Bewertungsaspekte und Tests in Java-Programmieraufgaben für Graja im ProFormA-Aufgabenformat

Robert Garmann[1], Peter Fricke[2] und Oliver J. Bott[3]

Abstract: Das ProFormA-Aufgabenformat wurde eingeführt, um den Austausch von Programmieraufgaben zwischen beliebigen Autobewertern (Grader) zu ermöglichen. Ein Autobewerter führt im ProFormA-Aufgabenformat spezifizierte „Tests" sequentiell aus, um ein vom Studierenden eingereichtes Programm zu prüfen. Für die Strukturierung und Darstellung der Testergebnisse existiert derzeit kein graderübergreifender Standard. Wir schlagen eine Erweiterung des ProFormA-Aufgabenformats um eine Hierarchie von Bewertungsaspekten vor, die nach didaktischen Aspekten gruppiert ist und entsprechende Testausführungen referenziert. Die Erweiterung wurde in Graja umgesetzt, einem Autobewerter für Java-Programme. Je nach gewünschter Detaillierung der Bewertungsaspekte sind Testausführungen in Teilausführungen aufzubrechen. Wir illustrieren unseren Vorschlag mit den Testwerkzeugen Compiler, dynamischer Softwaretest, statische Analyse sowie unter Einsatz menschlicher Bewerter.

Keywords: e-Assessment, Graja, Java, Programmieraufgabe, Bewertungsmaßstab, Bewertungsaspekt, Grader, Autobewerter, ProFormA-Aufgabenformat.

1 Einleitung

Von Studierenden im Rahmen des e-Assessments eingereichte Lösungen zu Programmieraufgaben können durch sog. Autobewerter (Grader) automatisch oder zumindest teilweise automatisch bewertet werden. Häufig setzen Autobewerter dabei Werkzeuge der Software-Qualitätssicherung ein, wie etwa xUnit-Testframeworks oder Werkzeuge der statischen Codeanalyse. Normalerweise werden solche Werkzeuge von Softwaretechnik-Profis genutzt. Ein- und Ausgaben dieser Werkzeuge sind in unveränderter Form in der Regel nicht unmittelbar für Programmiereinsteiger verständlich. Ein zentrales Qualitätskriterium eines Autobewerters ist eine für Studierende verständliche Aufbereitung der Ausgabe der eingesetzten Testwerkzeuge. Evaluierungen wie [WGH+15] belegen, dass eine für einreichende Studierende übersichtliche Ergebnisdarstellung entscheidend für einen lernförderlichen Einsatz von Autobewertern ist.

Für den Austausch von Programmieraufgaben zwischen verschiedenen Autobewertern und/oder Lernmanagementsystemen (LMS) existiert mittlerweile ein XML-basiertes

[1] Hochschule Hannover, Fakultät IV – Wirtschaft und Informatik, Ricklinger Stadtweg 120, 30459 Hannover, robert.garmann@hs-hannover.de
[2] Hochschule Hannover, ZSW – E-Learning Center, Expo Plaza 4, 30539 Hannover, peter.fricke@hs-hannover.de
[3] Hochschule Hannover, Fakultät III – Medien, Information und Design, Expo Plaza 12, 30459 Hannover, oliver.bott@hs-hannover.de

Aufgabenaustauschformat ProFormA [SSM+15], welches als wesentlichen Bestandteil *Tests* als Domänenobjekte vorsieht. Tests fokussieren auf die auszuführenden Werkzeuge und deren (technische) Konfiguration. In Ermangelung einer Möglichkeit, die Ergebnisdarstellung losgelöst von den Tests zu spezifizieren, gerät das vom Autobewerter erzeugte Feedback an den einreichenden Studierenden häufig zu einer bloßen Aneinanderreihung der Ausgaben der aufgerufenen Werkzeuge. Als Forschungsfrage ergibt sich, wie eine thematisch strukturierte und lernförderliche Darstellung der Bewertungsergebnisse erreicht werden kann, ohne dass der Vorteil der Nutzung bestehender und ausgereifter Softwaretechnikwerkzeuge aufgegeben werden muss.

