

Hochschullehre der Programmierung außerhalb eines Informatikstudiengangs: Herausforderungen und Perspektiven

Ralf Dörner¹, Ulrike Spierling²

Abstract: Increasing digitalization also raises the need for teaching programming at universities not only in courses related to computer science but other subjects as well. These new target groups for education in computer programming require a change of the classic didactic that is usually applied in universities today. This article presents a literature survey in the area of programming education. Moreover, it addresses challenges encountered when teaching programming in an interdisciplinary course program with media as major subject of study. These challenges comprise the fear of contact, the required ability of abstract thinking and the ability to tolerate frustrations. Based on this, solution approaches adopted and perspectives for future development are discussed.

Keywords: Programmierlehre, Programmieren Lernen, Digitalisierung, Interaktive Medien.

1 Einleitung

Digitalisierung transformiert alle Lebensbereiche – von der Berufswelt bis zum Privatbereich, von der schulischen Ausbildung in der Kindheit bis zum assistierten Leben im Alter. Von daher ist es nicht verwunderlich, dass die Fertigkeit, Computersysteme zu programmieren bzw. deren Verhalten auf formale Art so zu spezifizieren, dass dieses Verhalten maschinell umgesetzt wird, nicht mehr einer kleinen Gruppe von Spezialisten vorbehalten sein kann, sondern eine grundlegende Qualifikation darstellt. Während in Deutschland nicht nur über Informatik als Schulfach diskutiert wird, sondern auch über die Frage, ob eine Ausbildung im Programmieren überhaupt zu der in diesem Fach zu vermittelnden digitalen Kompetenz zählt, haben andere Länder wie z.B. Estland einen Programmierunterricht schon in der ersten Klasse der Grundschule eingeführt. In Deutschland gibt es auch außerschulische Programmierlehre für Kinder und Jugendliche, die z.T. von IT-Unternehmen angeboten werden [Sei18]. Eine ähnliche Heterogenität zeigt sich in der Hochschullehre für Nichtinformatiker etwa in Modulbeschreibungen aus Fächern wie z.B. Maschinenbau. Dabei stammen die Lehrenden teilweise aus einem Informatikfachbereich (die Lehrveranstaltung für Studierende der Informatik und Externe kann sogar identisch sein), in anderen Fällen

¹ Hochschule RheinMain, Fachbereich Design Informatik Medien, Unter den Eichen 5, 65195 Wiesbaden, ralf.doerner@hs-rm.de

² Hochschule RheinMain, Fachbereich Design Informatik Medien, Unter den Eichen 5, 65195 Wiesbaden, ulrike.spierling@hs-rm.de

wird eigens Lehrkräfte für Informatikaspekte berufen – oder das Fach wird prinzipiell nicht über wissenschaftliches Stammpersonal abgedeckt, sondern durch Lehrbeauftragte.

In diesem Artikel betrachten wir die Lehre von Programmierung an Hochschulen und dort speziell in Studiengängen, die nicht zum Bereich Informatik zählen. Die zu beobachtende Heterogenität ist ein Indiz, dass es hier keinen etablierten Goldstandard gibt, sondern noch darauf hingearbeitet wird. Basierend auf unseren Erfahrungen in der Programmierausbildung an einem interdisziplinär aufgestellten Fachbereich „Design, Informatik, Medien“ möchten wir einen Beitrag leisten, die Herausforderungen für diese Lehre zu identifizieren. Mit der Heterogenität der Vorkenntnisse der Studierenden haben wir schon einen Aspekt angesprochen, aber es gibt noch weitere neben der grundsätzlichen Herausforderung, dass Programmierenlernen per se nicht einfach ist – ein Jahrzehnt ist die geschätzte Zeitdauer von Programmierneuling zu Programmierexperte [Win96] und der bekannte Informatiker Donald E. Knuth sieht Programmieren sogar als Kunstform: „*We have seen that computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty.*“ [Knu74]. Andererseits möchten wir auch in diesem Workshop-Beitrag Ansätze und Ideen diskutieren, die Perspektiven für die Programmierlehre für Nichtinformatiker aufzeigen. Entsprechend ist dieser Artikel gegliedert in zwei Abschnitte, die sich mit Herausforderungen und Perspektiven beschäftigen. Davor geben wir einen kurzen Überblick verwandter Literatur.

