Metriken zur Messung von Lernerfolg im Informatik-Grundlagen-Unterricht

Simon Rupp[1], René Pawlitzek[1] und Carlo Bach[1]

Abstract: Es wird ein neu entwickeltes Werkzeug vorgestellt, das Metriken misst, mit denen der Lernerfolg der Studierenden in Übungsstunden abgeschätzt werden kann. Es handelt sich dabei um eine Erweiterung, die in der Entwicklungsumgebung BlueJ eingesetzt wird. Sie misst das Programmierverhalten der Studierenden anhand von Metriken, die aus Unterrichtsbeobachtungen und Dozierendenbefragungen hergeleitet wurden. Die Erweiterung berechnet aufgrund der gemessenen Metriken einen Hilfeindikator, der angibt, wie hoch das Bedürfnis nach Hilfe ist. Auf einer in Echtzeit generierten Webseite hat der Dozierende eine Übersicht über die gesamte Klasse. Alle Studierenden werden farblich, je nach Notwendigkeit einer Hilfestellung, eingefärbt. Vor Ort kann der Dozierende beim Studierenden über ein in die Entwicklungsumgebung integriertes Dashboard die Kennzahlen einsehen. Der Dozierende kann so seine Betreuungszeit auf sehr effiziente Art einsetzen und fördert damit die Wirkung des Unterrichts. Mit diesem Konzept werden potentiell die Studierenden unterstützt, welche die Hilfe am meisten benötigen. Die mit dieser Erweiterung gesammelten Daten wurden analysiert und führten zur Weiterentwicklung und Verbesserung der Metriken.

Keywords: Lernerfolg, Lernerfolgsmessung, Metriken, smart classroom, BlueJ

1 Einleitung

Im ersten Jahr des technischen Bachelorstudiengangs „Systemtechnik“ an der NTB, wird ein Grundlagenkurs in Ingenieur-Informatik im Umfang von 8 ECTS über zwei Semester geführt. Der Unterricht erfolgt im wöchentlichen Kontaktunterricht während der Vorlesungszeiten. Die Studierenden erlernen in einer Abfolge von kurzen Vorlesungsblöcken und betreuten Übungen ihre erste Programmiersprache. Während der individuellen Bearbeitung der Aufgaben am Rechner sind die Studierenden durchgehend durch einen Dozierenden betreut.

Das Ziel des Kurses ist, zukünftigen Maschinenbauern, Elektrotechnikern und Informatikern die Anwendung einer objekt-orientierten Programmiersprache zu vermitteln. Als Programmiersprache kommt Java zum Einsatz. Der Kurs verfolgt das Prinzip „Objects first“. Zum Einsatz kommen dabei das Buch „Java lernen mit BlueJ“ [BK09] sowie die Entwicklungsumgebung BlueJ [Bl17].

Während Unterrichtsbeobachtungen zur Analyse des Übungsteils hat sich gezeigt, was

[1] NTB Interstaatliche Hochschule für Technik Buchs, INF / PWO, Werdenbergstrasse 4, 9471 Buchs, Schweiz, simon.rupp@ntb.ch, rene.pawlitzek@ntb.ch, carlo.bach@ntb.ch

schon aus einigen Forschungsergebnissen bekannt ist: Der Dozierende kann sich bei Klassengrößen über 15 Studierenden nicht gleichzeitig auf alle Studierenden im Raum konzentrieren. Der Dozierende hilft häufig Studenten in seiner Nähe und übersieht dabei leicht andere.

