Ira Diethelm (Hrsg.): Informatische Bildung zum Verstehen und Gestalten der digitalen Welt,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 15

Analogien für Programmierkonzepte: Ein Weg zum Computational Thinking

Barbara Wieczorek[1], Liz Ribe[2], Christina B. Class[3] und Michael Brinkmeier[4]

Abstract: Im Rahmen einer Informatikausbildung ist das Erlernen einer Programmiersprache für Schüler und Studenten ein wichtiger Zugang, um Computational Thinking zu erlernen. Jedoch stellen die Grundlagen der Programmierung häufig eine schwer überwindbare Hürde dar, die Lernende vom eigentlichen Ziel abhält, Programmierkonzepte als Problemlöseoperatoren einzusetzen. In diesem Beitrag wird untersucht, inwieweit Analogien als Brücke zwischen dem Vorwissen des Lernenden und Programmierkonzepten dienen können. Es wird ein anschauliches Modell zur Verdeutlichung von Beziehungen und Wechselwirkungen von Lerner, Analogien und Fachkonzepten vorgestellt. In einem Workshop werden in Arbeitsgruppen Praxisbeispiele von Analogien für konkrete Programmierkonzepte zusammengestellt und deren Vor- und Nachteile sowie Grenzen und resultierende Herausforderungen diskutiert.

Keywords: Computational Thinking, Programmierung, Informatikunterricht, Analogien, Problemlösen, duale Kodierung

1 Einleitung

Computational Thinking wurde bereits in den 1950er und 60er Jahren als Teil der Informatik, unter dem Begriff *algorithmic thinking* (Algorithmisches Denken) gesehen [De09]. 2006 forderte Jeannette Wing, dass Computational Thinking neben Lesen, Schreiben und Rechnen als grundlegende Fähigkeit erlernt werden soll [Wi06]. Hierunter versteht sie eine Reihe von mentalen Werkzeugen, um Probleme zu lösen, Systeme zu entwerfen und menschliches Verhalten zu verstehen [Wi06]. Grundlagen des Computational Thinking sollen daher in die Curricula der Schule integriert werden [BS11]. Hierzu wurde eine Matrix mit Konzepten und verschiedenen Schulfächern entwickelt, in denen diese Konzepte angewandt und trainiert werden. Auch wenn Aspekte des Computational Thinking auf verschiedenste Wege gelehrt werden können, ist es zutiefst mit Programmieraktivitäten verbunden.

Das Erlernen einer Programmiersprache kann die Ausbildung von Computational-

[1] Ernst-Abbe-Hochschule Jena, Grundlagenwissenschaften, Carl-Zeiss-Promenade 2, 07745, Jena, Barbara.Wieczorek@eah-jena.de
[2] Ernst-Abbe-Hochschule Jena, Grundlagenwissenschaften, Carl-Zeiss-Promenade 2, 07745, Jena, Elizabeth.Ribe-Baumann@eah-jena.de
[3] Ernst-Abbe-Hochschule Jena, Grundlagenwissenschaften, Carl-Zeiss-Promenade 2, 07745, Jena, Christina.Class@eah-jena.de
[4] Universität Osnabrück, Institut für Informatik, Albrechtstr. 28, 49076, Osnabrück, Michael.Brinkmeier@uni-osnabrueck.de

Thinking-Fähigkeiten unterstützen, indem das automatisierte Lösen von Problemen geschult wird [Wo17]. Die Einführung in eine Programmiersprache findet häufig bereits zu Beginn einer Informatikausbildung statt.

Um genauer beleuchten zu können, wie Fachkonzepte der Programmierung so erschlossen werden können, dass sie letztendlich zur Lösung von Problemen eingesetzt werden können, betrachten wir die kognitive Psychologie des Lösens von Problemen. In diesem Kontext ist unser Ziel die Verankerung von Fachkonzepten in Wissensstrukturen, sodass sie als Operatoren im Sinne der Problemlösetheorie eingesetzt werden können. Hierbei gehen wir von folgender Begriffsbildung aus [Sp03]:

- *Problem als Zustand:* Ein Problem kann aufgefasst werden als ein Ausgangszustand. Die Lösung des Problems entspricht einem Zielzustand.
- *Lösungsprozess als Transformation:* Um den Ausgangszustand in den Zielzustand zu transformieren, müssen Problemlöseoperatoren angewendet werden. Eine Transformation besteht aus einer konkreten Reihenfolge von Operationen unter Einsatz geeigneter Operatoren und kann mehrere Übergangszustände beinhalten.