Ausgangspunkt der Betrachtung dieser Forschungsfrage ist das ProFormA-Aufgabenformat, das neben der Spezifikation von *Tests*, die im Rahmen der Bewertung einer Aufgabenlösung ausgeführt werden, die Vorgabe von Bewertungshinweisen (sog. gradinghints) erlaubt. In diesem Beitrag schlagen wir ergänzend zu den Tests, die durch Ausführung oder Teilausführung von Werkzeugen Prüfresultate erzeugen, als Startpunkt einer lernförderlichen Strukturierung des Feedbacks eines Graders die Einführung eines weiteren Domänenobjekts im ProFormA-Aufgabenformat vor: den *GradingAspect*. Die durch dieses Domänenobjekt bezeichneten Bewertungsaspekte spezifizieren die Struktur der Bewertung einer Einreichung und zwar losgelöst von der Testausführungs-Spezifikation. Bewertungsaspekte sind hierbei quasi „orthogonal“ als eine von der Testdimension unabhängige zweite Achse einer Matrix zu verstehen. Forschungsmethodisch zeigen wir an einem Fallbeispiel auf, welche Änderungen an bestehenden Informationssystemen zur Erreichung des Forschungsziels führen. Wir führen diese Änderungen an dem konkreten System Graja[4] [Gar15] durch und skizzieren die erreichbaren Ergebnisse. Ein ausführlicher Forschungsbericht ist unter [Gar16] verfügbar.

2 Tests im ProFormA-Aufgabenformat

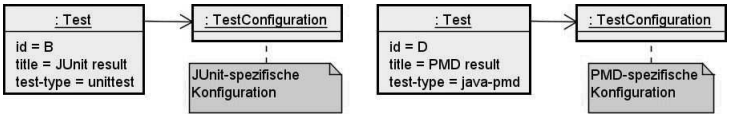

Abb. 1: Domänen-Instanzmodell für einen JUnit- und einen PMD-Test[5]

Ein *Test* im ProFormA-Format definiert eine automatisierte Prüfung der studentischen Einreichung. Gegenwärtig (Version 1.0.1) enthält die Liste der *Testtypen* Tests zur Compilierung, zur Quellcodeanalyse, zur Bytecodeanalyse, für (dynamische) Software-Tests, und für weitere Werkzeuge wie Codeabdeckungsanalyse und Anonymitätsprüfung. Ein Test besitzt neben seinem Testtyp eine testspezifische *Testkonfiguration* sowie einen *Titel* (vgl. Abb. 1). Der Titel wird von einem Autobewerter genutzt, um die vom Test generierte Ausgabe gegenüber dem einreichenden Studenten mit einem aussagekräftigen

[4] http://graja.hs-hannover.de
[5] http://junit.org/ bzw. https://pmd.github.io/

Titel zu versehen. Bewertungsvorgaben, wie etwa erreichbare Punkte, sind im ProFormA-Format nicht standardisiert, können aber in einem separaten, graderspezifischen Domänenbereich (*grading hints*) angegeben werden.

3 Bewertungsaspekte

Lehrpersonen geben beim formativen Assessment Feedback zu funktionalen Aspekten, aber häufig auch zu Qualitätsaspekten wie Wartbarkeit, Effizienz, etc. Das Feedback wird dabei meist unter gedachten Überschriften gruppiert. Eine mögliche Gruppierung bieten die folgenden, teilweise einer Softwarequalitätsnorm [ISO11] entlehnten Überschriften, wobei Feedback zu den drei erstgenannten Bereichen bei Programmieranfängern häufig den größten Raum einnimmt. Neben der Überschrift nennen wir beispielhaft einige Aspekte, zu denen Feedback gegeben werden kann:

