# Auf dem Weg zu variablen Programmieraufgaben: Requirements Engineering anhand didaktischer Aspekte

Benjamin Otto[1] und Michael Goedicke[1]

**Abstract:** E-assessment systems for automatic evaluation and feedback generation for programming tasks have become increasingly popular in recent years at universities. They allow a more individualized support of programming skills than it would be possible in personal contact with instructors even for lectures with a large number of students. A problem in the use of e-assessment systems is the number of available tasks and their lack of adaptivity. An approach to attack both problems could be to make programming tasks variable. Ideally, as many additional variants of a task as possible could be generated with preferably the least possible additional effort where the difficulty of each variant is well known. In order to work towards this ideal goal, in this first work we will discuss didactic aspects in the sense of requirements engineering for a description language for variable programming tasks which is yet to be developed. Furthermore, we will analyze problems specific to variability of programming tasks over existing approaches of variability e.g. of mathematics tasks.

**Abstract:** E-Assessment-Systeme zur automatischen Bewertung und Feedbackgenerierung für Programmieraufgaben haben sich in den letzten Jahren an Universitäten immer weiter durchgesetzt. Diese ermöglichen, auch bei Vorlesungen mit einer großen Anzahl von Studierenden, eine individuellere Förderung der Programmierkenntnisse als dies im persönlichen Kontakt mit Lehrenden möglich wäre. Ein Problem im Einsatz von E-Assessmentsystemen ist die Anzahl verfügbarer Aufgaben und deren mangelnde Adaptivität. Ein neuer Lösungsansatz kann es für beide Probleme sein, Programmieraufgaben variabel zu gestalten. Idealerweise könnte in Zukunft durch einen möglichst geringen Mehraufwand eine möglichst große Anzahl an möglichst unterschiedlichen Varianten einer Aufgabe erzeugt werden, deren Schwierigkeiten man gut kennt. Um auf dieses Idealziel hinzuarbeiten, werden wir in dieser ersten Arbeit zunächst didaktische Aspekte im Sinne eines Requirements Engineering für eine in zukünftigen Arbeiten zu entwickelnde Beschreibungssprache für variable Programmieraufgaben diskutieren. Ferner werden wir hier Probleme analysieren, die für Variabilität von Programmieraufgaben spezifisch sind gegenüber bereits existierenden Ansätzen der Variabilität z. B. von Mathematikaufgaben.

**Keywords:** Variabilität, Programmieraufgaben, Didaktik, Autobewertung

## 1 Einleitung

Ein limitierender Faktor bei der Verbreitung von E-Assessment-Systemen ist die Verfügbarkeit geeigneter Aufgaben. Eine mögliche Verbesserung hinsichtlich größerer

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

Aufgabenvielfalt können variabel gestellte Aufgaben sein, aus denen automatisiert zahlreiche Varianten generiert werden können.

Von solcherweise gestellten Aufgaben kann man sich ferner erhoffen, dass der Lerneffekt des in der jeweiligen Aufgabe behandelten theoretischen Konzeptes höher ist, da nicht nur eine Lösungsvariante auswendig gelernt werden kann.

Bereits existierenden Ansätzen für andere Domänen soll in folgenden Arbeiten die Domäne der Programmieraufgaben – zunächst für die Programmiersprache Java – hinzugefügt werden. Hierfür gilt es zunächst ein didaktisches Konzept zu entwerfen, um einen besseren Überblick über die auf uns zukommenden Anforderungen an solche Aufgaben und die noch zu entwickelnde Beschreibungssprache bekommen zu können.

Bezüglich der didaktischen Perspektive stellt sich die Frage nach Aufgabentypen, die einen erkennbaren didaktischen Mehrwert gegenüber statischen Aufgaben haben und die sich prinzipiell über Beschreibungssprachen abbilden lassen.

Des Weiteren ist aus didaktischer Sicht interessant, ob die Schwierigkeit einzelner Varianten[2] der gleichen variablen Aufgabe signifikant von anderen Varianten abweichen. Kann es zudem „Ausreißer-Varianten“ geben, deren Schwierigkeit durch eine bestimmte Belegung stark steigt oder sinkt?

