Über die Eignung der Programmiersprache *Scratch* zur Aneignung von Programmierungskompetenzen

Daniel Walter, Vechta und Dortmund

Obwohl ein breites Angebot an Programmiersprachen für Neueinsteiger existiert, wird die Programmierung im schulischen Kontext nur sehr selten und speziell in der Grundschule gar nicht aufgegriffen. Worin liegen die Ursachen dieser Beobachtung? Welche Entwicklungen zeichnen sich derzeit ab und welche Interventionsmöglichkeiten bestehen, den status quo zu beeinflussen? Dabei muss stets die Frage berücksichtigt werden, ob die Vermittlung grundlegender Aspekte der Programmierung in der Grundschule überhaupt sinnvoll ist.
Der vorliegende Beitrag liefert empirisch fundierte Antworten auf diese Kernfragen. Insbesondere wird ergründet, welchen Einfluss das Arbeiten mit *Scratch* auf Einstellungen und Kompetenzen angehender Lehrkräfte nehmen kann.

Einleitung

Informatik in der Grundschule? Über die Implementierung informatischer Aspekte in den Mathematikunterricht der Sekundarschulen wird häufig diskutiert. Für die Grundschule existiert ein derartiger Diskurs jedoch nicht.

Damit Kindern vergegenwärtigt wird, dass stundenlanges Spielen am Computer sowie das Nutzen von Standardprogrammen nicht zum Themengebiet der Informatik zu zählen ist, müssen konkrete Konzepte ausgearbeitet werden, die den Ideen der Informatik gerecht werden. Dabei ist es im Zeitalter von Tablets & Co. unerlässlich, Kinder frühzeitig an dieses Themenfeld heranzuführen.

Warum würde eine Suche nach Ideen der Informatik im Mathematikunterricht der Grundschulen derzeit überhaupt erfolglos bleiben? Entscheidende Interventionen könnten durch Mathematiklehrkräfte selbst vorgenommen werden. Allerdings ist es fraglich, ob diese einerseits die nötige positive Einstellung zu informatischen Themen aufweisen sowie andererseits über die nötigen Kompetenzen für eine unterrichtliche Vermittlung verfügen.

Der vorliegende Beitrag rückt Mathematiklehramtsstudierende in den Fokus. Mittels einer empirischen Studie wurde überprüft, in welchem Ausmaß ein Programmierungsverständnis sowie eine positive Einstellung zur Informatik bei angehenden Mathematiklehrkräften ausgeprägt ist. Dabei wurde insbesondere untersucht, ob Kompetenzen und Einstellungen der Probanden durch das Arbeiten mit der Programmiersprache *Scratch* positiv verändert werden können. Anhand der resultierenden Ergebnisse können Handlungsschritte für eine Weiterentwicklung des Mathematikunterrichts sowie der Lehrerausbildung abgeleitet

Zunächst werden allgemeine Informationen zur Programmiersprache *Scratch* dargelegt, bevor die Konzeption der empirischen Untersuchung offengelegt wird. Anschließend erfolgt die Analyse und Interpretation der Kernergebnisse. Der Beitrag schließt mit notwendigen Implikationen für den Umgang mit *Scratch* in Schule und Universität ab.

1 Was ist *Scratch*?

Scratch ist eine visuelle Programmiersprache, die an der *Lifelong Kindergarten Group* am *MIT Media Lab* entwickelt wurde. Die aktuelle *Scratch*-Version ist auf der Homepage (*http://scratch.mit.edu*) kostenlos erhältlich.

Warum wurde Scratch entwickelt?

Die Motivation für die Entwicklung *Scratch*s ist unter anderem in einigen, am *MIT* festgestellten Nachteilen bisheriger Programmiersprachen zu begründen, die gerade jungen Schülern während des Verinnerlichens grundlegender Ideen der Programmierung entgegenstehen. Das Programmieren in *Scratch* soll gerade dieser Schülergruppe ermöglicht werden (vgl. Resnick 2007).

