Integrating Programming into the Mathematics Curriculum: Combining Scratch and Geometry in Grades 6 and 7

Klaus-Tycho Foerster
ETH Zurich, Switzerland
foklaus@ethz.ch

ABSTRACT

Understanding, using, and applying algorithms and programming will be everyday necessities in a technological world of tomorrow, allowing participation in a changing society opposed to merely watching ongoing progress. While establishing Computer Science courses everywhere is a noble goal, other school subjects also can profit from teaching these two skills. We focus on Mathematics, where algorithms are inherently important, but often overlooked. Taking geometry as an example, we studied how to integrate programming and algorithms in the current curriculum in grades 6 and 7, and propose further application scenarios. We also perform a long-term evaluation, with our methods showing a significant improvement in the students' performance.

Keywords

Education, programming, geometry

1. INTRODUCTION AND MOTIVATION

Before Computer Science became a subject of its own, it was often Mathematics that brought programming into schools, cf. [38]. Nowadays, programming is however absent at large in Mathematics [22]: Specialized tools such as advanced calculators, spreadsheet software, computer algebra systems (CAS), and dynamic geometry environments have in some sense taken it's place. These are designed for their specific purposes in mind and are easy to use, so why bother with programming when teaching Mathematics?

Furthermore, programming languages can be difficult to learn for pupils, taking away even more of the limited time teachers have to fulfill the set standards. Should programming not thus be restricted to Computer Science classes?

This viewpoint ignores that programming itself has inherent advantages for teaching mathematics, as pointed out by Feurzeig et al. in their seminal article [4]. Among contributing to rigorous thinking, giving key insights to concepts such as variables and functions, and enabling the children to generalize problems, programming can also help with another important issue: Students often have difficulties to talk about mathematical problem solving (especially about the process afterwards) or to gather first own experiences in it. "*Programs are more discussable than traditional mathematical activities: one can talk about their structure, one can talk about their development, their relation to one another, and to the original problem*" [4].

Analogous to a function, a program performs a transformation from an input (argument) to an output (value), cf., e.g., [25]. Thus, tasking a student to write a program can be compared to giving a constructive proof [32]. As exercises asking for proofs are strongly underrepresented in today's classrooms (sometimes as low as just 1% [21]), programming can also be advantageous regarding this aspect.

Additionally, algorithms themselves are a fundamental core competence not just for Computer Science, but especially for Mathematics as well, cf., e.g., [16]. And how to better teach algorithms than to program them? When restricting students to the limited set of instructions given by a programming language, one forces them to formally implement their ideas, instead of just describing their thoughts on a high level – potentially skipping over important aspects or leaving them unclear: "*'telling the machine how to do it'*" engages the child in a cycle of modifying ideas based on the output the computer gives [12].

In practice, there is however one main obstacle regarding the usage of programming in the classroom: *Limited time.* An endless number of new topics and techniques could be included or extended in the classroom, e.g., discrete mathematics [14], with each having numerous learning opportunities. Adding programming into the Mathematics curriculum is also not a new idea, see, e.g., [5] for an "*integrated course in algebra*", or [11] for a more wide-spread approach, with many interesting algorithms waiting to be studied [35].

While a discussion on *what* and *in which depth* should be taught in today's classrooms is interesting for sure, the wheels of bureaucracy turn slowly, and for every person arguing for the removal of an item from the curriculum in favor of a replacement, there are multiple persons opposing the removal – often for good reasons.

We thus propose to integrate programming into the current mathematics curriculum, allowing the many advantages of programming to be experienced *today*, and not in a distant future.

We see programming as an essential technique that should be treated just like any other mathematical tool: To be used when appropriate, not to be applied when other tools are more useful, and not taught for it's own sake.

