Empirische Untersuchungen von Lückentext-Items zur Beherrschung der Syntax einer Programmiersprache

Michael Striewe und Matthias Kramer
Universität Duisburg-Essen, paluno – The Ruhr Institute
for Software Technology
Gerlingstraße 16
45127 Essen
michael.striewe@paluno.uni-due.de

Universität Duisburg-Essen, Didaktik der Informatik
Schützenbahn 70
45127 Essen
matthias.kramer@uni-due.de

Abstract: Lückentext-Items auf der Basis von Programmcode können eingesetzt werden, um Kenntnisse in der Syntax einer Programmiersprache zu prüfen, ohne dazu komplexe Programmieraufgaben zu stellen, deren Bearbeitung weitere Kompetenzen erfordert. Der vorliegende Beitrag dokumentiert den Einsatz von insgesamt zehn derartigen Items in einer universitären Erstsemestervorlesung zur Programmierung mit Java. Es werden sowohl Erfahrungen mit der Konstruktion der Items als auch empirische Daten aus dem Einsatz diskutiert. Der Beitrag zeigt dadurch insbesondere die Herausforderungen bei der Konstruktion valider Instrumente zur Kompetenzmessung in der Programmierausbildung auf. Die begrenzten und teilweise vorläufigen Ergebnisse zur Qualität der erzeugten Items legen trotzdem nahe, dass Erstellung und Einsatz entsprechender Items möglich ist und einen Beitrag zur Kompetenzmessung leisten kann.

Keywords: Programmierausbildung, Codeverständnis, C-Test, Lückentext, Kompetenzmessung, Empirische Untersuchung.

1 Einleitung

Beim Erlernen einer Programmiersprache stellt das Erlernen der Syntax oftmals eine Barriere für Anfänger dar [Mc01, SS13]. Da es jedoch nicht ausreichend erscheint, die Beherrschung von Syntaxelementen isoliert zu überprüfen oder aus den Lösungen für komplexe Programmieraufgaben indirekt auf die Beherrschung der Syntax zu schließen, können spezielle Lückentext-Items eingesetzt werden, die an sprachwissenschaftliche C-Test angelehnt sind [SKG17].

Der Einsatz von Lückentext-Items ist dabei grundsätzlich keine neue Idee, sondern wurde bereits in den 1980er Jahren erprobt [SEB82]. Sowohl diese als auch weitere Studien fokussieren jedoch Programmierwissen im weiteren Sinne und damit explizit mehr als die reine Syntax-Beherrschung. Die daraus resultierenden Lückentext-Items sind folglich so gestaltet, dass ein Verständnis der Semantik des vorliegenden Codes notwendig ist, um die Lücken zu schließen. Insbesondere die Ergebnisse von [De11] zu Schwierigkeiten von Anfängern legen jedoch nahe, dass korrekte Programmerstellung unabhängig vom konzeptuellen Wissensstand häufig an der Syntax scheitert. Deshalb soll im Fall von Lückentext-Items zur Beherrschung der Syntax einer Programmiersprache ein tiefgreifendes Verständnis der Semantik nicht notwendig sein. Stattdessen sollen diese Items so konstruiert sein, dass es zur Lösung jeder Lücke ausreicht, die Syntax der Programmiersprache zu beherrschen, d. h. insbesondere Schlüsselwörter zu kennen und Regeln für den Aufbau von Ausdrücken, Anweisungen und Referenzen zu beherrschen. Ein Beispiel für ein solches Item ist in Abbildung 1 zu sehen.

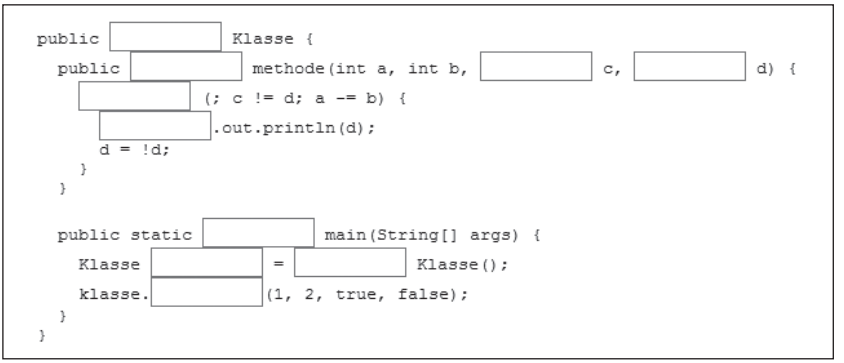

Abb. 1: Beispiel für ein Lückentext-Item in der Programmiersprache Java. In jeder Lücke fehlt genau ein Schlüsselwort oder Bezeichner

Der vorliegende Beitrag dokumentiert die Erstellung und den Einsatz von zehn solcher Items im Rahmen einer Erstsemestervorlesung zur Einführung in die Programmierung am Campus Essen der Universität Duisburg-Essen im Wintersemester 2017/18. Ziel der Diskussion ist die Beantwortung der Frage, ob die Items einen sinnvollen Beitrag zur Kompetenzmessung in der Programmierausbildung leisten können. Der Beitrag ist wie folgt gegliedert: Abschnitt 2 stellt den Forschungskontext und verwandte Arbeiten dar. Abschnitt 3 diskutiert die Erstellung der Items und Abschnitt 4 enthält die statistische Auswertung der empirisch gewonnenen Nutzungsdaten. Abschnitt 5 interpretiert diese Daten und bereitet damit das Fazit in Abschnitt 6 vor.