Objekte werden in vielen schülerorientierten Programmiersprachen (u.a. LOGO) durch bestimmte Befehle programmiert. Diese werden vom Nutzer üblicherweise über die Tastatur an die Programmiersprache übermittelt und dort verarbeitet. Am *MIT* sieht man in diesem Prozess einen entscheidenden Nachteil. Ein erfolgreicher Umgang mit derartigen Programmiersprachen ist nur denjenigen Nutzern möglich, welche die jeweilige Syntax und Semantik verstanden haben (vgl. Maloney 2008). Gerade für Schüler der Primarstufe dürfte das Verinnerlichen dieser Strukturen eine große Hürde darstellen.

Wie funktioniert Scratch?

Für das Programmieren von Objekten stehen in *Scratch* nach Kategorien geordnete Bausteine zur Verfügung. Diese werden mittels einer *Drag-and-Drop* Struktur logisch aneinandergegliedert. Die Inspiration für eine derartige Aufmachung des Programms gewannen die Entwickler aus der Idee der LEGO-Steine, mit denen Menschen seit Kindestagen vertraut sind (vgl. Resnick 2007). Hervorzuheben ist, dass *Scratch* Syntax- und Semantikfehlern automatisch vorbeugt. Dies soll anhand zweier Beispiele illustriert werden:

(a) (b)

Abb. 1: Syntaxfrage in *Scratch*

Das Aneinanderfügen jeglicher Programmierungsbausteine wird in *Scratch* automatisch überprüft. Ist die Angliederung eines Bausteins an einen anderen syntaktisch korrekt, so signalisiert *Scratch* dies durch einen weißen Balken. Im entsprechenden Beispiel (Abb. 1(a)) wird die Anweisung *„nächstes Kostüm“* an die Anweisung *„warte 0.2 Sek.“* angefügt. Erkennt *Scratch* hingegen einen möglichen syntaktischen Fehler, verwehrt die Programmiersprache die Verknüpfung der Bausteine. Beispielsweise ist das direkte Anfügen einer Bedingung an eine Anweisung nicht zulässig, wie Abb. 1(b) darlegt.

(a) (b)

Abb. 2: Semantikfrage in *Scratch*

Die „*warte*“-Anweisung wird stets mit einer Zeitangabe versehen. Diesbezüglich werden Nachkommastellen durch einen Punkt getrennt. Sollte der Nutzer fälschlicherweise ein Komma wählen, erkennt *Scratch* dies und verändert die Eingabe (Abb. 2(a)) dementsprechend automatisch. Obwohl Bausteine syntaktisch korrekt verknüpft wurden, ist das Begehen eines semantischen Fehlers im Allgemeinen nicht auszuschließen. Da *Scratch* dieses Problem allerdings bekannt ist und versteht, was der Nutzer programmieren möchte, erfolgt eine Vereinfachung der Handhabung.

Diese Beobachtung kann und muss aber auch kritisch betrachtet werden. Das automatische Vorbeugen von Syntax- und Semantikfehlern kann dazu führen, dass der Lernende selbst nicht auf diese Fragen aufmerksam wird. Selbstständiges Entdecken dieser Strukturen wird dadurch teilweise verhindert. Vor dem Hintergrund der Tatsache, dass *Scratch* vor allem für Neueinsteiger angeboten werden soll, ist diese Eigenart allerdings wiederum förderlich. Gerade junge Lernende müssen davor geschützt werden, während erster Erfahrungen mit informatischen Inhalten durch unzugängliche Programmieroberflächen demotiviert zu werden.

