Ansatz zur automatischen Generierung von Java-OOP-Aufgaben inkl. Bewertungsschemen

Ulf Döring[1]

Abstract: Im Fach „Algorithmen und Programmierung für Ingenieure“ werden an der TU Ilmenau noch papierbasierte Klausuren mit einem sehr geringen Multiple-Choice-Anteil geschrieben. Entsprechend hoch ist der Aufwand der händischen Korrektur. Zudem müssen selbst zum Zeitpunkt der Prüfung die meisten Studierenden noch als Programmieranfänger gesehen werden. Hierdurch führen Ansätze, welche compilierbaren Quellcode voraussetzen, sowohl beim Üben als auch bei Klausuren regelmäßig nicht zu angemessenen Bewertungen. Dieser Artikel beschreibt die Entwicklung eines Aufgabengenerators für einen bestimmten Aufgabentyp im Kontext von Java und OOP. Die automatische Generierung der Aufgaben soll in Bezug auf die zielgerichtete Vorbereitung auf die Prüfung den Studierenden Übungsmöglichkeiten bieten. Hinsichtlich der Klausurkontrolle sollen zudem auch Bewertungsschemen zur Anwendung bei der händischen Korrektur erzeugt werden.

Keywords: Ingenieurstudium; Java; Klassendefinition; Aufgabengenerator; Anfängerquellcode

1 Einführung

An der TU Ilmenau beginnen jährlich ca. 400 Studierende ihre Ausbildung in einem Ingenieurstudiengang, in dessen Rahmen sie die Lehrveranstaltung „Algorithmen und Programmierung für Ingenieure“ absolvieren. Ein grundsätzliches Problem gegenüber informatikbezogenen Studiengängen besteht in der oft fehlenden Motivation sowie einem durchschnittlich geringerem Vermögen, eine Programmiersprache zu erlernen und Algorithmen zu verstehen. Dementsprechend ist ein langsamerer Lernfortschritt typisch. Bezüglich des Erlernens einer Programmiersprache (für die Lehrveranstaltung wurde dazu Java ausgewählt) ist die Berücksichtigung der Probleme von Programmieranfängern mit den bereitgestellten Tools (insbesondere Eclipse) jedoch noch unzureichend. Von Seiten der Lehrenden wird hier ein entsprechend hoher Aufwand betrieben, um dieses Manko in Seminaren und Tutorien durch persönliche Hilfestellungen beim Finden und Beheben von Fehlern auszugleichen, siehe auch [SH15] zu Erfahrungen im Umgang mit Programmieranfängern. Um den Betreuungsaufwand zu senken, wurden bereits Angebote zum Üben typischer Verarbeitungsmuster für 1d-Felder ([DA19; DF17]) sowie zur Erstellung einfacher Java-Klassen entwickelt. Hilfestellungen werden bei diesen insbesondere durch das Aufzeigen der Beziehungen zwischen Aufgabenstellung und Beispiellösung sowie durch die Möglichkeit zur Simulation der Beispiellösung im Webbrowser gegeben. Der Generator für Aufgaben zur Erstellung von Java-Klassen soll nun erweitert werden. Ziel ist

[1] TU Ilmenau, IA/GDV, Helmholtzplatz 5, 98693 Ilmenau, ulf.doering@tu-ilmenau.de

doi:10.18420/abp2019-9

es dabei, sowohl die Studierenden beim Üben besser zu unterstützen, als auch die Qualität der Klausurkorrektur für derartige Aufgaben zu erhöhen.

