Debuggen im Unterricht – Ein systematisches Vorgehen macht den Unterschied

Tilman Michaeli[1], Ralf Romeike[2]

Abstract:

Selbstständig Fehler in Programmcode zu finden und zu beheben stellt sowohl eine wichtige Fähigkeit als auch eine große Herausforderung beim Programmierenlernen dar. Debuggen unterscheidet sich von allgemeinen Programmierfertigkeiten und muss explizit gelehrt werden. Dennoch gibt es überraschend wenige Studien, Materialien und Konzepte, die sich mit der expliziten Vermittlung von Debuggingfähigkeiten beschäftigen. Eine relevante Debuggingfähigkeit ist ein systematisches Vorgehen bei der Fehlersuche. Dieser Beitrag analysiert die Wirksamkeit einer expliziten Vermittlung eines solchen systematischen Vorgehens im Unterricht, insbesondere hinsichtlich der Selbstwirksamkeitserwartungen und der resultierenden Debuggingleistung der Schülerinnen und Schüler. Zu diesem Zweck haben wir eine Intervention entwickelt, pilotiert und anschließend in einem Pre-Post-Kontrollgruppen-Test-Design untersucht. Die Ergebnisse zeigen sowohl einen signifikanten Anstieg der Selbstwirksamkeitserwartungen als auch der Debuggingleistung in der Versuchsgruppe.

Keywords: Debugging; Interventionsstudie; Systematisches Vorgehen

1 Einleitung

Programmieren erfordert eine Vielzahl an Kompetenzen, entsprechend stellt deren Vermittlung eine zentrale Herausforderung des Informatikunterrichts dar. Dabei müssen Schülerinnen und Schüler nicht nur Programmierkonzepte erlernen, sondern auch dazu befähigt werden, Lösungen zu finden, wenn sie mit Fehlern konfrontiert werden. Programme systematisch auf Fehler zu untersuchen, sie zu finden und zu beheben, stellt eine zentrale Kompetenz professioneller Entwicklerinnen und Entwickler dar, die zwischen 20 und 40 Prozent ihrer Arbeitszeit dafür aufwenden [Pe17]. Allerdings haben gerade Programmieranfängerinnen und -anfänger große Probleme im Umgang mit Fehlern. Dies stellt ein erhebliches Hindernis beim Programmierenlernen dar.

Darüber hinaus wird Debuggen im Kontext von Computational Thinking diskutiert [Ya11], und findet sich prominent in neueren Curricula wie dem britischen „Computing Curriculum".

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Didaktik der Informatik, Martensstraße 3, 91058 Erlangen tilman.michaeli@fau.de

[2] Freie Universität Berlin, Didaktik der Informatik, Königin-Luise-Str. 24-26, 14195 Berlin ralf.romeike@fu-berlin.de

In deutschen Lehrplänen kommt der Begriff „debuggen“ nur selten vor, allerdings fordern fast alle die Fähigkeit, „mit Fehlern umzugehen“. Dennoch gibt es überraschend wenige Studien, die sich mit der expliziten Vermittlung von Debuggingfähigkeiten beschäftigen. Gleichzeitig mangelt es Lehrerinnen und Lehrern an Konzepten und Materialien für den Unterricht [MR19]. Eine relevante Debuggingfähigkeit ist ein systematisches Vorgehen bei der Suche und beim Beheben von Fehlern.

Das Ziel dieser Interventionsstudie ist es, den Einfluss einer expliziten Vermittlung eines systematischen Vorgehens zum Debuggen zu untersuchen. Im Fokus stehen dabei der Einfluss auf *(RQ1) Selbstwirksamkeitserwartungen* und *(RQ2) Debuggingleistung*. Dazu haben wir eine Intervention entwickelt, pilotiert und anschließend in einem Pre-Post-Kontrollgruppen-Test-Design untersucht.