2 Forschungskontext und Related Work

Neben dem Verstehen des Programmablaufs sowie dem Entwickeln allgemeiner Problemlösungsstrategien spielt das Erlernen einer *Notation,* also der Syntax und Semantik einer Programmiersprache, eine wichtige Rolle in der Programmierausbildung [DB86], ist aber allein nicht ausreichend, um funktionierende Programme zu erstellen [Wi96]. Der korrekte Umgang mit einer Programmiersprache, d. h. Programmierbausteine syntaktisch und semantisch sinnvoll miteinander kombinieren zu können, ist daher mutmaßlich eine notwendige Teilfertigkeit, die für das erfolgreiche Erlernen von Programmierung von wesentlicher Bedeutung ist. Betrachtet man Programmierkompetenz daher als multidimensionales latentes Konstrukt [KHB16], so erscheint es sinnvoll, durch vielfältige Testitems im Rahmen eines formativen Assessments festzustellen, welche Ausprägungen in den verschiedenen Teildimensionen zu (Miss-)Erfolgen führen. Lückentext-Items als Adaption sprachwissenschaftlicher C-Tests können als Baustein in einem größeren Assessment wertvolle Hinweise auf die Leistung in Kompetenzfacetten wie *Verstehen gegebener Quelltexte* bzw. *syntaktisches Verständnis einer Programmiersprache* liefern. Zudem stellen sie durch das Format eine ideale Schnittstelle zwischen den Fertigkeiten *Quelltexte interpretieren* und *Quelltexte produzieren* dar, was ebenfalls vermuten lässt, dass durch einen solchen Test verschiedene Fertigkeiten abgeprüft werden [SKG17].

Lückentext-Items wurden allgemein in der Programmierung bereits in verschiedenen Forschungskontexten eingesetzt. Beispiele sind die Erkennung unterschiedlicher Strategien bei der Programmierung zwischen Novizen und Experten [SEB82] oder die Auswirkung von Geschlecht und Lernstil [LY09]. Im Gegensatz zum vorliegenden Beitrag leiten diese Studien aus den Ergeb-

nissen von Lückentext-Tests weitere Erkenntnisse ab, während sich der vorliegende Beitrag mit der Evaluation des Item-Typs als solchem befasst.

3 Item-Konstruktion

Alle zehn Items, die im Rahmen dieses Beitrags diskutiert werden, wurden manuell mit Hilfe eines Editors erzeugt, der das passende Datenformat für das verwendete Übungssystem erzeugt. Der Editor ist so gestaltet, dass ein Quellcode eingelesen wird und anschließend durch den Aufgabenautor einzelne Wörter in diesem Quellcode durch Anklicken als Lücke ausgewählt werden können. Als Wörter werden dabei primär Schlüsselworte der Programmiersprache behandelt. Im Kontext von Java ist es zudem sinnvoll, Zeichenketten wie `String`, `System` oder `println` äquivalent zu Schlüsselworten zu behandeln, da sie als vorgegebene Typen und Methoden der Java API ebenfalls eine feststehende Bedeutung unabhängig vom Programmkontext haben. Grundsätzlich erlaubt es der Editor jedoch, spezifisch pro Programmiersprache oder sogar pro Quelltext beliebige Zeichenketten als Kandidaten für Lücken zu definieren. Dadurch wird es auch möglich, beliebige Klassen- oder Methodennamen in Lücken umzuwandeln. Dies ist bei der Item-Konstruktion jedoch höchstens dann sinnvoll, wenn deren Lösung trotzdem noch rein syntaktisch aus dem Programmcode geschlossen werden kann.

Zu Beginn der Untersuchung wurde der Versuch unternommen, die Items automatisiert zu erzeugen, indem einfache Regeln für die Erzeugung der Lücken definiert wurden. Nach diesen Regeln hätte dann beispielsweise jedes zweite Schlüsselwort oder jeder dritte Bezeichner automatisch in eine Lücke umgewandelt werden können. Es stellte sich jedoch schnell heraus, dass auf diese Weise erzeugte Items häufig nicht eindeutig lösbar sind. Beispielsweise dürfen Zugriffsmodifikatoren nicht in Lücken umgewandelt werden, da nicht immer aus dem Kontext ersichtlich ist, ob die entstehende Lücke mit `public`, `private` oder `protected` zu füllen ist. Auch das Offenlassen einer solchen Lücke könnte eine syntaktisch gültige Lösung sein. An anderen Stellen könnte das Entfernen eines primitiven Datentyps eine Lücke hinterlassen, die syntaktisch sowohl mit `int` als auch mit `long` oder sowohl mit `double` als auch mit `float` gefüllt werden könnte. Während in diesen Beispielen die Lösungsmenge zumindest endlich ist, können jedoch auch Situationen entstehen, in denen unendlich viele gültige Lösungen möglich sind. Entfernt man beispielsweise den Ausdruck innerhalb einer `return`-Anweisung, kann die entstande-

ne Lücke syntaktisch korrekt mit jedem beliebigen Ausdruck gefüllt werden, der zum Rückgabetyp der Methode passt.