Überschrift	**Aspekt**	
Sprache und Syntax	1a	Ist das studentische Programm syntaktisch korrekt?
	1b	Werden Bezeichner konventionsgemäß verwendet?
	1c	Ist das Codelayout (Einrückung, Leerräume) einheitlich / lesbar?
Funktion	2a	Liefert das Programm korrekte Ausgaben?
	2b	Werden auch Grenzfälle und Fehleingaben berücksichtigt?
	2c	Werden funktionale Vor- und Nachbedingungen und funktionale Invarianten eingehalten?
Wartbarkeit	3a	Wurde der Code angemessen kommentiert?
	3b	Ist redundanter Code vorhanden?
	3c	Werden Grundprinzipien wie lose Kopplung / hohe Kohäsion berücksichtigt?
	3d	Ist der Code lesbar?
Effizienz	4a	Ist das Laufzeitverhalten und der Ressourcenverbrauch des Programms für große Eingaben angemessen?
Sicherheit	5a	Werden Daten vor unberechtigten Zugriffen geschützt?
Benutzbarkeit	6a	Funktioniert die Benutzungsschnittstelle wie angefordert?
	6b	Entspricht die Benutzungsoberfläche softwareergonomischen Gestaltungsgrundsätzen?

Tab. 1: Beispielhafte und gruppierte Bewertungsaspekte

Für Grader ist anzustreben, Feedback strukturiert zu liefern, z. B. wie in Tab. 1, ergänzt um erreichte (Teil-)punkte. Alternative Strukturierungen von Bewertungsaspekten sind etwa die Fehlerbehebungsschwierigkeit, die vermittelten Programmierkompetenzen, oder schlicht die Teilaufgaben-Nummern. Ein Autobewerter sollte der Lehrperson erlauben, das Feedback dem Lehrkontext entsprechend und unabhängig von der Testausführungssequenz zu strukturieren. Graja, der Autobewerter den wir im Rahmen dieses Beitrages weiter entwickelt haben, ermöglicht ein solches Feedback.

4 Orthogonalität von Tests und Bewertungsaspekten

Manche der in Tab. 1 genannten Aspekte lassen sich gut durch Testwerkzeuge automatisiert prüfen. Ein Compiler etwa ist ideal geeignet zur Prüfung des Aspekts 1a. Statische Code- und Bytecodeanalyse-Werkzeuge eignen sich z. B. für die Prüfung der Aspekte 1b, 1c, 2c[6], 3a-c und 5a. Einem dynamischen Softwaretest sind die Aspekte 2a-c, 4a und 6a[7] gut zugänglich. Schließlich bleiben einige Aspekte übrig, die teilweise oder ausschließlich durch menschliche Gutachter[8] bewertet werden sollten: 1c, 3d, 5a, 6b. Dabei ist die o. g. Zuordnung von Testwerkzeugen zu Bewertungsaspekten nicht fix: Für jeden Aspekt lassen sich ggf. geeignetere, spezialisierte Werkzeuge und Algorithmen zur Bewertung des betreffenden Aspekts finden. Die obige Zuordnung von Testwerkzeugen zu Bewertungsaspekten wurde vorgenommen, um zu verdeutlichen, wie jedes Testwerkzeug Bewertungsbeiträge zu verschiedenen Aspekten unter verschiedenen Überschriften leisten kann. Bewertungsaspekte und Tests lassen sich daher als zwei Dimensionen einer Matrix auffassen (vgl. Tab. 2).

↓ Testwerkzeug Aspektgruppe →	1	2	3	4	5	6
Compiler	a	-	-	-	-	-
Statische Code-/Bytecodeanalyse	b,c	c	a,b,c	-	a	-
Dynamischer Softwaretest	-	a,b,c	-	a	-	a
Mensch	c	-	d	-	a	b

Tab. 2: Von ausgewählten Testwerkzeugen geleistete Bewertungsbeiträge zu den Aspekten

5 Vorschlag zur Abbildung in der Aufgabenbeschreibung

Für die Abbildung der Bewertungsaspekte in Tab. 1 bietet sich insb. vor dem Hintergrund potenziell weiterer Unterteilungen von Bewertungsaspekten eine Baumstruktur an, deren Blätter vom Domänen-Entitätstyp *GradingAspect* sind. Innere Knoten sind *GradingAspectGroup*s. Beide Objekte besitzen einen Titel und ggf. eine Beschreibung[9].

*Test*s können nun in Test-Teilausführungen (Subtests) aufgebrochen werden, um sie der Referenzierung durch GradingAspects zugänglich zu machen. Für Teilausführungen können zusätzliche Tests mit einem angepassten Testtyp eingesetzt werden. Subtests

[6] Bspw. ist die Einhaltung des Java-hashCode/equals-Kontrakt prüfbar durch die PMD-Rule OverrideBothEqualsAndHashcode.