## 2 Verwandte Arbeiten

Den Autoren sind zur Zeit keine weiteren Arbeiten bekannt, die das Konzept der variablen Programmieraufgaben in E-Assessment-Systemen realisieren. Zur Thematik variabel gestellter Aufgaben im E-Assessment gibt bereits existierende Ansätze für Mathematik: für mathematische Fill-In Aufgaben (vgl. [Ku15]), für mathematische Multiple-Choice-Aufgaben (vgl. [Sc15]).

Zur Analyse der Schwierigkeit von Objektorientierten Programmieraufgaben in Java (vgl. [Hu17]).

Zu der Thematik der Variabilität in Software hat sich in den letzten Jahren die Methodik des Software Product Line Engineering (SPLE) [PBL05] etabliert. Wir werden uns auch an dem Fachvokabular des SPLE orientieren, sofern sich dies problemlos auf unsere Domäne übertragen lässt[3]. Zur Analyse der Widerspruchsfreiheit von Anforderungen an Variabilität und sich dadurch ergebende Einschränkungen (vgl. [La09]).

[2] D. h. der Konkretisierung aller Variationspunkte einer Variablen Aufgabe
[3] Insbesondere sind hier Begriffe wie „Variationspunkt“, „Variante“, „Ausprägung“, „Artefakt“ etc. gemeint

# 3 Didaktische Aspekte

Die Zielgruppe des didaktischen Konzepts sind zunächst Bachelorstudierende[4] im ersten Semester Informatik / Wirtschaftsinformatik / Lehramt Informatik. Im Wintersemester 2017/18 wird für diese Gruppe an der Universität-Duisburg-Essen die Vorlesung „Programmierung“ gehalten werden, in welcher das erste Mal variabel gestellte Aufgaben für das E-Assessmentsystem JACK [SZG15] verwendet werden sollen. Hierbei wird JACK einerseits formativ als Übungstool mit wöchentlich freiwillig zu bearbeitenden Übungsaufgaben eingesetzt. Andererseits aber auch als summatives Assessment für ca. zweiwöchentliche stattfindende Miniprojekte und deren zugehörige Testate. Miniprojekte sind umfangreichere Übungsaufgaben, die den in Testaten abgefragten Fähigkeiten sehr nahekommen. Um für die Testate zugelassen zu sein, muss das zugehörige Miniprojekt nur bearbeitet worden sein, eine Mindestpunktzahl gibt es nicht. Die Testate werden unter Prüfungsbedingungen an PC-Arbeitsplätzen in einer Halle mit ca. 150 Plätzen geschrieben. In allen Testaten kombiniert muss allerdings eine Mindestpunktzahl erreicht werden, um an der Abschlussklausur (die noch auf Papier geschrieben wird) teilnehmen zu dürfen.

Da solche Anfängervorlesungen an der Universität Duisburg-Essen aber in der Größenordnung 500-600 Teilnehmer liegen, müssen Testate notwendigerweise in mehreren Durchgängen durchgeführt werden. Um Abschreiben und das Bekanntwerden von Lösungen zu verhindern, werden solche Aufgaben deswegen bisher manuell variiert. Variabel gestellte Aufgaben könnten das Weitergeben und Abschreiben von Lösungen in Zukunft nicht nur zwischen großen Gruppen von Studierenden erschweren, sondern auch zwischen einzelnen Studierenden.

**Variantenschwierigkeit**
Ein wesentlicher didaktischer Aspekt der Variabilität von Programmieraufgaben ist die Variantenschwierigkeit[5] innerhalb der gleichen Aufgabe.

Zunächst stellt sich die Frage, wie möglichst objektiv die Schwierigkeit einer Aufgabenvariante bestimmt werden kann. Hierzu hat sich die Methode der Item-Response-Theorie (IRT) [ER13], insb. durch Rasch-Modelle [St15] als geeignet erwiesen. Es ist hierbei allerdings nötig, eine gewisse Datenbasis von Bearbeitungen der jeweiligen Aufgabe zu haben, um mit dem Modell sinnvolle Aussagen treffen zu können.