Wir können diese Begriffe auf die Programmierung übertragen: Ein zu lösendes Ausgangsproblem kann durch eine geeignete Repräsentation [An13] als Ausgangs*zustand* eines Programmes betrachtet werden. Die Durchführung von Operationen, die durch das Programm festgelegt sind, stellt den *Lösungsprozess* dar. Der Zustand des Programms wird während seiner Durchführung transformiert, bis ein definierter Endzustand erreicht ist. Der Endzustand ist eine Repräsentation der Lösung des Problems. Um geeignete Operationen für den Lösungsprozess zu finden, müssen entsprechende Fachkonzepte aus der Programmierung als Problemlöseoperatoren eingesetzt werden.

Ein zentrales Problem in der Ausbildung von Programmieranfängern ist also der Erwerb von entsprechenden Problemlöseoperatoren. Nach der Theorie des Problemlösens gemäß Anderson [An13] erfolgt dieser Erwerb durch Entdecken, direkte Instruktion oder den Einsatz von Analogien. Im Abschnitt 2 werden Probleme diskutiert, die bei der allgemeinen Vermittlung von Programmierkonzepten auftreten. Danach richten wir den Fokus auf den Einsatz von Analogien, denen eine möglichst große Bildhaftigkeit innewohnt. Im Abschnitt 3 werden unsere Erwartungen an bildhafte Analogien und die Voraussetzungen für deren Verwendung erläutert. Anschließend werden im Abschnitt 4 spezifische Aspekte, die bei der Verwendung von bildhaften Analogien auftreten können, anhand von Beispielen geschildert.

Die Inhalte dieses Beitrags sollen in einem Workshop vertieft werden. Dazu werden in Arbeitsgruppen Praxisbeispiele von Analogien für konkrete Programmierkonzepte zusammengestellt. Deren Vor- und Nachteile sowie Grenzen und resultierende Herausforderungen werden anschließend diskutiert.

2 Herausforderungen bei der Vermittlung von Programmierkonzepten

Trotz der Tatsache, dass einige Lernende auch ohne Vorkenntnisse zum Bereich der Programmierung einen leichten Zugang finden und die vermittelten Konzepten schnell einsetzen können, stellen Inhalte, die bereits am Anfang der Programmierausbildung stehen (z.B. Variablen, Datentypen, Verzweigung, Iteration), für einen erheblichen Teil der Studierenden oft schwer überwindbare Hürden dar.

Diese Beobachtung wird in Abbildung 1 illustriert durch die Auswertung von Prüfungen des ersten Semesters im Bereich Informatik von Studierenden einer Ingenieurwissenschaft. Während man möglicherweise andere Verteilungen der Noten erwarten würde, tritt ein in der landläufigen Meinung von Lehrenden als typisch angesehenes Phänomen auf: Es gibt eine Häufung von guten und sehr guten ebenso wie von schlechten und sehr schlechten Noten bei vergleichsweise wenigen mittelmäßigen Noten. Nach Meinung der Autoren deutet diese Notenverteilung auf eine große Anzahl an Studierenden hin, die keinerlei oder fast keinen Zugang zu den entsprechenden Konzepten finden konnten. Insbesondere gibt es recht wenige Studierende, deren Leistungen im Mittelfeld liegen und die recht passabel mit dem Stoff und damit den Konzepten zurechtkommen.

Wir führen dies darauf zurück, dass Fachkonzepten aus dem Bereich der Programmierung oft ein hoher Grad an Abstraktion innewohnt. Um Konzepte erfolgreich einsetzen zu können, müssen in hohem Ausmaß formale Denkoperationen durchgeführt werden. Bereits der Entwurf einfacher Programme geht oft mit einem Prozess der Hypothesenbildung einher, zum Beispiel um Nutzereingaben, die vor Ausführung des Programmes unbekannt sind, zu verarbeiten oder Iterationen, deren Häufigkeit vom Zustand des Programmes abhängt, zu realisieren. Auch die Verwendung von