Anders als ITS (intelligente Tutor-Systeme) [Kö07], unterstützende Analysen von Gruppenarbeiten [STL13] oder Grader wie der Praktomat [BHS16] wurde ein Werkzeug entwickelt, das den Dozierenden in den Übungslektionen dabei unterstützt, den hilfsbedürftigsten Studierenden zu identifizieren. Das Werkzeug misst dafür während des Bearbeitens der Übungen kontinuierlich das Programmierverhalten der Studierenden und bereitet es so auf, dass die Lehrperson die verfügbare Betreuungszeit effizienter einsetzen kann. Die Metriken werden autonom im Unterrichtsbetrieb gemessen und nicht anhand von Beobachtungen wie in [Hu06]. Einen ähnlichen Ansatz verfolgt Ulrich Kiesmüller bei der Untersuchung der Lösungsstrategien von Sekundarschülern. [Ki13]

2 Konzept

Bei jedem Studierenden läuft innerhalb der BlueJ-Entwicklungsumgebung eine Erweiterung, die Messdaten zum Programmierverhalten und zur Interaktion mit der Entwicklungsumgebung sammelt. Die Daten werden zu einem Indikator aggregiert und dem Dozierenden in Echtzeit auf einer Webseite zur Verfügung gestellt. Daraus kann der Dozierende ableiten, wo seine Hilfestellung den größten Mehrwert bringt. Die gemessenen Daten können in einem Dashboard abgerufen werden und geben Hinweise über die Aktivität und den Erfolg der einzelnen Studierenden.

Im Rahmen dieser Arbeit wurden Metriken entwickelt und verifiziert, die den größten Aussagegehalt für den Hilfeindikator bieten.

2.1 Metriken zur Messung von Lernerfolg

Um geeignete erfassbare Metriken zu finden, wurden Unterrichtsbeobachtungen und Dozierenden-Befragungen durchgeführt. An der Befragung nahmen fünf Dozierende teil. Aus den daraus gewonnen Erkenntnissen entstanden die zu messenden Metriken. Es handelt sich dabei um Aspekte, die von den Dozierenden entweder als wichtig beurteilt wurden oder von ihnen bei der Kontrolle von Quellcode angewendet werden. In Tabelle 1 ist ein Ausschnitt aus den zu messenden Metriken dargestellt. Die Metriken wurden anhand ihrer Wichtigkeit sortiert und entsprechende Maßnahmen eingeleitet.

Diese Arbeit beschreibt die grau hinterlegten Metriken (Tab. 1) und deren Ergebnisse. Die erste Version kann neben dem Messen der Interaktion der Studierenden mit der Entwicklungsumgebung auch Schlüsselwörter zählen und Kompiliereregnisse interpretieren. Für die Quellcodeanalyse wurden mehrere statische Code Analyse Werkzeuge evaluiert und für die Verwendung in Version 2 vorbereitet.

Metrik	Wichtigkeit	Umsetzung in…
Quellcodeanalyse / Codier Konventionen	7	Vorarbeit / in Evaluation
Benutzung des Hilfeknopfs	5	Version 2
Fokus, Interaktion mit IDE	4	Version 1
Existenz von Schlüsselwörtern im Code	4	Version 1
Kompiliermeldungen	2+	Version 1

Tab. 1: Metriken, die durch Dozierendenbefragungen und Unterrichtsbeobachtungen zusammengetragen wurden.

Kompiliermeldungen: Gemessen wird die Anzahl Kompiliervorgänge, die ein Studierender im Verlauf der Übungsphase durchführt. Diese Zahl kann in erfolgreiche und fehlgeschlagene Vorgänge aufgeschlüsselt werden. Bei Fehlgeschlagenen kann außerdem die Anzahl Fehler und Warnungen des Compilers ermittelt werden. Neben den Zählern, die sich von Beginn weg inkrementieren, gibt es abgeleitete Metriken wie „compiles per minute“. Diese setzt die Anzahl Kompiliervorgänge ins Verhältnis zur Zeit, die für die Übung aufgewendet wurde. Anhand dieser Werte kann auf den Erfolg geschlossen werden: ist die Anzahl der fehlgeschlagenen Kompiliervorgänge innert kurzer Zeit sehr hoch, versucht der Studierende vermutlich durch Ausprobieren ein Problem zu lösen. Dies wäre für den Dozierenden ein Indiz bei diesem Studierenden vorbei zu gehen und ihn bei der Problemsuche zu unterstützen.