Durch kombinatorische Explosion kann die Kombination mehrerer Variationspunkte sehr viele Varianten ermöglichen bzw. ein Variationspunkt kann sehr viele Ausprägungen besitzen. Deshalb kann nicht erwartet werden, für jede Variante genug Lösungen für eine gute statistische Schwierigkeitsschätzung zu haben. Ein Ansatz zur Lösung dieses Problems kann es sein, manuell Klassen von Ausprägungen zu definieren, welche eine

[4] Fast alle der Beobachtungen dürften jedoch ähnlich auf Schüler zutreffen, bei diesen wird man jedoch weniger Vorwissen in Mathematik usw. voraussetzen.
[5] Im Folgenden werden wir bei „Variantenschwierigkeit“ immer die Schwierigkeit von Varianten in der gleichen Aufgabe meinen

ähnliche Schwierigkeit produzieren. Um diese Annahme über Schwierigkeitsklassen glaubhaft zu machen, sollten idealerweise zufällig Schwierigkeitsrepräsentanten[6] bestimmt werden, für die man genug Daten für eine Rasch-Analyse sammelt.

### Summatives Assessment

In einem summativen E-Assessment sollte sich die Varianz nicht wesentlich auf die Schwierigkeit auswirken, da andernfalls die Bewertung unfair wird. Es sollten deshalb anhand der Variantenschwierigkeit automatisierte Entscheidungen in der Beschreibungssprache möglich sein. Eine zu schwierige Variante könnte so automatisiert neu „ausgewürfelt“ werden oder die Punktezahl könnte angepasst werden[7]. Ferner sollte analysiert werden, ob man bestimmte Klassen von Variationspunkten bzw. Klassen von Ausprägungen je Variationspunkt identifizieren kann, die die Schwierigkeit nur gering variieren. Hierbei ist zu beachten, dass eventuell durch Kombination solcher Variationspunkte unerwartet hohe oder niedrige Schwierigkeiten entstehen können. Um Variation in summativen Szenarien einsetzen zu können, soll deshalb in erster Iteration davon ausgegangen werden, dass variable Aufgaben gestellt werden können, deren Varianten nur kleine Variationen der Schwierigkeit haben. Dies kann in jedem Fall dadurch erreicht werden, dass sich der Lehrende semimanuell alle möglichen Varianten der Aufgabe erstellt und ihrer Schwierigkeit nach einordnet. Um diese Ad-hoc Einschätzungen zu verifizieren, sollte zumindest nach dem Assessment an [Hu17] orientierend eine auf Item-Response-Theorie basierende Einschätzung der Schwierigkeit erfolgen.

Damit zwischen Studierenden, die in Prüfungssituationen nebeneinander sitzen, immer andere Varianten erzeugt werden können, folgt einerseits: es sollten Constraints in der Beschreibungssprache möglich sein. Constraints sollen als Einschränkungen an die möglichen Ausprägungen eines Variationspunkts verstanden werden. Ferner sollten diese auch physisch umgebungsbezogene Einschränkungen im Sinne eines Sitzplans enthalten, um Abschreiben zu vermeiden.

### Formatives Assessment

Erfahrungsgemäß sind in einer Informatik-Anfängerveranstaltung die Vorkenntnisse, algorithmisches Denken und Computeraffinität stark unterschiedlich bei den Studierenden verteilt. Die Variabilität von Programmieraufgaben soll hier auch zu einer Verbesserung der Lehre beitragen, in dem die Studierenden in die Lage versetzt werden sollen, adaptiv an schwächen in bestimmten Aufgaben zu arbeiten. Hierbei muss zunächst eine rudimentäre Unterstützung für Adaptivität in JACK integriert werden. Da im Rasch-Modell Persönlichkeitstraits eng gesteckt sind, müssen in zukünftigen Arbeiten diese anhand von konkreten Aufgaben identifiziert und in ein noch zu programmierendes Usermodell aufgenommen werden. Dann könnten, sofern man die Schwierigkeitsklassen von variablen Aufgaben grob kennt, jeweils auf den Studierenden zugeschnittene Varianten generiert werden, um die Studierenden optimal auf deren Lernweg zu unterstützen.

