

10 Jahre automatische Bewertung von Programmieraufgaben mit JACK – Rückblick und Ausblick

Michael Goedicke¹ und Michael Striewe²

Abstract: Das E-Assessment-System JACK ist seit nunmehr 10 Jahren für die Programmierausbildung von Studierenden im ersten Fachsemester im Einsatz. Der Beitrag blickt auf die dabei gemachten Erfahrungen zurück, ordnet sie in den Kontext der aktuellen Forschung ein und stellt einige Thesen über die zukünftige Entwicklung automatischer Übungs- und Prüfungssysteme in der Informatik auf.

Keywords: E-Assessment, Programmierausbildung, Feedback

1 Einleitung

Die frühen 2000er-Jahre bescherten vielen Informatik-Studiengängen stark wachsende Teilnehmerzahlen und damit auch viele Studierende in den Einführungsvorlesungen zur Programmierung. Auch an der Universität Duisburg-Essen ging dieser Trend nicht vorbei. Es konnte jedoch bald beobachtet werden, dass sich eine große Mehrheit der Studierenden nicht intensiv mit der Materie beschäftigte und insbesondere auf praktische Übungen verzichtete sowie entsprechende Übungsaufgaben nur unzureichend bearbeitete. Dies kann freilich nicht zum gewünschten Lernerfolg führen [Ko15]. Um dem entgegen zu wirken, wurden zunächst semesterbegleitende Testate in Form kurzer Prüfungsgespräche eingeführt, die sich jedoch sehr schnell als viel zu hohe organisatorische und zeitliche Belastung für den Lehrstuhl herausstellten.

Daraus entstand der Bedarf nach einer automatisierten Lösung, zu deren Realisierung ab 2006 das System JACK entwickelt wurde. Ein besonderer Fokus der Eigenentwicklung lag im Vergleich zu damals bereits verfügbaren vergleichbaren Systemen auf einer flexiblen Systemarchitektur und der Ermöglichung aufgabenspezifischer statischer Überprüfungen. Sehr schnell kam auch eine Integration in die Programmierumgebung Eclipse hinzu, da diese als Standardsoftware in den Rechnerpools zur Verfügung steht, über die die semesterbegleitenden Testate abgewickelt werden. Den ersten Einsatz im Testatbetrieb hatte JACK im Wintersemester 2006/07. Ein Jahr später wurde JACK auch in vollem Umfang im Übungsbetrieb eingesetzt [SGB08].

¹ paluno – The Ruhr Institute for Software Technology, Universität Duisburg-Essen, Gerlingstraße 16, 45127 Essen, michael.goedicke@paluno.uni-due.de

² paluno – The Ruhr Institute for Software Technology, Universität Duisburg-Essen, Gerlingstraße 16, 45127 Essen, michael.striewe@paluno.uni-due.de

2 Didaktisches Konzept und Einsatzerfahrungen

Für die Studiengänge „Angewandte Informatik“ und „Wirtschaftsinformatik“ ist die Einführung in die objektorientierte Programmierung mit Java Pflichtveranstaltung im ersten Semester. Der Einsatz von JACK ist zunächst genau auf diese Situation zugeschnitten und an mehreren Stellen in das Veranstaltungskonzept der Programmiervorlesung eingebettet: Alle zwei Wochen im Semester finden Testate statt, die von den Studierenden in 45 Minuten unter Aufsicht im Rechnerpool bearbeitet werden. Die eingereichten Lösungen werden von JACK bewertet und mit Feedback versehen, das nach dem Testat eingesehen werden kann. Aus den Testaten muss eine Mindestpunktzahl erworben werden, um zur Abschlussklausur zugelassen zu werden. Zu jedem Testat gibt es eine größere vorbereitende Übungsaufgabe, die von zu Hause aus bearbeitet werden soll. Auch diese werden von JACK bewertet und mit Feedback versehen. Die Übungsaufgaben müssen bearbeitet werden, um am jeweiligen Testat teilnehmen zu dürfen. Die erreichte Punktzahl ist dabei jedoch unerheblich.

