Lernumgebungen für den Einstieg ins Programmieren: Versuch einer Klassifikation

Michael Hielscher
Pädagogische Hochschule Schwyz
michael.hielscher@phsz.ch

Beat Döbeli Honegger
Pädagogische Hochschule Schwyz
beat.doebeli@phsz.ch

Abstract: Der Einstieg ins Programmieren ist nicht einfach. Lernende sind gleichzeitig mit der Problemlösung in Form eines Algorithmus, den Befehlen und der Syntax der Programmiersprache, der Handhabung der Programmierumgebung, der Programmausführung und Fehlersuche konfrontiert. Seit rund fünfzig Jahren werden deshalb spezielle Lernumgebungen entwickelt. Dieses Angebot ist mit Smartphones und Tablets und kostengünstigen Robotern (z.B. BeeBots, Thymio oder Lego Mindstorms) in den letzten Jahren stark gewachsen. Im Beitrag stellen wir sieben Kriterien vor, die Lehrkräfte bei der Wahl einer Lernumgebung für den Einstieg ins Programmieren unterstützen können.

1 Einleitung

Programmieren als wichtiger Teilaspekt der Informatik wird zunehmend als Teil der Allgemeinbildung gesehen. Es wird sowohl verstärkt in die Lehrpläne einzelner Länder aufgenommen als auch mit breit angelegten Initiativen wie der europäischen Code Week oder der amerikanischen Hour of Code gefördert. Millionen von Schülerinnen und Schülern weltweit haben im Rahmen dieser Programme erste Programmiererfahrung mit verschiedenen Lernumgebungen sammeln können, auch wenn die Schulen keine eigenen Zeitgefässe für Informatik anbieten.

Während Programmieren auf der Sekundarstufe II bereits seit längerem praktiziert und erforscht wird, ist in letzter Zeit insbesondere das Interesse an Programmiereinführungen bei jüngeren Schülerinnen und Schülern gewachsen. Begründet wird diese frühe Einführung mit Berufswahlargumenten, einer allgemeinen MINT-Förderung sowie einem mit zunehmendem Alter schwindenden intrinsischen Interesse am Programmieren (vgl. [She03]). Verschiedentlich wird dabei die Grundschule - oft das achte Altersjahr - als ideales Einstiegsalter zur Vermittlung erster Programmierkonzepte genannt ([She03], [EGR+08], [Sch01]). In diesem Alter interessieren sich die Kinder nicht primär für das Thema programmieren an und für sich. Hingegen sind Kinder sehr motiviert Programmieren als Werkzeug und Hilfsmittel zu erlernen, wenn sie damit Dinge nach Ihren Vorstellungen konstruieren können (vgl. [Lan06]).

2 Programmieren lernen - eine Herausforderung

Unter dem Begriff Programmieren verstehen wir den systematischen Prozess des Entwurfs, der Implementation und des Testens eines Computerprogramms. Mindestens die letzten beiden Schritteerfolgen typischerweise innerhalb einer Programmierumgebung. In der Praxis werden Programmiersprachen wie C++, Java oder Phyton mit professionellen Entwicklungswerkzeugen wie Eclipse, Visual Studio oder XCode verwendet. In den letzten 50 Jahren wurden unzählige Lernumgebungen für den Einstieg ins Programmieren entwickelt, die mit altersgerechter Darstellung und spielerischen Ansätzen versuchen das Bedürfnis der Kinder zu erfüllen, eigene Ideen relativ schnell und mit Erfolgserlebnis umsetzen zu können. Lernumgebungen für den Einstieg ins Programmieren basieren deshalb häufig auf einer stark vereinfachten Programmiersprache und unterstützen die Lernenden durch zusätzliche Visualisierungen und Feedbacks während aller Phasen des Programmierprozesses. Lernumgebungen für den Einstieg existieren nicht nur als reine Softwarelösungen, sondern auch in Kombination mit spezieller Hardware oder gar als reine Hardwarelösungen. Bei Primo.io können beispielsweise die Bewegungen eines kleinen motorisierten Roboters per Funk über das Einsetzen von unterschiedlich gestalteten Befehlsbausteinen auf einem Spielbrett aus Holz programmiert werden (Abb. 1, links). Auch Brettspiele wie Robot Turtles (Abb. 1, rechts) ermöglichen das Entwerfen, Implementieren und Testen einfacher Programme ganz ohne Strom (dies erfordert jedoch die Nutzung eines Tutors als Schildkröten-steuernden analogen „Computer“).

