Accepted Manuscript, to appear in:
INFOS 2025 – 21. GI-Fachtagung Informatik und Schule

Eine theoriegeleitete Analyse von Debugging-Prozessen in Scratch

Elena Spörer [1] und Tilman Michaeli [1]

Abstract: Debugging stellt eine große Herausforderung für Programmieranfänger:innen dar und endet nicht selten in Resignation und Frustration. In diesem Beitrag wird ein Vorgehen entwickelt, dass ermöglicht die Debugging-Prozesse von Lernenden zu untersuchen. Im Unterschied zu anderen Untersuchungen, die zumeist induktiv das Verhalten beschreiben, wird hier das Verhalten auf einen idealtypischen Prozess gemappt, um Probleme und Herausforderungen zu erkennen und Unterstützungsbedarf zu identifizieren. Dazu wurden Screenrecordings qualitativ analysiert, die die Bearbeitung von Debugging-Aufgaben in Scratch von Schüler:innen der Sekundarstufe zeigen. Aus den Ergebnissen lassen sich Indizien für einen nicht erfolgreichen Debugging-Prozess ableiten.

Keywords: Debugging, Debugging-Prozess, Screenrecordings, Prozessanalyse

1 Motivation

Debugging stellt für Schüler:innen eine zentrale Herausforderung beim Programmieren lernen dar und wird oftmals als frustrierend wahrgenommen [10]. Dabei unterscheiden sich Programmier- und Debuggingfähigkeiten, sodass auch Debuggingfähigkeiten explizit erlernt werden müssen [2]. Ein zielführender Debugging-Prozess ist dabei für das erfolgreiche Finden und Beheben von Fehlern entscheidend [4] und stellt häufig das Ziel von unterrichtlichen Interventionen dar [7].

Vor diesem Hintergrund bietet die Untersuchung von Debugging-Prozessen von großes Potenzial, da analysiert werden kann, wie die Debugging-Prozesse von Schüler:innen aussehen. Dies ermöglicht typische Probleme beim Debugging zu identifizieren und Ansatzpunkte für gezielten Unterstützungsbedarf abzuleiten. Bisher wurden in informatikdidaktischer Forschung dafür vor allem induktiv auffällige Muster im Verhalten der Lernenden identifiziert. Im Gegensatz dazu wird in diesem Beitrag ein Vorgehen vorgestellt, um Debugging-Prozesse theoriegeleitet zu analysieren. Dazu wird das Verhalten der Lernenden auf die Schritte eines idealtypischen Debugging-Prozesses gemappt und mit diesem verglichen, um anschließend Unterschiede identifizieren zu können. Zudem werden exemplarische Ergebnisse beschrieben, die mit diesem Vorgehen analysiert wurden.

[1] TU München, School of Social Sciences and Technology, Professur für Didaktik der Informatik, Arcisstraße 21, 80333 München, elena.spoerer@tum.de, https://orcid.org/0009-0004-5100-368X; tilman.michaeli@tum.de, https://orcid.org/0000-0002-5453-8581

2 Forschungsstand

Debugging, also das systematische Finden und Beheben von Fehlern, ist eine praktische Notwendigkeit der Softwareentwicklung. Der idealtypische Ablauf eines Debugging-Prozesses wurde in verschiedenen empirisch wie theoretisch entwickelten Modellen beschrieben (z.B. [4, 12]). In all diesen Modellen lassen sich zusammenfassend die folgenden vier zentralen Schritte identifizieren: (1) Beobachtung des Fehlverhaltens, (2) Formulierung von Hypothesen, (3) Identifikation des Fehlers und (4) Verifikation der Hypothese. Dabei werden die Schritte (2) bis (4) in einem idealtypischen Debugging-Prozesses solange iterativ durchlaufen, bis die zugrunde liegende Ursache für das Fehlverhalten identifiziert und der Fehler vollständig behoben ist. Das Modell von Araki u. a. (1991) gliedert diese Schritte feingranularer auf. So werden Finden und Beheben von Fehlern in zwei getrennten Phasen dargestellt und entsprechend die Schritte (2) bis (4) in weitere Teilschritte aufgeteilt (siehe Abb. 1). Wenn statt eigenem fremder Code debuggt werden soll, muss das Vorgehen um zusätzliche Schritte des Codeverstehens erweitert werden [3].