2 Hintergrund

Debuggen beschreibt den Prozess der Suche und des Behebens von Fehlern. Debuggingfähigkeiten unterscheiden sich von allgemeinen Programmierfertigkeiten, wie [AEH05] oder [Mu08b] zeigen. Während ausgeprägte Debuggingfähigkeiten in der Regel auf entsprechende Programmierfähigkeiten schließen lassen, gilt die Umkehrung nicht unbedingt: Gute Programmierer sind nicht notwendigerweise auch gute Debugger. Dies wirft die Frage auf: Was macht einen „guten“ Debugger aus?

Als eine relevante Debuggingfähigkeit wird die Anwendung eines **systematischen Vorgehens**, also die systematische Verfolgung eines zielgerichteten Plans zur Fehlersuche und -behebung, gesehen. Dazu werden wiederholt Hypothesen formuliert, in Experimenten überprüft und gegebenenfalls verfeinert, bis die Ursache des Fehlers gefunden ist (vgl. z.B. [Gi91]). Oftmals kann dieser Prozess abgekürzt werden: Durch Erfahrung und damit die **Anwendung von Heuristiken und Pattern**, sind typische Fehler und deren mögliche Ursachen bekannt. Um dieses „Lernen aus früheren Fehlern“ zu unterstützen, führen viele professionelle Entwickler ein Debugging-„Tagebuch“ mit dem sie ihre Debugging-Erfahrung dokumentieren [Pe17]. Darüber hinaus spielt die Anwendung von **Debuggingstrategien** eine große Rolle im Debuggingprozess: Durch Strategien, wie beispielsweise dem Tracen des Kontrollflusses durch *print-f*-Debugging, dem Auskommentieren von Code oder Slicing, können Informationen gewonnen werden, die bei der Formulierung von Hypothesen und damit der Lokalisierung des Fehlers helfen [Sp18]. Ähnliches wird durch die Verwendung von **Werkzeugen** wie dem Debugger ermöglicht.

Murphy et al. [Mu08a] genauso wie Kessler and Anderson [KA86] argumentieren, dass Debuggingfähigkeiten explizit unterrichtet werden sollten. Nichtsdestotrotz gibt es überraschend wenige Studien sowohl in Bezug auf die universitäre Lehre als auch auf den Unterricht, die sich mit der expliziten Vermittlung von Debugginginhalten beschäftigen.

2.1 Debuggen in der Hochschullehre

Chmiel und Loui [CL04] verwendeten freiwillige Debuggingaufgaben, um die Debuggingfähigkeiten der Studierenden zu fördern. Es stellte sich heraus, dass Studierende, die die freiwilligen Debuggingaufgaben bearbeitet hatten, deutlich weniger Zeit für das Debuggen ihrer eigenen Programme benötigten. Dieser Zusammenhang spiegelte sich jedoch nicht in den Klausurergebnissen wider, die entgegen der Erwartungen nur leicht besser waren.

Katz und Anderson [KJ87] untersuchten den Effekt der Vermittlung verschiedener Vorgehensweisen (forward-reasoning, backward-reasoning) beim Debuggen. Unterschiedlichen Studierendengruppen wurde zunächst jeweils eine der Vorgehensweisen vermittelt, ehe sie ihr Vorgehen frei wählen konnten. Dabei zeigte sich, dass Studierende weiterhin das ihnen vermittelte Vorgehen anwendeten.

Allwood und Björhag [AB91] untersuchten, inwieweit schriftliche Debugging-Hinweise den Prozess unterstützen können. Während sich die Anzahl der Fehler zwischen Versuchs- und Kontrollgruppe nicht unterschied, war die Anzahl der beseitigten Fehler (insbesondere semantischer und logischer Art) bei Verfügbarkeit schriftlicher Hinweise signifikant höher. Da gleichzeitig keine Unterschiede in den verwendeten Strategien zwischen den Gruppen erkennbar waren, folgerten die Autoren, dass die Unterschiede auf einer höheren Ebene liegen müssen und vor allem ein systematisches Vorgehen beim Debuggen entscheidend sei.