Neben dem Zählen der Kompiliervorgänge wird auch die Zeit dazwischen gemessen. Dafür wurden zwei Kennwerte entwickelt. Der Erste misst die Zeit zwischen zwei Kompiliervorgängen, der Zweite misst die Zeit, bis ein erfolgreicher Kompiliervorgang ausgeführt wurde. Neben der Zeitdauer kann aus dieser Metrik das Kompilierverhalten des Studierenden abgelesen werden: arbeitet der Studierende nach dem Versuchsprinzip oder kompiliert er erst nach langer Zeit ein großes, komplexes Programmstück?

Fokus, Interaktion mit IDE: Um die Interaktion mit der Entwicklungsumgebung zu messen, wird die Textzeigerposition in regelmäßigen Abständen ausgelesen. Dadurch kann in Erfahrung gebracht werden, wie oft sich diese innerhalb eines Texteditors oder zwischen mehreren Editoren verändert. Mit diesen Messdaten kann auf die Aktivität geschlossen werden.

Es wird zudem gezählt, wie viele Texteditoren offen sind und ob der Fokus auf der BlueJ Anwendung liegt. Über solche Messwerte kann eruiert werden, ob der Studierende aktiv die Übung bearbeitet. Sind viele Fenster offen und wechselt der Studierende sehr oft den Editor, kann dies ein Hinweis sein, dass der Studierende nicht weiss, wo er ein Stück Code umsetzen soll. Oder es kann ein Hinweis sein, dass er von einem anderen Programm Codestücke entlehnt.

Weitere Metriken messen die Anzahl Zeichen in einer Datei und die Schreibgeschwindigkeit der Studierenden im Editor. Damit soll gemessen werden, ob Studierende viel mit Copy-Paste arbeiten oder ob sie über längere Zeit nichts zu ihrem Quellcode hinzufügen.

3 Analyse

Die BlueJ-Erweiterung wurde in zwei Unterrichtseinheiten mit einer Klasse von 20 und 19 Studierenden getestet und ausgewertet. Die gesammelten Daten wurden periodisch in ein für den Dozierenden zugängliches Verzeichnis abgelegt. Diese Daten wurden für die anschließende Analyse verwendet. Während die Erweiterung in Betrieb war, wurde fortlaufend ein Indikator berechnet, welcher die Notwendigkeit einer Hilfestellung darstellte. Der Dozierende konnte während den Übungslektionen über einen Computer oder Tablet auf den Indikator zugreifen und erkannte unmittelbar, welcher Studierende seine Hilfe am dringendsten benötigte. Bei den Studierenden vor Ort konnte der Dozierende ein Dashboard öffnen, um sich einen Überblick über alle Metriken zu verschaffen.

3.1 Studierendenprofile

Die durch die Anwendung gesammelten Daten wurden verwendet, um Profile zu entdecken und daraus Schlüsse zum Lernerfolg zu erzielen. Die resultierenden Hypothesen gilt es in weiteren Testläufen zu überprüfen bzw. zu widerlegen. Die Profile wurden einerseits durch die grafische Visualisierung in Diagrammen und andererseits durch die tabellarische Darstellung der Messwerte entwickelt.