Zur Untersuchung von Debugging-Prozessen werden häufig Screenrecordingaufnahmen sowohl mit [11] als auch ohne [8] Think-aloud-Protokolle herangezogen. Darüber hinaus wird das Verhalten der Lernenden beim Debugging mit Hilfe von Programmsnapshots untersucht, also mehreren Momentaufnahmen eines Programms zu unterschiedlichen Zeitpunkten, die ausgewertet und verglichen werden [6]. Vereinzelt werden auch Eyetracking-Daten [5] oder händische Beobachtungs-Protokolle [2] erhoben. Zur Auswertung dieser Prozessdaten wurde bisher vor allem *induktiv* vorgegangen, das heißt in den Daten wurden ohne zugrunde liegendes theoretisches Framework auffällige Verhaltensweisen identifiziert, z.B. im Hinblick auf den Erfolg oder Misserfolg des Debugging-Prozesses. Auf diese Weise wurden etwa erfolgreiches Verhalten wie systematisches Program-Tracing [2, 5, 11], Pattern Matching [2] oder die Verwendung von externen Ressourcen [2, 11] gefunden, als problembehaftete Verhaltensmuster etwa unsystematisches Tinkering [2, 11], Neuschreiben von Code, der nicht verstanden wurde [2, 11] oder unstrukturiertes im Code „herumspringen“ [5, 11]. Eine *deduktive*, also theoriegeleitete Analyse des gesamten Debugging-Prozesses unter Verwendung von idealtypischen Prozess-Modellen, wie etwa in vielen anderen Bereichen der Bildungsforschung üblich, wurde bislang nicht durchgeführt. Eine solche Auswertung ermöglicht es, Unterschiede im Vorgehen von Lernenden gegenüber dem idealtypischen

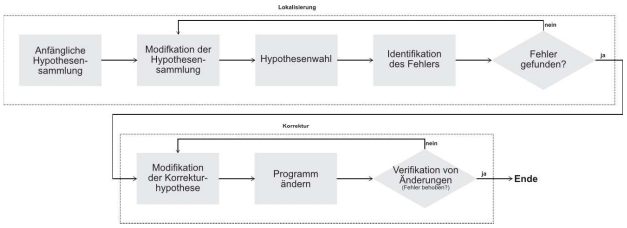


Abb. 1: Modell des Debugging-Prozesses in Anlehnung an Araki u. a. (1991)

Prozess systematisch zu erfassen und damit Lücken aus induktivem Vorgehen zu füllen [9], um daraus mögliche Ursachen für Schwierigkeiten beim Debugging abzuleiten. Diese können anschließend als Ausgangspunkt für gezielte Interventionen dienen und dazu beitragen die Schwierigkeiten von Lernenden gezielter zu adressieren.

3 Ein theoriegeleitetes Vorgehen zur Analyse von Debugging-Prozessen

In diesem Beitrag wird ein Vorgehen für eine theoriegeleitete Analyse von Debugging-Prozessen auf Grundlage eines idealtypischen Debugging-Prozess-Modells entwickelt.

Neben der Beschreibung der Auswertungsmethodik werden exemplarische Ergebnisse aus einer Pilotierung dieses Vorgehens vorgestellt.