[7] Die Funktion kann unter Einsatz von Drittbibliotheken wie UIspec4J dynamisch getestet werden.

[8] Menschliche Bewerter als „Testwerkzeug“ explizit vorzusehen kann vom Autobewerter oder vom LMS genutzt werden, um zum einen den Bewertungs-Workflow durch Nachrichten an menschliche Bewerter zu steuern, und zum anderen um Punkte als Platzhalter für von Menschen zu vergebende Teilpunkte zu reservieren und für den einreichenden Studenten sichtbar auszuweisen.

[9] Darüber hinaus ist denkbar, Bewertungsmaßstäbe wie einfache Summengewichte oder kompliziertere Verrechnungsregeln an den Knoten des Baumes zu modellieren (vgl. auch Abschnitt 6). In diesem Beitrag konzentrieren wir uns jedoch auf das qualitative, meist textuelle Feedback eines Testwerkzeuges und lassen das quantitative Feedback (Punkte, Note) zunächst unberücksichtigt.

müssen einem übergeordneten Test und dessen Testkonfiguration über ein neu einzuführendes Attribut *parentId* zugeordnet werden. Der Titel eines Tests bezeichnet nun den technischen Vorgang der Werkzeugdurchführung und nicht notwendigerweise einen didaktischen Hintergrund. Aus diesem Grunde ist der Titel optional.

Ein Beispiel soll dies verdeutlichen. Eine JUnit-Testklasse *de.hsh.example.Grader* besitze mehrere Testmethoden. Eine davon namens *attrsShouldBePrivate* untersuche mittels Java Reflection, ob alle Instanzattribute der eingereichten Klassen privat deklariert sind. Es werden mehrere Instanzen des Domänentyps *Test* zur Beschreibung der Aufgabe genutzt. Das in Abb. 2 mit der *id* B bezeichnete Objekt spezifiziert die Durchführung von JUnit für die o. g. Testklasse. Das Objekt B.i spezifiziert, dass der JUnit runner das Testergebnis für die o. g. Testmethode als Ergebnis einer Teilausführung separat speichern und auf Anfrage zur Verfügung stellen muss. Weitere Objekte spezifizieren die Ausführung des Compilers (A) sowie die nachträgliche menschliche Bewertung (C).

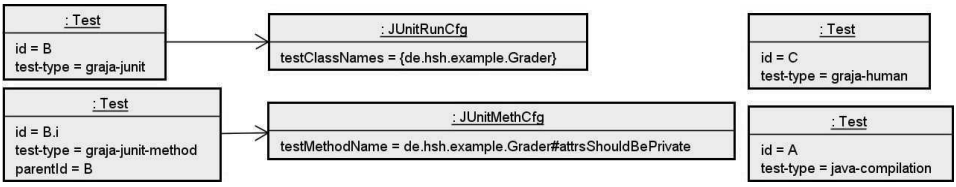

Abb. 2: Domänen-Instanzmodell für detaillierte Test-Teilausführung[10]

Um nun ein komplettes Bewertungsschema einer Aufgabe wie es in Tab. 1 skizziert ist in dem *grading-hints*-Bereich des ProFormA-Formats unterzubringen, werden mehrere Instanzen der Domänentypen *GradingAspect* und *GradingAspectGroup* benötigt, die in Abb. 3 aus Platzgründen nur unvollständig dargestellt sind. Durch Verweise auf Test-(Teil-)Ausführungen (*testref-id*) können die dort zwischengespeicherten Resultate bei der Ermittlung einer Gesamtbewertung herangezogen werden. Ein konkretes Beispielfeedback, welches durch Graja unter Einsatz einer solchen Hierarchie von Bewertungsaspekten erzeugt wurde, ist in [Gar16] dokumentiert.