SIGITE'16, September 28-October 01, 2016, Boston, MA, USA
© 2016 ACM. ISBN 978-1-4503-4452-4/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2978192.2978222

We build our work on [6], extending their approaches and evaluating the long-term impact of combining programming and geometry. We start with Section 2, where we describe the advantages of using Geometry as a topic when teaching algorithms and programming. In Section 3, we discuss why we use the graphical language *Scratch* in the classroom. Then, in Section 4, we present our integrated approach, which we studied in the classroom in the grades 6 and 7. We conclude our paper in Sections 5 and 6, summarizing our study and providing a positive long-term evaluation. Lastly, we give an outlook on further possibilities in Section 7.

2. GEOMETRY AND PROGRAMMING

As Holland pointed out [10], describing a geometrical construction is nothing less than giving an appropriate algorithm for it; furthermore, only describing the construction algorithmically enforces checking the correctness of every single step of the construction. However, it is akin to chalk and cheese [28] between what (*i*) students write when describing a construction, and (*ii*) mathematically correct solutions: Early digital thinking in this area leads to fascinating new possibilities [28], with strong improvements in the students' descriptive skills when the construction is performed in the computer first [36].

According to Schmidt-Thieme [30], the final form of the description of a geometric construction is the algorithm, which then can be translated to a computer language. When constructing step by step, the geometric computer program can be developed iteratively, with errors being attributable to elements of the own code. As in a constructive proof, the students can modularize different parts of the construction, testing them separately, and lastly joining them together.

These are just a few reasons why programming with *Turtle graphics* (e.g., in *LOGO*, cf., [1]) was popular in Mathematics in the 70's to the 90's for school children, and is still used to promote Computer Science in primary schools, e.g., in Switzerland [31].

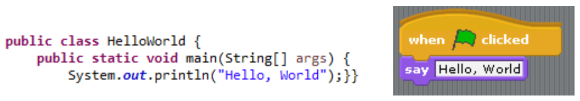
While the educational value of Turtle graphics was and is widely undisputed regarding Computer Science[1], there was a heavy debate on the concrete usage of Turtle graphics in teaching Mathematics. Most of this discussion stemmed from the *Logo Philosophy* [3], which focuses on *constructivism* or *discovery learning*. Some argued that the classical curriculum might vanish, being replaced with an *educational utopia* [2]. On the other hand, e.g., Hromkovic [13] points out that one can directly achieve a high interaction between Turtle graphics and teaching Geometry.

We do not want to take any side in this discussion, but rather reap the best of both worlds: We will use the classic and well proven concept of Turtle graphics, but integrate it into a standard curriculum of Mathematics in the grades 6 and 7, considering it not as a replacement, but as a valuable tool for a deeper understanding of the topic itself and using programming at the same time.

3. THE CHOICE OF SCRATCH

Many standard programming languages come with a large initial overhead, which is of no great concern when teaching (sub-)college level introductory Computer Science courses. For smaller children however, languages such as *C*, *C++*, or

[1]"*The good old idea of Turtle graphics still has an enormous potential*" [23].



```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");}}
```

```
when [flag] clicked
say Hello, World
```

(a) Java **(b) Scratch**

Figure 1: Comparison of a "Hello, World" program in Java and Scratch. The command blocks of Scratch are also available in dozens of different spoken languages from all around the world, e.g., German.

Java can provide starting difficulties, especially when not much time should be spent on the programming language itself, but rather on directly programming algorithms.

Especially not being able to fix "*brackets out of synch*" seems to be a common problem [34]. Furthermore, our main audience are not students who will take Computer Science courses in parallel, but rather those who will not encounter CS courses in their curriculum.

As such, there has been a wide variety of specially designed languages/environments: *LOGO*, *Karel the Robot*, *Etoys and Squeak*, *Lego Mindstorms*, *Alice*, *Kara*, and *Greenfoot*, to list just a few in historical order. However, we would like to use a language capable of performing Turtle graphics, being easy to use right away, and universal enough to apply it also after the grades 6–7 for diverse mathematical algorithms.