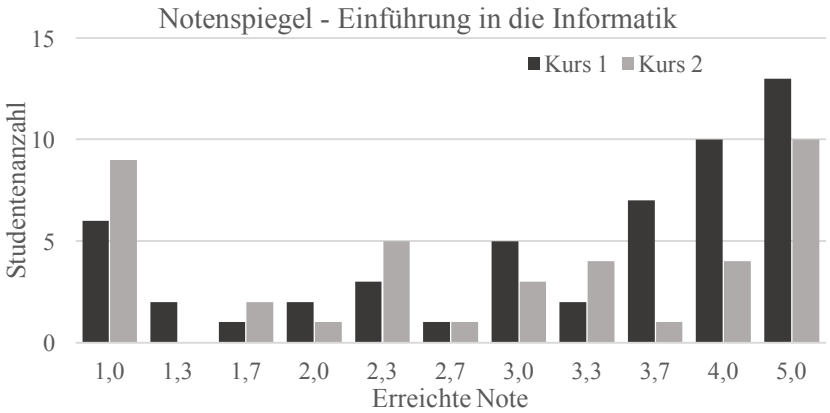

Abbildung 1: Notenspiegel von zwei Informatik-Prüfungen im ersten Studiensemester

Hilfsvariablen, zum Beispiel für die Bildung einer Summe über die Elemente einer Liste, geht über das konkret-operative Denken hinaus, auf welches der Lernende hinsichtlich des Problemlöseprozesses zurückgreifen kann.

Hinsichtlich der Hilfestellung für Lernende mit Schwierigkeiten haben die Autoren wenig Erfolg mit schrittweisem, ausführlichem Erläutern von einfachen Quelltextbeispielen beobachtet. Wesentlich bessere Erfahrungen wurden hingegen mit bildhaften Vergleichen gemacht, um betroffene Konzepte anschaulich zu machen. Bei der Veranschaulichung wird entweder Bezug auf Objekte aus der Lebenswelt der Lernenden genommen oder auf Prozesse, die aufgrund der Erfahrung der Lernenden bekannt oder leicht zugänglich sind. Somit werden Verstehensprozesse hinsichtlich deklarativen oder prozeduralen Wissens unterstützt.

3 Erwartungen und Voraussetzungen an die Verwendung bildhafter Analogien

Schwierigkeiten beim Zugang zu Fachkonzepten sollen also durch den Einsatz von bildhaften Analogien gemindert werden. Die Nutzung von bildhaften Analogien hat folgende Vorteile:

- Ist eine Analogie ausreichend bildhaft und aus dem Erfahrungsbereich des Lernenden gegriffen, so kann eine Einbindung an die Wissensstrukturen des Lernenden erleichtert werden. Dies stellt wichtige Dimensionen im verständnisintensiven Lernen dar [ViL17]. Betrachten wir das Vorwissen des Lerners und das zu vermittelnde Fachwissen als Objekte, so kann mit einer bildhaften Analogie eine Verknüpfung zwischen Vorwissen und Fachwissen wie in Abbildung 2 hergestellt werden.

- Die Anknüpfung an vorhandenes Wissen kann beim Lernenden bereits zu einer positiven Selbsteinschätzung hinsichtlich des Lernprozesses führen und damit zu einer Erhöhung der Lernmotivation [DR13].

- Im fortgeschrittenen Lernprozess, nachdem das Fachwissen und das Verständnis für die Analogie sich gefestigt haben, kann eine duale Kodierung des Wissens erreicht werden: Das erworbene verbale Wissen wird durch die nonverbale Bildhaftigkeit der Analogie ergänzt. Es ist anzunehmen, dass dual kodiertes Wissen besser abgerufen werden kann [Ma01] und dadurch ebenso besser auf andere Situation übertragen und somit als Problemoperator eingesetzt werden kann.