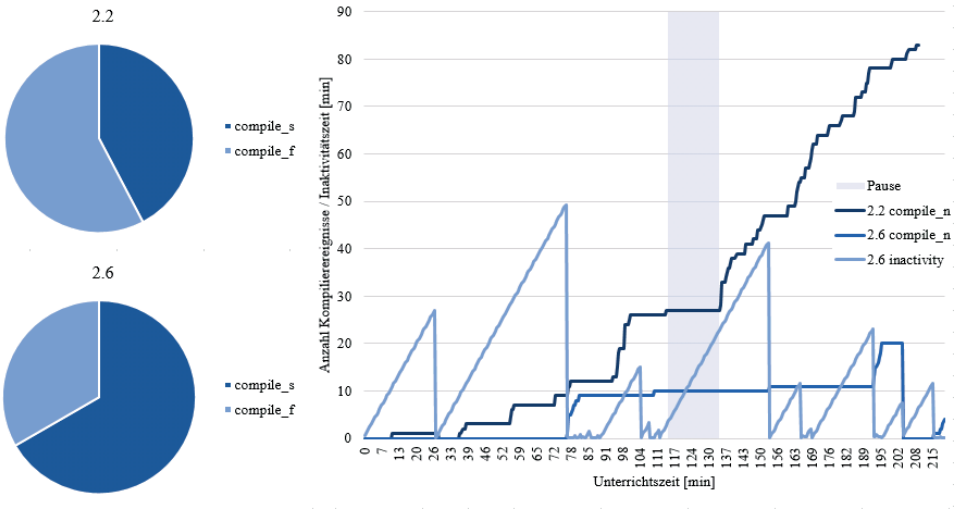

Abb. 1: Kreisdiagramm mit der Totalanzahl von erfolgreichen (compile_s) und fehlgeschlagenen (compile_f) Kompiliervorgängen (links) und Liniendiagramm mit der kombinierten Anzahl Kompiliervorgänge (compile_n) für zwei Studierende über den Zeitraum zweier Doppellektionen (rechts). Im Liniendiagramm ist außerdem die Inaktivitätskurve von Student 2.6 aufgetragen. Steigende Zacken zeigen die steigende Inaktivitätsdauer an. Grau hinterlegt ist die Pause.

In Abbildung 1 sind die beiden Studierenden 2.2[2] und 2.6 einander über den gesamten Verlauf der Übungsstunde gegenübergestellt. Der Studierende 2.6 hat mehrere Phasen von Inaktivität, was in dieser Grafik am waagrechten Verlauf der Kurve „compile_n“ und den hohen Peaks der Kurve „inactivity“ erkennbar ist. Durch das Betrachten der kleinen Anzahl Kompiliervorgänge könnte darauf geschlossen werden, dass der Studierende 2.6 kaum programmiert und Motivationsprobleme hat. Im Diagramm links wird aber ersichtlich, dass der Studierende 2.6 meist erfolgreich kompiliert. Eine mögliche Schlussfolgerung wäre, dass er mit den Aufgaben unterfordert ist. Werden die erfolgreichen (compile_s) und fehlgeschlagenen (compile_f) Kompiliervorgänge des Studierenden 2.2 in Relation zu seinem sehr häufigen Kompilieren gestellt, zeigt sich, dass dieser Studierende Unterstützung durch den Dozierenden benötigt.

Eine mögliche Klassifizierung der Studierenden in Gruppen ist in Tabelle 2 dargestellt. Die Messwerte wurden daraufhin untersucht, wie lange ein Studierender die Anwendung nutzte. Die Studierenden wurden in fünf Gruppen eingeteilt. Die größte Gruppe stellt die Gruppe von Studierenden dar, die zwei Doppellektionen durchgehend gearbeitet haben.

<50min	51-100min	101-150min	151-200min		>201min
1.3	1.14	1.16	1.1	1.11	2.2
1.6	1.19	2.11	1.5	1.12	2.6
1.15	2.13		1.7	1.17	2.9
	2.17		1.9	2.5	2.12
			1.10	2.14	

Tab. 2: Studierendengruppen anhand der Dauer in Minuten, wie lange die Studierenden die Entwicklungsumgebung nutzten.

<10min	11-20min	21-30min	31-40min	41-50min	>51min
1.3	1.14	1.5	1.1	1.9	2.7
1.6	1.19	1.12	1.10	2.6	2.16
1.15		1.17	1.11		2.9
		2.2	2.13		2.12
		2.5			2.14
		2.11			
		2.17			

Tab. 3: Studierendengruppen anhand der Dauer in Minuten, wie lange sie inaktiv waren.