The graphical programming language Scratch[2] meets all these criteria, and is even used by universities for introductory courses (e.g., Harvard, Berkeley etc.). [7, 26, 29, 33] By combining blocks in a Lego-like structure, syntax errors are impossible, meaning that the errors can be restricted to the algorithmic thoughts themselves. Even object-oriented programming and complex algorithms are possible, e.g., Dijkstra's algorithm. [19]

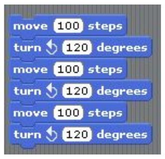
We would like to strongly emphasize that Scratch does not replace or improve all these languages and environments listed before, but just suits our intended purposes better. Furthermore, Scratch has been used before to augment the teaching of Mathematics using extra-curricular activities, cf., e.g., [15, 18, 37]. We are not aware of a long-term approach to use Scratch in an unchanged Mathematics curriculum.

Scratch is available for free for *Windows*, *MacOS X*, and *Linux*. There are multiple variants/re-implementations available, e.g., *Snap!* with further features, or *Scratch Jr* for preschool children. We chose Scratch in the older version 1.4, as it still runs on outdated (but still used in offline school environments) operating systems, such as *Windows 2000*. Furthermore, we changed the language settings to German, as our studies were performed at a German school. For ease of reading, nearly all pictures use English block descriptions in this paper.

4. OUR SCRATCH APPROACH

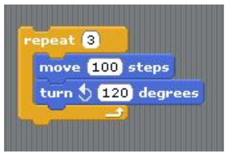
Based on our thoughts noted in Section 2, we chose the standard mathematical topics of (*i*) polygons and tessellations for grade 6, and (*ii*) congruent triangle constructions for grade 7. Our classroom studies were performed in a high school in northern Germany, in the same class with roughly

[2]Available at http://scratch.mit.edu.



```
move 100 steps
turn ↺ 120 degrees
move 100 steps
turn ↺ 120 degrees
move 100 steps
turn ↺ 120 degrees
```

(a) Blocks to create a regular triangle, step by step.



```
repeat 3
    move 100 steps
    turn ↺ 120 degrees
```

(b) The same construction, but now with a loop.

Figure 2: The natural notion of a loop can be introduced when covering regular triangles, as the needed commands will repeat multiple times.

one year of time in between. In grade 6, there were 15 girls and 13 boys, while in grade 7, the number was slightly reduced to 12 girls and 12 boys, due to some children leaving the school/repeating grade 6. Most students had no prior knowledge in programming and Computer Science was not available as a subject for these grades at the school. The programming part was integrated into the normal Mathematics lectures and covered only subjects part of the standard curriculum, though from the new viewpoint of programming algorithms. Due to the facilities available in the school, most students shared a computer as a pair respectively.

4.1 Polygons and Tessellations

After introducing the students to the Scratch environment (especially how to draw lines using simple moving commands), the first two lectures (45 minutes each) covered the construction of triangles and regular polygons, followed by two lectures about tessellations.

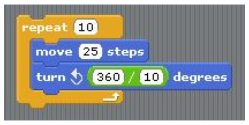
The correctness of the first programs can be verified by the output on the screen, whether it is a triangle or not. As noted before, this concept spans over most geometrical algorithms, as the output is produced on the screen iteratively (the drawing object takes some time to move along its path), errors can be directly correlated to program blocks. However, drawing a simple triangle does not give any deep algorithmic insights. When advancing to regular triangles (and polygons), the concepts of loops can be introduced in a natural way, as depicted in Figure 2.

While the advantages of using a loop are not strongly noticeable when constructing a regular polygon with just three corners, it becomes more clear when the number of corners are increased. Then, one can directly cover that the sum of the exterior angles has to be 360°, leading to a more intricate program depicted in Subfigure 3a. Now the program can be used as a sort of a black box: One just changes the number of desired corners, and the program draws the regular polygon.