2 Programmierlehre in der Literatur

Die Lehre von Programmierung ist Gegenstand einer großen Zahl wissenschaftlicher Veröffentlichungen [RRR03]. Dabei liegt ein Fokus auf der Zielgruppe von Kindern und Jugendlichen sowie von Informatikstudierenden in den ersten Fachsemestern. Veröffentlichungen, die Studierende außerhalb von MINT-Fächern in der Hochschullehre adressieren, sind selten.

Ein häufig gewählter Ansatz in der Literatur ist die Verwendung von Visual Programming in der Programmierausbildung, bei der Programmcode nicht durch Schreiben eines Textes in einer formalen Sprache erzeugt wird, sondern durch Anordnung grafischer Elemente. Grafische Repräsentationen wie Nassi-Shneiderman-Diagramme oder UML-Diagramme haben eine lange Tradition in der Informatik. Ein anderer visueller Aspekt ist die Programmierung eines graphischen Systems (z.B. eines Computerspiels) oder von Tangibles (z.B. Robotern). Hier wird neben einem motivierenden spielerischen Effekt auch das direkte visuelle Feedback angestrebt, so dass die Lernenden unmittelbar das von ihnen programmierte Verhalten wahrnehmen und hinsichtlich Zielerreichung überprüfen können. Eine Kombination von beiden bietet Alice [CPGB94]. Ursprünglich gedacht als Rapid Prototyping System für Virtual Reality wurde es sowohl für die Programmierausbildung von Kindern als auch von Studierenden

eingesetzt. Eine Meta-Studie [CM17], die 232 Veröffentlichungen zum Einsatz von Alice auswertet, weist positive Effekte hinsichtlich Effektivität verglichen mit der Nutzung konventioneller Programmiersprachen in der Lehre nach. Ein Vorteil von Visual Programming ist dabei, dass die Erstellung syntaktisch fehlerhafter Programme verhindert werden kann und damit eine typische Quelle von Fehlern und Frustration vermieden wird.

Im Gefolge von Alice gibt es zahlreiche Programmiersprachen und Programmierumgebungen, die speziell für die Programmierausbildung gedacht sind. Scratch [Scr18] zielt auf Kinder und erlaubt die Programmierung multimedialer Anwendungen, die in einem sozialen Netzwerk verbreitet und mit anderen Lernenden diskutiert werden können. Scratch kann dabei nicht nur die Lernmotivation erhöhen [EK17], sondern neben Programmieren auch grundsätzliche Kompetenzen in Problemlösungsstrategien fördern [KG14]. EOS [EOS18] fokussiert auf die Vermittlung objektorientierter Konzepte an Jugendliche, wobei Features aus Programmierwerkzeugen wie Syntax Highlighting oder Auto Completion unterstützt werden. Blockly [Blo18a] dagegen ist eine Bibliothek für den Bau visueller Code-Editoren, die Programmcode z.B. in Javascript erzeugen. Darauf aufbauend gibt es mit code.org [Cod18] eine Umgebung zum spielerischen Erlernen von Programmierung mit Unterrichtsmaterial zum Selbstlernen und für Schulen. Die Initiative arbeitet auch an einer Verbesserung des Images von Programmieren, indem prominente wie Bill Gates, Mark Zuckerberg oder Barack Obama sie bewerben. Während code.org von Google, Microsoft, Facebook und Twitter unterstützt wird, bietet Apple mit Playgrounds [App18] ein ähnliches Angebot, um einen möglichst breiten Personenkreis für Programmierung zu interessieren. Playgrounds dient dem Erlernen der Programmiersprache Swift und zeichnet sich durch ein ansprechendes Design aus, das auch Interaktionsmöglichkeiten durch Touchscreens ausnutzt. Eine besondere Motivation liefert das Projekt freeCodeCamp [Fre18], das nicht nur eine interaktive Lernplattform und soziale Medien für das Erlernen von Web Engineering bereit stellt, sondern auch gemeinnützige Organisationen an die Lernenden vermitteln, so dass das Gelernte direkt angewendet werden kann, indem ehrenamtlich Webseiten für diese Organisationen erstellt werden.