Abbildung 1: Enaktive Lernumgebungen am Beispiel Primo.io (links) und Robot Turtles (rechts)

Kelleher und Pausch [KP05] untersuchten im Jahr 2005 rund 75 Programmierlernumgebungen aus den Jahren 1960 bis 2003. Sie identifizierten mehrere Herausforderungen die beim Einstieg ins Programmieren adressiert werden müssen und klassifizierten Lernumgebungen anhand der vier Aspekte Komplexität, Bedienbarkeit, Nachvollziehbarkeit und Motivation, wobei ihnen nicht immer eine eindeutige Einordnung möglich war. Insbesondere neuere Lernumgebungen versuchen meist mehrere oder alle Aspekte gleichzeitig zu adressieren.

Komplexitätsaspekt:
Professionell genutzte Programmiersprachen sind sehr komplex, das Erlernen der Syntax ist aufwändig und die Interpretation von Fehlermeldungen erfordert häufig ein gutes Verständnis der Sprache und der verwendeten Programmbibliotheken. Bereits für einfa-

che Programme sind viele Rahmenbedingungen zu erfüllen, die vom eigentlichen Lernziel ablenken können. Die Klasse der Mini-Programmiersprachen speziell für Lernzwecke versucht durch Reduktion der Komplexität Abhilfe zu schaffen (vgl. [BCH+98]). Auch komplexe Sprachen wie Java lassen sich mit Hilfe von Lernumgebungen für den Einstieg so reduzieren, dass zunächst keine Objektorientierung oder ein „public static void main" benötigt wird (vgl. [HW09]).

Bedienbarkeitsaspekt:
Die Eingabe von Programmtexten über die Tastatur ist für Kinder mühsam und fehleranfällig. Ein fehlendes Semikolon führt in klassischen Programmiersprachen schnell zu unverständlichen Fehlermeldungen. Die Klasse der eingabeunterstützenden Lernumgebungen versucht diese Probleme zu vermeiden, indem visuelle Programmiersprachen und -umgebungen entwickelt wurden. Einige ermöglichen deren Einsatz noch vor dem Erlernen der Schriftsprache. Professionelle Programmierumgebungen können Lernende auch bereits auf Grund der Menge an Menüs und Schaltflächen überfordern. Mason und Cooper [MC13] konnten zeigen, dass bereits eine gezielte Reduktion der angebotenen Bedienelemente in einer Programmierumgebung zu signifikanten Verbesserungen des Lernerfolgs und des Transfers auf andere Umgebungen führen kann. Gerade diese Reduktion der sichtbaren Elemente wird jedoch teilweise auch als Argument für textorientierte Programmierumgebungen verwendet, da in Textumgebungen nicht gleich der gesamte Sprachumfang sichtbar sein muss (vgl. auch [Mod11]).

Nachvollziehbarkeitsaspekt:
Was der Computer bei der Abarbeitung eines Programms genau tut ist nicht immer einfach nachvollziehbar. Die Klasse der Ablaufsimulationsumgebungen versucht durch spezielle Visualisierungen die Arbeitsweise des Computers zu verdeutlichen. Dies kann zum Beispiel über die Hervorhebung des gerade ausgeführten Befehls oder einer Übersicht aller aktuellen Variablen und deren Werte erfolgen. Ebenso können physische Roboter dazu beitragen, die Abarbeitung eines Programms Schritt für Schritt nachvollziehbar zu machen.

Motivationsaspekt:
Auch Kinder unter 10 Jahren können durchaus - und zwar geschlechtsunabhängig - Interesse an der Programmierung eines Computers haben. Dabei stehen insbesondere die Herstellung eigener Computerspiele, Zeichnungen, Animationen und andere kreative Dinge im Mittelpunkt (vgl. [She03]). Unabhängig vom Alter ist die Entwicklung von Algorithmen für mathematische Probleme jedoch nur für wenige Schülerinnen und Schüler motivierend (vgl. [BCH+98]). In der Klasse der motivationssteigernden Lernumgebungen findet man deshalb besonders ansprechend gestaltete Werkzeuge die häufig den kreativen Aspekt in den Mittelpunkt rücken.