Da das Abfangen derartiger Sonderfälle eine enorme Komplexität in den Erstellungsregeln erfordern würde, wurde auf weitere Versuche zur automatischen Erstellung der Items verzichtet. Stattdessen wurden zehn Items durch manuelle Auswahl von Lücken erstellt. Das in Abbildung 1 gezeigte Item ist auf diese Weise mit den folgenden Begründungen für die Wahl der Lücken entstanden:

- Die erste Lücke verlangt das Schlüsselwort `class`. Dieses ist für Klassendeklarationen zwingend und aus der Zeile ist ersichtlich, dass es sich hier nicht um eine andere Deklaration (Feld oder Methode) handeln kann.
- Die zweite Lücke verlangt das Schlüsselwort `void`. Aus der Zeile ist ersichtlich, dass es sich hier um eine Methodendeklaration handelt, die eine Angabe zum Rückgabetyp benötigt. Aus der Betrachtung der folgenden Zeilen ist ersichtlich, dass die Methode keine `return`-Anweisung enthält und somit `void` die einzig richtige Angabe zum Rückgabetyp sein muss.
- Die dritte und vierte Lücke verlangen jeweils das Schlüsselwort `boolean`. Aus der Betrachtung des Methodenkörpers ist erkennbar, dass `d` vom Typ `boolean` sein muss, damit `!d` ein erlaubter Ausdruck ist. Dann muss auch `c` `boolean` sein, damit auch `c != d` ein erlaubter Ausdruck ist.
- Die fünfte Lücke verlangt das Schlüsselwort `for`. Dieses ist das einzige Schlüsselwort, das die Zeile zu einer syntaktisch gültigen Anweisung ergänzt.
- Die sechste Lücke verlangt den Bezeichner `System`. Dieser ist Teil der Standard-API von Java und wurde im Rahmen der Vorlesung eingeführt. Da der Quellcode keine weiteren Imports deklariert, kann davon ausgegangen werden, dass `System` als einziges den statischen Zugriff auf ein Feld namens `out` ermöglicht.
- Die siebte Lücke verlangt das Schlüsselwort `void`. Hier greift dieselbe Argumentation wie bei Lücke zwei.
- Die achte Lücke verlangt den Bezeichner `klasse`. Dieser wird in der folgenden Zeile verwendet und kann syntaktisch nur hier eingeführt werden.
- Die neunte Lücke verlangt das Schlüsselwort `new`. Dieses ist das einzige Schlüsselwort, das die rechte Seite der Zuweisung zu einem syntaktisch gültigen Ausdruck ergänzt.
- Die zehnte Lücke verlangt den Bezeichner `methode`. Dieser wird in Zeile 2 als Name einer Methode mit passender Signatur eingeführt. Auf dem Objekt `klasse` vom Typ `Klasse` können zwar auch Methoden der

allgemeinen Oberklasse `Object` aufgerufen werden, aber von diesen hat keine eine passende Signatur.

Es ist zu erwarten, dass die Lücken unterschiedlich schwierig zu schließen sind, da sie eine unterschiedlich umfangreiche Betrachtung des Kontexts erfordern. Damit ist ebenso zu erwarten, dass die Items insgesamt unterschiedlich schwierig zu lösen sind, da sie auf unterschiedlichen Quelltexten basieren und somit unterschiedlich schwierige Lücken enthalten.

4 Empirische Untersuchung und statistische Auswertung

4.1 Einsatzszenario und Datenerhebung

Die zehn Lückentext-Items wurden parallel zur Durchführung der Erstsemestervorlesung „Einführung in der Programmierung“ entworfen und in einem elektronischen Übungssystem zur Verfügung gestellt. Das in den Lückentexten verwendete Vokabular des Programmcodes (d. h. die Menge an verschiedenen Schlüsselwörtern und Sprachkonstrukten) orientiert sich daher am Fortgang der Vorlesung und ist bei den ersten Items geringer als bei den später erzeugten. Das erste Item wurde kurz nach Beginn der Vorlesung im Oktober freigeschaltet und bis zum Ende des Semesters von 222 Studierenden insgesamt 350 Mal bearbeitet. Das letzte Item wurde Anfang Dezember freigeschaltet und von 70 Studierenden insgesamt 85 Mal bearbeitet. Insgesamt haben 47 Personen alle 10 Items bearbeitet.

Die Bearbeitung der Items erfolgte auf freiwilliger Basis und ohne Verknüpfung mit Bonuspunkten oder sonstigen Anreizen. Als Rückmeldung erhielten die Studierenden die Information, wie viele Lücken sie korrekt ausgefüllt hatten. Musterlösungen oder Hinweise wurden nicht zur Verfügung gestellt. Alle Einreichungen wurden im elektronischen Übungssystem gespeichert und konnten für die statistische Auswertung herangezogen werden.