Some took a longer time to understand the concept of the algorithmically more intricate program in Subfigure 3a, as it implicitly uses variables on a propaedeutic level. One should not aim for a program as in Subfigure 3b, unless the concept of variables has been introduced in depth in Mathematics (usually not the case in grade 6 in Germany).

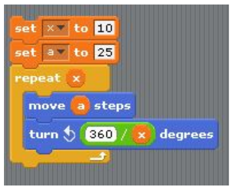
For the following topic of creating finite regular tessellations, the three respective building blocks (regular triangle, square, and hexagon) had now already been programmed by the students.

Nonetheless, putting the individual polygons together multiple times to a tessellation is not as simple a task as it might seem. The student has to take the perspective of the drawing



```
repeat 10
    move 25 steps
    turn ↺ (360 / 10) degrees
```

(a) To construct regular polygons with a differing number of corners, one would replace the 10s with the desired number.



```
set x to 10
set a to 25
repeat x
    move a steps
    turn ↺ (360 / x) degrees
```

(b) A generalized form with added variables.

Figure 3: Extending the program from Subfigure 2b to deal with regular polygons in general.

object itself; else, e.g., multiple hexagons might be drawn, but the final product does not resemble a regular tessellation at all. A modularizing approach is needed, drawing first a sequence of polygons, and then joining them multiple times to a tessellation. An iterative approach can still be helpful though, first figuring out the ideas needed for the solution, before simplifying them in a concise program, cf. Figure 4.

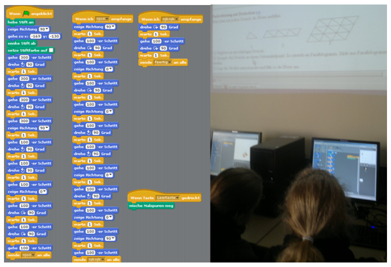


Figure 4: A first approach by students to generate a tessellation.

While a square tessellation can be performed as the easiest of the three, the regular triangle tessellation is already intricate, with the hexagon tessellation usually taking over 20 lines of correct code, see Figure 5.

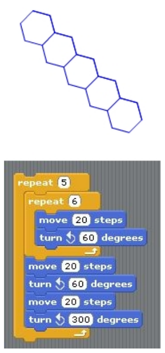
4.2 Congruent Triangle Constructions

Describing congruent triangle constructions is usually split into four cases, cf., e.g. [17], in the Mathematics curriculum:

1. SSS: All three sides are given (side-side-side).
2. SSA: side-side-angle.
3. SAS: side-angle-side.
4. ASA: angle-side-angle (or SAA or AAS).

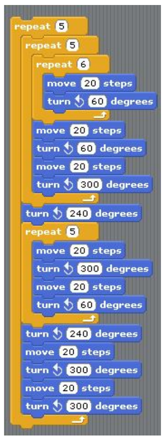
The first two items, SSS and SSA, are not suited for implementation in Scratch in grade 7, as the intersection of two cycles cannot be calculated yet by the students. However, these two items can be handled by dynamic geometry environments such as, e.g., *GeoGebra* [9] or *Cinderella* [27].

In fact, following the arguments at the beginning of Section 2, one can choose a reverse approach: First let the students program SAS and ASA in Scratch (and analogous parametrized families of triangles), then switch to a dynamic geometry environment for SSS and SSA, and lastly, let the students describe congruent triangle constructions the classical way, i.e., by hand. We now focus on the programming part with Scratch in this subsection.



```
repeat 5
  repeat 6
    move 20 steps
    turn ↺ 60 degrees
  move 20 steps
  turn ↺ 60 degrees
  move 20 steps
  turn ↺ 300 degrees
```

(a) By adding five additional blocks of code, multiple hexagons can be drawn in a row.



```
repeat 5
  repeat 5
    repeat 6
      move 20 steps
      turn ↺ 60 degrees
    move 20 steps
    turn ↺ 60 degrees
    move 20 steps
    turn ↺ 300 degrees
  turn ↺ 240 degrees
  repeat 5
    move 20 steps
    turn ↺ 300 degrees
    move 20 steps
    turn ↺ 60 degrees
  turn ↺ 240 degrees
  move 20 steps
  turn ↺ 300 degrees
  move 20 steps
  turn ↺ 300 degrees
```

(b) However creating a 5×5 tessellation requires many more additional blocks of code.