Darüber hinaus gibt es im Semesterverlauf in loser Folge weitere freie Übungsaufgaben sowie eingebettete Übungen in der Vorlesung, die ebenfalls in JACK bearbeitet werden können, um Feedback zu erhalten. Ferner wird ein freiwilliges Programmier Tutorium angeboten, in dem studentische Tutoren bei allen Fragen rund um den Vorlesungsstoff und die Übungsaufgaben weiterhelfen. Auch die Einsicht in die Bewertung der Testate erfolgt im Rahmen dieses Tutoriums.

Insgesamt wurden vom Wintersemester 2006/07 bis einschließlich Wintersemester 2016/17 mehr als 65.000 Testat-Lösungen und über 100.000 Lösungen zu vorbereitenden Übungsaufgaben geprüft und mit Feedback versehen. Im Schnitt reichen die Studierenden 3,6 Lösungen zu Übungsaufgaben und 2,5 Lösungen zu Testataufgaben ein, wobei es in den ersten Semestern eine gewisse Zurückhaltung gab und die Zahlen in den jüngeren Jahrgängen steigen. Nicht berücksichtigt sind in diesen Zahlen die Einreichungen zu freien Übungsaufgaben und eingebetteten Übungen.

Der Einsatz von JACK wurde in mehreren Semestern durch eine umfangreiche Evaluation begleitet um herauszufinden, ob JACK von den Studierenden als hilfreich für Übungen oder Testate betrachtet wird. In der Summe mehrerer Befragungen, in die die Meinungen von über 300 Studierenden einfließen, wurde JACK von 75% als vollständig oder weitgehend nützlich im Übungsbetrieb sowie von 68% als vollständig oder weitgehend nützlich im Testatbetrieb betrachtet.

Diese positiven Erfahrungen haben dazu geführt, dass der Funktionsumfang von JACK kontinuierlich erweitert wurde. Die flexible Architektur ermöglichte es insbesondere, mit geringem Aufwand – zum Teil im Rahmen studentischer Projekt- und Abschlussarbeiten – die automatische Überprüfung weiterer Programmiersprachen (z.B. Python und C#) zumindest prototypisch zu realisieren [Str16]. Mangels Lehrveranstaltungen in diesen Sprachen wurden diese Erweiterungen jedoch bisher nicht großflächig eingesetzt. Die Erfahrungen konnten aber genutzt werden, um vergleichbare technische und didaktische

Konzepte auch auf Übungen in der Statistik-Sprache R für den Einsatz in Statistik-Vorlesungen der Wirtschaftswissenschaften zu realisieren. Auch dafür wurden inzwischen Übungen und Testate in Grundlagenvorlesungen erfolgreich absolviert.

3 Stand der Forschung und Technik

Weder mit seinem Alter noch mit dem Einsatzszenario hebt sich JACK signifikant von vergleichbaren Systemen ab [Iha+10] [KJH16]. Tatsächlich kann der Einsatz von automatischen Bewertungs- und Feedbacksystemen für den Übungs- oder Prüfungsbetrieb heute sowohl von den technischen Möglichkeiten als auch von der didaktischen Einbettung her als etablierter Stand der Technik betrachtet werden. Es gibt jedoch noch eine große Spannbreite bei der Art und Weise, wie diese Systeme Feedback ermitteln und bereitstellen sowie bei der Granularität der erzeugten Bewertungen.

Aus didaktischer Sicht ist möglichst umfangreiches Feedback wünschenswert, um selbstgesteuertes Lernen zu ermöglichen [BW95] [Fal+14]. Auch wenn klar ist, dass ein automatisiertes System einen menschlichen Tutor nicht ersetzen kann und soll, sollte der Qualitätsverlust, der mit dem Ersetzen eines menschlichen Tutors durch ein automatisiertes System einhergeht, doch so gering wie möglich gehalten werden. Die genauen Stärken und Schwächen automatisierter Systeme im Vergleich zu menschlichem Feedback sind jedoch auch heute noch Gegenstand der Forschung [GDG14].