Seit der Untersuchung von Kelleher und Pausch ist sowohl die Anzahl als auch die Bandbreite an Programmierumgebungen für Kinder und Jugendliche deutlich grösser geworden und nimmt weiter zu. Treiber dieser Entwicklung dürfte einerseits die technische Entwicklung, andererseits die bereits in der Einleitung erwähnten bildungspolitische Initiativen für mehr Informatik in der Schule sein. Während die untersuchten Lernumgebungen aus den Jahren 1960 bis 2003 fast ausschließlich Softwarelösungen für Desktop-Computer waren,

sind inzwischen deutlich mehr enaktive Umgebungen und Apps für Tablets und Smartphones hinzugekommen. Der Trend geht zum Anfassen und be-greifbar machen. Dazu zählen neben den bekannten Lego Mindstorm Kästen auch verschiedenste Education-Kits aus der Arduino-Familie und diverse Bodenroboter wie zum Beispiel Roamer oder Bee-Bots. Ein enaktiver Zugang zum Programmieren mit physischen Robotern ist nicht neu und wurde bereits in den 70er Jahren von Seymour Papert mit seinen Floor-Turtles beschrieben (vgl. [Pap80]). Auf Grund der hohen Kosten und fehlender Verbindlichkeiten in den Lehrplänen konnten sich diese Werkzeuge bislang jedoch nicht flächendeckend durchsetzen. Durch neue günstige Elektronikkomponenten (z.B.: Arduino Projekt) und das steigende Interesse von Eltern, ihre Kinder auf die digitale Welt vorzubereiten, werden zunehmend mehr physische Roboter, Baukästen und Spielzeug für den Programmiereinstieg angeboten. Auf Crowdfunding-Plattformen wie Kickstarter finden sich jeden Monat neue Projekte, die eine frühkindliche Förderung von Programmierkompetenzen zum Ziel haben und auch mit grossem Zuspruch von interessierten Eltern finanziert werden. Mit Preisen von teilweise unter 100€ sind diese enaktiven Lernumgebungen auch für Schulen inzwischen erschwinglicher geworden. Aber auch Softwarelösungen ohne spezielle Hardware-Komponenten werden durch Technologien wie Tablets und Smartphones mit ihrer einfachen Touch-Bedienung für Kinder zugänglicher, wie es vor wenigen Jahren in dieser Form noch nicht möglich war.

3 Kriterien zur Einschätzung von Programmierlernumgebungen

Diese sowohl qualitativ als auch quantitativ grössere Vielfalt an Programmierlernumgebungen erhöht zwar die Möglichkeiten für Lehrpersonen im Unterricht, erschwert aber gleichzeitig auch die Übersicht. Um die Potenziale bestehender und zukünftiger Lernumgebungen rascher einschätzen zu können, schlagen wir in dieser Arbeit nach der Evaluation von über 50 aktuell nutzbarer Lernumgebungen eine neue Klassifikation anhand von sieben Kriterien vor (siehe Abb. 2).

Repräsentationskriterium:
Auch im Informatikunterricht kann das E-I-S-Prinzip unterschiedlicher Repräsentationsformen nach J. Bruner Anwendung finden. Alle drei Formen (enaktiv - Erfassen von Sachverhalten durch eigene Handlungen, durch „Greifen“, „Begreifen“; ikonisch - Erfassen von Sachverhalten mit Hilfe von Bildern; symbolisch - Erfassen von Sachverhalten durch Symbole wie Text, Zeichen, Formeln) können bei der Vermittlung grundlegender Programmierkonzepte eingesetzt werden. Die meisten Lernumgebungen verwenden zur besseren Veranschaulichung einen oder mehrere „Roboter“ (Akteure), die von den Lernenden programmiert und damit gesteuert werden. Die physischen oder simulierten Roboter werden je nach angedachter Altersstufe unterschiedlich gestaltet zum Beispiel als Tier (Biene, Schildkröte, Katze usw.) oder als schlichtes Dreieck auf dem Bildschirm. Die Lernumgebungen lassen sich in vier Gruppen unterteilen:

- **enaktiv programmiertes physisches System:**
  Die Lernumgebung (Hardware) wird direkt über Knöpfe oder eine Tastatur am Gerät programmiert. Es ist keine weitere Infrastruktur nötig. Die eingegebenen Befehle