Figure 5: Extending the program creating a hexagon to first create a "line" of hexagons in Subfigure 5a, before assembling the final program to create a 5×5 hexagon tessellation in Subfigure 5b.

A major change compared to the programming in grade 6 (see Subsection 4.1) is the use of variables. The students already know them from previous topics in Mathematics in grade 7, allowing variables to be used in programming. Of course, one could also introduce variables earlier, e.g., in grade 6, but then additional time would be devoted to a topic which is not covered by the curriculum in grade 6. Our approach is an integrated one, i.e., we do not intend to add any topic to the curriculum itself.

To start with variables in Scratch, we picked up on the known topic of regular triangles. The length of each side can be replaced by a common variable, allowing the program to represent all regular triangles at once, see Figure 6.

This idea can then be applied to SAS and ASA: First, the students generate a program for a specific triangle, and secondly, extend it to a program that can generate all triangles of the sort by adding variables, cf. Figure 7 for the case of SAS. Thus, the classic concept of describing a specific geometric construction for congruent triangles is generalized.

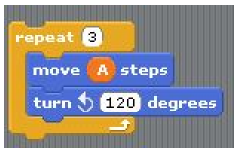
So far, variables have been only used as placeholders for values, they were not changed during the execution of the program; yet, this is a concept fundamental to programming advanced algorithms. As before, we introduce this concept by extending previous programs, see Figure 8.

Parametrized families of triangles can then also be programmed by the students for constructions of the type ASA and SAS, see Figure 9.

As noted before, the remaining types of congruent triangle constructions can afterwards be handled by dynamic geometry environments, followed by hand-written descriptions of the different construction types.

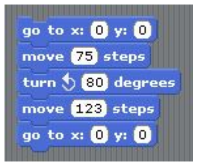
5. OBSERVATIONS OF OUR APPROACH

As expected, introducing the students in grade 6 to the programming language Scratch was intuitive and unproblematic. The concepts of Turtle graphics were directly trans-



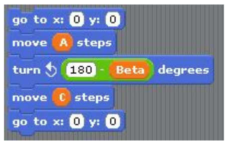
```
repeat 3
  move A steps
  turn ↺ 120 degrees
```

Figure 6: The program from Subfigure 2b, with the length of each side replaced by the variable A.



```
go to x: 0 y: 0
move 75 steps
turn ↺ 80 degrees
move 123 steps
go to x: 0 y: 0
```

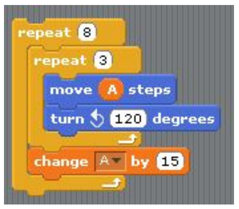
(a) Construction of a triangle with SAS given by $75 - 100° - 123$.



```
go to x: 0 y: 0
move A steps
turn ↺ 180 - Beta degrees
move C steps
go to x: 0 y: 0
```

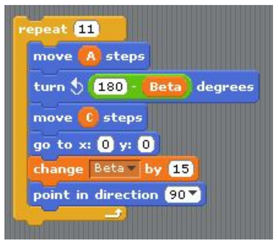
(b) Generalized construction of all triangles given by SAS.

Figure 7: By replacing the explicit values in Subfigure 7a with variables in Subfigure 7b, all SAS constructions can be described at once.