JACK setzt aufgrund der flexiblen Systemarchitektur auf den kombinierten Einsatz verschiedener Analysetechniken, um möglichst vielfältiges Feedback erzeugen zu können. Die klassische Ausführung von Testfällen wird dabei ergänzt durch eine Aufzeichnung von Traces, die es den Studierenden ermöglichen, das Verhalten ihrer Lösung auch ohne Einarbeitung in Debugging-Werkzeuge Schritt für Schritt nachzu-vollziehen. Ferner kann JACK Visualisierungen von Datenstrukturen erzeugen, die im Rahmen der Aufgabe von den Studierenden manipuliert werden. Dies ermöglicht neben der indirekten Kontrolle des Programmierergebnisses über Feedback und Bewertung sowie der textuellen Kontrolle über Konsolenausgaben auch eine visuelle Kontrolle.

Ein zweiter Schwerpunkt von JACK sind detaillierte statische Checks. Hier hebt sich JACK von vergleichbaren Systemen ab, indem keine externen Werkzeuge oder fertigen Kataloge von statischen Prüfungen zum Einsatz kommen, sondern statische Code-Checks individuell pro Aufgabe konfiguriert werden können. So können nicht nur stilistische Schwächen oder typische Programmierfehler erkannt werden, sondern es kann beliebig detailliertes Feedback zu einzelnen Konstrukten gegeben werden.

4 Ausblick

Aus didaktischer Sicht besonders interessant und vielversprechend ist, dass mit der zu-

nehmenden Verbreitung automatisierter Systeme zur Programmbewertung nun eine hinreichend große und umfassend analysierte (oder zumindest analysierbare) Menge an Daten verfügbar ist, auf deren Basis umfassende Studien zu Kompetenzmodellen und –strukturen für verschiedene Aspekte der Programmierung betrieben werden können [Iha+15]. Erste Ansätze, mit denen statistische Modelle wie das Rasch-Modell auf Programmieraufgaben angewendet werden können, wurden bereits im Kontext von JACK erprobt [Hub+17]. Es besteht die begründete Hoffnung, dass mit weiterer Forschung in den kommenden Jahren eine weitere Ebene des Feedbacks erschlossen werden kann, indem automatisierte Systeme nicht mehr nur Feedback zur konkreten Lösung einzelner Aufgaben geben, sondern auch Hinweise auf fehlenden Kompetenzen, die sowohl Lernenden als auch Lehrenden wertvolle Informationen liefern können.

Eine weitere sowohl didaktische als auch technische Forschungs herausforderung besteht in der Variabilität von Programmieraufgaben. Während es beispielsweise in der Mathematik möglich und üblich ist, Varianten von Aufgaben durch das geschickte Austauschen von Zahlenwerte zu erzeugen, fehlen derartige Konzepte für Programmieraufgaben nahezu vollständig. Angesichts des Aufwandes, der für das Erstellen einer guten Programmieraufgabe notwendig ist und der immer noch hohen Teilnehmerzahlen in Vorlesungen und Testaten sind Fortschritte in diesem Bereich jedoch sehr wünschenswert. Programmieraufgaben sind jedoch in der Regel so gestellt, dass der zu entwickelnde Programmcode eine Klasse von Problemen löst und nicht wie in der Mathematik eine konkrete Instanz eines Problems. Variabilisierung muss hier also auf einer anderen Ebene erfolgen und erfordert daher weitere intensive Forschung.

Verwandt mit der Frage der Variabilisierung ist die Frage nach dem Austausch von Aufgaben zwischen verschiedenen Systemen. Auch dazu gibt es in anderen Bereichen bereits standardisierte Formate, die für Programmieraufgaben noch weitgehend fehlen. Erste Lösungen werden vom ProFormA-Format angeboten, dass zumindest die flexible Beschreibung von Aufgaben und Bewertungsverfahren systemunabhängig unterstützt und von mehreren Systemen implementiert wird [Str+15]. Eine teilweise Unterstützung des Formats ist auch in JACK verfügbar, aber auch hier ist weder technisch noch inhaltlich-fachlich das Ende der Entwicklung bereits erreicht.