Forschungsstand

Forschungsvorhaben bezüglich eines wirksamen Umgangs mit *Scratch* werden vor allem durch das *MIT* selbst vorgenommen. Im Rahmen eines Computer *Clubhouse-Projekts* konnten Kinder ausgewählte Angebote zur Mediennutzung erproben, zu denen auch *Scratch* zählte. Dabei stellte sich heraus, dass der Schlüssel zum Erlernen grundlegender Konzepte der Programmierung

- in der eigentätigen und spielerische Auseinandersetzung sowie
- in der Abwesenheit programmierungserfahrener Mentoren zu finden ist (vgl. Maloney 2008).

Auch im deutschsprachigen Raum ist *Scratch* bereits zum Forschungsgegenstand geworden. Beispielsweise werden im Rahmen des Projekts *InfoSphere - Schülerlabor Informatik* an der *RWTH* Aachen zahlreiche *Scratch*-Unterrichtsmaterialien entwickelt und anschließend in der Praxis erprobt (vgl. Schröder 2013).

2 Empirische Studie

Die Eignung der Programmiersprache *Scratch* zur Aneignung von Programmierungskompetenzen wurde im Rahmen einer empirischen Studie untersucht.

2.1 Ziele der empirischen Untersuchung

Die folgenden Forschungsfragen stehen vor dem Hintergrund der dargelegten Überlegungen im besonderen Interesse der empirischen Untersuchung:

- Verfügen angehende Lehrkräfte über ein hinreichendes Programmierungsverständnis, um Ideen der Programmierung vermitteln zu können?
- Vertreten zukünftige Lehrer eine positive Einstellung zum PC-Einsatz in der Schule sowie zu Ideen der Programmierung?
- Welchen Einfluss hat die Programmiersprache *Scratch* auf die Programmierungskompetenzen sowie auf die Einstellung zum PC-Einsatz und zur Programmierung der Studierenden?

2.2 Forschungsdesign

Um eine adäquate Antwort auf obige Forschungsfragen generieren zu können, wurde eine empirische Untersuchung durchgeführt, die eine Stichprobe von $N = 88$ Probanden umfasste. Zum Untersuchungszeitraum (2. und 3. KW 2013) besuchten die Probanden eine Lehrveranstaltung des Bachelor „Combined Studies“ im Fach Mathematik an der Universität Vechta. Die Vorerfahrungen der Probanden beliefen sich auf Grundkenntnisse der Programmierung und PC-Anwendung, die sie bereits im Rahmen einer bisherigen Lehrveranstaltung erworben haben sollten.

Die Konzeption der empirischen Untersuchung erfolgte nach dem Muster einer Panelstudie. Der Rahmen der Untersuchung wurde durch zwei Datenerhebungen festgelegt, einem Pre- und Posttest. Nachdem die Probanden den Pretest abgelegt haben, besuchten sie eine Vorlesung. Diese thematisierte hauptsächlich die Bedeutung der Programmierung für Individuum und Gesellschaft. Abschließend wurde ein mögliches Angebot an Programmier-

sprachen für junge Lernende vorgestellt[1], bevor das Potential *Scratch*s dargelegt wurde. In der folgenden Kalenderwoche besuchten die Probanden die zur Vorlesung begleitenden Seminare, in denen Übungsaufgaben in *Scratch* bearbeitet wurden. Unmittelbar nach der Seminarzeit erfolgte die Durchführung des Posttests.

Gestaltung der Erhebungsinstrumente

Um sowohl das Programmierungsverständnis als auch die Einstellungen und Haltungen zur Programmierung sowie zum PC-Einsatz zu erheben, wurden Pre- und Posttest in jeweils differierende Abschnitte gegliedert.

Die Erhebung der Einstellungen erfolgte mittels likertskalierten Itembatterien. Damit eine interne Konsistenz der Daten gewährleistet werden kann, wurden die jeweiligen Cronbachs α-Werte generiert. Die entstandenen Ergebnisse belegten zufriedenstellende Reliabilitäten.