4.2 Datenaufbereitung

Für die Auswertung wurden alle Einreichungen zunächst aus dem Übungssystem exportiert und noch einmal unabhängig vom Ergebnis durch das Übungssystem automatisiert analysiert. Dabei wurden insbesondere Eingabefehler der Testpersonen insofern bereinigt, dass führende und folgende Leerzeichen bei der Feststellung der Korrektheit einer Eingabe ignoriert wurden.

Durch die automatische Analyse entstanden insbesondere zwei Arten von Tabellen: (1) Je eine *Lösungstabelle* pro Item mit allen Eingaben der Lücken und deren jeweiliger Häufigkeit. (2) Eine *Gesamttabelle* mit den Ergebnissen derjenigen Studierenden, die alle zehn Items bearbeitet haben. In dieser Tabelle ist pro Lücke dichotom kodiert, ob die Lücke korrekt gefüllt wurde oder nicht. Ferner ist in dieser Tabelle polytom kodiert, wie viele Lücken eine Testperson in jedem Item korrekt gefüllt hat. Alle Tabellen wurden in Excel gespeichert, in die Statistiksoftware R importiert und dort unter Verwendung des Pakets *psych* [Re17] weiter analysiert.

4.3 Deskriptive Statistik – Betrachtung auf Testebene

Von den 47 Testpersonen, die alle zehn Items bearbeitet haben, haben acht Personen die Aufgaben zur Vorbereitung auf die Nachklausur benutzt, d. h. nicht parallel zur Vorlesung sondern nach der Erstklausur bearbeitet. Alle folgenden statistischen Aussagen beziehen sich auf die 47 Personen und die Betrachtung der polytomen Werte. Aufgrund der unterschiedlichen Anzahl der Lücken pro Item (4–12) und den daraus resultierenden möglichen Punktzahlen, war ein direkter Vergleich nicht möglich. Daher erfolgte in einem ersten Schritt eine Normierung der Werte durch Teilen der erreichten Punktzahl pro Item und pro Proband durch die Anzahl der Lücken pro Item. Dadurch ergibt sich für jeden Probanden die prozentuale Korrektheit für jedes Item. Die Ergebnisse dazu sind in Tabelle 1 dargestellt.

Tab. 1: Die normierten Werte der polytomen Rohdaten für die ersten vier Probanden

TP	LT1	LT2	LT3	LT4	LT5	LT6	LT7	LT8	LT9	LT10
1	0,857	1,000	0,400	1,000	1,000	0,750	1,000	0,900	0,857	0,667
2	0,857	1,000	1,000	1,000	0,778	0,750	1,000	0,900	1,000	0,917
3	1,000	1,000	0,800	1,000	1,000	0,750	1,000	0,800	0,857	0,833
4	1,000	1,000	0,800	1,000	1,000	0,750	1,000	0,600	1,000	1,000
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Aus diesen Werten lassen sich nun erste deskriptivstatistische Werte durch die Boxplots zu den jeweiligen Items darstellen. Diese sind in Abbildung 2 festgehalten.

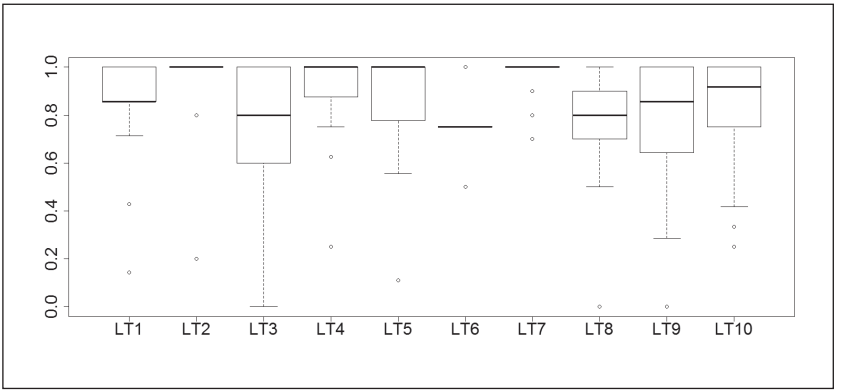

Abb. 2: Boxplot der normierten Ergebnisse. Lückentext-Item 8 (LT8) ist dasjenige, das auch in Abbildung 1 zu sehen ist

Als Maß für die testpsychologische Qualität der zehn Items im Rahmen der internen Konsistenz wurde anschließend Cronbachs Alpha berechnet und ein Wert von 0.77 erreicht. Dabei wiesen die Lückentext-Items 6 und 7 eine auffallend schlechte, tlw. sogar negative Trennschärfe (Korrelation mit dem Gesamtergebnis) auf. Dies weist auf die Notwendigkeit einer inhaltlichen Betrachtung hin und wird im Abschnitt 5 vorgenommen. Auch drei weitere Items (2, 3 und 5) zeigten eine Trennschärfe von unter 0.5, was auf inhaltlicher Ebene zu diskutieren ist.

Aus Homogenitätssicht spricht eine durchschnittliche Inter-Item-Korrelation r_{ii} von 0.21 gerade noch für die Annahme eines eindimensionalen Tests [BD06].

