SCHREITER, Saskia & DENNHARD, Jens
Schwäbisch Gmünd, Heidelberg

Integration von Computational Thinking im Mathematikunterricht zu arithmetischen Zahlenfolgen

Computational Thinking (CT) beschreibt die Denkprozesse, die notwendig sind, um Probleme so zu analysieren und Lösungen zu gestalten, dass sie mit digitalen Geräten umgesetzt werden können (Shute et al., 2017). Die Bedeutung von CT zeigt sich in der wachsenden Nachfrage nach digital kompetenten Menschen, die in der Lage sind, Technologie kritisch zu nutzen und weiterzuentwickeln (Wing, 2006). Parallel dazu steigt das Interesse, CT bereits früh in der Schulbildung zu fördern und in den Lehrplänen zu verankern, besonders im MINT-Bereich, wie z.B. in der Mathematik (Lv et al., 2023). Im Projekt CoM-MIT (Coden im Mathematikunterricht – Mathematik Informatik Transfer) werden fächerverbindende Lernumgebungen zum Programmieren im Mathematikunterricht der Klassenstufen 3/4 und 5/6 für Anfänger*innen auf Seiten der Lehrenden und Lernenden entwickelt. Dieser Beitrag gibt Einblicke in eine entwickelte Lernumgebung zu arithmetischen Zahlenfolgen und präsentiert Ergebnisse aus empirischen Erprobungen mit Viertklässlern. Hierbei wurde qualitativ untersucht, welche CT-Aspekte und -Praktiken beobachtet werden können, während die Schüler*innen Codes für arithmetische Folgen in einer Unterrichtseinheit analysierten und erstellten.

Zentrale Computational Thinking Aspekte in der K-12 Bildung

CT umfasst kognitive Strategien, die es erlauben, Probleme effektiv und effizient zu lösen. In der Bildungsforschung werden verschiedene CT-Aspekte als wichtig für die K-12-Bildung hervorgehoben. Der vorliegende Beitrag orientiert sich inhaltlich an den fünf CT-Aspekten nach Shute et al. (2017):

- Abstraktion: Die Komplexität eines Problems auf die wesentlichen Merkmale reduzieren, irrelevante Details werden ignoriert oder entfernt.
- Umgang mit Algorithmen: Beinhaltet ein schrittweises Verfahren zur Lösung eines Problems. Dies umfasst die Analyse eines Algorithmus (d. h. die schrittweise gedankliche Ausführung der einzelnen Schritte) sowie die Erstellung eines Algorithmus (d. h. ein eigenes schrittweises Verfahren designen), um ein gewünschtes Ergebnis zu erzielen.
- Debugging: Fehler im Algorithmus erkennen, analysieren und beheben.
- Generalisierung: Verallgemeinerung von Mustern und Lösungen aus spezifischen Problemen auf ein breiteres Spektrum von ähnlichen Problemen.
- Dekomposition: Zerlegen komplexer Probleme in kleinere, besser handhabbare Teile, um sie leichter lösen zu können.

In: L. Schick, M. Platz & A. Lambert (Hrsg.),
Beiträge zum Mathematikunterricht 2025.
58. Jahrestagung der Gesellschaft für Didaktik der Mathematik. WTM.
https://doi.org/10.37626/GA9783959873307.0

Integration von Computational Thinking im Mathematikunterricht

Die enge Verbindung zwischen CT und mathematischem Denken bietet zahlreiche Möglichkeiten, CT im Mathematikunterricht zu fördern. Beide Ansätze teilen eine zentrale Ausrichtung auf Problemlösung (Wing, 2006) sowie gemeinsame Konzepte wie Modellierung, Datenanalyse, Statistik und Wahrscheinlichkeitsrechnung (Sneider et al., 2014).

Aktuelle Forschungsergebnisse zeigen, dass CT-Aspekte integriert im Mathematikunterricht gefördert werden können (z.B., Büscher, 2024). Hierbei ist jedoch hervorzuheben, dass in bisherigen Studien nicht alle mathematischen Inhaltsbereiche gleichermaßen berücksichtigt wurden; ein Großteil der Forschung konzentriert sich auf Geometrie, während Potenziale in Bereichen wie Arithmetik und Algebra bisher weniger erforscht sind (Barcelos et al., 2018; Lv et al., 2023). Umgekehrt weisen Studien auch darauf hin, dass über die gezielte Integration von CT-Aspekten in den Mathematikunterricht das domänenspezifische mathematische Lernen unterstützt werden kann. Es zeigte sich, dass hierbei nicht nur die Problemlösungsfähigkeit und das mathematische Verständnis der Schüler*innen gestärkt, sondern auch das Interesse an der Mathematik gefördert werden kann (z.B., Sung & Black, 2020).