Böttcher et al. [Bo16] vermittelten ein systematisches Debuggingvorgehen sowie die Verwendung des Debuggers in einer expliziten Einheit. Dabei wurde das Debuggingverfahren in einer Live-Demonstration verdeutlicht und eine Übung mit Debuggingaufgaben durchgeführt. Die Auswertung zeigte, dass nur wenige Studierenden den vermittelten systematischen Ansatz anwendeten, sondern schnell zu einem unsystematischen „Herumstöbern“ zurückkehrten.

2.2 Debuggen im Unterricht

Carver und Risinger [CR87] vermittelten einen Debuggingprozess mit LOGO mit vielversprechenden Ergebnissen: Sie gaben den Schülerinnen und Schülern eine Stunde Debugging-Training als Teil eines größeren LOGO-Curriculums. Sie nutzten ein Flow-Chart, das den Debuggingprozess charakterisiert, „Bug Mappings“ und Debugging-Tagebücher, die während der gesamten Zeit im Klassenzimmer vorhanden waren. Die Ergebnisse (ohne Kontrollgruppe) zeigten einen Wechsel von Brute-Force hin zu einem systematischen Vorgehen bei der Suche nach Fehlern. Darüber hinaus wurde für die Fehlersuche deutlich weniger Zeit benötigt. Die Schülerinnen und Schüler formulierten vor dem Ausprobieren des Codes mehr Hypothesen, achteten stärker auf den Kontrollfluss, nahmen weniger Code-Änderungen vor (insbesondere an fehlerfreien Stellen) und machten weniger neue Fehler.

In einer qualitativen Interviewstudie unter Informatiklehrkräften untersuchten [MR19] die aktuelle Rolle, die Debuggen im Unterricht spielt. Dabei ist festzustellen, dass Lehrkräfte bei Programmierproblemen viel und oft individuelle Hilfestellung leisten müssen. In der Folge eilen sie häufig von Schüler-Rechner zu Schüler-Rechner und versuchen zu helfen (gemeinhin als „Turnschuhdidaktik“ bekannt). Weiterhin mangelt es Lehrkräften an Konzepten und Materialien für die Vermittlung von Debugging. Insbesondere ist die Vermittlung eines systematischen Vorgehens – obgleich dessen Bedeutung – in der Schule unterrepräsentiert.

Zusammenfassend zeigt sich, dass nur wenige Untersuchungen zur expliziten Vermittlung von Debuggingfähigkeiten existieren und ein Großteil der Studien älter als 25 Jahre ist. Nichtsdestotrotz demonstrieren die vorhandenen Ergebnisse, dass sich Debuggen explizit vermitteln lässt. Sie deuten zudem darauf hin, dass ein systematisches Vorgehen eine entscheidende Rolle in einem erfolgreichen Debuggingprozess spielen kann. Eine empirische Untersuchung, inwieweit Programmieranfängerinnen und -anfänger innerhalb eines Unterrichtssettings von der Vermittlung eines solchen systematischen Vorgehens profitieren, fehlt bislang.

3 Vorgehen

Ziel dieser Untersuchung ist es, den Einfluss einer expliziten Vermittlung eines systematischen Vorgehens zum Debuggen im Unterricht zu untersuchen. Wir adressieren damit folgende Forschungsfragen:

- (**RQ1**) Hat die Vermittlung eines systematischen Vorgehens einen positiven Effekt auf die Selbstwirksamkeitserwartungen der Schülerinnen und Schüler?
- (**RQ2**) Hat die Vermittlung eines systematischen Vorgehens einen positiven Effekt auf die Debuggingleistung der Schülerinnen und Schüler?