Für alle Testpersonen lagen auch Daten der statischen Tests aus Übungs- und Testataufgaben vor, in die unter anderem Fehlermeldungen des Compilers bzgl. der syntaktischen Korrektheit einer Lösung einfließen. Daher wurde die Korrelation zwischen den Ergebnissen bei den Lückentexten und dem Anteil kompilierfähiger Lösungen in den Programmieraufgaben berechnet. Für Übungsaufgaben wurde eine Trennschärfe von 0.24 erreicht; für Testaufgaben eine Trennschärfe von 0.26. Die kompletten Item-Korrelationen sowie die Korrelationen mit den erreichten Punkten aus Übungen und Testaten sind visualisiert in Abbildung 3.

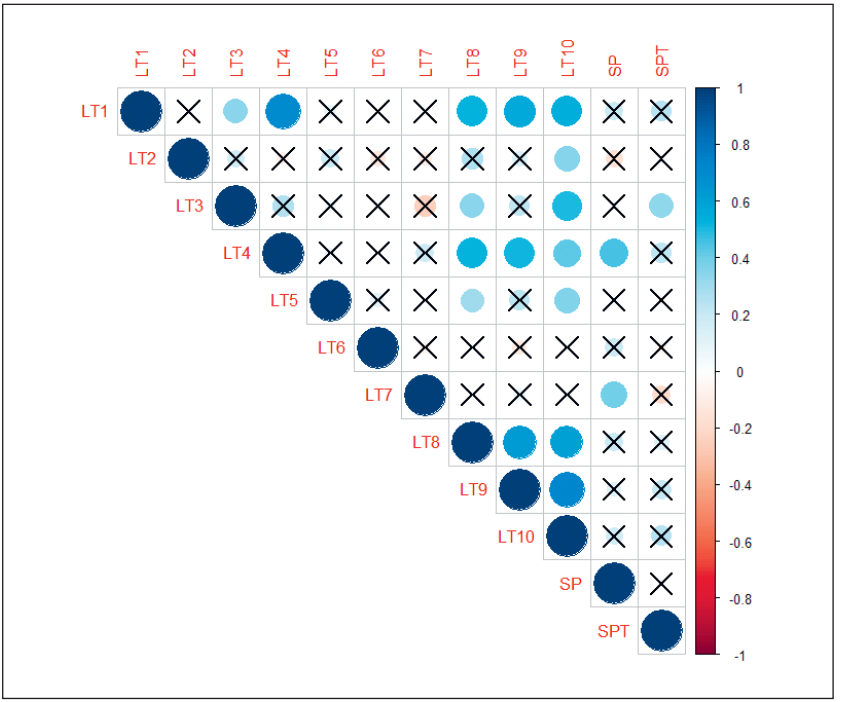

Abb. 3: Visualisierung der Korrelationsmatrix aller standardisierten Items incl. Punktzahlen aus statischen Tests (SP) sowie statischen Tests in Testatsituationen (SPT), mit X sind Korrelationen markiert, die das Signifikanzniveau verletzen ($p < .05$)

4.4 Deskriptive Statistik – Betrachtung auf Itemebene

In einem weiteren Schritt wurden die produzierten Ergebnisse auf Itemebene betrachtet. Für jedes Item ergab sich entsprechend eine Matrix, in der dichotom die produzierten Lösungen für alle Lücken und alle Probanden kodiert waren. Anschließend wurden für alle zehn Items die Kovarianzen sowie die sich daraus ergebenen Korrelationen der Lösung zu den jeweiligen Lücken ausgewertet, sowie die biseriellen Korrelationen zwischen den Lücken und der Item-Gesamt-Punktzahl als Maß für die Trennschärfe der Lücke auf Itemebene berechnet. Letztere wurden mit dem Befehl `biserial.cor()` aus dem R-Paket *ltm* [Ri06] ermittelt. Für die zehn Lücken des Items 8 (siehe Abbildung 1) ergaben sich bspw. folgende Varianzen, Kovarianzen und Korrelationen:

Tab. 2: Kovarianzen (untere Dreiecksmatrix) und Korrelationen (obere Dreiecksmatrix) für jede der zehn Lücken (L1 – L10) in Item 8, Varianzen entsprechen den Kovarianzen einer Lücke mit sich selbst. Die letzte Zeile zeigt die Trennschärfe der Lücke als biserielle Korrelation bzgl. der zu erreichenden Gesamtpunktzahl im Item. Die Nummerierung der Lücken ergibt sich aus dem Auftauchen in normaler Lesart.

	L1	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**	**L8**	**L9**	**L10**
L1	0.097	0.114	−0.171	−0.171	0.130	0.142	0.427	0.395	0.269	−0.028
L2	0.016	0.194	0.161	0.161	0.160	0.171	0.252	0.254	0.360	0.368
L3	−0.027	0.036	0.253	1	−0.172	0.186	0.164	0.048	0.023	0.211
L4	−0.027	0.036	0.253	0.253	−0.172	0.186	0.164	0.048	0.023	0.211
L5	0.020	0.035	−0.043	−0.043	0.241	0.240	0.116	−0.109	0.166	0.125
L6	0.012	0.021	0.026	0.026	0.033	0.080	0.483	0.268	0.313	0.011
L7	0.019	0.016	0.012	0.012	0.008	0.020	0.021	0.326	0.699	0.267
L8	0.047	0.043	0.009	0.009	−0.020	0.029	0.018	0.144	0.465	0.151
L9	0.017	0.032	0.002	0.002	0.017	0.018	0.021	0.036	0.042	0.132
L10	−0.004	0.069	0.045	0.045	0.026	0.001	0.017	0.025	0.012	0.183
r_{bis}	0.292	0.600	0.600	0.600	0.298	0.504	0.600	0.474	0.518	0.521