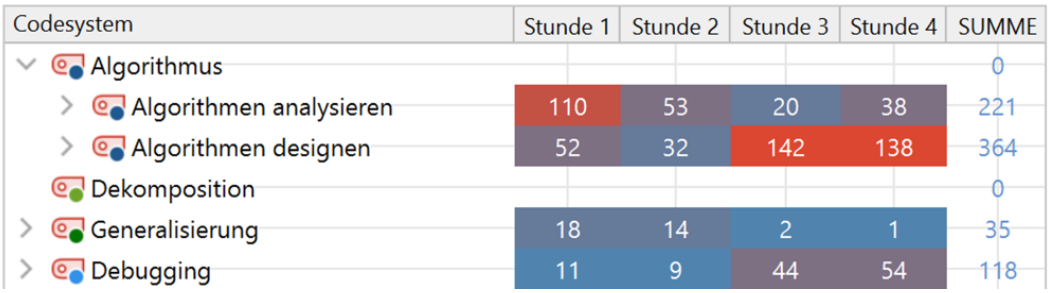
Blockbasierte Programmierumgebungen wie Scratch oder NEPO haben sich als effektive Werkzeuge zur Förderung von CT erwiesen (Lv et al., 2023). Vorteile von visuellen Programmiersprachen liegen in ihrer benutzerfreundliche Drag-and-Drop-Oberfläche, die Anfänger*innen eine sinnvolle und kontextbezogene Lernumgebung bieten (Dennhard & Schreiter, 2024).

Methode

Eine explorative Fallstudie wurde durchgeführt, um zu untersuchen, welche CT-Aspekte und Praktiken im Rahmen einer Einheit im Mathematikunterricht beobachtet werden können. Eingesetzt wurde eine im Projekt CoMMIT entwickelte Lernumgebung zu arithmetischen Zahlenfolgen. Diese führt die Schüler*innen schrittweise von der Analyse blockbasierter Codes zu Zahlenfolgen, über das Vervollständigen dieser, hin zum eigenständigen Erstellen von Codes. Die Stichprobe umfasste 36 Schüler*innen aus zwei vierten Klassen an Grundschulen in Deutschland. Die Unterrichtseinheit umfasste vier Sitzungen à 60 bis 90 Minuten, die in beiden Klassen durchgeführt wurden. Die Schüler*innen arbeiteten mit der Methode des "Pair-Programming" (Iskrenovic-Momcilovi, 2019) immer zu zweit an einem Tablet und nutzten die blockbasierte Programmiersprache NEPO. Während des Unterrichts wurden Audio-, Bildschirmaufnahmen und Schülernotizen für eine anschließende qualitative Datenanalyse gesammelt.

Ergebnisse

Die Ergebnisse zeigten, dass die CT-Aspekte *Umgang mit Algorithmen* (Algorithmen analysieren & designen), *Dekomposition*, *Generalisierung* und *Debugging* in beiden Klassen während jeder Unterrichtsstunde mit unterschiedlichen Häufigkeiten beobachtet wurden (vgl. Abb. 1). Durch mehrere iterative Analyseschleifen konnten aus dem Datenmaterial induktiv verschiedene Subkategorien zu den jeweiligen Praktiken der einzelnen CT-Aspekte herausgearbeitet werden. Der CT-Aspekt der *Abstraktion* wurde hingegen nicht beobachtet. Dies könnte darauf zurückzuführen sein, dass die Codes zu den Zahlenfolgen aus nur wenigen Bausteinen bestehen (vgl. Abb. 2a), wodurch eine Reduzierung oder Vereinfachung nicht erforderlich war.



Codesystem	Stunde 1	Stunde 2	Stunde 3	Stunde 4	SUMME
Algorithmus					0
Algorithmen analysieren	110	53	20	38	221
Algorithmen designen	52	32	142	138	364
Dekomposition					0
Generalisierung	18	14	2	1	35
Debugging	11	9	44	54	118

Abb. 1: Codesystem

Der folgende Schülerdialog gibt einen exemplarischen Einblick in zwei Subkategorien zu den CT-Aspekten Umgang mit Algorithmen und Debugging. In der Aufgabe sollten die Schüler*innen zur Zahlenfolge 20, 18, 16, …, 2, 0 einen Code mit vorgegebenen Blöcken vervollständigen (vgl. Abb. 2b):

[Max setzt den Matheblock mit der Null in die Lücke]

Max [startet die Simulation]: „20… 20… 20… mal 0. Natürlich kommt da immer 20, weil du 20 mal 0 [richtig: 20 minus 0] machst."