- In umgekehrter Richtung kann ein Fachkonzept Einzug in alltägliche Situationen

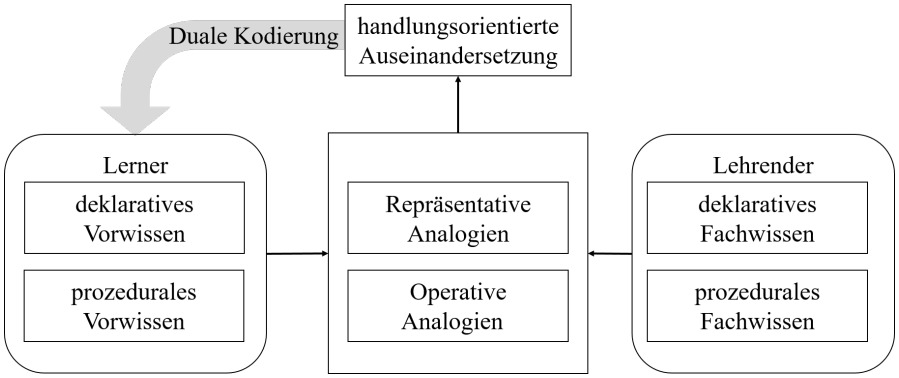

Abbildung 2: Vorwissen und Fachwissen mit Analogien zusammenführen und mithilfe handlungsorientierten Auseinandersetzungen zum dual-kodierten Wissen des Lerners führen.

> erhalten, wenn ein Objekt als Analogon verwendet wird, welchem ein Lernender häufig begegnet. Somit kann eine Veränderung der Betrachtungsweise alltäglicher Gegenstände und Prozeduren hinsichtlich des Computational Thinking möglich sein. Wird z.B. ein Snackautomat als Analogon für eine Funktion verwendet (siehe Abschnitt 4.4), erinnert sich der Lernende ggf. bei künftigen Snackautomaten-Einkäufe an den Funktionsbegriff.

Analogien werden in der Lehre und in Lehrbüchern schon lange eingesetzt, und bereits im letzten Jahrtausend gab es umfangreiche Untersuchungen zu deren Einsatz in den Naturwissenschaften. In einer Literaturstudie stelle Duit die Ergebnisse verschiedenster Untersuchungen zusammen und wies auf diverse Probleme hin [Du91]. Bereits 1982 wurde vor dem unbedachten Einsatz von Analogien in der Informatikbildung gewarnt, da diese das Lernen erschweren könnten [HM82].

Voraussetzung für den Einsatz einer Analogie ist daher eine hohe Passgenauigkeit. Hierzu müssen die Elemente eines Fachkonzeptes, die durch den Einsatz einer Analogie abgedeckt werden, identifiziert werden. Um die erfolgreiche Verknüpfung der Analogie zum Fachkonzept herzustellen, ist laut Anderson sicherzustellen, dass bei „jedem Akt des Analogiebildens“ die „Elemente von der Quelle auf das Ziel […] übertragen“ werden [An13]. Eine bloße Nennung eines Bildes im Umfeld der Vorstellung eines Fachkonzeptes ist demzufolge nicht ausreichend. Studien haben gezeigt, dass oft eine Anleitung notwendig ist, um die Analogien richtig in die Denkprozesse zu integrieren [Du91]. Auch ist wichtig, auf die Unterschiede zwischen Analogie und der Quelle hinzuweisen, um deutlich zu machen, wo die Gemeinsamkeiten nicht helfen. Hier kann es hilfreich sein, verschiedene Analogien für die einzelnen Aspekte zu verwenden [Du91].

Das Verständnis von Konzepten ist im Bereich der Programmierung oftmals nicht ausreichend. Weinberg [We98] stellte in seiner Beobachtung von Lernen von Programmiersprachen fest, dass ein Wissenserwerb über Fachkonzepte möglicherweise nicht stattfinden kann ohne handlungsorientierte Auseinandersetzung mit dem

entsprechenden System.

Entscheidend ist also die Verzahnung der Wissensvermittlung durch den Lehrenden und des eigenständigen Handelns durch den Lernenden. Hierbei kann die Verwendung von Analogien zum Einstieg in ein Programmierkonzept hilfreich sein, ein handelnder Bezug zum Programmieren muss dennoch hergestellt werden. Für den Wissenserwerb soll ein Lernender jedes neue Konzept aktiv anwenden und entsprechende Programme am Rechner implementieren. Der Bezug zur Analogie kann während dieses Prozesses immer wieder unterstützend hergestellt werden. Dies ist jedoch nur so lange sinnvoll, wie die relevanten Gemeinsamkeiten der Analogien zum Tragen kommen und Lernende nicht verwirrt werden. Aus diesem Grund kommen Auswahl und zielführendem Einsatz von Analogien eine große Bedeutung zu. Hierbei gilt es sowohl die Frage nach der Wahl der Analogien zu betrachten, als auch zu entscheiden, wann von der Verwendung der Analogie abgesehen werden und der Lernende auf relevante Unterschiede hingewiesen werden soll.