Techniken des Software Engineerings wie z.B. Pair Programming können auch als Ausgangspunkt für neue Ansätze der Programmierausbildung dienen [MWB02]. Daneben gibt es Ansätze, die das Erreichen allgemeiner didaktischer Zielsetzungen wie z.B. ein selbstbestimmtes Lerntempo für die Lehre der Programmierung anstreben [OAD17]. Allgemeinere Ansätze denken darüber nach, wie Methoden aussehen, die verschiedene Abstraktionsebenen im Kontext von Programmierung und insbesondere den Zusammenhang zwischen Code und dem daraus hervorgehenden Verhalten visualisieren [CAP17]. Grundsätzliche Überlegungen gehen der Frage nach, wie der Spalt zwischen Technikdidaktik und Bildung in Informatikthemen (wie z.B. Modellbildung) einerseits und der Fachdidaktik von Nicht-Informatikfächern (wie z.B. Medienpädagogik und Mediendidaktik) überbrückt werden kann [Rei17]. Derartige, mehr grundsätzliche Überlegungen sind in der Literatur selten. Sie sollten auf einer

sorgfältigen Erhebung von Herausforderungen bei der Programmierlehre gerade für Nicht-Informatiker im Hochschulbereich basieren.

3 Herausforderungen im Bereich Medien

Bei Studierenden außerhalb Informatikfächern sind sehr unterschiedliche Motivationslagen bezüglich des Erlernens von Programmierung anzutreffen. In den seltensten Fällen wird Programmieren um seiner selbst willen erlernt, oder auf Vorrat, damit man das Können später einmal nutzen kann. Erfahrungen, die z.B. im Bereich von Medienstudiengängen oder im digitalen Mediendesign gemacht wurden, deuten darauf hin, dass die Zielsetzung beim Arbeiten häufig in der erfolgreichen Erschaffung eines digitalen Werks liegt, das bestimmte interaktive Verhaltensqualitäten haben soll. Um dies zu erreichen, ist Programmierung nötig – und wird zumindest zu Beginn zuweilen als notwendiges Übel empfunden. Zudem ist hierfür nicht nur das Programmieren im Sinne einer technischen Implementierung eine Herausforderung, sondern es werden bereits bei der Konzeption prozedurale Denkweisen nötig, die als unbequem und kompliziert erlebt werden. Deutlich wird letzteres insbesondere in Gebieten, in denen durch graphische Autorenwerkzeuge das eigentliche Coding erleichtert wird, in dem Codezeilen – wenn überhaupt – erst nach Bedienen oder Durchdringen einer graphischen Oberfläche sichtbar gemacht werden.

Im Studiengang Media Management der Hochschule RheinMain besteht ca. die Hälfte der Fächer aus Wirtschaftsfächern und die andere Hälfte aus technisch-gestalterischen Fächern und Projekten des "Medien Machens". Im Bereich der letzteren wählen Studierende im 3. Semester ein Schwerpunkt-Interessensgebiet zwischen Bewegtbildmedien/Film oder Interaktiven Medien. In Informationsveranstaltungen und Diskussionen vor der Wahl war wiederholt feststellbar, dass obwohl eine große Gruppe Begeisterung für das Design von "Apps" äußerte, eine weit geringere Anzahl an Studierenden aber schlussendlich den Schwerpunkt der Interaktiven Medien wählte – hier wurden Programmierkurse für Einsteiger angekündigt.

Im Kontext von Konzept-Diskussionen über "Gamification" – als einem typischen motivationsfördernden Instrument, das speziell durch interaktive Medien ermöglicht wird – sollten Studierende im Selbsttest vier Kapitel von Blockly Games [Blo18b] durchspielen, und über ihre Erfahrungen berichten. Die Aufgabe war an dieser Stelle jedoch nicht, auf den entstehenden Code zu achten bzw. auf diese Weise JavaScript zu lernen, sondern es sollten die abgestimmten Fortschritts-Bedingungen und das Gamification-Konzept bewertet werden. Letzteres erlaubt einen schnellen und sehr leichten Einstieg, wobei die konkreten einzelnen Aufgaben aber eine zunehmende Abstraktion im Denken erfordern. Beispielsweise muss eine Figur mit zu spezifizierenden Regeln durch ein Labyrinth geführt werden, oder eine klassische Turtle-Grafik [Pap80] durch Längen- und Winkeleinheiten geskriptet werden. Überraschenderweise (für die Lehrenden) wurden nur weniger als die Hälfte der Teilnehmenden von

dem zu Grunde liegenden Konzept angesprochen; noch weniger äußerten sich begeistert. Einige Studierende scheiterten gar an manchen Aufgaben, bzw. fanden die nötigen Überlegungen zu abstrakt und für Kinder entschieden zu schwer. In der Diskussion wurde deutlich, dass dies Studierende waren, die in ihrer Jugend nie mit ähnlichen Aufgaben in Kontakt kamen und die Denkweise als unnatürlich empfanden, während manche der Begeisterten in der Schule bereits mit Logo [Pap80] o.ä. zu tun hatten.