Demgegenüber erfolgte die Überprüfung des Programmierungsverständnisses der Probanden durch speziell entwickelte Aufgaben. Die empirische Untersuchung umfasste die Bearbeitung zweier Aufgaben, die jeweils auf verschiedene Programmierungskompetenzen abzielen.

Aufgabe 1: Kompaktes und effizientes Programmieren:

$a \leftarrow 3$
$e \leftarrow a + 2$
$a \leftarrow \frac{e}{2} + 1$
$m \leftarrow 5 \cdot 8$
$x \leftarrow m \cdot 2$
$m \leftarrow m + e$
$e \leftarrow e - 1$
$m \leftarrow m + e$
$e \leftarrow e - 1$
$m \leftarrow m + e$
$e \leftarrow e - 1$
$t \leftarrow x + 3$
Gib aus: m

Abb. 3: Struktogramm zu Aufgabe 1

[1] Exemplarisch wurden in der Vorlesung die Programmiersprachen LOGO, Small Basic und Alice thematisiert

Welche Anweisungen sind für die Ausgabe einer bestimmten Variablen relevant? Wie kann ein vorgegebener Algorithmus kompakter aufbereitet werden? Um die erste Aufgabe erfolgreich bewältigen zu können, mussten diese Leitfragen berücksichtigt werden. Den Probanden wurde ein fiktiver Algorithmus in Struktogrammdarstellung vorgegeben, der unter Berücksichtigung von Strukturelementen kompakter aufbereitet werden sollte.

Aufgabe 2: Korrektur logischer Fehler in Algorithmen

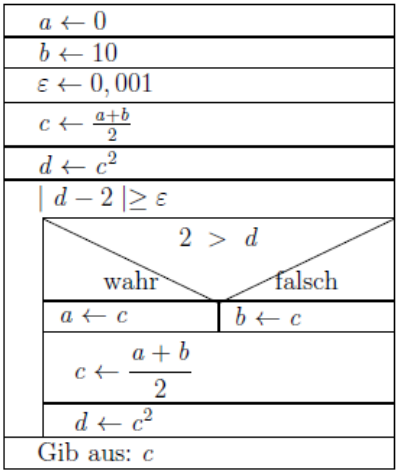

Abb. 4: Struktogramm zu Aufgabe 2

Der zweite Algorithmus wurde den Probanden unter Berücksichtigung eines Kontextes vorgelegt. Dieser besagte, dass ein Algorithmus zur Ermittlung eines Näherungswertes von $\sqrt{2}$ mittels des Intervallhalbierungsverfahrens entwickelt wurde. Die Probanden wurden daraufhin angeleitet, zu entscheiden, ob der vorgegebene Algorithmus das intendierte Ziel erfüllt. Sollte dies nicht der Fall sein, war eine Korrektur des fehlerverursachenden Bausteins im Struktogramm gefordert.

Obiger Algorithmus erfüllt die Intention, einen Näherungswert von $\sqrt{2}$ auszugeben. Die Algorithmen der Tests wurden allerdings derart verändert, dass dieses Ziel nicht erfüllt sein kann. Während im Pretest eine Negation der Schleifeneinbruchsbedingung erfolgte, wurde im Posttest die in der Schleife enthaltene Selektion mit einer veränderten Bedingung versehen. Die erforderliche Kompetenz, derartige Aufgaben lösen zu können, ist im

Schulalltag omnipräsent. Lehrer müssen tagtäglich in der Lage sein, die Rechenwege ihrer Schüler nachzuvollziehen und zu korrigieren.

Für beide Aufgabenformate wurden die Ansätze der Probanden verschiedenen Lösungskategorien zugeordnet. Diese sind qualitativ differenziert geordnet, so dass Lösungen erster Kategorie den Anforderungen in höherem Maße entsprechen als Ansätze der zweiten oder dritten Kategorie.