---

[6] Als Schwierigkeitsrepräsentant einer Kasse ist hier eine Aufgabe mit Ausprägung der variablen Merkmale gemeint, die in der gleichen Schwierigkeitsklasse liegen.

[7] Was allerdings zu weiteren Problemen hinsichtlich der Gesamtpunkzahl usw. führt

Das E-Assessment-System sollte es den Studierenden anbieten, die gleiche Aufgabe mehrfach in unterschiedlichen Schwierigkeiten auszuführen. Dies könnte gezielt im Rahmen eines Selfassessments genutzt werden, um ein bestimmtes Lernobjekt besser zu durchdringen.

Ein Problem der Verbreitung von E-Assessment Systemen ist die mangelnde Anzahl verschiedener Aufgaben für diese Systeme. Dies liegt unter anderem daran, dass oftmals Aufgaben vom Lehrenden über schwierig manuell zu editierende Formate wie XML encodiert werden müssen. Während es zu erwarten ist, dass dieser Aufwand durch variable Aufgaben zunächst sogar erst steigt, da zusätzlich noch der Variabilität Rechnung getragen werden muss, sollte sich jedoch die die Gesamtzeit pro Aufgabe zur Erstellung drastisch verkürzen lassen, wenn man jede Variante einer Aufgabe als neue Aufgabe ansieht. Hierbei ist nicht wohldefiniert, wie unterschiedlich eine Variante sein sollte, um als eigenständige Aufgabe zu gelten.

## 4 Beispiele und Beobachtungen

Im Folgenden soll eine Beispielaufgabe analysiert werden, anhand derer Anforderungen und Besonderheiten untersucht werden sollen.

**Beispiel: Temperaturarray**

Betrachten Sie den Sourcecode. Gegeben ist hier ein Array als Klassenattribut

```
double[] temperaturMessung = new double[525599];
```

welches Messwerte in Grad Celsius, die minütlich über einen Zeitraum von einem Jahr an einer Wetterstation gemessen wurden, enthält. Die erste Messung vom 01. Januar 2016 um 00:01 steht in `TemperaturMessung[0]`.

1) Schreiben Sie eine Java Methode
   `double berechneDurchschnittsTemperatur(int untererIndex, int obererIndex)`
   welche die Durchschnittstemperatur der Werte zwischen `temperaturMessung[obererIndex]` und `temperaturMessung[untererIndex]` (einschließlich dieser Werte) ausrechnet.
2) Schreiben Sie eine Methode
   `double bestimmeHoechsteTemperatur(LocalDate tag)`
   welche die höchste Temperatur des übergebenen Datums errechnet.
3) Schreiben Sie eine Methode
   `Month getMonatKleinsteDurchschnittsTemp()`
   welche den Monat mit der kleinsten Durchschnittstemperatur des Arrays `temperaturMessung` zurückgibt.

In Unteraufgabe 1) lässt sich ein wesentlicher Unterschied von Programmieraufgaben gegenüber anderen Domänen wie der Mathematik beobachten. Bei einer Mathematikaufgabe wäre es z.B. eine sinnvolle Aufgabenstellung, konkrete Zahlenwerte für Koeffizienten von Polynomen auszuwürfeln und für diese jeweils eine Kurvendiskussion anfertigen zu lassen. In der Programmierung wäre eine Aufgabe der Art von 1) ohne Untermethode und mit vorgegebenen Zahlenwerten für `untererIndex` und `obererIndex` aus verschiedenen Gründen nicht sinnvoll. Einerseits widerspräche dies dem Ziel, guten Programmierstil zu fördern. Um die Programmierparadigmen der Kapselung und DRY[8] zu erfüllen, sollte hier also eine Unterfunktion mit (variablen) Übergabeparametern geschrieben werden. Des Weiteren ist auch aus technischer Sicht das Schreiben von Unterfunktionen mit Parametern nötig, damit ein dynamischer Test zur Verifikation der Korrektheit (bzw. zur Bepunktung und Feedbackgenerierung) diese mit verschiedenen Parametern aufrufen kann. Das bedeutet, dass durch in der Regel möglichst allgemein geschriebene Funktionen der Spielraum der Variabilität in Programmieraufgaben eingeschränkt wird.