4 Probleme und Fragen hinsichtlich des Einsatzes von bildhaften Analogien

Um zu verdeutlichen, wie ein Programm einen Problemlöseprozess im Sinne von Anderson darstellt, soll das folgende Beispiel betrachtet werden. Es soll für eine Menge von Zahlen in einem Programm das Minimum dieser Zahlen bestimmt werden. Eine Automatisierung des Lösungsprozesses in der Programmiersprache Python könnte folgendermaßen aussehen:

```
liste = [2,4,6,3,5]
minimum = liste[0]
zaehler = 1
while zaehler<len(liste):
    if minimum > liste[zaehler]:
          minimum = liste[zaehler]
    zaehler = zaehler + 1
print(minimum)
```

Die Repräsentation des Problems beinhaltet die Variablen *liste*, *zaehler* und *minimum* mit ihren anfänglich definierten Werten. Der Zielzustand ist erreicht, wenn die Variable *minimum* das tatsächliche Minimum der Zahlen bezeichnet. Um durch geeignete Operationen zum Zielzustand zu gelangen, wird eine Reihe von Transformationen angewendet:

- *zaehler* wird in jeder Iteration inkrementiert.
- Ein Vergleich von aktuellem Listenelement und aktuellem Minimum findet statt, falls nötig, wird *minimum* angepasst.

- Ein Abbruchkriterium wird überprüft.

Notwendige Konzepte für die Operationen sind hierbei die Wertzuweisung, die Auswertung logischer Ausdrücke sowie die Iteration.

4.1 Funktionale Fixierungen

Unter funktionaler Fixierung wird verstanden, dass für einen Problemlöseoperator nicht der volle Umfang seiner Möglichkeiten erkannt wird, sondern er nur in ausgewählten (bekannten) Kontexten eingesetzt wird. Der von Anderson [An12] beschriebene Effekt der funktionalen Fixierung im Zusammenhang mit Problemlöseoperatoren hat womöglich auch Bedeutung für die Verwendung von bildhaften Analogien.

Während Programmierkonzepte allgemeingültigen Charakter haben, sind bildhafte Analogien häufig in einer konkreten Situation verankert. Für jede bildhafte Analogie muss untersucht werden, inwieweit die Gefahr der Einschränkung des Operators im Sinne einer funktionalen Fixierung durch fehlende allgemeingültigen Charakter der Analogie besteht.

Mit der Vielfalt an Ausprägungen von Programmierkonzepten stellen sich daher beim didaktischen Einsatz von Analogien immer die Fragen:

- Wie lange kann diese als übergangsweise Veranschaulichung dienen, um später zu einer Abgrenzung zu gelangen?
- Wie sehr ist eine Vereinfachung eines entsprechenden Programmierkonzeptes möglich und wie wird diese später korrigiert?

4.2 Deklarative Analogie: Schachtel als Analogie für Variablenkonzept

Unter einer *deklarativen Analogie* verstehen wir eine Analogie, welche Verstehensprozesse unterstützt, die auf deklaratives Wissen bezogen sind. Häufig wird z.B. das Bild eines beschrifteten Gefäßes (oder Schachtel) verwendet, um das Konzept einer Variablen einzuführen. In obigem Bespiel könnte man mit einer Schachtel darstellen, welche Rolle *minimum* im Algorithmus hat.

Aber auch wenn hierfür die von Anderson geforderten Passgenauigkeit einer Analogie erfüllt ist, repräsentiert das Bild der Schachtel das Konzept der Variable als Ganzes noch nicht völlig korrekt:

- In der Realität kann eine Schachtel leer sein. Dies trifft auf Variable nicht zu, da sie als Speicherbereich immer einen, wenn auch zufälligen, Wert enthalten. In Sprachen wie C, welche das Lesen einer Variable ohne vorherige Initialisierung ermöglichen, ist dies ein relevanter Unterschied. Dies kann also zu einer falschen Vorstellung einer Variablenzuweisung führen.