5 Interpretation & Schlussfolgerungen

Zuerst bleibt festzuhalten, dass die Populationsgröße verallgemeinernde Spekulationen verbietet. Dennoch können erste Hinweise auf sehr gute bzw. schlechte oder nicht funktionierende Items gewonnen werden. Diese werden zuerst auf Ebene des gesamten Testes und anschließend auf Itemebene diskutiert.

5.1 Interpretation auf Testebene

Insgesamt lässt sich erkennen, dass die Items auf Gesamttestebene für die untersuchte Population relativ leicht zu lösen waren. Dies wird ersichtlich am Deckeneffekt, der sich im Boxplot widerspiegelt. Dies kann vor dem Hintergrund der freiwilligen Bearbeitung einerseits dahingehend interpretiert werden, dass die zehn Items nur von den Studierenden bearbeitet wurden, welche

sich auch selbst zugetraut haben, alle Items zu bearbeiten und diese auch gut zu lösen. Studierende, für die der Umgang mit Syntax ein Problem darstellt, sind daher evtl. nur teilweise oder gar nicht durch die Items erfasst worden. Wie zuverlässig diese Interpretation ist, könnte in zukünftigen Untersuchungen durch Betrachtung externer Faktoren ergänzt werden, wie bspw. der Abschlussnote. Die Ergebnisse können deshalb auch nicht als Nachweis dafür gewertet werden, dass der Umgang mit Java-Syntax als besonders einfach zu bewerten ist. Gegen eine solche Interpretation sprechen auch einschlägige Ergebnisse der fachdidaktischen Forschung. So konnten [De11] zeigen, dass Probleme mit der Syntax unabhängig vom konzeptuellen Wissen auf allen Fähigkeitsebenen zu finden sind. Neben dem Stichprobenumfang sind des Weiteren auch die Korrelationen mit den Punkten aus den statischen Tests zu gering. D. h. Probanden, die hohe Werte in den Items erreicht haben, lieferten nicht zwangsweise syntaktisch fehlerfreie Testataufgaben ab (und umgekehrt). Daraus folgt, dass die Items nicht als Prädiktor zulässig sind, sondern allenfalls als weiteres Hilfsinstrument um Rückschlüsse auf die Ausprägung der Programmierkompetenz zu ziehen. Gleichzeitig muss jedoch auch erwähnt werden, dass die Punktzahlen aus statischen Tests allgemein nur ein wenig geeignetes Maß für die Beherrschung der Syntax sein können, da hier nur derjenige Anteil an Lösungen erfasst wird, die ins Übungssystem hochgeladen werden. Über die Häufigkeit von Syntaxfehler insgesamt können daraus nur sehr begrenzte Rückschlüsse gezogen werden.

Bei der Analyse der Korrelationen untereinander fällt zudem auf, dass einzelne Items sowohl schlecht mit dem Test insgesamt korrelieren, und daher eine schlechte Trennschärfe (siehe bspw. Item 6) aufweisen, als auch sehr schlecht mit einer Großzahl an anderen Items korrelieren. Dies muss nicht zwangsweise auf ein gänzlich ungeeignetes Item hinweisen, sondern könnte auch ein Indiz für eine zu geringe Anzahl oder eine zu homogene Gruppe von Probanden sein. Bei zukünftigen höheren Probandenzahlen wäre es nach Überarbeitung problematischer Items zu überlegen, mittels faktorenanalytischer Maßnahmen die Itemanzahl zu minimieren und nur solche Items zuzulassen, die sowohl hohe Trennschärfen, als auch hohe durchschnittliche Item-Korrelationen garantieren.

Exemplarisch wurde dies bereits über iteratives Entfernen derjenigen Items durchgeführt, die nur sehr geringe positive ($r < .1$) bzw. negative Trennschärfen aufweisen. Als Richtmaß für die Verbesserung wurden wiederum Cronbachs Alpha sowie die durchschnittliche Inter-Item-Korrelation r_{ii} betrachtet. Durch Entfernen der Items 6 und 7 ergeben sich ein Alpha von 0.81 sowie ein $r_{ii} = .34$, was durchaus als Verbesserung des Tests gewertet werden

kann. Entfernt man zusätzlich noch die Items mit mittleren Trennschärfen bzw. betrachtet nur solche, mit sehr starken Trennschärfen (Items 1, 4, 8, 9, und 10), ergibt sich zwar ein Alpha von 0.87 aber dafür auch eine durchschnittliche Inter-Item-Korrelation r_{ii} von 0.58, was bereits als partiell redundanter Test interpretiert werden kann [BD06, S. 220]. Von einer Reduktion der Testitems bis zu dieser Stufe muss daher abgesehen werden.