Lia: „Okay, ich glaub mal 1".

[Lia tauscht die Blöcke mit Null und Eins. Sie startet die Simulation]

Max: „So, und jetzt bitte… [Es laufen die Zahlen 20, 19, 18, 17 über das Display]

Lia: „Hä, da muss doch minus 2."

Max: „Also wir müssen mal 2 meinst du?

Lia: „Ja mein ich doch." [Die Schüler*innen schieben den Matheblock mit der 2 in die Lücke und starten die Simulation]

Max: „Schau, wir haben's richtig."

In diesem Beispiel wurde der CT-Aspekt Algorithmus mit der Subkategorie "Algorithmus designen" vergeben: Die Kinder entwickeln ein schrittweises Verfahren (hier: indem sie einen lückenhaften Code vervollständigen), um

ein gewünschtes Ergebnis zu erzielen (hier: eine arithmetische Zahlenfolge mit Startwert 20, Endwert 0 und einer Differenz von 2).

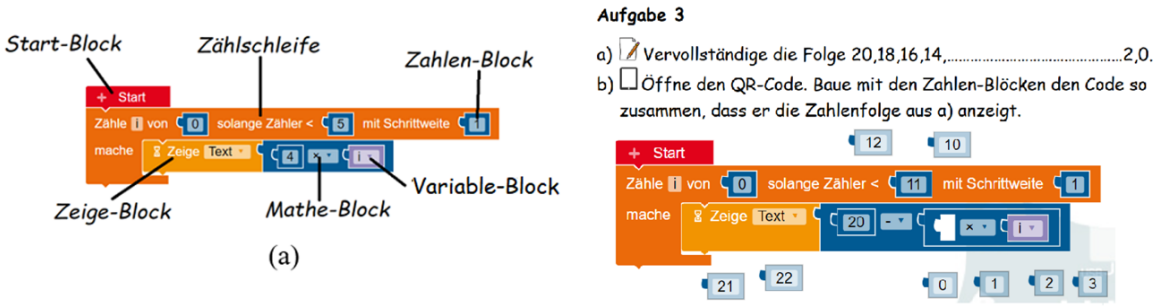


Abb. 2: Aufbau eines Codes (a) und beispielhafte Aufgabe aus der Lernumgebung (b)

Weiterhin konnte der CT-Aspekt Debugging beobachtet werden, mit den Subkategorien "Fehler identifizieren" und "Fehler beheben". Auf diese Weise konnten weitere Subkategorien herausgestellt und die in der Unterrichtseinheit beobachteten CT-Aspekte konkretisiert werden. Diese und weitere Ergebnisse werden auf der Konferenz präsentiert und diskutiert.

Literatur

Barcelos, T. S., Muñoz-Soto, R., Villarroel, R., Merino, E., & Silveira, I. F. (2018). Mathematics learning through computational thinking activities: a systematic literature review. *The Journal of Universal Computer Science*, *24*(7), 815–845.

Büscher, C. (2024). Differences in Students' Computational Thinking Activities when Designing an Algorithm for Drawing Plane Figures. *International Journal of Science and Mathematics Education.* https://doi.org/10.1007/s10763-024-10465-3

Dennhard, J., & Schreiter, S. (2024). Den Bogen spannen: Programmieren und Mathematik im Kontext von endlichen Zahlenfolgen. *MNU Journal*, *77*(5), S. 383 – 389.

Iskrenovic-Momcilovic, O. (2019). Pair programming with scratch. *Education and Information Technologies*, *24*, 2943–2952.

Lv, L., Zhong, B. & Liu, X. (2023). A literature review on the empirical studies of the integration of mathematics and computational thinking. *Education and Information Technologies*, *7*(28), 1–23.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). *Demystifying computational thinking. Educational Research Review*, *22*, 142–158.

Sneider, C., Stephenson, C., Schafer, B., & Flick, L. (2014). Computational thinking in high school science classrooms: Exploring the science "framework" and "NGSS. *The Science Teacher*, *81*(5), 53–59. https://www.learntechlib.org/p/155904.rom

Sung, W., & Black, J. B. (2020). Factors to consider when designing effective learning: Infusing computational thinking in mathematics to support thinking-doing. *Journal of Research on Technology in Education*, *53*(4), 404–426.

Wing, J. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.