Hohe Motivation wird hingegen häufiger durch das gewünschte Erreichen einer eigenen gestalteten Umsetzung von interaktiven Medien freigesetzt. Hier wurden im genannten Studiengang sowohl Erfahrungen im Bereich aktueller Methoden des Responsive Webdesign als auch mit alternativen Interaktionsformen wie im Bereich "Interactive Storytelling" oder auch der Erstellung kleiner Games oder VR-Welten gemacht. Webdesign gehört dabei zum Grundkanon bei der breiten Vermittlung im Bereich interaktiver Medien, während die Bereiche Games und VR/AR eher in kleineren Projektgruppen aus geneigten Studierenden höherer Semester stattfindet.

Beide Gruppen von Studierenden haben regelmäßig und erwartungsgemäß Frustrations-Erlebnisse während verschiedener Stufen ihrer Umsetzungen. In den ersten Code-Übungen wird bereits die Erfahrung mit Klammer- und anderen Syntaxfehlern gemacht, deren Suche als erster Schritt des "Debugging" – insbesondere wenn es länger dauert – spaßreduzierend wirkt. Dies wird von den Lehrenden jedoch als eine sinnvoll zu erwerbende Erfahrung angesehen. Absolventen des Studiengangs, die später im Bereich der interaktiven Medien professionell arbeiten wollen, werden voraussichtlich in interdisziplinären Teams arbeiten, in denen sie vermutlich selbst nicht programmieren müssen, aber Verständnis für sämtliche Aufgaben benötigen, um managen zu können.

Debugging kann bei interaktiven Medien auch nötig werden, wenn nicht direkt mit Code gestaltet wird, sondern mit grafischen Autoren-Werkzeugen, sowie auf der konzeptionellen Ebene. Dies wird zum Beispiel deutlich, wenn ein Interaktionsfortschritt unmöglich wird auf Grund nicht erfüllbarer eingestellter Vorbedingungen, bei Endlos-Schleifen in falsch konzipierten Abläufen, oder auf Grund fehlender Medienbausteine, o.ä. Diese Art von Problemen wird insbesondere im Gebiet des interaktiven Geschichtenerzählens thematisiert, bei dem mehrere Erzähl- oder Aktionspfade abhängig von komplexeren Nutzerinteraktionen ermöglicht werden. Beispiele hierfür sind Spiele mit Dialogen mehrerer Chatbots, bei denen auch Muster von Freitext-Eingaben der Nutzenden abgeglichen werden [GJK+18]. In der Endnutzenerfahrung kann dies zur Wahrnehmung von "emergentem" Verhalten führen, wobei auch Autoren durch ihre eigenen Kreationen überrascht werden können, wenn z.B. Bot-Antworten an Stellen kommen, wo sie sie nicht vorgesehen haben. An dieser Stelle ist für die Konzeption prozedurales Denken unter Berücksichtigung von komplexen Vor- und Nachbedingungen erforderlich. Auch wenn man hierbei nicht direkt und textlich "coden" müsste, sind Erfahrungen mit dem Programmieren nützlich, um Konzeptionsanforderungen besser erfüllen zu können.

Zusammengefasst bestehen Herausforderungen in vorgelagerten Berührungspunkten, in eventueller mangelnder Technik- und Logikaffinität bzw. auch geäußerter Unlust oder vermuteter Schwerfälligkeit bei der Abstraktion von Abläufen, in der notwendigen Motivation durch Ziele, die außerhalb des Programmierens selbst liegen, und in der notwendigen Frustrationstoleranz bei langwieriger Fehlersuche.

4 Ansätze und Perspektiven

Bei den festgestellten Herausforderungen führte bereits eine Mischung aus Ansätzen, die zum Teil auch bei der Literaturstudie (Abschnitt 2) benannt wurden, zu Teilerfolgen. Wie sich in zahlreichen Medienkategorien gezeigt hat, sind so genannte grafische Authoring-Tools oder visuelle Programmierwerkzeuge für Nichtprogrammierer geeignet, das Problem der vorgelagerten Berührungspunkte vor Syntaxfehlern zu reduzieren.