Aus anderen fachlichen Domänen – insbesondere der Mathematik – gibt es zudem beim Einsatz von JACK gute Erfahrungen mit mehrstufigen Aufgaben mit vermischten Item-Formaten, bei denen in späteren Stufen auf frühere Eingaben zurückgegriffen wird oder automatisch Folgefragen generiert werden. Es ist technisch denkbar und didaktisch wünschenswert, vergleichbare Konzepte auch für Programmieraufgaben zu entwickeln. So könnten in einer komplexen Aufgabe zunächst konzeptionelle Aspekte zu einer gegebenen Problemstellung abgefragt werden, um auf Basis der Antworten eine individuelle Programmieraufgabe zu generieren. Nach Eingabe eines entsprechenden Lösungsversuchs werden dann wiederum automatisch Folgefragen generiert, in denen die Studierenden ihren eigenen Code erläutern sollen, oder in denen eine alternative Lösung vergleichend diskutiert werden soll. Die automatische Bewertung von Programmieraufgaben würde damit im Sinne einer umfangreichen Analyse der abgegebenen Lösungen eine

noch deutlich weitergehende Einbettung in verschiedene Lernszenarien erfahren, als dies heute der Fall ist.

Literaturverzeichnis

- [BW95] Butler, D.; Winne, P.: Feedback and Self-Regulated Learning: A Theoretical Synthesis. *Review of Educational Research*, 65 (3), 1995; S. 245-281.
- [Fal+14] Falkner, N. et.al.: Increasing the Effectiveness of Automated Assessment by Increasing Marking Granularity and Feedback Units. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education. SIGCSE '14*. ACM, 2014, S. 9–14. DOI: 10.1145/2538862.2538896.
- [GDG14] Gaudencio, M.; Dantas, A.; Guerrero, D. D. S.: Can Computers Compare Student Code Solutions as Well as Teachers? *Proceedings of the 45th ACM Technical Symposium on Computer Science Education, SIGCSE'14*, 2014.
- [Hub+17] Hubwieser, P.; Striwe, M.; Berges, M.; Goedicke, M.: Towards Competency Based Testing and Feedback. *Proceedings of IEEE Global Engineering Education Conference (EDUCON)*, 2017
- [Iha+10] Ihantola, P. et.al.: Review of Recent Systems for Automatic Assessment of Programming Assignments. In: *Proceedings of the 10th Koli Calling International Conference on Computing Education Research. Koli Calling '10*. ACM, 2010, S. 86–93. DOI: 10.1145/1930464.1930480.
- [Iha+15] Ihantola, P.; Vihavainen, A.; Ahadi, A.; Butler, M.; Börstler, J.; Edwards, S. H.; Isohanni, E.; Korhonen, A.; Petersen, A.; Rivers, K.; Rubio, M. Á.; Sheard, J.; Skupas, B.; Spacco, J.; Szabo, C.; Toll, D.: Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. *Proceedings of the 2015 ITiCSE Working Group Reports*, 2015, 41-63
- [KJH16] Keuning, H.; Jeurig, J.; Heeren, B.: Towards a Systematic Review of Automated Feedback Generation for Programming Exercises - Extended Version. *Technical Report*. Utrecht University, 2016
- [Ko15] Koedinger, K. et. al.: Learning is not a Spectator Sport. In *Proceedings of the Second ACM Conference on Learning @ Scale*. ACM, New York, 2015; S. 111-120.
- [SGB08] Striwe, M.; Goedicke, M.; Balz, M.: Computer Aided Assessments and Programming Exercises with JACK. *Techn. Ber. 28. ICB*, University of Duisburg-Essen, 2008.
- [Str+15] Strickroth, S.; Striwe, M.; Müller, O.; Priss, U.; Becker, S.; Rod, O.; Garmann, R.; Bott, J. O.; Pinkwart, N.: ProFormA: An XML-based exchange format for programming tasks. *eled*, 2015, 11
- [Str16] Striwe, M.: An architecture for modular grading and feedback generation for complex exercises. In: *Science of Computer Programming 129* (2016), S. 35–47. DOI: 10.1016/j.scico.2016.02.009.