2.3 Ergebnisse der Untersuchung

Im Folgenden werden die Kernergebnisse der empirischen Untersuchung dargelegt:

Programmierungskompetenzen:

Kompaktes, effizientes Programmieren

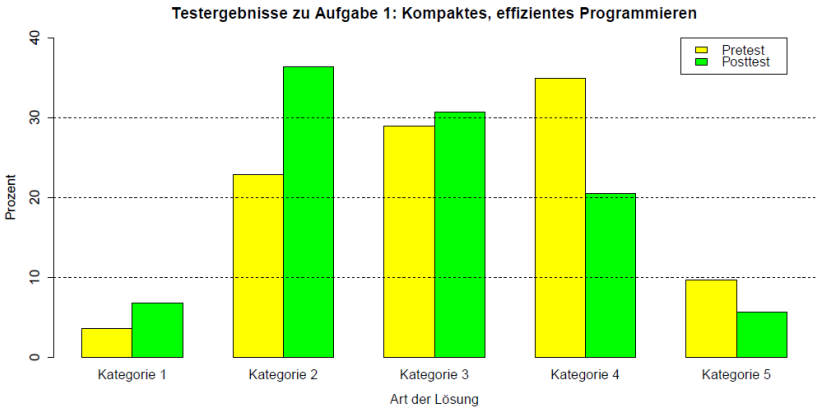

Abb. 5: Testergebnisse: Effizientes Programmieren

Aus den Daten der vergleichenden Balkendiagramme ist abzuleiten, dass die Probanden im Posttest besser abschnitten als im Pretest. Darüber hinaus zeigt die Durchführung des *Wilcoxon*-Tests, dass der Leistungsunterschied als signifikant eingestuft werden muss (p =0,3%).

Weiterhin ist festzustellen, dass der Prozentsatz derjenigen Studierenden, welche die Aufgabe nicht bearbeiteten, sank. Während 5,7% im Pretest keinen Ansatz generierten, haben ausnahmslos alle Teilnehmer die Aufgabe im Posttest zumindest bearbeitet. Folglich stieg die Bereitschaft, sich mit Problemen der Programmierung auseinanderzusetzen.

Korrektur logischer Fehler

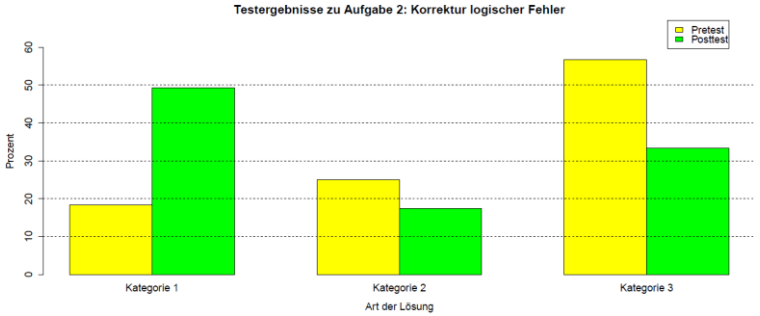

Abb. 6: Testergebnisse: Korrektur logischer Fehler

Die Testergebnisse der Studierenden zur Korrektur logischer Fehler offenbaren ebenfalls einen signifikanten Leistungszuwachs ($p \approx 0{,}03\%$). Dieser wird vor allem durch die Häufigkeitsverteilung der qualitativ besten Lösungen in Pre- und Posttest ersichtlich. Während im Pretest nur jeder fünfte Ansatz den gesetzten Erwartungen entsprach, konnten im Posttest fast 50 % diese Leistung erbringen.

Weiterhin stieg auch in diesem Aufgabenformat die Bereitschaft, sich mit Problemen der Informatik auseinanderzusetzen. Dennoch muss bemängelt werden, dass die relative Häufigkeit derjenigen Probanden, die keinen Ansatz generierten, zu beiden Erhebungszeitpunkten erschreckend hoch ist (Pretest: 31,8% ; Posttest: 21,6%).