Die Daten zur Pilotierung des Vorgehens wurden in einer Workshop-Woche für begabte und motivierte Schüler:innen erhoben. Dabei bearbeiteten die fünf Teilnehmenden (2x weiblich, 3x männlich – Jahrgangsstufen 9 bis 11), drei Debugging-Challenges in Scratch. Hierfür wurden modifizierte Versionen der Spiele Pong und Snake verwendet, in die, analog zu Whalley u. a. (2023), gezielt fünf bis sechs Fehler integriert wurden. Die Bearbeitung der Debugging-Challenges erfolgte, wie von Fitzgerald u. a. (2008) angeregt, in Paaren, ein Teilnehmer arbeitete dabei alleine. Neben den Bildschirmen der Schüler:innen, wurden die Gespräche während der Bearbeitung aufgezeichnet. Aufgrund technischer Probleme konnte bei etwa einem Drittel der Videos die Tonspur nicht ausgewertet werden. Das schriftliche Einverständnis aller Teilnehmenden und Erziehungsberechtigten wurde eingeholt und liegt vor.

Um die Debugging-Prozesse der Schüler:innen in den Screenrecordings qualitativ zu erfassen, wurden in diese in fünf aufeinander aufbauenden Schritten ausgewertet:

Schritt 1: Zuerst wurden alle Actions der Lernenden kodiert, dafür wurden alle in den Screenrecordings sichtbaren Handlungen (Block-Bewegungen; Mausbewegungen; Sprites, in denen gearbeitet wurde) herangezogen. Eine *Action* beschreibt dabei eine Interaktion auf der Programmieroberfläche, die aus einem oder mehreren Einzelschritten besteht. Beispiele dafür sind das Ausführen des Programms, das Hinzufügen von Blöcken oder das Ändern von Variablenwerten. Die Visualisierung der Actions (siehe Abb. 2(a)) diente als Basis für die folgenden Auswertungs- und Interpretationsschritte.

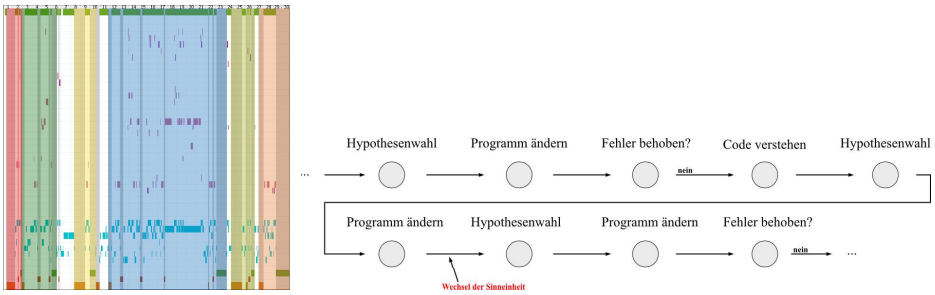
Schritt 2: Im Anschluss wurden alle Actions, die denselben Fehler adressieren zu einer Sinneinheit zusammengefasst. In Abbildung 2(a) steht jedes farbige Kästchen für eine Sinneinheit. Zudem wurde in diesem Schritt festgehalten, ob der jeweilige Fehler erfolgreich behoben werden konnte oder nicht.

Schritt 3: Darauf aufbauend wurden alle Actions pro Sinneinheit auf die Prozess-Schritte des Modells von Araki u. a. (1991) gemappt, welches sich aufgrund seiner Feingranularität besonders gut eignet. Dabei kann ein Prozess-Schritt durch eine oder mehrere Actions

repräsentiert werden. Weil die Lernenden mit fremden Code arbeiteten, wurden Actions auch auf zusätzliche Schritte des *Code Verstehens* gemappt. Ein Beispiel für das Mapping ist die Interpretation der Action *Programm ausführen*: Wenn im Programm keine Änderung seit dem letzten Klick auf die grüne Flagge erfolgt ist, dann wurde das Segment als *Identifikation des Fehlers* interpretiert, wenn vorher Änderungen vorgenommen wurden als *Verifikation von Änderungen*. Ein weiteres Beispiel ist das Mapping von Mausbewegungen, die den Code stückweise abfahren (Program-Tracing) und das Inspizieren von verschiedenen Drop-Down Werten und Kommentaren auf den Prozess-Schritt *Code verstehen*.

Schritt 4: Anschließend wurden die Audioaufnahmen für die Validierung der Interpretationen aus den Schritten 2 und 3 herangezogen. Bis auf wenige Ausnahmen, v.a. Phasen in denen keine Mausbewegungen stattgefunden haben, war keine Anpassung nötig.