2 Grundsätzliche Arbeitsweise des Aufgabengenerators

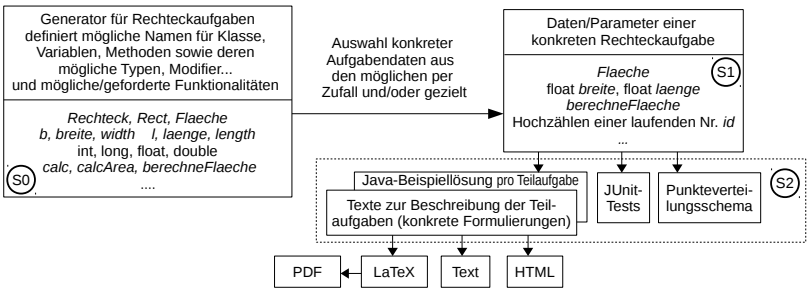

Abb. 1: Generierungsablauf am Beispiel eines Generators für Rechteckaufgaben

- Der Aufgabengenerator basiert auf spezifischeren Generatoren, welche jeweils eine grundsätzliche Aufgabenstruktur definieren (Stufe S0). Abb. 1 zeigt ein Beispiel.
- Eine solche Aufgabenstruktur wird in einer ersten Stufe (S1) durch die Vergabe konkreter Namen für die zu entwickelnde Klasse, für die Objekt- und Klassenvariablen und für die Methoden spezifischer festgelegt („verpackt“)[2]. Hier erfolgt auch die Auswahl der zu implementierenden Methoden (z. B. zufällig im sinnvollen Rahmen).
- In einer zweiten Stufe (S2) erfolgt dann die Generierung der passenden Java-Beispiellösung, der JUnit-Tests, des Bewertungsschemas[3] sowie die Formulierung der konkreten Aufgabentexte. Auch die Auswahl der konkreten Formulierung ist zufallsgesteuert auf Basis der Menge der jeweils für die Teilaufgaben zur Verfügung stehenden parametrisierten Satzbausteine (siehe auch Fußnote auf nächster Seite).
- Die generierte Aufgabenstellung kann online wahlweise als HTML-Seite und/oder als PDF gelesen werden. Lösungen sind bei Bedarf anzeigbar. Erläuterungen zum Zusammenhang zwischen bestimmten Teilen der Aufgabentexte und den Beispiellösungen wie in [DF17] sind wieder angestrebt.
- Um den oft geringen Programmierkenntnissen und -fähigkeiten der Übenden Rechnung zu tragen, wird ein Eingabeformular angeboten, welches die einzugebende Lösung auf mehrere Textfelder verteilt, siehe Abb. 2.

[2] Die Festlegungen lassen sich dabei jeweils gezielt händisch oder per Zufall aus vorgegebenen Mengen treffen. Aufgabenstrukturen, die momentan erarbeitet werden, beziehen sich z. B. auf die Modellierung einer Fläche (mit Ausprägungen wie Rasenfläche, Tafelfläche oder Chipfläche) oder eines Volumens (mit Ausprägungen wie Wasserbecken, Container oder Verpackungsbox).

[3] Für die Papierklausur wird bisher eine per Hand erzeugte Liste mit Hinweisen an die Korrektoren zur Punktevergabe bereitgestellt. Diese soll durch eine generierte Version ersetzt werden, welche der Prüfer selbstverständlich anpassen kann. Ziel ist es dann jedoch, dass die der Änderung zugrunde liegende Idee wieder in den Generator einfließt und das entsprechende Bewertungswissen erhalten bleibt.

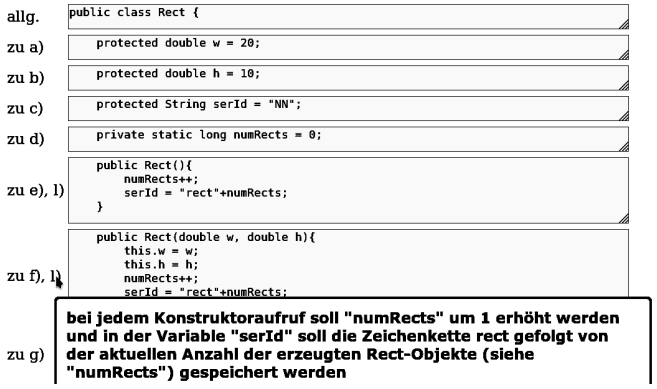

Abb. 2: Ansicht der Eingabemasken für die Lösungen mit Aufgabentext als Tooltip

3 Aktueller Stand der Umsetzung und Umsetzungsdetails

Die Implementierung des Aufgabengenerators erfolgt in Java. Die Java-Klassen lassen sich lokal verwenden, z. B. zur Erzeugung von Aufgaben auf dem Rechner des Lehrenden oder Prüfenden. Auf einem Aufgabenserver wird der Generator via Tomcat (Java Server Pages) in die Übungsoberfläche eingebunden.

Bei der Erzeugung der Aufgabentexte werden Einleitung und einzelne Teilaufgaben auf Basis der abstrakten Aufgabenbeschreibung durch unterschiedliche Satzbausteine wiedergegeben[4]. Dies soll helfen, das Verständnis von „Informatik“-Deutsch zu verbessern. Um verschiedene Ausgabeformate (HTML, Text, LaTeX) angemessen zu unterstützen, werden die Hervorhebungen semantisch wichtiger Texte (z. B. Variablennamen) oder textstrukturierende Elemente (z. B. Aufzählungen) pro Ausgabeformat anders erzeugt. Ein gekürztes Beispiel für einen vollständig generierten Aufgabentext könnte wie folgt aussehen:

Erstellen Sie eine öffentliche Klasse namens `Rect`*, welche Folgendes enthalten soll:*

a) eine geschützte `double`*-Objektvariable mit dem Namen* `w`*, wobei* `w` *am Anfang stets den Wert 20 haben soll und eine Breite speichert,*

c) eine geschützte `String`*-Variable, welche* `serId` *heißen soll und pro Objekt separat zu speichern ist, wobei* `serId` *am Anfang stets den Wert NN enthalten soll,*

d) eine private `long`*-Klassenvariable namens* `numRects`*, wobei* `numRects` *am Anfang stets den Wert 0 haben soll und die aktuelle Anzahl der erzeugten Objekte speichert,*

f) einen öffentlichen Konstruktor mit einer Parameterliste, welche zur Initialisierung von `w` *und* `h` *dienen soll (die Reihenfolge der Parameter genau so wie eben aufgezählt),*

[4] `public static String[] coD_V = { "einen ${deM1}1 Defaultkonstruktor"  // gekürztes Beispiel`
`, "einen Konstruktor, welcher parameterlos und ${deX0}1 ist"`
`, "einen ${deM1}1 Konstruktor, welcher keine Parameter hat" };`
Hier ist andeutungsweise zu sehen, dass auch grammatikalische Aspekte bei der Parametersubstitution Berücksichtigung finden können (z. B. wird aus `public` durch `{deX0}` „öffentlich“ und durch `{deM1}` „öffentlichen“)

l) bei jedem Konstruktoraufruf soll `numRects` *um 1 erhöht werden und in der Variable* `serId` *soll die Zeichenkette rect gefolgt von der aktuellen Anzahl der erzeugten* `Rect`*-Objekte (siehe* `numRects`*) gespeichert werden.*

Die Aufteilung der Teillösungen auf einzelne Felder erhöht die Chance für besseres Feedback erheblich („teile und herrsche“). Bei einer derartigen Eingabeform wirkt sich z. B. eine vergessene schließende geschweifte Klammer am Methodenende nur auf die entsprechende Teilaufgabe aus. Da die Aufgabe (inkl. den einzelnen Teilaufgaben und -lösungen) dem Generator bekannt ist, lassen sich einzelne Tests grundsätzlich auch bei nicht komplett compilierbarem Quellcode generieren und aufrufen. Der bisherige Ansatz einer JUnit-Testklasse, welche nur compiliert, wenn alle zu testenden Methoden und Variablen in der Klasse zugreifbar sind, wird momentan durch eine wesentlich flexiblere reflection-basierte Lösung ersetzt. Pro Teilaufgabe hängt die Art der Bewertung davon ab, ob der jeweilige Code compiliert (ggfs. unter Einbeziehung der Lösungsbsp. anderer Teilaufg.). Falls ja, werden die zugehörigen generierten funktionalen Tests gestartet. Falls nein, sollen Pattern bekannter/typischer Fehler eingesetzt werden, um Lösungshinweise zu geben. Diese Pattern dienen auch als Basis für die Erstellung des zur Aufgabe gehörenden Korrekturschemas.

4 Zusammenfassung

Der Beitrag beschreibt kurz den Stand der Arbeiten am Aufgabengenerator im August 2019. Der Generator sollen eine gerechtere automatische Bewertung von nicht-compilierfähigem Quellcode (typischerweise Anfängerquellcode) unterstützen - aber durchaus auch in Verbindung mit dynamischen Tests der compilierfähigen Lösungsteile.

Literatur

[DA19] Döring, U.; Artelt, B.: Web-Based Tools for Interactive Training in Implementing Java Methods. In: Proceedings of INTED 2019, 13th annual International Technology, Education and Development Conference. Valencia, Spain, S. 3974–3979, März 2019.

[DF17] Döring, U.; Fincke, S.: Interaktive Ansätze zur Vermittlung von Programmierfähigkeiten im Rahmen des Ingenieurstudiums. In: Tagungsband der 12. Ingenieurpädagogischen Regionaltagung 2017. Ilmenau, Germany, S. 171–176, Mai 2017, ISBN: 978-3-9818728-1-1.

[SH15] Schulten, B.; Höppner, F.: Zur Einschätzung von Programmierfähigkeiten - Jedem Programmieranfänger über die Schultern schauen. In (Priss, U.; Striewe, M., Hrsg.): 2nd Workshop on „Automatische Bewertung von Programmieraufgaben“ (ABP 2015), Wolfenbüttel, Germany 6. Nov. 2015. CEUR Workshop Proceedings 2015, Wolfenbüttel, Germany, 2015, URL: http://ceur-ws.org/Vol-1496/#paper7.