Differenzierte Ergebnisanalyse

Um genauere Aussagen über das Abschneiden einzelner Gruppen treffen zu können, erfolgt nun eine differenzierte Analyse der Daten. Hierbei wird insbesondere Bezug auf die Merkmale Geschlecht, Informatikkenntnisse, angestrebte Schulform sowie Expertenstatus[2] genommen.

- Geschlecht

Frauen zeigten im Rahmen des Pretests bessere Leistungen. Diese Leistungsunterschiede sind allerdings als nicht signifikant einzustufen. Demge-

[2] Diejenigen Studierenden, die die begleitenden Übungen zur Vorlesung moderierten, hatten den *Expertenstatus* inne, da diese sich zwingend umfassender mit den Inhalten auseinandersetzen mussten.

genüber wurden beide Aufgaben des Posttests von den Geschlechtern nahezu gleich gut bearbeitet.

- Informatikkenntnisse

Hier lassen sich obige Beobachtungen erneut erkennen. Im Pretest schnitten diejenigen Studierenden etwas besser ab, die bereits in ihrer Schulzeit im Fach Informatik unterrichtet wurden. Während der zweiten Erhebungsphase egalisierte sich dieser Unterschied allerdings.

- Expertenstatus

Studierende mit Expertenstatus konnten im Posttest signifikant bessere Ergebnisse erzielen, als Nicht-Experten. Während des Pretests zeigten sich hingegen keine nennenswerten Unterschiede.

- angestrebte Schulform

Im Rahmen der empirischen Untersuchung ließen sich keine signifikanten Unterschiede zwischen Studierenden verschiedener angestrebter Schulform erkennen. Grundschullehramtsstudierende schnitten ähnlich gut ab, wie Studierende der Sekundarstufe.

Hervorzuheben ist insgesamt, dass keine Teilpopulation im Posttest schwächere Ergebnisse erzielte, als im Pretest. Ein Kompetenzzuwachs war stets erkennbar. Weiterhin sind die Leistungsdifferenzen der Merkmale Geschlecht, Informatikkenntnisse und angestrebte Schulform im Posttest deutlich geringer als im Pretest.

Einstellung zum PC-Einsatz

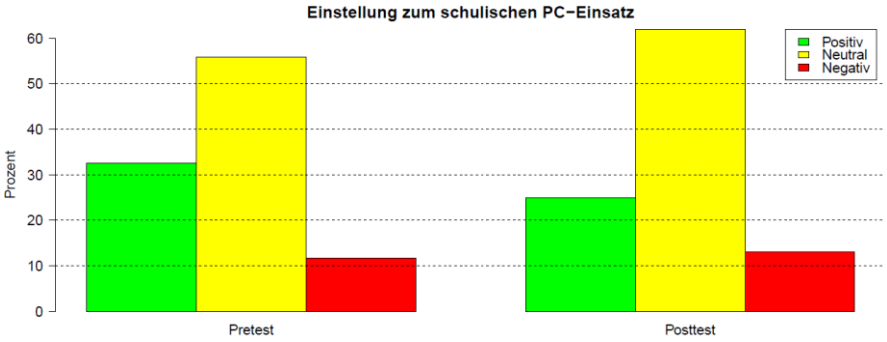

Abb. 7: Einstellung zum schulischen PC-Einsatz

Die Abbildung illustriert, dass sich zwischen den Erhebungsdaten keine nennenswerten Unterschiede zur Einstellung zur PC-Nutzung erkennen lassen. Zwar sinkt der Anteil der Probanden, die den PC-Einsatz positiv einschätzen von 31 % auf 25 %. Allerdings ist dieser Unterschied statistisch als nicht signifikant einzustufen (p = 16,24%).