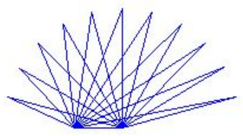
```
repeat 8
  repeat 3
    move A steps
    turn ↺ 120 degrees
  change A by 15
```

Figure 8: The program from Subfigure 6, with the variable A changing during the execution of the program. In this example, there will be eight squares, with the length of each side increasing by a length of 15 each time. If desired, the program can be even more generalized by replacing the 8 and 15 with further variables.



```
repeat 11
  move A steps
  turn ↺ 180 - Beta degrees
  move C steps
  go to x: 0 y: 0
  change Beta by 15
  point in direction 90
```

(a) Program for a parametrized family of constructions of the type SAS.



(b) Example output of the program from Subfigure 9a.

Figure 9: The program in Subfigure 9a is an extension of the program for SAS from Subfigure 7b, with an encompassing loop added and the variable $Beta$ changing during the execution.

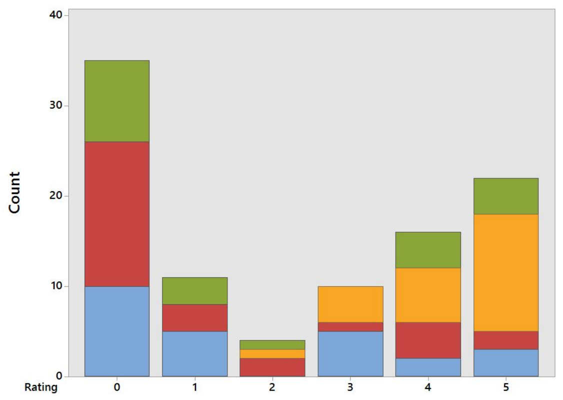


Figure 10: Stacked column chart of the results of the four classes, sorted from 0 (none or barely any knowledge) to 5 (correct solution). While all classes have some correct solutions, the class 9a (using our Scratch approach) has mostly high ratings, with the other classes having many children that scored low.

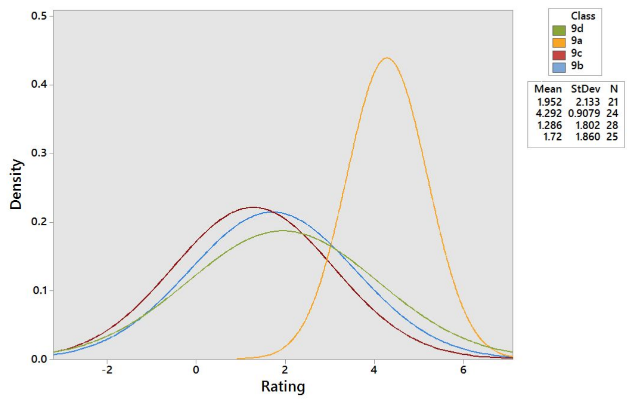


Figure 11: Probability density functions of our data sets. Note that these are fitted curves, therefore the interval extends beyond 0 and 5. The three classes without Scratch perform quite similar, with the class 9a using our approach outperforming them.

latable into Scratch, with regular polygons giving a first start into programming algorithms. Furthermore, the subsequent topic of tessellations illustrated the relevance of exactness and the need for modularization in programming.

In grade 7, roughly a year later, the students were quickly familiar with Scratch again, maybe also because some students continued to use it at home from time to time. Scratch proved to be a viable tool in constructing congruent triangles, with variables now also being used from a Computer Science viewpoint. By reversing the classical order with first programming, then describing by hand, the students were able to provide more concise descriptions of geometric constructions than students from other classes. Additionally, by programming the constructions themselves, the students gained a propaedeutic view on the function of dynamic geometry environments.

While the computers were mostly shared pairwise among the students due to the facilities provided by the school, this turned out to be beneficial, cf. [20]. The students could discuss their programs with each other, as they now had a common mathematical language provided by Scratch.

6. LONG-TERM EVALUATION

To study the long-term impact of our approach, we performed an evaluation in four grade 9 classes at the same school. All classes were taught according to the same Mathematics curriculum by different teachers, except for one class (9a), which used our Scratch approach in grades 6 and 7[3]. The $N = 98$ pupils were tasked with describing a congruent triangle construction, using pen and paper, of the type SAS: 3cm, 50°, 4cm.