3.1 Untersuchsungsdesign

Um diese Forschungsfragen zu beantworten, haben wir ein Pre-Post-Kontrollgruppen-Test-Design gewählt. Zunächst wurde die Intervention in einer 10. Klasse für besonders leistungsstarke Schülerinnen und Schüler ($n = 14$, Greenfoot und Stride) pilotiert, um ausgehend von den gewonnenen Erkenntnissen der Durchführung Anpassungen vorzunehmen. Ergebnisse aus einer solchen Untersuchung ohne Kontrollgruppe helfen uns bei der Beantwortung der Forschungsfragen allerdings nur eingeschränkt, da mögliche Zuwächse in Selbstwirksamkeitserwartungen und Leistung der Schülerinnen und Schüler auch lediglich auf die zusätzliche Übung im Debuggen zurückzuführen seien könnten. Um den Einfluss der Intervention im Gegensatz zum reinen Üben von Debuggen, z.B. durch Debuggingaufgaben, zu untersuchen, haben wir zwei 10. Klassen als Versuchs- ($n = 13$) und Kontrollgruppe

($n = 15$) herangezogen. Dabei wurden explizit zwei Klassen ausgewählt, die von derselben Lehrkraft mit dem identischen Unterrichtskonzept (unter Verwendung von BlueJ und Java) unterrichtet wurden und die im Curriculum zum Untersuchungszeitpunkt gleich weit fortgeschritten waren.

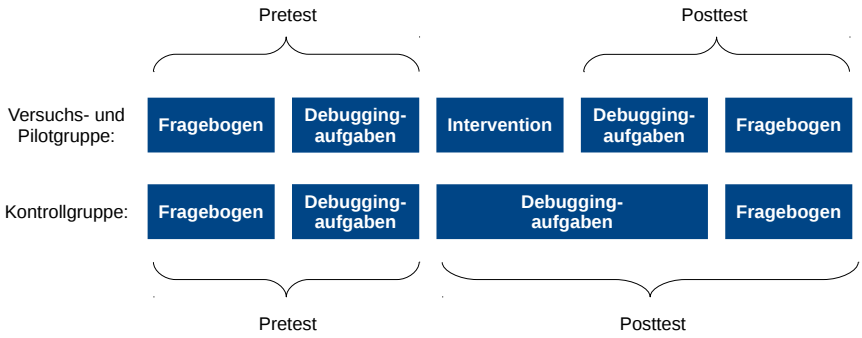

Abb. 1: Untersuchungsdesign

Der Ablauf der jeweils 90-minütigen Unterrichtseinheiten bestand aus einem Pretest, der Intervention (außer in der Kontrollgruppe) und einem Posttest. Wie in Abbildung 1 dargestellt, unterteilten sich Pre- und Posttest jeweils in einen Fragebogen zur Erhebung der Selbstwirksamkeitserwartungen (vier Items mit fünfstufiger Likert-Skala) und der Lösbarkeit der Aufgaben (nur im Posttest) sowie in Debuggingaufgaben zur Erhebung der Leistung der Schülerinnen und Schüler. Für Letzteres wurde die Anzahl der behobenen Fehler (analog zu [Mu08b]) herangezogen, wofür sowohl die Arbeitsblätter, auf denen Fehler und deren Behebung von allen Gruppen notiert werden sollten, als auch der Code ausgewertet wurde.

3.2 Intervention

Die Intervention vermittelt ein systematisches Vorgehen für die Suche und das Beheben von Fehlern. Das Vorgehen lehnt sich an die sogenannte *wissenschaftliche Methode* [Ze05] an, die von professionellen Entwicklerinnen und Entwicklern typischerweise implizit angewendet wird [Pe17]: Es werden wiederholte Hypothesen formuliert, in Experimenten verifiziert und gegebenenfalls verfeinert, bis die Ursache gefunden ist. Wir verwenden dabei eine didaktisch angepasste Variante dieses Vorgehens und unterscheiden explizit zwischen unterschiedlichen Fehlertypen: Compilezeit-, Laufzeit- und logischen Fehlern (vgl. Abbildung 2). Es wird das *Rückgängigmachen* von Änderungen betont, falls Maßnahmen zur Fehlerbehebung nicht erfolgreich sind – gerade da dieses Vorgehen für Schülerinnen und Schüler unnatürlich ist [Si08]. Hierdurch soll vermieden werden, dass die Schülerinnen

und Schüler im Zuge einer fehlgeschlagenen Fehlerbehebung zusätzliche Fehler einbauen – ein typisches Phänomen für Programmieranfängerinnen und -anfänger [GO86].