5.2 Interpretation auf Itemebene

Auf Itemebene gilt es mehrere Punkte zu beachten. Dazu gehört zum einen die Betrachtung der Lückenanzahl als möglicher Einfluss auf die Sekundärvarianz. Wie im Boxplot zu sehen ist, sind die Items 2 und 6 nicht nur besonders leicht zu lösen, sie leisten auch keinen Beitrag zur Unterscheidung zwischen den Probanden (bis auf wenige Ausnahmen). Dies kann durch die geringe Anzahl der Lücken erklärt werden. Bereits wenige korrekte Einträge liefern hohe prozentuale Anteile. Im Gegensatz bspw. zum Item 8 mit zehn Lücken, weist Item 2 nur fünf Lücken und Item 6 nur vier Lücken auf. Es muss daher bereits auf dieser Ebene hinterfragt werden, inwiefern sich die Items durch die vorgenommene Normierung vergleichen lassen. In zukünftigen Untersuchungen muss daher berücksichtigt werden, welchen Einfluss die Variation der Lückenanzahl potentiell auf das Ergebnis und dessen Interpretation hat.

Für eine detailliertere Betrachtung wurden zum anderen die Kovarianzen und die daraus resultierenden Korrelationen der Items untereinander betrachtet (siehe Tabelle 2 beispielhaft für Item 8). Niedrige Varianzen lassen an dieser Stelle auf Lücken schließen, die entweder offensichtliche oder sehr schwere Wörter verlangen. So musste in der siebenten Lücke das Schlüsselwort `void` hinzugefügt werden. Dies wurde aber nicht durch das Fehlen des Schlüsselwortes `return` im Methodenrumpf signalisiert, sondern vermutlich bei einem Großteil durch den stets gleichen Aufbau des `main`-Methodenkopfes induziert. Bestätigt wird dies durch die eher niedrige Korrelation von 0.252 mit der Lücke 2, die ebenfalls das Schlüsselwort `void` verlangt. Die siebente Lücke prüft also eher, wie gut jemand den Methodenkopf der `main`-Methode in Java auswendig gelernt hat. Es ist daher wenig verwunderlich, dass sich gute Korrelationen mit den Lücken 1, 6 und 9 ergeben, da diese ebenfalls nach bestimmten Schlüsselworten in „Standard-Situationen“ fragen (Klassendeklaration, Ausgabe auf der Konsole, Objektinstanziierung). Diese Lücken lassen sich ausfüllen, ohne den restlichen Quelltext zu betrachten. Ebenfalls wenig überraschend sind die perfekten Korrelationen der Lücken 3

und 4 miteinander, da beide dieselbe Lösung im selben Kontext verlangen. Interessanter sind jedoch negative Korrelationen. Diese geben zwar Hinweise auf problematische Kombinationen, verbieten jedoch eine generelle Identifikation des problematischen Konzeptes. Für Item 8 ergeben sich bspw. (schwache) negative Korrelationen zwischen der ersten Lücke sowie den Lücken 3, 4 und 10. Gerade die erste Lücke weist auch bei weiteren Items negative Korrelationen mit den restlichen Lücken auf, was den Schluss zulässt, dass häufig erste, einfache Schlüsselworte gerade noch gewusst werden, jedoch mit steigender Komplexität keine Antworten mehr gegeben werden können. Negative Korrelationen sind jedoch zusätzlich insofern problematisch, als dass dadurch möglicherweise miteinander vermischte Konzepte abgefragt werden. Im Item 7 wurden bspw. fünf von zehn Lücken von allen Probanden korrekt gelöst und wiesen daher gar keine Varianz auf, was wiederum die schlechte Trennschärfe des Items erklärt.

Schlussendlich wurden für jedes Item auch die biseriellen Korrelationen als Maß für die Trennschärfen der Lücken bzgl. der Gesamtpunktzahl pro Item betrachtet. Diese weisen mittlere bis gute Trennschärfen auf, so dass der zukünftige Ausschluss potentieller Lücken inhaltsgeleitet geschehen muss. Dabei ist des Weiteren darauf zu achten, welche Schlüsselworte an welchen Positionen abgefragt werden, um nicht möglicherweise auswendig gelernte Muster, sondern die syntaktisch korrekten Inhalte abzufragen.

6 Fazit und Ausblick

Es wurden zehn Items nach dokumentierten Richtlinien erstellt und in einer ersten empirischen Erhebung semesterbegleitend von 47 Testpersonen vollständig bearbeitet. Die Items wurden mittels Berechnung der internen Konsistenz sowie der mittleren Inter-Item-Korrelation auf Testebene und mittels Kovarianz- und Korrelationsberechnungen auf Itemebene ausgewertet. Dabei wurden zum einen für die untersuchte Population schlecht funktionierende Items identifiziert und durch iteratives Entfernen eine Subgruppe von Items gebildet, die starke Korrelationen untereinander aufweisen. Teilweise sind die daraus resultierenden Korrelationen allerdings so stark, dass über eine redundante Testung spekuliert werden muss. Zum anderen wurden Lücken mit niedriger Varianz und schlecht oder gar negativ korrelierenden Lücken identifiziert. Der aus der Itemkonstruktion gewonnene Eindruck, dass die Erstellung der Items nicht automatisiert erfolgen kann, wurde damit bestätigt. Durch die Spezifizierung auf syntaktische Konstrukte sowie das erste Auf-

zeigen einer Subgruppe von Items mit guter Inter-Item-Korrelation und guter interner Konsistenz vertreten die Autoren die Auffassung, dass Lückentext-Items sinnvoll zu einer gesamtheitlichen Kompetenzmessung beitragen können. Im Rahmen weiterer Analysen ist zu untersuchen, inwiefern die Items genutzt werden können, um bessere und gezieltere Rückschlüsse auf generelle Fehlvorstellungen und Probleme der Lernenden zu ziehen, was wiederum zu einer möglichen Verbesserung der Lehre beiträgt.