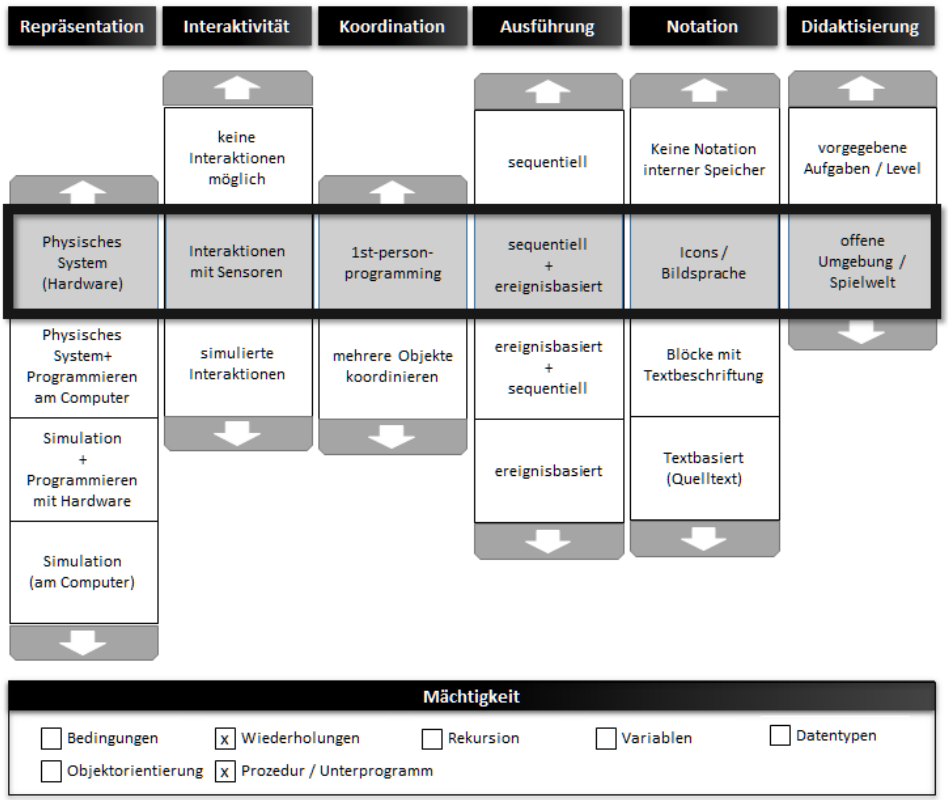

Abbildung 2: Sieben Kriterien zur Einschätzung von Programmierlernumgebungen: Repräsentation, Interaktivität, Koordination, Ausführung, Notation, Didaktisierung und Mächtigkeit

werden zum Beispiel direkt in Bewegungsaktionen in der physischen Welt übersetzt.

- **physisches System mit Programmierung am Computer:**
  Die physische Hardware muss zur Programmierung mit einem Computer (Notebook, Tablet usw.) verbunden werden. Dort wird mit einer meist bildhafte Sprache programmiert. Die Hardware führt anschließend das Programm als Aktionen in der physischen Welt aus.
- **simuliertes System mit physischer Programmierung:**
  Das System wird ausschließlich auf dem Bildschirm eines Computers dargestellt, die Programmierung erfolgt jedoch mit physischen Karten oder Bauklötzchen, die über ein Lesegerät oder Kamera zum Computer und damit zum virtuellen Teil der Lernumgebung übertragen werden. Die Ausführung des Programms erfolgt in der simulierten Umgebung am Bildschirm.
- **simuliertes System mit Programmierung am Computer:**
  Die Lernumgebung wird ausschließlich am Bildschirm eines Computers dargestellt und programmiert. Die Programmierung erfolgt mit bildhaften oder textuellen Spra-

chen. Die Ausführung des Programms erfolgt ebenfalls in der simulierten Umgebung am Bildschirm.

Physische, enaktive Lösungen erleichtern es Anfängerinnen und Anfängern, sich sowohl gedanklich wie auch körperlich in die Rolle des Roboters zu versetzen und mögliche Lösungen zum Beispiel auf einem Spielplan durchzuprobieren. Zudem wirkt der Einsatz physischer Roboter bei Kindern im Alter zwischen 7 und 9 Jahren stark motivierend (vgl. [BCH+98]). Dafür sind bei rein physischen Lernumgebungen die Stückkosten vergleichsweise hoch und die Möglichkeiten relativ beschränkt. Je mehr Anteile der Programmierumgebung virtualisiert werden, desto kostengünstiger kann diese werden und desto grösser sind ihre Möglichkeiten - allerdings müssen dafür Computer verfügbar und deren Nutzung geläufig sein. Allein durch die Möglichkeit, eine Simulation am Computer schneller oder langsamer ablaufen zu lassen, können komplexere Probleme bearbeitet werden, für die ein physischer Roboter viel zu lange umherfahren müsste. Gewisse Programmierkonzepte wie die Verwendung von Variablen lassen sich mit den bisherigen physischen Robotern in der Regel nicht vermitteln. Mit simulierten Umgebungen lassen sich hingegen auch weitere Problemstellungen der Informatik thematisieren (z.B. Sortieralgorithmen mit Kara oder CargoBot) die über die reine Bewegungssteuerung des Roboters hinausgehen. Es scheint somit naheliegend, erste Programmiererfahrungen in der Primarstufe mit enaktiven Umgebungen zu sammeln und später zu virtuellen Simulationsumgebungen zu wechseln.