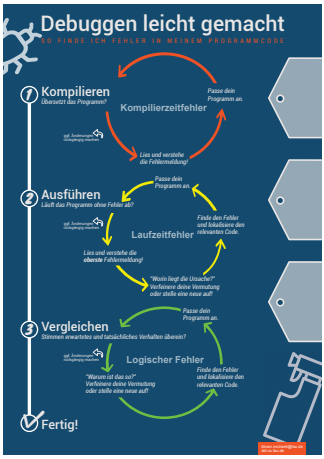

Abb. 2: Vermitteltes Vorgehen

Zum Üben des Debuggens werden typischerweise Debuggingaufgaben verwendet. Allerdings konfrontieren diese die Schülerinnen und Schüler mit einer großen Menge an fremdem Programmcode. Das Verständnis und die Einarbeitung in fremden Code stellt jedoch eine große Herausforderung dar [AB90]. Um sich dem eigentlichen Lern- und Untersuchungsziel – dem Debuggen *eigener* Programme – anzunähern, verwenden wir daher mehrere aufeinander aufbauende Prototypen eines Programms. Auf diese Art und Weise sehen sich die Schülerinnen und Schüler in jedem neuen Prototypen nur mit vergleichsweise wenig „fremdem" Code konfrontiert und kennen sich im „alten" Code bereits aus. Beispielsweise ist im ersten Prototypen des in der Pilotgruppe verwendeten Pongspiels lediglich die Bewegung des Balles umgesetzt, und im nächsten werden zusätzlich die Schläger und deren Steuerung eingefügt. Da die Debugging- und nicht die Testfähigkeiten der Schülerinnen und Schüler untersucht werden sollten, war die Anzahl der vorhandenen Fehler je Prototyp gegeben. Aus dem gleichen Grund wurde darauf geachtet, dass das Fehlverhalten des Programms schnell ersichtlich war, sodass direkt mit der Fehlerlokalisation begonnen werden konnte.

4 Ergebnisse

4.1 (RQ1) Hat die Vermittlung eines systematischen Vorgehens einen positiven Effekt auf die Selbstwirksamkeitserwartungen der Schülerinnen und Schüler?

Zunächst untersuchen wir den Zuwachs der Selbstwirksamkeitserwartungen für Pilot-, Versuchs- und Kontrollgruppe pre und post, der sich als Mittelwert der vier Items ergibt. Die Antworten der fünfstufigen Likert-Skala wurden auf die Skala 0 (stimme nicht zu) bis 4 (stimme zu) abgebildet. Die Mittelwerte bewegen sich folglich zwischen 0 und 4.

Wir ermitteln, ob ein signifikanter Anstieg der Selbstwirksamkeitserwartungen zwischen Pre- und Post-Test innerhalb der einzelnen Gruppen feststellbar ist. Aufgrund der Stichprobengrößen verwenden wir dazu stets nicht-parametrisierte Verfahren zur Prüfung auf Signifikanz [Ra10]. Mithilfe des Wilcoxon-Vorzeichen-Rang-Tests – einem nicht-parametrischen Test für abhängige Stichproben – analysieren wir die Rangfolgen im Pre- und Posttest. In Tabelle 1 sind die jeweiligen Mediane und der p-Wert des Wilcoxon-Vorzeichen-Rang-Tests (H_0: Kein oder negativer Versuchseffekt) dargestellt [3].