We rated the students' answers forwarded to us by the teachers as follows: 5: Correct, 4: few errors, 3: moderate amount of errors, 2: many errors, but correct approach visible, 1: some information about the triangle supplied, 0: none or barely any knowledge. The Figures 10 and 11 show the distribution of the rated answers.

[3]Our integrated approach in grade 7 was $\approx$ two years before.

Based on these results, we further analyzed[4] the data using Levene's test to test for homogeneity of variance using a significance level of 0.05 and found that equal variances can be assumed ($p = 0.062$). Therefore the one-way Analysis of Variance ($ANOVA$) with a significance level of 0.1 was chosen. We found significant differences ($F(3, 94) = 15.06$, $p < 0.001$) in the performance of the students. Tukey's honest significant differences post-hoc test revealed that the class 9a (which used our Scratch approach) performed significantly better than the other classes. We also found no significant differences between the performance of the other classes 9b, 9c, and 9d.

Based on these results, we believe that using our Scratch approach has positive long-term effects regarding the childrens' expertise in the combination of algorithms and geometry. Maybe it is the deeper understanding of the involved algorithmic techniques using programming that allows the knowledge to persist, opposed to it being largely forgotten without recent repetition.

7. OUTLOOK

There are many further options where programming with Scratch can be integrated into a standard Mathematics curriculum: The Bisection method, random experiments, the Euclidean algorithm, the Babylonian method, the sieve of Eratosthenes, approximation of π, numerical algorithms, introduction of negative numbers [24] and coordinate systems, or of course also further topics, e.g., aspects of discrete mathematics (graph theory), space-filling curves, and the Sierpinski gasket, to list just a few.

We envision that programming should be a standard tool in Mathematics in schools[5], just as a calculator, compass, or ruler is; a cultural technique that is available to and useable by everyone.

Acknowledgments We would like to thank the anonymous reviewers for their helpful comments.

[4]We used the popular statistics package *Minitab*, available at http://www.minitab.com/.

[5]To quote C.A.R. Hoare [8]: "*I hold the principle that the construction of computer programs is a mathematical activity like the solution of differential equations.*"

8. REFERENCES

[1] H. Abelson and A. A. DiSessa. *Turtle geometry : the computer as a medium for exploring mathematics.* The MIT Press series in artificial intelligence. Cambridge, Mass. MIT Press, 1981.

[2] P. Bender. Critizing the LOGO philosophy (Kritik der LOGO-Philosophie). J. Math.-Didakt. (1987) v. 8(1/2) p. 3-103., 1987.

[3] C. Almeida et al. *Logo Philosophy and Implementation.* Logo Computer Systems, 1999.

[4] W. Feurzeig, S. Papert, M. Bloom, R. Grant, and C. Solomon. Programming-languages as a conceptual framework for teaching mathematics. *SIGCUE Outlook*, 4(2):13–17, Apr. 1970.

[5] W. Feurzeig, S. A. Papert, and with a preface by Bob Lawler. Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments*, 19(5):487–501, 2011.

[6] K.-T. Förster. Programming in scratch and mathematics: Augmenting your geometry curriculum, today! In *SIGITE*, 2015.

[7] B. Harvey and J. Mönig. Bringing No Ceiling to Scratch: Can One Language Serve Kids and Computer Scientists? In *Constructionism*, 2010.

[8] C. Hoare. The mathematics of programming. In *FSTTCS*, 1985.

[9] M. Hohenwarter and J. Preiner. Dynamic mathematics with geogebra. *AMC*, 10:12, 2007.

[10] G. Holland. Die Bedeutung von Konstruktionsaufgaben für den Geometrieunterricht. *Der Mathematikunterricht*, 20(1):71–86, 1974.