Interaktivitätskriterium:
Unabhängig vom Repräsentationskriterium bieten sowohl einige physische als auch einige simulierte Umgebungen die Möglichkeit das laufende Programm durch Aktionen zu beeinflussen. Eine Aktionen kann zum Beispiel ein einfacher Tastendruck auf der Tastatur, das Abdecken eines optischen Sensors, oder eine Touchgeste auf dem Tablet sein. Auch akustische Sensoren können Ereignisse wie „in die Hände klatschen“ messen, die die Lernenden in der physischen Welt auslösen können. Die Fähigkeit zur Reaktion auf Ereignisse in der Umwelt bietet Raum zum spielerischen experimentieren und kann zusätzlich motivierend wirken. Anhand von interaktiven Umgebungen lassen sich u.a. auch Konzepte der Mensch-Maschinen-Interaktion als weiteres Teilgebiet der Informatik aufzeigen. Sind keine Interaktionen (Beeinflussungen) während der Abarbeitung eines Programms möglich, so arbeitet das Programm hingegen immer exakt gleich (randomisierte Programmteile ausgenommen). Dieses rigorose Verhalten kann speziell beim Einstieg didaktisch wünschenswert sein, entspricht es doch der Vorstellung der „dummen“ Maschine und vereinfacht die Fehlersuche. Für Interaktionen muss das Programm entsprechend Ereignisse und Reaktionen vorsehen und implementieren. Dieser zusätzliche Anspruch kann Anfangs überfordern. Dies gilt auch für Lernumgebungen, die zwar keine Interaktion mit der realen Welt zulassen, dafür aber ein vergleichbares Verhalten mit virtuellen Sensoren simulieren.

Koordinationskriterium:
Die meisten Lernumgebungen stellen einen einzelnen Roboter (Akteur) in den Mittelpunkt und gestalten ggf. eine Mikrowelt um ihn herum. Die Lernenden können sich bei der Programmierung unmittelbar in die Rolle des Roboters versetzen und dessen Bewegungen und Befehle auch selbst nachspielen. Das Quadratprogramm: „wiederhole vier

mal: gehe gerade aus, drehen dich 90° nach links“ lässt sich beispielsweise sehr gut als Rollenspiel nachstellen, bevor es in den Computer eingegeben wird. In Anlehnung an 1st-Person-Spiele, die aus den Augen der Spielfigur heraus gespielt werden, sprechen wir kurz von 1st-person-programming. Die übrigen Lernumgebungen stellen meist die Spielwelt und die Koordination mehrerer zu programmierender Objekte ins Zentrum. Dies trifft zum Beispiel auf Lego Mindstorm Roboter zu, bei denen nicht der Roboter als Ganzes, sondern einzelne Motoren und Sensoren als System programmiert werdenmüssen. Dies wird in der Programmierumgebung deutlich, wenn nicht Befehle wie „rechts drehen 90°“ sondern „Motor A vorwärts für 1s, Motor B rückwärts für 1s“ verwendet werden. Für den Einstieg ins Programmieren bietet 1st-person-programming einige Vorteile, etwa eine einfachere Fehlersuche im Rollenspiel. Hingegen bieten Umgebungen mit mehreren Objekten vielfältigere Möglichkeiten für die Gestaltung von Spielen oder Animationen.