	Median pre	Median post	Wilcoxon-Test
Pilotgruppe	2, 75	3, 25	$p = 0,044^*$
Kontrollgruppe	2, 25	2, 50	$p = 0,083$
Versuchsgruppe	2, 25	2, 75	$p = 0,001^*$

Tab. 1: Einfluss auf Selbstwirksamkeitserwartungen

Wir sehen also in allen drei Gruppen einen Anstieg der Selbstwirksamkeiterwartungen. Dieser ist allerdings nur für die Pilot- und die Versuchsgruppe signifikant auf einem Signifikanzniveau von $\alpha = 0,05$. Die Effektstärken nach Cohen liegen bei $d = 0,56$ (Pilot) bzw. $d = 0,54$ (Versuch), dies entspricht einem mittleren Effekt [Co88].

Obgleich das aktive Üben von Debugging die Selbstwirksamkeitserwartungen verbessert, scheint ein systematisches Vorgehen die Selbstwirksamkeitserwartungen stärker positiv zu beeinflussen.

4.2 (RQ2) Hat die Vermittlung eines systematischen Vorgehens einen positiven Effekt auf die Debuggingleistung der Schülerinnen und Schüler?

Für Unterschiede in der Debuggingleistung vergleichen wir Versuchs- und Kontrollgruppe in Pre- und Posttest. Die Debuggingleistung wird anhand der Anzahl der behobenen Fehler gemessen. Ein pre-post-Vergleich der Debuggingleistung innerhalb der einzelnen Gruppen analog zur Untersuchung der Selbstwirksamkeitserwartungen ist nicht zielführend, da in Pre- und Posttest unterschiedliche Fehler zu beheben waren.

Um festzustellen, ob sich die Leistung der Schülerinnen und Schüler der Versuchsgruppe signifikant von der Leistung der Schülerinnen und Schüler der Kontrollgruppe unterscheidet, prüfen wir, ob die beiden Stichproben derselben Grundgesamtheit entstammen. Nur wenn dies nicht der Fall ist, können wir von einem signifikanten Unterschied ausgehen. Auch hier greifen wir ob der Stichprobengröße wieder zu einem nicht-parametrisierten Test, dem Mann-Whitney-U-Test. Im Gegensatz zum Wilcoxon-Vorzeichen-Rang-Tests ist dieser Test für unabhängige Stichproben ausgelegt. Die p-Werte des Mann-Whitney-U-Tests (H_0: Stichproben kommen aus derselben Grundgesamtheit) sind in Tabelle 2 dargestellt.

[3] Signifikante Testergebnisse zu einem Signifikanzniveau von $\alpha = 0,05$ sind durch ein $*$ gekennzeichnet.

	Mann-Whitney-U-Test
Versuch- vs. Kontrollgruppe Pre	$p = 0,191$
Versuch- vs. Kontrollgruppe Post	$p = 0,049^*$

Tab. 2: Einfluss auf Debuggingleistung

Dementsprechend können wir die Nullhypothese für den Vergleich der Pre-Tests auf einem Signifikanzniveau von $\alpha = 0,05$ nicht ablehnen: Die Debuggingleistung der Schülerinnen und Schüler unterscheidet sich vor Durchführung der Intervention nicht signifikant. Im Gegensatz dazu zeigt sich ein signifikanter Unterschied im Posttest: Die Schülerinnen und Schüler der Versuchsgruppe weisen eine höhere Debuggingleistung (Median = 4, bei insgesamt 9 zu behebenden Fehlern) auf als die Schülerinnen und Schüler der Kontrollgruppe (Median = 2). Im Posttest wurden für das Ermitteln der Debuggingleistung Aufgaben mit höherem Schwierigkeitsgrad herangezogen, da in beiden Gruppen ein Lerneffekt zwischen Pre- und Posttest anzunehmen ist. Die Effektstärke nach Cohen liegt bei $d = 0,69$ und entspricht einem mittleren Effekt [Co88].