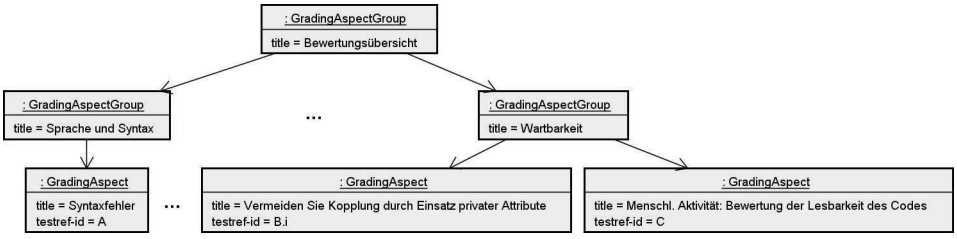

Abb. 3: Domänen-Instanzmodell für Bewertungsaspekte. Das Blatt ganz links bildet den Aspekt 1a ab, das in der Mitte ist dem Aspekt 3c zugeordnet, das rechte Blatt dem Aspekt 3d.

[10] Die im ProFormA-Format vorgesehene *test-configuration* mit graderspezifischen Erweiterungen stellen wir hier verkürzt durch einen einfachen UML-Objektlink dar.

6 Zusammenfassung und Ausblick

Wir haben den Unterschied zwischen Testausführung auf der einen Seite und Interpretation eines Testergebnisses als Bewertungsaspekt und Ausgangspunkt eines strukturierten Feedbacks an den Lernenden erörtert. Die verschiedenen Sichten auf ein Testergebnis führten zum Vorschlag, im ProFormA-Aufgabenformat neben Domänenobjekten zur Spezifikation einer Testausführung zusätzliche Domänenobjekte zur Spezifikation der Bewertungsaspekte vorzusehen und damit ein strukturiertes Feedback vorzubereiten.

Offen geblieben sind a) Fragen der Darstellung eines Teilausführungsergebnisses eines Tests sowie b) die Frage, wie quantitative Bewertungen an inneren Knoten vom Typ *GradingAspectGroup* aggregiert werden können. Zu a) ist zu bemerken, dass in Graja ein Test-(Teil-)Ausführungsergebnis derzeit als zusammengesetzter Datentyp mit den Elementen Erfolgsgrad (0-100%) und Kommentar gespeichert wird. Die Entwicklung eines grader-übergreifend verwendbaren Antwortformats mit Bewertungsergebnissen ist Gegenstand aktueller Forschung im eCULT[11]-Projekt. Zu b) nutzt Graja in der Hierarchie der Bewertungsaspekte zurzeit gewichtete Summen der Bewertungsergebnisse in Unteraspekten. Letzteres erleben wir als in bestimmten Bewertungssituationen zu unflexibel und soll zukünftig überarbeitet werden.

Literaturverzeichnis

[WGH+15] Werner, P.; Garmann, R.; Heine, F.; Kleiner, C.; Reiser, P.; De Vere Peratoner, I.; Grzanna, S.; Wübbelt, P.; Bott, O.: Grading mit Grappa – Ein Werkstattbericht. In: Workshop „Automatische Bewertung von Programmieraufgaben“, 6.11.2015, Wolfenbüttel, 2015.

[SSM+15] Strickroth, S.; Striewe, M.; Müller, O.; Priss, U.; Becker, S.; Rod, O.; Garmann, R.; Bott, O.; Pinkwart, N.: ProFormA: An XML-based exchange format for programming tasks. eleed e-learning & education, 11(1), 2015.

[Gar15] Garmann, R.: E-Assessment mit Graja – ein Vergleich zu Anforderungen an Softwaretestwerkzeuge. In: Workshop „Automatische Bewertung von Programmieraufgaben“, 6.11.2015, Wolfenbüttel, 2015.

[Gar16] Garmann, R.: Bewertungsaspekte und Tests in Java-Programmieraufgaben für Graja im ProFormA-Aufgabenformat, Forschungsbericht, http://serwiss.bib.hs-hannover.de/frontdoor/index/index/docId/834, 2016.

[ISO11] ISO/IEC 25010:2011: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

[11] Dieser Beitrag wurde als Teil des Projekts „eCompetence and Utilities for Learners and Teachers“ (eCULT) vom Bundesministerium für Bildung und Forschung (BMBF) gefördert (Förderkennzeichen 01PL11066D).