Ausführungskriterium:
Die einzelnen Befehle eines Programms werden in jeder Lernumgebung nach einem bestimmten Schema abgearbeitet. Aus Sicht der Informatik kann zwischen Single- und Mutli-Thread-Laufzeitverhalten unterschieden werden. Single-Thread bedeutet in diesem Fall, dass alle Befehle stets sequentiell nacheinander abgearbeitet werden. Multi-Thread hingegen, dass mehrere Programmstränge parallel voneinander ausgeführt werden - beispielsweise um ständig Sensorwerte zu prüfen, während das Hauptprogramm abläuft. Die Form der Abarbeitung hat Auswirkungen auf die Art wie Programme entworfen, implementiert und getestet werden müssen. Die sequentielle Abarbeitung begünstigt das imperative Programmierparadigma und kann gut mit Rollenspielen nachgespielt werden. Das Lesen sequentieller Programme lässt sich mit dem Lesen von Bauanleitungen oder Backrezepten vergleichen. Ein Programm könnte zum Beispiel so aussehen: fahre drei mal geradeaus, drehe dich nach rechts, fahre geradeaus, schalte Licht an, warte kurz, schalte Licht aus. Die Ereignis-basierte Abarbeitung (Event-driven-programming) entspricht eher der Vorstellung eines regelbasierten Systems welches ständig alle angegebenen „WENN Bedingung DANN Anweisung“-Regeln überprüft. Sobald oder solange eine Bedingung eintritt, wird die Anweisung ausgeführt. Ein solches Programm könnte zum Beispiel so aussehen (die Reihenfolge der Regeln spielt keine Rolle):

WENN kein Hindernis voraus DANN fahre nach vorn;
WENN Hindernis voraus DANN drehe dich nach rechts;
WENN Helligkeitssensor < 10 DANN schalte Licht an;
WENN Helligkeitssensor >= 10 DANN schalte Licht aus;

Bei einigen Lernumgebungen werden beide Ansätze verwendet, wobei jedoch in der Regel ein Ansatz vorherrschend ist. Für das Ausführungskriterium ergeben sich somit die vier Ausprägungen: sequentiell, sequentuell / ereignisbasiert, ereignisbasiert / sequentiell und ereignisbasiert. Für den Einstieg ins Programmieren eignet sich sowohl ein streng sequentieller als auch ein streng ereignisbasierter Ansatz - die gleichzeitige Verwendung beider Ansätze erfordert in der Regel zunächst Erfahrung mit der in der Lernumgebung vorherrschenden Variante. Der sequenzielle Ansatz legt mehr Augenmerk auf die strukturierte Planung von Programmen, ermöglicht dafür eine einfachere Fehlersuche - die Programmiersprache Logo und BeeBots-Roboter sind gute Beispiele dafür. Der eventbasierte-Ansatz fördert stärker das Experimentieren und einfache Erweitern des Programms wenn

mehrere Objekte in einer Mikrowelt miteinander interagieren (zum Beispiel in Kodu oder AgentCubes).

Notationskriterium:
Ein Programm besteht aus einer Folge von Befehlen, die von den Lernenden in irgendeiner Form in der Lernumgebung eingegeben werden müssen. Von Programmiersprachen wie Java, Pascal oder C++ kennen wir die klassische textuelle Repräsentation als editierbaren Quelltext. Im Gegensatz zu dieser symbolischen Programmrepräsentation verwenden viele Lernumgebungen für Kinder eine visuelle, ikonische Darstellung. Befehle werden als bebilderte Blöcke bzw. Text-Blöcke dargestellt die aneinandergereiht oder wie Puzzelteile aneinandergefügt werden können. Syntaktische Fehler lassen sich vollständig vermeiden, da nur passende Blöcken zusammengefügt werden können. Je nach Darstellungsform und Eingabemethode sind entsprechende Vorkenntnisse der Lernenden erforderlich. Bei ikonischen Darstellungen (zum Beispiel als Richtungspfeile) sind häufig weder Lese- noch Schreibkompetenzen nötig, wodurch sie bereits im Kindergarten eingesetzt werden können. Umgebungen mit Text-Blöcken erfordern zumindest grundlegende Lesekompetenzen. Die klassischen textuellen Umgebungen wie zum Beispiel XLogo erfordern zusätzlich entsprechende Schreibkompetenzen und Sorgfalt bei der Eingabe von Programmtexten. Einige wenige physische Roboter (z.B. BeeBot, Roamer Too) verzichten - vermutlich aus Kostengründen - vollständig auf eine Darstellung des entwickelten Programms. Das Programm steckt ausschliesslich im internen Speicher des Geräts. Dies verhindert jedoch das Lesen und Überarbeiteneines einmal eingegebenen Programms - ein Fehler muss in der Regel durch eine komplette Neueingabe des ganzen Programms ausgebessert werden - bei komplexeren Aufgabenstellung wirkt dies schnell demotivierend.