Des Weiteren wurden nützliche Erkenntnisse für die Item-Konstruktion abgeleitet. So ist im Rahmen der Varianzanalyse zu überlegen, ob weiterhin Items eingesetzt werden, welche unterschiedliche Anzahlen von Lücken aufweisen, oder ob eine andere Normalisierung bzw. Standardisierung zu einer hinreichenden Verbesserung der Datenqualität führen würde. Möglicherweise wird die Schwierigkeit eines Items teilweise durch die abgefragten Inhalte und teilweise durch die Lückenanzahl beeinflusst, welche als Sekundärvarianz in die Gesamtvarianz eingeht. Die geforderte Lückenreduktion kann dabei durch Verwendung verschiedener Kriterien, bspw. der Varianz einzelner Lücken selbst, der Korrelation mit anderen Lücken, der Trennschärfe sowie aus inhaltlichen Aspekten erfolgen. Für die nächsten Experimente erscheint es daher insbesondere sinnvoll, mehrere Items mit identischer Anzahl von Lücken und unter Verzicht auf Lücken in bekannten Mustern zu erstellen. In jedem Fall muss bei einem erneuten Einsatz der Items die Unkorreliertheit der Fehlervariablen und der essentiellen τ-Äquivalenz als Voraussetzung für die Verwendung von Cronbach's Alpha überprüft werden, um die Teststatistik sinnvoll interpretieren zu können. Dass Interpretationen von Alpha trotz Verletzung dieser Voraussetzungen valide sein können, wurde berichtet in [Wa15].

Die bisher erhaltenen Rohdaten lassen zusätzlich im Rahmen einer Sekundäranalyse die Auswertung nach Konzepten zu, die in mehreren Items vorkommen.

Die Items sind über den Erstautor für den Einsatz in nicht-kommerziellen Untersuchungen kostenfrei erhältlich.

Literaturverzeichnis

[BD06] Bortz, J.; Döring, N.: Quantitative Methoden der Datenerhebung. In (Bortz, J.; Döring, N. Hrsg.): Forschungsmethoden und Evaluation. 4. Auflage, Springer, Berlin, S. 137–293, 2006, S. 220.

[De11] Denny, P. et al.: Understanding the syntax barrier for novices. In: Proceedings of the 16th annual joint conference on Innovation and technology in computer science education (ITiCSE), Darmstadt 2011. ACM, New York, S. 208–212, 2011.

[KHB16] Kramer, M.; Hubwieser, P.; Brinda, T.: A Competency Structure Model of Object-Oriented Programming. In: Proceedings of the International Conference on Learning and Teaching in Computing and Engineering (LaTICE), Mumbai. IEEE, S. 1–8, 2016.

[LY09] Lau, W.; Yuen, A.: Exploring the effects of gender and learning styles on computer programming performance: implications for programming pedagogy. British Journal of Educational Technology 40/4, S. 696–712, 2009.

[Mc01] McCracken, M. et al.: A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. ACM SIGCSE Bulletin 33/4, S. 125–180, 2001.

[Re17] Revelle, W.: psych: Procedures for Personality and Psychological Research. Northwestern University, Evanston, Illinois, 2017.

[Ri06] Rizopoulos, D.: ltm: An R package for Latent Variable Modelling and Item Response Theory Analyses. Journal of Statistical Software 17/5, S. 1–25, 2006.

[SEB82] Soloway, E.; Ehrlich, K.; Bonar, J.: Tapping into Tacit Programming Knowledge. Proceedings of the 1982 Conference on Human Factors in Computing Systems. ACM, New York, S. 52–57, 1982.

[SS13] Stefik, A.; Siebert, S.: An empirical investigation into programming language syntax. ACM Transactions on Computing Education (TOCE) 13/4, 19, 2013.

[SKG17] Striewe, M.; Kramer, M.; Goedicke, M.: Ein Lückentext-Test zur Beherrschung einer Programmiersprache. In: DeLFI 2017 – Die 15. e-Learning Fachtagung Informatik der Gesellschaft für Informatik, Volume 273 of Lecture Notes in Informatics. Köllen, Bonn, S. 261–266, 2017.

[Wa15] Warrens, M.J.: Some relationships between Cronbach's alpha and the Spearman-Brown formula. Journal of Classification 32/1, S. 127–137, 2015.

[Wi96] Winslow, L.: Programming Pedagogy – A Psychological Overview. ACM SIGCSE Bulletin 28/3, S. 17–22, 1996.