- In der Realität kann eine Schachtel durchaus mehrere Elemente enthalten. Eine Zuweisung eines Wertes an eine Variable führt aber dazu, dass der vorherige Wert unwiderruflich überschrieben wird.
- In der Realität hat eine Schachtel eine gegebene Größe. In C/C++ wird zwar durch Deklaration einer Variablen ein bestimmter Speicherbereich bezeichnet, in welchem Daten des deklarierten Typs gespeichert sind. In Python kann das Bild der Schachtel aber zu einer fehlerhaften funktionalen Fixierung führen, da die gleiche Variable im Laufe der Zeit Werte unterschiedlichen Typs und unterschiedlicher Größe speichern kann. Eine Variable in Python stellt lediglich den Namen eines Objektes im Speicher dar - durch Zuweisung eines anderen Wertes bezeichnet die Variable meist ein anderes Objekt im Speicher.

Hat sich die Vorstellung einer Schachtel als Analogon zu einer Variable manifestiert, können Probleme auftreten, wenn weitere Konzepte eingeführt werden, z.B. statische und dynamische Variablen oder Listen. Diese sind mit dem Schachtelmodell nicht unmittelbar zugänglich und bei Einführung dieser muss Abstand von der bisher verwendeten Analogie genommen werden oder weitere Analogien hinzugefügt werden, um die anderen Eigenschaften aufzuzeigen. Solche multiplen Analogien können notwendig sein, um einen Konzept umfassend zu erlernen [Du91].

4.3 Deklarative Analogie: Snackautomat als Analogie für Funktionskonzept

Die Kapselung von Quelltext in Funktionen ist ein wichtiges Konzept in der Programmierung. Funktionen stellen bereits ein relativ hohes Abstraktionsniveau dar und bereiten Lernenden häufig Schwierigkeiten. Ein grundlegendes Problem ist die Trennung von *Definition* und *Ausführung* von Funktionen. Lernenden fällt es oft schwer zu verstehen, dass Funktionen erst durch einen Funktionsaufruf ausgeführt werden und Parameter erst durch den Funktionsaufruf konkrete Werte erhalten.

Als bildhafte Analogie für eine Funktion mit Rückgabewert, unter Verwendung zweier Parameter, kann ein Snackautomat dienen. Ein Funktionsparameter steht für den Geldbetrag, der in den Automaten eingeworfen wird, ein weiterer für die Nummer, die zur Auswahl des Produktes über eine Tastatur eingegeben wird. Während der Funktionsabarbeitung werden verschiedene Operationen von dem Snackautomaten durchgeführt und als Rückgabewert erhält man das gewünschte Produkt. Das Ausgabefach kann als Analogie für das *return*-Statement in der Funktionsdefinition betrachtet werden.

Das Beispiel knüpft an die Vorstellungswelt der Lernenden an und ist hilfreich, um das Konzept von Funktion, Funktionsaufruf und Aufrufer zu verdeutlichen. Dennoch kann auch hier eine funktionale Fixierung erfolgen, wenn der Lernende zu einer der folgenden Schlussfolgerungen gelangt:

- Eine Funktion hat immer zwei Parameter.
- Eine Funktion muss immer einen Rückgabewert liefern.
- Der Rückgabewert einer Funktion muss immer von allen Parametern abhängen.

Um eine Manifestation dieser Fixierung zu verhindern, müssen gezielt Verallgemeinerungen anhand geeigneter Programmieraufgaben stattfinden, z.B. von:

- Einer Funktion mit nur einem Argument.
- Einer Funktion ohne Parameter.
- Einer Funktion ohne Rückgabewert.
- Einer Funktion, deren Rückgabewert nur von einer Teilmenge ihrer Parameter abhängt.

4.4 Operative Analogie: Finden der kürzesten Spielzeit aus einer CD-Regal als Analogie für Bestimmung des Minimums

Unter einer *operativen Analogie* verstehen wir eine Analogie, welche Verstehensprozesse unterstützt, die auf prozedurales Wissen bezogen sind. Das Abarbeiten eines Rezepts kann z.B. als operative Analogie für das Abarbeiten eines Algorithmus dienen.