Einstellung zur Programmierung

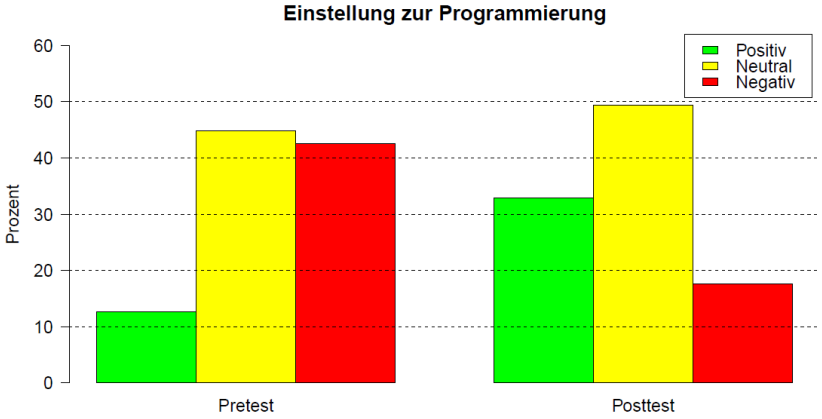

Abb. 8: Einstellung zur Programmierung

Demgegenüber zeigt Abb. 8, dass sich die Einstellung zum Themenfeld Programmierung allgemein positiv entwickelt hat. Während nur etwa 10% der Probanden im Pretest eine positive Einstellung erkennen lassen, sind es im Posttest bereits etwa ein Drittel der befragten. Durch die Datenanalyse mittels des *Wilcoxon*-Tests ($p \approx 2{,}1 \cdot 10^{-5}$) lässt sich eine signifikante Einstellungsänderung belegen.

Einstellung zu Scratch

Mehrheitlich befürworten angehende Mathematiklehrkräfte die Frage, ob *Scratch* für die Grundschule geeignet sei. Mehr als 60% der Probanden stimmten dem entsprechenden Item zu. Besonders interessant wird dieses Ergebnis unter Bezugnahme der Frage, ob die Studierenden *Scratch* als bedeutsam für Ihren weiteren Werdegang als Lehrkraft einstufen. Es zeigt sich, dass lediglich 15 % die Auffassung vertreten, dass *Scratch* für den Werdegang als Lehrkraft bedeutsam ist.

Zunächst erscheint es paradox, dass die Probanden *Scratch* einerseits eine Eignung für die Grundschule aussprechen, während sie andererseits keine Bedeutung für (ihre eigene) Zukunft erkennen. Ein möglicher Interpretationsgang hierfür wäre, dass keine Möglichkeit erkannt wird, *Scratch* im Mathematikunterricht einzubringen.

3 Schlussfolgerungen für den Umgang mit *Scratch*

Die empirische Untersuchung lieferte folgende Erkenntnisse:

- *Scratch* ist dafür geeignet, um das Programmierungsverständnis angehender Lehrkräfte zu verbessern.
- Die Einstellung zur Programmierung wird durch die Arbeit mit *Scratch* positiv verändert.
- Der Einsatz von *Scratch* ist nach Ansicht der Studierenden auch für die Grundschule angemessen. Dennoch sehen sie keine Bedeutung für ihren weiteren Werdegang als Lehrkraft.

Aufbauend zu diesen Erkenntnissen sind Implikationen für den Umgang mit *Scratch* in der universitären Lehre zu formulieren. *Scratch* ist geeignet, um Lehramtsstudierenden

- Programmierungskompetenzen zu vermitteln,
- eine positive Einstellung gegenüber der Programmierung zu tradieren,
- ein Instrument aufzuzeigen, das die unterrichtliche Aufbereitung von Aspekten der Programmierung ermöglichen kann.

Weiterhin stellte sich heraus, dass *Scratch* unabhängig von den Leistungsständen der Nutzer angeboten werden kann. Nach dem Arbeiten mit *Scratch* werden die Kompetenzen auf ein ähnliches Level angehoben.

