre dies der Fall, müssten diese entweder vom Autor oder vom Generator separat berücksichtigt werden. Da unterschiedliche Themen und Konzepte als unterschiedlich schwierig angenommen werden können, werden im Folgenden nur die drei oben genannten Schleifenaufgaben betrachtet. Dabei haben die ersten beiden Beispielaufgaben dieselbe Fragestellung, während die zweite und dritte auf demselben Programmcode basieren. Dies erlaubt einen Vergleich, der Indizien für Einflussfaktoren liefern könnte. Da eine Schwierigkeitsbeurteilung die Fähigkeiten der Testpersonen berücksichtigen muss, werden im Folgenden nur die Daten derjenigen 21 Studierenden berücksichtigt, die alle drei Aufgaben bearbeitet haben.

Aufgabe	Anzahl Bearbeitungen	Anzahl Versuche	Anzahl erfolgreiche Bearbeitungen	Anteil erfolgreiche Bearbeitungen	Schnitt Versuche
Quiz 1	27	122	21	78 %	4,52
Quiz 4	27	136	21	78 %	5,04
Quiz 5	23	110	19	83 %	4,78

Tab. 5: Bearbeitungszahlen für die drei Beispielaufgaben für 21 Studierende, die alle drei Aufgaben bearbeitet haben

Mit Ausnahme von 2 Fällen bei der letzten Aufgabe haben alle 21 Studierenden die Aufgaben erfolgreich bearbeitet (siehe Tabelle 5). Die Anzahl der Versuche unterscheidet sich dabei nur wenig. Die Erfolgsrate bei den ersten beiden Aufgaben ist identisch. Auch wenn die Aussagekraft der Daten aufgrund der Größe der Stichprobe gering ist, kann geschlossen werden, dass der höhere Umfang des Programmcodes in der zweiten Beispielaufgabe keinen entscheidenden Einfluss auf die Schwierigkeit hat. Ferner kann vermutet werden, dass der leichte Unterschied zwischen der zweiten und dritten Beispielaufgabe in der veränderten Aufgabenstellung liegen könnte, wobei diese These zweifellos einer genaueren Untersuchung bedarf. Es ist aber festzuhalten, dass diese drei inhaltlich vergleichbaren Aufgaben auch von annähernd gleicher Schwierigkeit sind.

5 Fazit und Ausblick

In diesem Paper wurde ein Generator zur automatischen Erzeugung von Multiple-Choice-Aufgaben zum Codeverständnis vorgestellt und anhand von drei Beispielaufgaben evaluiert. Alle drei damit verbundenen Forschungsfragen konnten positiv beantwortet werden: Die Aufgaben wurden von die Studierenden ernsthaft und motiviert bearbeitet, die automatisch erzeugten Antwortoptionen sind zum deutlich überwiegenden Teil von einer angemessenen Schwierigkeit und es konnten keine weiteren wichtigen Einflussfaktoren identifiziert werden, die bei der Aufgabenstellung beachtet werden müssen. Der Generator kann damit als reif genug für den Produktivbetrieb und den Einsatz der Aufgaben in einer Lehrveranstaltung angesehen werden.

Für weitere Arbeiten ergeben sich daraus zwei Perspektiven: Auf der technischen Seite können weitere Generierungsmechanismen ergänzt werden, die zusätzliche Antwortoptionen für die vorhandenen Fragen sowie Antwortoptionen für gänzlich neue Fragen erzeugen. Dazu wird es insbesondere nötig sein, weitere Antwortoptionen zu generieren und ihre didaktische Tauglichkeit zu untersuchen, um eine möglichst große Menge sinnvoller Optionen zur Verfügung zu haben. Auf der wissenschaftlichen Seite können weitere Studien zur Schwierigkeit der Aufgaben und zur Verknüpfung mit den zur Lösung der Aufgaben benötigten Kompetenzen durchgeführt werden. Interessant ist dabei auch die Frage, wie sich durch gezielte Auswahl der angebotenen Antwortoptionen bewusst leichtere oder schwierigere Aufgaben erzeugen lassen. Auch die Wirkung von Feedback bei der wiederholten Bearbeitung bzw. mehreren Versuchen wurde für die Aufgaben bisher nicht betrachtet und kann noch untersucht werden.

Danksagung:

Die Autoren danken Alexander Mardorf für die Entwicklung des ersten Prototypen des Generators im Rahmen seiner Bachelorarbeit.

Teile der Forschungsarbeit in diesem Paper wurden vom BMBF unter Förderkennzeichen 01PL16075 gefördert.

Literaturverzeichnis

[BS05] Brusilovsky, P. und Sosnovsky, S. Individualized Exercises for Self-assessment of Programming Knowledge: An Evaluation of QuizPACK J. Educ. Resour. Comput., ACM, 2005, 5.

[LAJ05] Lahtinen, E., Ala-Mutka, K. und Järvinen, H-M. A study of the difficulties of novice programmers. In: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '05). 14-18. DOI: 10.1145/1067445.1067453

[Li01] Lister, R. Objectives and Objective Assessment in CS1, School of Computing and Information Technology, In Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education (2001), Seite 292-296, Charlotte, North Carolina, USA, DOI 10.1145/364447.364605

[LL03a] Lister, R. und Leaney, J. Introductory Programming, Criterion-Referencing, and Bloom, Proceedings of the 34th SIGCSE technical symposium on Computer science education, Seite 143-147, 2003, DOI: 10.1145/611892.611954

[LL03b] Lister, R. und Leaney, J. First Year Programming: Let All the Flowers Bloom, In Proceedings of the fifth Australasian conference on Computing education (ACE '03), Volume 20, 221-230.

[LR98] Lienert, G. A. und Raatz, U. Testaufbau und Testanalyse, BeltzPVU, 1998.

[SAC+13] Sanders, K., Ahmadzadeh, M., Clear, T., Edwards, S., Goldweber, M., Johnson, C., Lister, R., McCartney, R., Patitsas, E. und Spacco, J. The Canterbury QuestionBank: building a repository of multiple-choice CS1 and CS2 questions. In Proceedings of the ITiCSE working group reports conference on Innovation and technology in computer science education-working group reports (ITiCSE -WGR '13). ACM, New York, NY, USA, 33-52. DOI: 10.1145/2543882.2543885

[TG05] Traynor, D. und Gibson, J. P. Synthesis and Analysis of Automatic Assessment Methods in CS1: Generating intelligent MCQs, In Proceedings of the 36th SIGCSE technical symposium on Computer science education, ACM, New York, NY, USA, 495-499, 2005, DOI: 10.1145/1047344.1047502