Als Variationspunkt kommt die Berechnung weiterer statistischer Kennwerte in Betracht, also beispielsweise statt Durchschnitt den Median oder den harmonischen Mittelwert, aber auch andere Werte wie Minimum oder Maximum. Kann man beim Durchschnitt noch davon ausgehen, dass diesen alle Studierenden berechnen können, wird beim Median oder dem harmonischen Mittel noch ein erklärender Text notwendig sein, der die jeweilige Berechnung erläutert. Der Variationspunkt sollte also weitere Artefakte beinhalten können.

Als weitere Variationspunkte kommen subtilere Änderungen in Betracht, die sich aber möglicherweise substantiell auf die Lösung auswirken können. Es kann z.B. variiert werden, ob von korrekten Übergabewerten ausgegangen werden darf, oder welche Fehler (negative Indizes, Indizes außerhalb der Arraygrenzen, obererIndex < untererIndex) abgefangen werden sollen und wie diese Fehler behandelt werden sollen (bestimmte Exception werfen, 0 zurückgeben, Warnung loggen etc.).

Eine weitere Besonderheit bei Programmieraufgaben ist ferner, dass bereitgestellter Java Code oftmals abhängig von Variationspunkten mitvariiert werden muss. Im Code müssen z. B. Methodennamen[9] oder -signaturen verändert werden oder Datenstrukturen anpassbar sein.

Hier offenbart sich eine weitere spezifische Problematik der Variabilität von Programmieraufgaben: Da der bereitgestellte Sourcecode variabel ist, sich Datenstrukturen, Eigenschaften und Anforderungen an den Code pro Variante ändern können, kann dies zu großem Mehraufwand beim Erstellen von statischen oder dynamischen Prüfregeln führen. Zu jeder Kombination von Variationspunkten müssen Prüfregeln generiert werden, die automatisiert die Bewertung und Feedback für diese Variante bereitstellen. Dabei ist ferner zu beachten, dass Ausprägungen von Variationspunkten eventuell Ausprägungen

[8] „Don‘t Repeat Yourself“
[9] z.B. sollte eine Methode für die Minimumsbestimmung anders heißen, als für die Maximumsbestimmung

anderer Variationspunkte einschränken, d. h. in einer Beschreibungssprache müssen Variationspunkte Constraints beinhalten können, vgl. auch [La09].

Bezüglich der Variation der Schwierigkeit ist zu beobachten, dass die Schwierigkeit sich bei der Variation von Minimum oder Maximum nicht wesentlich ändern dürfte, da hier der jeweilige Algorithmus zur Bestimmung desselbigen sehr ähnlich ist. Des Weiteren sind sich Durchschnitt und harmonisches Mittel algorithmisch zwar ähnlich, vermutlich wird aber das harmonische Mittel Aufgrund der unbekannteren Berechnungsart als schwerer empfunden werden. Die Berechnung des Medians ist algorithmisch etwas anders als die vorgenannten Ausprägungen des Variationspunkts und kann von den Autoren nur schwer in der Schwierigkeit für Studierende eingeschätzt werden. Fehlerbehandlung für die Übergabeparameter dürfte jede Aufgabe in der Schwierigkeit erhöhen. Um hierfür nicht nur Ad-hoc-Aussagen treffen zu können, wird dies anhand der Rasch-Modellierung in der in Kap. 3 erwähnten Vorlesung überprüfen sein.

In den Unteraufgaben 2) und 3) ließe sich zusätzlich zum bereits Genannten durch Variation z.B. noch die weiteren Übergabeparameter `double[] temperaturMessung, int Jahr` einführen, so dass auch beliebige Temperaturarrays anderer Jahre übergeben werden könnten. Eine Lösung, die die Funktion aus der ersten Aufgabe nutzt, um die zweite und dritte Aufgabe zu lösen, müsste also abhängig von der Jahreszahl die Indizes des Funktionsaufrufs der ersten Aufgabe anpassen, was die Aufgabe erschweren würde.