Obwohl *Scratch* zahlreiche positive Eigenschaften erkennen lässt, ist ein kritischer Umgang unerlässlich. Beispielsweise lassen sich nicht alle Ziele mit *Scratch* verwirklichen. Möchte der Lernende beispielsweise eine dreidimensionale Animation erzeugen, muss eine andere Programmiersprache herangezogen werden. Dieses Beispiel verdeutlicht, dass die Intentionen des Programmierens die Wahl der Programmiersprache bestimmt. Daher müssen zukünftigen Lehrkräften Alternativen angeboten werden. *Scratch* darf nicht blind in den Unterricht integriert werden, ohne die zu erreichenden Ziele zu betrachten.

Zusammenfassung

Die vorliegende empirische Untersuchung fokussierte explizit Mathematiklehramtsstudierende der Grund-, Haupt- und Realschulenden. Es wurde herausgestellt, dass *Scratch* dafür geeignet ist, das Programmierungsverständnis der Studierenden zu verbessern und gleichzeitig eine positivere Haltung zum Themenfeld aufzubauen. Darüber hinaus wurde nachgewiesen, dass *Scratch* unterschiedliche Leistungs- und Kenntnisstände angehender Lehrkräfte berücksichtigt. Das Arbeiten mit *Scratch* spricht unterschiedliche Lerngruppen mit ihren jeweiligen Neigungen und Interessen an. Somit weist die Programmiersprache ein großes Differenzierungspotential im universitären Kontext auf.

Aber wie arbeiten Schüler mit *Scratch*? Eignet sich die Programmiersprache auch für den Einsatz in der (Grundschul-) praxis? Um die Wirksamkeit des schulischen Einsatzes von *Scratch* vollkommen aufzudecken, sind weiterführende Studien vonnöten, die positive Effekte auf das Lernen von Schülern empirisch fundiert sichern.

Darüber hinaus sollte die Frage ergründet werden, inwieweit ein fundiertes Programmierungsverständnis sich auf die Mathematikleistungen der Schüler auswirkt. Hierbei könnte diese Beziehung durchaus bereits in der Grundschule erforscht werden. Ideen der Programmierung, wie das algorithmische Denken sowie Schleifen und Selektionen, lassen sich auch in dieser Schulform, beispielsweise im Rahmen der schriftlichen Rechenverfahren, erkennen. Sind Grundschüler eher in der Lage, die Verfahren zu verinnerlichen, wenn sie auch ein Programmierungsverständnis herausgebildet haben? Da diese Fragen im wissenschaftlichen Diskurs bisher unbeantwortet bleiben, muss es ein Anliegen sein, dieses Forschungsfeld zu begehen.

Danksagung

Ein abschließender Dank geht an Dr. Andreas Kirsche, der mich durch konstruktive Impulse und Ideen während der Planung und Durchführung der empirischen Untersuchung sowie der Erstellung dieses Artikels unterstützt hat.

Literatur

Maloney, John (2008). Programming by Choice. Urban Youth Learning Program ming with Scratch. http://web.media.mit.edu/~mres/papers/sigcse08.pdf

Resnick, Mitchel (2007). All I Really Need to Know (About Creative Thinking) I Learned (By Studying How Children Learn) in Kindergarten. http://web.media.mit.edu/~mres/papers/kindergarten-learning- ap proach.pdf

Resnick, Mitchel (2009). Scratch: Programming for all. http://web.media.mit.edu/~mres/papers/Scratch-CACM-final.pdf

Schröder, Ulrik (2013). Spielend pro- grammieren lernen mit Scratch. http://schuelerlabor.informatik.rwthaachen.de/modul/spielend-programmieren-lernen-mit-scratch

Auf getrennter Seite [kommt dann ans Ende des Buchs]

Adresse des Autors:
Daniel Walter
Institut für Entwicklung und Erforschung des Mathematikunterrichts
Technische Universität Dortmund
Vogelpothsweg 87
47227 Dortmund
dwalter@math.tu-dortmund.de