Mit Hilfe einer operativen Analogie kann der Prozess zur Entwicklung des obigen Algorithmus zur Minimumbestimmung unterstützt werden. In diesem Zusammenhang wird folgende analoge Aufgabe gestellt: Die kürzeste Spieldauer von CDs in einem Regal soll gefunden werden. Durch eigene Überlegungen kommen Lernende schnell auf die Idee, dass es sinnvoll ist, alle CDs der Reihe nach zu betrachten und sich die bisherige kürzeste Spieldauer zu merken. Wenn das Problem des Minimums mehrerer Zahlen gestellt wird, muss der Lernende Gemeinsamkeiten mit der Analogie identifizieren, um einen Transfer des Lösungsprozesses vorzunehmen: Das Regal ist eine Liste; die CDs sind die Zahlen; die gemerkte kürzeste Spielzeit ist eine Variable. Durch die Analogie werden die Lernenden dabei unterstütz, eigene Algorithmen mithilfe ihrem prozeduralen Vorwissen zu entwickeln.

5 Zusammenfassung

Die Nutzung von bildhaften Analogien für Fachkonzepte, die im Rahmen der Programmierausbildung vermittelt werden sollen, dient als Hilfsmittel, um anfängliche Verständnisschwierigkeiten zu überwinden. Wichtig hierbei ist eine wiederholte Auseinandersetzung mit der Analogie während der Bearbeitung von Programmieraufgaben, welche im Zusammenhang mit dem zu lernenden Fachkonzept

stehen, um Fehlvorstellungen durch mangelnde Passgenauigkeit und Effekte funktionaler Fixierung zu vermeiden. Analogien sollen helfen, Programmierkonzepte besser in Wissensstrukturen zu verankern, um Problemlösefähigkeiten in Bezug auf Programmierung zu verbessern. Ebenso sollen Analogien es ermöglichen, vermehrt alltägliche Situationen aus dem Blickwinkel des Computational Thinking zu betrachten.

Literaturverzeichnis

[An13] Anderson, J.R.: Kognitive Psychologie. VS Verlag für Sozialwissenschaften, S. 165, 249, 2013.

[BS11] Barr, V; Stephenson, C.: Bringing Computational Thinking to K-12: What is Involved and What ist he Role of the Computer Science Education Commmunity?. acm Inroads, 2/1, S. 48, 2011.

[De09] Denning, P.J.: The Profession of IT Beyond Computational Thinging, Communcations of the ACM, 52/6, S. 28-30, 2009.

[DR93] Deci, E.L.; Ryan, R.M.: Die Selbstbestimmungstheorie der Motivation und ihre Bedeutung fuer die Paedagogik. Zeitschrift für Pädagogik, 39/2, S. 223-238, 1993.

[Du91] Duit, R.: On the Role of Analogies and Metaphors in Learning Science, Science Education, 75/6, S. 649-672, 1991.

[HM82] Halasz, F.; Moran, T.: Analogy Considered Harmful, CHI '82 Proceedings of the 1982 Conference on Human Factors in Computing Systems, S. 383-386, 1982.

[Ma01] Martschinke, S.: Aufbau mentaler Modelle durch bildliche Darstellungen. Waxmann Verlag, Münster [u.a.], S. 61, 2001.

[Sp03] Spering, M.: "Praktisches Problemlösen", 2003, http://www.psychologie.uni-heidelberg.de/ae/allg/mitarb/ms/PH_Einfuehrung.pdf, zuletzt besucht am 9.6.2017.

[ViL17] Verein für Verständnisintensives Lernen e.V., http://www.verstehenlernen.de/?page_id=672, zuletzt besucht am 6.6.2017.

[We98] Weinberg, G.M.: The Psychology of Computer Programming. Silver Anniversary ed, Dorset House Publ., 1998.

[Wi06] Wing, J.: Computational Thinking, Communications of the ACM, 49/3, S. 33-35, 2006.

[Wo17] Stephan Wolfram Blog, http://blog.stephenwolfram.com/2016/09/how-to-teach-computational-thinking/, zuletzt besucht am 9.3.2017.

[YSH17] Yadav, A; Stephenson C.; Hong, H.: Computational Thinking for Teacher Education, Communications of the ACM, 60/4, S. 55-62, 2017.