Mächtigkeitskriterium:
In praxisrelevanten Programmiersprachen lassen sich in der Regel alle gebräuchlichen Programmierkonzepte wie Schleifen, Unterprogramme, Rekursionen usw. verwenden. Lernumgebungen mit Mini-Programmiersprachen bieten hingegen teilweise nur eingeschränkte Möglichkeiten und sind somit weniger mächtig. In Anlehnung an Kelleher und Pausch [KP05] schlagen wir die folgenden Konzepte als Unterscheidungsmerkmale für Programmierlernumgebungen vor:

- **Bedingte Anweisung:** Es können Befehle mit einer Bedingung verknüpft werden, sodass diese nur abgearbeitet werden, wenn die Bedingung erfüllt (true) ist. Einige Umgebungen bieten zusätzlich auch die erweiterte Form, die Verzweigung (if then else) an.
- **Wiederholung:** Eine Folge von Befehlen lassen wiederholt ausführen. Sind in der Umgebung auch bedingte Anweisungen verfügbar, lässt sich in der Regel auch eine Bedingung (solange true) verwenden.
- **Prozedur / Unterprogramm:** Es lassen sich Teilprogramme definieren und in verschiedenen Kontexten wiederverwenden. Ein Programm kann damit zudem in selbstgewählte Sinnabschnitte gegliedert werden (etwa eine Prozedur Blatt die wiederholt für das zeichnen einer Blume aufgerufen wird).
- **Rekursion:** Prozeduren können sich rekursiv aufrufen und bieten damit eine Alternative zur Wiederholung. Häufig wird entweder die Wiederholung oder die Rekursion in einer Lernumgebung verwendet.

- **Variablen:** Variablen können selbst angelegt, ein Wert zugewiesen und wieder ausgelesen werden. Vordefinierte und veränderbare Objektattribute (z.B. Position, Drehung, Transparenz) sollen hier nicht als Variablen verstanden werden.
- **Datentypen:** Variablen und Prozedurparameter verwenden verschiedene Datentypen wie Zahl, Zeichenkette oder Wahrheitswert, mit einem zugehörigen Definitionsbereich. Sie müssen entsprechend richtig gewählt und ggf. ineinander transformiert werden.
- **Objektorientierung:** Bei einer objektorientierten Lernumgebung sind alle Elemente der Mikrowelt als Objekte mit Attributen und Methoden definiert und Objekte können miteinander interagieren (z.B. durch den Austausch von Nachrichten). Die objektorientierte Programmierung mit Klassen und Vererbung ist hingegen nur in ganz wenigen Lernumgebungen angedacht. Eine objektorientierte Lernumgebung bietet jedoch gute Voraussetzungen für die spätere objektorientierte Programmierung in einer Hochsprache.

Mit Blick auf die Alterststufe der Lernendenist ist eine umfangreichere Sprache nicht immer auch die didaktisch Sinnvollere. Umgekehrt stoßen aber Umgebungen, die sehr wenige oder keine der aufgeführten Konzepte unterstützen (z.B. BeeBots) nach kurzer Zeit an ihre Grenzen. Die zur Verfügung stehende Zeit, die Lernziele sowie Vorkenntnisse und die Altersstufe entscheiden letztendlich darüber, welche Konzepte die zu wählende Programmierumgebung mindestens unterstützen muss.

Didaktisierungskriterium:
Für praktisch alle Lernumgebungen werden Unterrichtsmaterialien und Aufgabenstellungen für die Lehrperson angeboten. Einige Umgebungen bieten vorgefertigte und in der Regel aufeinander aufbauenende Aufgaben / Rätsel (Spiellevel) mit automatisiertem Feedback und Hilfestellungen innerhalb der Lernumgebung an. Diese können von Lernenden sehr selbstständig bearbeitet werden und eignen sich somit auch für die außerschulische Nutzung. Lehrpersonen mit bisher wenig Programmier-Erfahrung bzw. der Vermittlung von Programmierkonzepten können von derartig vorstrukturierten Aufgaben profitieren. Umgekehrt schränken vorgegebene Aufgabenstellungen mit automatisiertem Feedback die Kreativität der Lernenden ein. Teilweise wird nur eine ganz bestimmte Lösung akzeptiert, obwohl mehrere Programme zum geforderten Resulat führen. Im Bereich der enaktiven Lernumgebungen fehlen derzeit noch Systeme, welche vorgefertigte Aufgaben zum Beispiel in Form gesprochener Aufgabenstellungen „Lass uns ein Viereck malen“ und entsprechendem automatisiertem Feedback „ich glaube wir haben einen Schritt vergessen“ bereitstellen. Die meisten Lernumgebungen bieten ausschließlich oder optional den freien Sandkasten-Modus an, bei dem keine Aufgabenstellung vorgegeben wird. Bei dieser Variante steht entweder die Kreativität und Experimentierfreude im Zentrum, wenn Lernende selbstständig damit arbeiten oder aber die Lehrperson gibt Aufgabenstellungen vor. In beiden Fällen sind damit deutlich vielfältiger Einsatzszenarien als bei fixen Vorgabeaufgaben denkbar.