Dies allein reicht jedoch nicht, um sich auch der Unbequemlichkeit einer abstrakten Konzeption zu widmen. Dagegen wurden (bei heruntergeschraubter Erwartungshaltung seitens der Lehrenden überraschende) Erfolge erzielt, wenn bei Studierenden eine intrinsische Motivation im Laufe eines Projekts entstand, ein gewisses kreatives Ziel erreichen zu wollen. Zunächst musste allerdings das Selbstvertrauen und die Vorstellungskraft für ein Ziel unterstützt werden. Dies kann mit Autorenwerkzeugen erreicht werden, wobei insbesondere ein direkt abspielbares Feedback motiviert und Sicherheit verleiht. Eine wiederholte Erfahrung war dann die, dass Autorenwerkzeuge bezüglich während der Arbeit neu entstehender Ideen in ihrer Funktionalität begrenzt sind, oder ihre Usability bei steigendem Inhaltsumfang schlecht skaliert. In oben genanntem Chatbotprojekt [GJK+18], in einem weiteren anspruchsvollen Projekt mit location-based Augmented Reality, sowie in Projekten des Responsive Webdesign, gaben involvierte Studierende bei anwachsendem Inhalt freiwillig die Nutzung grafischer Tools auf und widmeten sich direkt dem Code im Texteditor (meist XML-Dialekten oder Java Script), weil nach Einarbeitung die Kontrolle über den Umfang bei gleichzeitiger Fehlersuche leichter fiel. Dies betraf eine kleine Teilmenge an Studierenden, die sich für derartige Wahl-Projekte interessierte.

Eine weitere Erfahrung dabei war die Nützlichkeit von physischem Paper-Prototyping außerhalb des Computers für die Vermittlung von bestimmten non-linearen Denk-Konzepten. Hier wurde in der Lehre mehrere Male erfolgreich ein bereits 2010 selbst entwickeltes Kartenspiel [HSS11] eingesetzt, das beim Spiel anschaulich eine Konzeption mit Plan-Operatoren abhängig von Vorbedingungen und Effekten vermittelt. Während eine Gruppe Studierender gemeinsam die Funktion einer Planungssoftware, nämlich hier die Suche in den Karten nach nächst-möglichen Ereignisfortschritten simuliert, entsteht durch den Karteninhalt eine humorvolle Geschichte – jedes Mal mit sich anders entwickelndem Verlauf. Dabei werden erste typisch auftretende Verständnis-Irrtümer durch unvollständiges Analysieren von Vorbedingungen besprochen. Auch

dient das Spiel als Anregung, in einer interaktiven Konzeption selbst zuerst Papierprototypen anzufertigen und iterativ auf anfängliche inhaltliche Konsistenz zu testen, bevor dann darauf aufbauend eine Implementierung gestartet wird, die erfolgversprechender ist als ein direktes erstes Probieren im Code. Selbstredend sind Papierprototypen in ihrer möglichen Dynamik begrenzt, wecken aber auch wiederum die Aussicht, selbige durch eine Implementierung in Software dann zu erhöhen.

Ein weiterer zukünftiger Ansatz, verwandt mit dem planerischen Denken, könnte darin bestehen, andere Formen der Verhaltensbeschreibung speziell aus dem Bereich der Artificial Intelligence einzusetzen. Regelbasierte Systeme, die z.B. aus Sammlungen von ECA (Event Condition Action)-Regeln bestehen, sind ein alternativer Ansatz zum Spezifizieren von Systemverhalten. Der Vorteil ist, dass Regeln unabhängig und parallel zueinander ausgeführt werden, so dass keine Abläufe beim Programmieren zu bedenken sind. Zwei Regelmengen können auch einfach zu einer größeren Regelmenge gemischt werden. Daher kann man Verhalten spezifizieren in einer Reihenfolge wie es einem einfällt und man muss nicht das Systemverhalten als Ganzes betrachten. Game Engines wie Unity3D nutzen in Ergänzung zu Skripting in einer Programmiersprache bereits Artificial Intelligence Methoden wie Entscheidungsbäume oder Agenten [Uni18]. Allerdings ist es schwer, unterspezifiziertes Verhalten zu erkennen und mit eventuell unerwünschtem emergentem Verhalten umzugehen. Weitere Methoden der Artificial Intelligence wie Maschinelles Lernen oder Programming by Example können einen neuen Zugang zum Programmieren eröffnen. Dabei sind Methoden wie z.B. evolutionäre Algorithmen, die ein iteratives Vorgehen im Sinne einer Zusammenarbeit von Mensch und Maschine fördern, von besonderem Interesse.