Zuletzt stellt sich hier die Frage, wann man noch von einer Aufgabenvariante sprechen kann oder diese vom Charakter her schon eher eine neue Aufgabe darstellt. Als Kriterium könnte hier dienen, ob der Aufwand dies mit allen abhängigen Artefakten (Sourcecode, Prüfcode, Aufgabentext, Constraints etc.) in eine Aufgabe zu integrieren mehr Aufwand erzeugt, als für das Erstellen einer neuen Aufgabe nötig wäre. Eine mögliche Variation, die für eine solche Betrachtung infrage käme, wäre das Ersetzen des Temperaturarrays durch ein mehrdimensionales Array, in dem der erste Index für das Jahr steht und im zweiten Index die Messwerte, die über das Jahr aufgezeichnet wurden. Dies soll in einer späteren Arbeit weiter untersucht werden.

## 5 Fazit und Ausblick

In dieser Arbeit konnten wir wichtige Anforderungen und Probleme für Variabilität von Programmieraufgaben aufzeigen und diverse Anknüpfungspunkte für zukünftige Arbeiten identifizieren. Bisher kann nicht abgeschätzt werden, ob sich das im Abstract skizzierte Idealziel erreichen lässt, also ob der zusätzliche Aufwand, Variabilität von Programmieraufgaben zu ermöglichen, den Nutzen rechtfertigt. Deshalb ist als Nächstes geplant, einen ersten Prototyp einer Beschreibungssprache als Domain Specific Language (DSL) zu erstellen. Darauf basierend soll ein rudimentärer Variantengenerator erstellt werden, mit dem aus der DSL JACK-Aufgabenvarianten erzeugt werden können. Anhand dessen sollten wir eine bessere Idee davon gewinnen, wie sich Aufwand und Nutzen die Waage halten.

Als nächster Schritt sollen dann Aufgabenvarianten in der in Kap. 3 erwähnten Vorlesung eingesetzt und durch Item-Response-Theorie auf Schwierigkeit analysiert und die Idee der Schwierigkeitsklassen näher untersucht werden. Hierbei soll ein kleiner Aufgabenpool variabler Aufgaben zu einem bestimmten Thema zur Messung einer latenten Variable mit IRT erstellt werden, der auf Rasch-Skalierbarkeit geprüft wird.

## Literaturverzeichnis

[ER13] Embretson, S. E., & Reise, S. P. (2013). Item response theory. Psychology Press.

[Hu17] Hubwieser, P., Berges, M., Striewe, M., & Goedicke, M. (2017, April). Towards competency based testing and feedback: Competency definition and measurement in the field of algorithms & data structures. In Global Engineering Education Conference (EDUCON), 2017 IEEE (pp. 517-526). IEEE.

[Ku15] Kurt-Karaoglu, F., Schwinning, N., Striewe, M., Zurmaar, B., & Goedicke, M. (2015, April). A Framework for Generic Exercises with Mathematical Content. In Learning and Teaching in Computing and Engineering (LaTiCE), 2015 International Conference on (pp. 70-75). IEEE.

[La09] Lauenroth, K. (2009). Konsistenzprüfung von Domänenanforderungsspezifikationen. Logos Verlag Berlin GmbH.

[PBL05] Pohl, K., Böckle, G., & van Der Linden, F. J. (2005). Software product line engineering: foundations, principles and techniques. Springer Science & Business Media.

[Sc15] Schwinning, N., Striewe, M., Savija, M., & Goedicke, M. (2015, October). On Flexible Multiple Choice Questions With Parameters. In European Conference on e-Learning (p. 523). Academic Conferences International Limited.

[St15] Strobl, C. (2015). Das Rasch-Modell: Eine verständliche Einführung für Studium und Praxis. Rainer Hampp Verlag.

[SZG15] Striewe, M., Zurmaar, B., & Goedicke, M. (2015). Evolution of the E-Assessment Framework JACK. In Software Engineering (Workshops) (pp. 118-120).