Schritt 5: Abschließend wurden die identifizierten Debugging-Prozesse visualisiert (siehe Abb. 2(b)) und mit dem idealtypischen Debugging-Prozess verglichen. Daraus wurden Unterschiede im Vorgehen abgeleitet und interpretiert.



(a) Actions inkl. Sinneinheiten (b) Ausschnitt eines Debugging-Prozesses von Schüler:innen

Abb. 2: Visualisierungen, die zur Analyse der Debugging-Prozesse erstellt und herangezogen wurden.

4 Exemplarische Ergebnisse

Im Folgenden werden exemplarische Ergebnisse dargestellt, um zu illustrieren wie mit Hilfe dieses Vorgehens Unterschiede in den Prozessen der Schüler:innen im Vergleich zu einem idealtypischen Vorgehen ermittelt werden können. Im Allgemeinen konnten in den ausgewerteten Screenrecordings die wesentlichen Schritte eines idealtypischen Debugging-Prozesses wiedergefunden werden. Gleichzeitig konnten auch einige Abweichungen vom idealtypischen Prozess festgestellt werden.

Als zentralen Unterschied zu einem idealtypischen Debugging-Prozess fehlten in den Debugging-Prozessen der Schüler:innen wesentliche **Prozess-Schritte**. Zum einen ließen die Schüler:innen häufig die explizite *Identifikation des Fehlers* aus. Bei diesem Verhalten sind die Schüler:innen unmittelbar von der Identifikation des Fehlverhaltens dazu übergegangen den Fehler zu beheben. Jedoch hatten sie diesen häufig nicht korrekt lokalisiert.

Damit einhergehend konnten keine Debugging-Strategien wie zum Beispiel das Ausgeben von Variablenwerten identifiziert werden. In der Folge konnte keine zielführende Korrekturhypothese aufgestellt werden, so dass die Lernenden in unseren Daten den Fehler häufig nicht korrekt beheben konnten. Zum anderen wurden *Änderungen* häufig nicht *verifiziert*. Bezogen auf den Prozess sind die Lernenden direkt vom Ändern des Programms entweder zur Modifikation der Korrekturhypothese oder zur Adressierung des nächsten Fehlers übergegangen, ohne die Änderungen zu überprüfen. Das zeigte sich dann, wenn die Schüler:innen Änderungen im Code vornahmen, mit dem Ziel einen Fehler zu beheben und sich dann, ohne das Programm auszuführen, direkt dem nächsten Fehler widmeten. Aber auch, wenn die Schüler:innen eine Annäherung an die Korrektur eines Fehlers nicht überprüften und sehr viele Änderungen vornahmen, bevor sie diese zum ersten Mal verifizierten. Dies führte häufig dazu, dass neue Fehler im Programm enthalten waren, die zusätzlich noch identifiziert und behoben werden mussten. Zudem konnte ein Fehlverhalten nicht eindeutig einer Änderung zugeordnet werden, wenn diese zuvor nicht verifiziert wurde. Als Konsequenz zeigt sich in unseren Daten, dass das Auslassen der Identifikation des Fehlers und der Verifikation von Änderungen häufig einen nicht erfolgreichen Debugging-Prozess zur Folge hatte und entsprechend ein Indikator für einen erfolglosen Debugging-Prozess ist.

Neben fehlenden Prozess-Schritten zeigte sich in den Daten der Schüler:innen außerdem, dass sie häufig von der idealtypischen Reihenfolge abweichen. Zudem wurden, bedingt durch die Arbeit mit fremdem Code, **zusätzliche Verständnis-Schritte** identifiziert. Darüber hinaus konnten in der Phase der Fehlerkorrektur weitere Abweichungen vom idealtypischen Vorgehen in Form von **Work arounds** anstelle gezielter Fehlerbehebungen sowie das fehlende Rückgängigmachen (**Undo**) nicht erfolgreicher Änderungen festgestellt werden.