Die höhere Debuggingleistung spiegelt sich auch in der wahrgenommenen Schwierigkeit der Aufgaben durch die Schülerinnen und Schüler wider. Diese wurde ex post im Fragebogen mit Hilfe einer fünfstüfigen Likert-Skala erhoben. Wiederum auf die Skala 0 (stimme nicht zu) bis 4 (stimme zu) abgebildet, ergeben sich folgende Mittelwerte:

	Aufgaben Pre	Aufgaben Post
Kontrollgruppe	3,07	1,47
Versuchsgruppe	3,23	2,92

Tab. 3: Mittelwerte für *Aufgaben gut lösbar*

Die Ergebnisse lassen darauf schließen, dass ein systematisches Vorgehen den Unterschied machen kann: Wird Schülerinnen und Schülern ein solches systematisches Vorgehen an die Hand gegeben, so können diese ihren Erfolg beim Lokalisieren und Beheben von Fehlern signifikant verbessern.

5 Diskussion und Fazit

Welche Bedeutung haben diese Ergebnisse für den Informatikunterricht? Nach [MR19] fehlt es Informatiklehrkräften an geeigneten Konzepten für den Themenkomplex *Debuggen*: Zwar sind verschiedene Debuggingstrategien sowie der Umgang mit Werkzeugen wie dem Debugger teils Unterrichtsgegenstand, allerdings spielt die Vermittlung eines systematischen Vorgehens im Unterricht bisher kaum eine Rolle. Diese Studie unterstreicht dabei, wie wichtig es ist, ein systematisches Vorgehen zu vermitteln und liefert Ansatzpunkte für eine Umsetzung. Dabei ist die Unabhängigkeit von Werkzeugen und (textbasierten)

Programmiersprachen – die in der Schulpraxis eine große Heterogenität aufweisen – zu betonen, so wurde dieselbe Intervention sowohl mit Java und BlueJ, als auch mit Stride und Greenfoot erfolgreich durchgeführt. Weiterhin legt dieser Ansatz einen Fokus auf die *Selbstständigkeit* der Schülerinnen und Schüler in der Fehlersuche und -behebung, um das Problem der „Turnschudidaktik“ zu adressieren.

Die vorgestellte Intervention stellt einen ersten Baustein für die Förderung von Debuggingfähigkeiten dar. Dieser sollte um die Vermittlung von konkreten Debuggingstrategien, Werkzeugen und Heuristiken erweitert werden.

Eine mögliche Einschränkung der Validität dieser Studie stellt die geringe Stichprobengröße und die nicht vorhandene Randomisierung der Schülerinnen und Schüler dar. Sie wurden von derselben Lehrkraft nach demselben Konzept unterrichtet und stammen von derselben Schule. Dies könnte die Aussagekraft bezüglich der Verallgemeinerung der Ergebnisse auf die Grundgesamtheit einschränken. Wir planen daher, diese Studie auf eine größere Stichprobe auszuweiten.

Zusammenfassend zeigt unsere Untersuchung, dass eine solche Intervention einen vielversprechenden Ansatz für die Vermittlung von Debuggingfähigkeiten darstellt. Die Vermittlung eines systematischen Vorgehens zum Finden und Beheben von Programmierfehlern hat einen positiven Einfluss auf die Debugging-Selbstwirksamkeitserwartungen. Schülerinnen und Schüler, die ein systematischen Vorgehen vermittelt bekommen haben, zeigen zudem höhere Leistungen im Debuggen als Schülerinnen und Schüler, die Debuggen ausschließlich geübt haben: Ein systematisches Vorgehen der Schülerinnen und Schüler macht den Unterschied.

Literaturverzeichnis

[AB90] Allwood, Carl Martin; Björhag, Carl-Gustav: Novices' debugging when programming in Pascal. International Journal of Man-Machine Studies, 33(6):707–724, 1990.

[AB91] Allwood, Carl Martin; Björhag, Carl-Gustav: Training of Pascal novices' error handling ability. Acta Psychologica, 78(1-3):137–150, 1991.