Literaturverzeichnis

- [App18] Apple Inc.: Homepage von Swift Playgrounds. <https://www.apple.com/swift/playgrounds/> Letzter Zugriff: 27.06.2018.
- [Blo18a] Google Inc.: Blockly. A library for building visual programming editors. <https://developers.google.com/blockly/> Letzter Zugriff: 27.06.2018.
- [Blo18b] Google Inc.: Blockly Games. <https://blockly-games.appspot.com/> Letzter Zugriff: 27.06.2018.
- [CAP17] López, P. E. M.; Ciolek, D.; Arévalo, G.; Pari, D.: The GOBSTONES method for teaching computer programming. In *Computer Conference (CLEI), 2017 XLIII Latin American* (pp. 1-9). IEEE, 2017.
- [CM17] Costa, J. M.; Miranda, G. L.: Relation between Alice software and programming learning: A systematic review of the literature and meta-analysis. *British Journal of Educational Technology*, 48(6), 1464-1474, 2017.
- [Cod18] Code.org: Homepage von code.org. <http://code.org> Letzter Zugriff: 27.06.2018.

- [CPGB94] Conway, M.; Pausch, R.; Gossweiler, R.; Burnette, T.: Alice: a rapid prototyping system for building virtual environments. In *Conference Companion on Human Factors in Computing Systems* (pp. 295-296). ACM. Conway, M., Pausch, R., Gossweiler, R., & Burnette, T. (1994, April). Alice: a rapid prototyping system for building virtual environments. In *Conference Companion on Human Factors in Computing Systems* (pp. 295-296). ACM, 1994.
- [EK17] Erol, O.; Kurt, A. A.: The effects of teaching programming with scratch on pre-service information technology teachers' motivation and achievement. *Computers in Human Behavior*, 77, pp. 11-18, 2017.
- [EOS18] Pabst Software: Homepage von EOS. <https://www.pabst-software.de/doku.php?id=programme:eos:start> Letzter Zugriff: 27.06.2018.
- [Fre18] freeCodeCamp: Homepage von freeCodeCamp. <https://www.freecodecamp.org/> Letzter Zugriff: 27.06.2018.
- [GJK+18] Glock, F.; Junker, A.; Kraus, M.; Lehrian, C.; Schäfer, A.; Hoffmann, S.; Spierling, U.: "Office Brawl" – A Conversational Storytelling Game and its Creation Process. Demonstration at ACE/DIMEA 2011, Proceedings of the Eighth International Conference on Advances in Computer Entertainment Technology, ACM DL, 2011.
- [HSS11] Hoffmann, S.; Spierling, U.; Struck, G.: A Practical Approach to Introduce Story Designers to Planning. In: Proceedings of GET 2011, IADIS International Conference Game and Entertainment Technologies, 22-24 July 2011, Rome, Italy, 2011.
- [KG14] Kalelioglu, F.; Gülbahar, Y.: The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective. *Informatics in Education*, 13(1), 2014.
- [Knu74] Knuth, D. E.: Computer programming as an art. *Commun. ACM* 17, 12, 667-673, 1974.
- [MWB02] McDowell, C.; Werner, L.; Bullock, H.; Fernald, J.: The effects of pair-programming on performance in an introductory programming course. *ACM SIGCSE Bulletin*, 34(1), 38-42, 2002.
- [OAD17] Offutt, J.; Ammann, P.; Dobolyi, K.; Kauffmann, C.; Lester, J.; Praphamontriphong, U.; White, L.: A Novel Self-Paced Model for Teaching Programming. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale* (pp. 177-180). ACM, 2017.
- [Pap80] Papert, S.: *Mindstorms: Children, Computers and Powerful Ideas*. Basic Books, New York, 1980.
- [Rei17] Reimann, D.: Informatische Modellbildung als Dimension einer künstlerisch-technisch konzipierten Medienbildung. *MedienPädagogik: Zeitschrift für Theorie und Praxis der Medienbildung*, 25, 109-130, 2017.
- [RRR03] Robins, A.; Rountree, J.; Rountree, N.: Learning and teaching programming: A review and discussion. *Computer science education*, 13(2), 137-172, 2003.
- [Scr18] Scratch Foundation: Homepage von Scratch. <https://scratch.mit.edu/> Letzter Zugriff: 27.06.2018.

Hochschullehre der Programmierung außerhalb eines Informatikstudiengangs

- [Sei 18] //SEIBERT/MEDIA: Homepage des Angebots "Programmieren lernen für Kinder und Jugendliche in Wiesbaden". www.programmieren.de. Letzter Zugriff: 01.06.2018.
- [Uni18] Unity Technologies: Maschinelles Lernen in Unity3D, <https://unity3d.com/de/machine-learning>. Letzter Zugriff: 01.06.2018.