5 Diskussion & Ausblick

In diesem Beitrag wurde ein Vorgehen zur theoriegeleiteten Analyse von Debugging-Prozessen beschrieben. Im Vergleich zu bisheriger Forschung wurden die Schritte aus dem Vorgehen der Schüler:innen auf Debugging-Prozess-Schritte aus einem idealtypischen Prozess-Modell gemappt. Dieses Vorgehen bietet Potenzial um grundlegende Ähnlichkeiten zu einem idealtypischen Debugging-Prozess festzustellen, zudem zeigten sich auch merkliche Abweichungen, die häufig mit einem Misserfolg in der Fehlerbehebung einhergehen. Zum Beispiel konnte festgestellt werden, dass im Vergleich zu einem idealtypischen Debugging-Prozess viele Schritte, insbesondere in der Lokalisierungsphase, fehlen. Die Schüler:innen springen direkt von der Identifikation des Fehlverhaltens zur Änderung des Programms ohne, dass sie zuvor verifiziert haben, ob sie den Fehler wirklich gefunden haben. Außerdem ist auffällig, dass Änderungen häufig nicht verifiziert wurden. Die Schüler:innen scheinen sich sicher zu sein, dass sie den Fehler korrekt behoben haben und sehen keine Notwendigkeit darin dies zu überprüfen.

Zusammenfassend beschreibt das vorgestellte Vorgehen eine theoriegeleitete Analyse des gesamten Debugging-Prozesses. Durch den Vergleich mit einem idealtypischen Debugging-

Prozess, werden zudem Lücken aus induktiven Vorgehensweisen gefüllt. Dadurch lassen sich mögliche Ursachen für Probleme von Schüler:innen beim Debugging gezielt identifizieren und darauf aufbauend passgenaue Unterstützungsmöglichkeiten ableiten. Darüber hinaus bietet dieser Ansatz Potenzial zur Evaluation von Interventionen auf der Prozessebene und schafft eine fundierte Grundlage für weiterführende Untersuchungen des Debugging-Prozesses, insbesondere durch den Einsatz automatisierter Analysen mittels Log-Daten.

Literatur

[1] K. Araki u. a. „A general framework for debugging“. In: *IEEE Software* (1991).

[2] S. Fitzgerald u. a. „Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers“. In: *Computer Science Education* (2008).

[3] I. Katz und J. Anderson. „Debugging: an analysis of bug-location strategies“. In: *Human-Computer Interaction* (1987).

[4] C. Li u. a. „Towards a Framework for Teaching Debugging“. In: *Proceedings of the Twenty-First Australasian Computing Education Conference*. 2019.

[5] Y. Lin u. a. „Tracking Students' Cognitive Processes During Program Debugging—An Eye-Movement Approach“. In: *IEEE Transactions on Education* (2016).

[6] R. Mansur u. a. „Exploring the Bug Investigation Techniques of Intermediate Student Programmers“. In: *Proceedings of the 20th Koli Calling International Conference on Computing Education Research*. 2020.

[7] T. Michaeli und R. Romeike. „Debuggen im Unterricht – Ein systematisches Vorgehen macht den Unterschied“. In: *INFOS 2019 - 18. GI-Fachtagung Informatik und Schule - Informatik für alle*. 2019.

[8] M. Parkinson u. a. „Exploring debugging processes and regulation strategies during collaborative coding tasks among elementary and secondary students“. In: *Computer Science Education* (2024).

[9] N. Pearse. „An Illustration of Deductive Analysis in Qualitative Research“. In: *Proceedings of the 18th European Conference on Research Methodology for Business and Management Studies*. 2019.

[10] D. N. Perkins u. a. „Conditions of Learning in Novice Programmers“. In: *Journal of Educational Computing Research* (1986).

[11] J. Whalley u. a. „A Think-Aloud Study of Novice Debugging“. In: *ACM Transactions on Computing Education* (2023).

[12] A. Zeller. *Why programs fail. A guide to systematic debugging*. 2009.