[AEH05] Ahmadzadeh, Marzieh; Elliman, Dave; Higgins, Colin: An analysis of patterns of debugging among novice Computer Science students. Proceedings of the 10th annual SIGCSE conference on Innovation and Technology in Computer Science Education (ITiCSE '05), 37(3):84–88, 2005.

[Bo16] Bottcher, Axel; Thurner, Veronika; Schlierkamp, Kathrin; Zehetmeier, Daniela: Debugging students' debugging process. Proceedings - Frontiers in Education Conference, FIE, 2016-Novem, 2016.

[CL04] Chmiel, Ryan; Loui, Michael C: Debugging: from Novice to Expert. Proceedings of the 35th SIGCSE technical symposium on Computer science education - SIGCSE '04, 36(1):17, 2004.

[Co88] Cohen, Jacob: Statistical power analysis for the behavioural sciences. Hillsdale, NJ: erlbaum, 1988.

[CR87] Carver, McCoy Sharon; Risinger, Sally Clarke: Improving children's debugging skills. In: Empirical studies of programmers: Second workshop. Ablex Publishing Corp., S. 147–171, 1987.

[Gi91] Gilmore, David J: Models of debugging. Acta psychologica, 78(1-3):151–172, 1991.

[GO86] Gugerty, Leo; Olson, G.: Debugging by skilled and novice programmers. ACM SIGCHI Bulletin, 17(4):171–174, 1986.

[KA86] Kessler, Claudius M; Anderson, John R: A model of novice debugging in LISP. In: Proceedings of the First Workshop on Empirical Studies of Programmers. S. 198–212, 1986.

[KJ87] Katz, Irvin R.; John R. Anderson: Debugging: An Analysis of Bug-Location Strategies. Human-Computer Interaction, 3(4):351–399, 1987.

[MR19] Michaeli, Tilman; Romeike, Ralf: Current Status and Perspectives of Debugging in the K12 Classroom: A Qualitative Study. In (IEEE, Hrsg.): Global Engineering Education Conference (EDUCON). 2019.

[Mu08a] Murphy, Laurie; Lewandowski, Gary; McCauley, Renée; Simon, Beth; Thomas, Lynda; Zander, Carol: Debugging: the good, the bad, and the quirky – a qualitative analysis of novices' strategies. Proceedings of the 39th SIGCSE technical symposium on Computer Science Education (SIGCSE '08), 40:163, 2008.

[Mu08b] Murphy, Laurie; Simon, Beth; Fitzgerald, Sue; Lewandowski, Gary; Thomas, Lynda; Zander, Carol: Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers ´. Computer Science Education, 18(2):93–116, 2008.

[Pe17] Perscheid, Michael; Siegmund, Benjamin; Taeumel, Marcel; Hirschfeld, Robert: Studying the advancement in debugging practice of professional software developers. Software Quality Journal, 25(1):83–110, 2017.

[Ra10] Rasch, Björn; Friese, Malte; Hofmann, Wilhelm; Naumann, Ewald: , Quantitative Methoden 2: Einführung in die Statistik für Psychologen und Sozialwissenschaftler (3., erweiterte Auflage), 2010.

[Si08] Simon, Beth; Bouvier, Dennis; Chen, Tzu-yi; Lewandowski, Gary; Mccartney, Robert; Sanders, Kate: Common sense computing (episode 4): debugging. Computer Science Education, 18(2):117–133, 2008.

[Sp18] Spinellis, Diomidis: Modern debugging: the art of finding a needle in a haystack. Communications of the ACM, 61(11):124–134, 2018.

[Ya11] Yadav, Aman; Zhou, Ninger; Mayfield, Chris; Hambrusch, Susanne; Korb, John T: Introducing Computational Thinking in Education Courses. In: Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education. SIGCSE '11, ACM, New York, NY, USA, S. 465–470, 2011.

[Ze05] Zeller, Andreas: Why Programs Fail: A Guide to Systematic Debugging. Elsevier, 2005.