Tabellenübergreifend wurden anschließend einzelne Studierende hervorgehoben, um weitere Schlussfolgerungen daraus zu erhalten. Werden Tabelle 2 und Tabelle 3 miteinander verglichen, kann erkannt werden, dass der Studierende 2.12 die Anwendung zwar sehr lange offen hatte, jedoch auch beinahe eine Stunde am Stück inaktiv war. Der Studierende 2.2 war hingegen nur solange inaktiv, wie die Pause lang war.

[2] Die Zahl vor dem Punkt gibt an, aus welcher Testgruppe der Studierende stammt. Die Zahl danach separiert die Studierenden innerhalb der Testgruppe.

4 Zusammenfassung und Ausblick

Als besonders aussagekräftig haben sich die Metriken rund um die Kompilier-Ereignisse und die Aktivitätsmessung gezeigt. Sieht der Dozierende den Bedarf für eine Hilfestellung, so kann er entsprechend reagieren.

In der nächsten Version sollen zudem einige Auswertungen nur die letzten Minuten repräsentieren und können pro Aufgabe zurückgesetzt werden. So wirken sich lokale Maxima oder Schwierigkeiten mit einer einzelnen Aufgabe nicht über den gesamten Übungsverlauf aus. Das Dashboard wird um eine Zeitkomponente ergänzt, damit auch vergangene Werte sichtbar gemacht werden.

Im Frühlingssemester 2017 wird die zweite Version der Erweiterung zum Einsatz kommen. Diese ermöglicht neben den bisher vorgestellten Metriken auch Quellcode zu analysieren und für die Auswertung zu verwenden. Testweise wurde die Bibliothek Checkstyle [Ch17] zur Überprüfung des Programmierstils eingefügt. Die Anwendung soll in größerem Umfang in mehreren Klassen getestet werden. Die während dieser Zeit gewonnenen Erkenntnisse können für die Weiterentwicklung und Verbesserung berücksichtigt werden. Das Ziel ist es, dieses Werkzeug flächendeckend im Informatik-Grundlagen-Unterricht einzusetzen.

Literaturverzeichnis

[BK09] Barnes, D. J.; Kölling, M.: Java lernen mit BlueJ. Eine Einführung in die objektorientierte Programmierung. 4. aktualisierte Auflage, Pearson Studium (Informatik), München, 2009.

[Bl17] BlueJ Java Development Environment, www.bluej.org, Stand: 20.02.2017.

[Ch17] Checkstyle, www.checkstyle.sourceforge.net/, Stand: 20.02.2017.

[Hu06] Hundhausen, C. D.; Brown, J. L.; Farley, S.; Skarpas, D.: A Methodology for Analyzing the Temporal Evolution of Novice Programs Based on Semantic Components. In ACM: ICER '06, Canterbury, United Kingdom, S. 59-71, 2006.

[STL13] Slotta J. D.; Tissenbaum M.; Lui M.: Orchestrating of complex inquiry: Three roles for learning analytics in a smart classroom infrastructure. In ACM: LAK '13, Leuven, Belgium, S. 270-274, 2013.

[Kö07] Kölle Ralph: Java lernen in virtuellen Teams. Kompensation defizitärer Rollen durch Simulation. 1. Auflage, Hülsbusch, Boizenburg, 2007.

[BHS16] Breitner J.; Hecker M.; Snelting G.: Der Grader Praktomat, In (Bott O. J. et al., Hrsg.): Automatisierte Bewertung in der Programmierausbildung, 2016.

[Ki13] Kiesmüller U.: Prozessbegleitende automatisierte Identifizierung von Problemlösestrategien Lernender beim Lösen algorithmischer Probleme mit Programmierumgebungen in der Sekundarstufe I. Shaker, 2013.