4 Zusammenfassung

Mit Hilfe der aufgestellten Kriterien lassen sich bisherige und zukünftige Lernumgebungen für den Einstieg ins Programmieren klassifizieren und vergleichen. Wir sehen diese Kriterien in erster Linie als Hilfestellung für Informatikdidaktiker in der Aus- und Weiterbildung sowie Beratung. Die Kriterien helfen zwar, konkrete Programmierlernumgebungen rascher einordnen und einschätzen zu können, erfordern aber trotzdem noch einiges an informatikdidaktischem Wissen, welches Lehrkräften insbesondere in unteren Schulstufen meist fehlt. Für diese Zielgruppe müssten aufrund von konkreten lokalen Bedingungen wie Schulstufe, Wissensstand der Lehrkräfte und der Lernenden, vorhandener Infrastruktur etc. mit Hilfe der vorgeschlagenen Kriterien Empfehlungen abgeleitet werden.

Hinweis: Alle in dieser Arbeit genannten Lernumgebungen und viele weitere sind auf der Webseite http://programmingwiki.de/Lernumgebungen exemplarisch anhand der Kriterien eingeordnet und mit Empfehlungen für die Schulstufe ergänzt.

Literaturverzeichnis

[BCH+98] Peter Brusilovsky, Eduardo Calabrese, Jozef Hvorecky, Anatoly Kouchnirenko und Philip Miller. Mini-languages: A Way to Learn Programming Principles. *Education and Information Technologies*, 2(1):65–83, Januar 1998.

[EGR+08] Markus Esch, Patrick Gratz, Steffen Rothkugel, Jörg Jakoby, Ingo Scholtes und Peter Sturm. CrePes - Warum und wie Schüler ab 8 Jahren Programmieren lernen sollten. In *DeLFI 2008: Die 6. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V., 07. - 10. September 2008 in Lübeck, Germany*, Seiten 413–424, 2008.

[HW09] Michael Hielscher und Christian Wagenknecht. Programming–Wiki: Online programmieren und kommentieren. *Zukunft braucht Herkunft. 13. GI-Fachtagung Informatik und Schule*, INFOS'09:281–292, 2009.

[KP05] Caitlin Kelleher und Randy Pausch. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Comput. Surv.*, 37(2):83–137, Juni 2005.

[Lan06] Claudio Landerer. Die Nintendo-Generation lernt Programmieren – Der Versuch einer didaktischen Rekonstruktion des Programmierens für den Unterricht. *Diplomarbeit*, Universität Salzburg, 2006.

[MC13] R. Mason und G. Cooper. Distractions in Programming Environments. In Angela Carbone und Jacqueline Whalley, Hrsg., *Fifteenth Australasian Computing Education Conference (ACE 2013)*, Jgg. 136 of *CRPIT*, Seiten 23–30, Adelaide, Australia, 2013. ACS.

[Mod11] Eckart Modrow. Visuelle Programmierung - oder: Was lernt man aus Syntaxfehlern? In Marco Thomas, Hrsg., *INFOS*, Jgg. P-189 of *LNI*, Seiten 27–36. GI, 2011.

[Pap80] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA, 1980.

[Sch01] Andreas Schwill. Ab wann kann man mit Kindern Informatik machen? Eine Studie über die informatischen Fähigkeiten von Kindern. In Reinhard Keil-Slawik und Johannes Magenheim, Hrsg., *Informatik und Schule – Informatikunterricht und Medienbildung INFOS 2001 – 9. GI-Fachtagung 17.–20. September 2001, Paderborn*, GI-Edition – Lecture Notes in Informatics – Proceedings, Seiten 13–30, Bonn, September 2001.

[She03] Robert Sheehan. Children's Perception of Computer Programming As an Aid to Designing Programming Environments. In *Proceedings of the 2003 Conference on Interaction Design and Children*, IDC '03, Seiten 75–83, New York, NY, USA, 2003. ACM.