[11] J. Howe, F. Plane, and T. O'Shea. Teaching mathematics through logo programming : an evaluation study. Technical Report DAI-RP-115, University of Edinburgh (Edinburgh, GB), 1979.

[12] J. Howe, P. Ross, K. Johnson, F. Plane, and R. Inglis. Teaching mathematics through programming in the classroom. *Computers & Education*, 6(1):85 – 91, 1982.

[13] J. Hromkovic. *Einführung in die Programmierung mit LOGO: Lehrbuch für Unterricht und Selbststudium.* Vieweg+Teubner, Wiesbaden, 2nd edition, 2012.

[14] S. Hussmann and B. Lutz-Westphal, editors. *Diskrete Mathematik Erleben.* Springer, 2015.

[15] F. Ke. An implementation of design-based learning through creating educational computer games: A case study on mathematics learning during design and computing. *Computers & Education*, 73:26 – 39, 2014.

[16] U. Kortenkamp and A. Lambert. Arbeitskreis Mathematikunterricht und Informatik. *GDM-Mitteilungen*, 94:32–37, 2013.

[17] S. Krauter. *Erlebnis Elementargeometrie.* Spektrum Verlag, 2007.

[18] C. M. Lewis and N. Shah. Building upon and enriching grade four mathematics standards with programming curriculum. In *SIGCSE*, 2012.

[19] E. Modrow. *Informatik mit BYOB / Snap!* Universität Göttingen, Lehrerbildungszentrum Informatik, 2013.

[20] N. Nagappan, L. Williams, M. Ferzli, E. Wiebe, K. Yang, C. Miller, and S. Balik. Improving the cs1 experience with pair programming. *SIGCSE Bull.*, 35(1):359–362, Jan. 2003.

[21] J. Neubrand. *Eine Klassifikation mathematischer Aufgaben zur Analyse von Unterrichtssituationen.* Franzbecker, Hildesheim, 2002.

[22] R. Oldenburg. *Mathematische Algorithmen im Unterricht. Mathematik aktiv erleben durch Programmieren.* Wiesbaden: Vieweg+Teubner., 2012.

[23] R. Oldenburg, M. Rabel, and J. Schuster. A Turtle's Genetic Path to Object Oriented Programming. In *Proceedings to Constructionism*, 2012.

[24] A. Pallack. Die Multiplikation ganzer Zahlen – mit oder ohne Kontext? *Mathematik lehren*, 31(183):25–27, 2014.

[25] H. Puhlmann. Funktionales Programmieren: eine neue Verbindung von Informatikunterricht und Mathematik. Technical report, Technische Universität Darmstadt, FB Mathematik, 1998.

[26] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: Programming for all. *Commun. ACM*, 52(11):60–67, Nov. 2009.

[27] J. Richter-Gebert and U. Kortenkamp. *The interactive geometry software Cinderella.* Springer, 1999.

[28] W. Riemer. Erziehen im Mathematikunterricht. In R. Kaenders and R. Schmidt, editors, *Mit GeoGebra mehr Mathematik verstehen*, pages 13–20, Wiesbaden, 2011. Vieweg+Teubner Verlag.

[29] M. Rizvi, T. Humphries, D. Major, M. Jones, and H. Lauzun. A cs0 course using scratch. *J. Comput. Sci. Coll.*, 26(3):19–27, Jan. 2011.

[30] B. Schmidt-Thieme. Erklären als fachspezifische Kompetenz in fächerübergreifender Perspektive. In *Beiträge zum Mathematikunterricht*, Hildesheim, 2009. Franzbecker.

[31] G. Serafini. Teaching programming at primary schools: Visions, experiences, and long-term research prospects. In *ISSEP*, 2011.

[32] K. M. Strecker. *Informatik für Alle - wie viel Programmierung braucht der Mensch?* PhD thesis, University of Göttingen, 2009. d-nb.info/999618229.

[33] S. Uludag, M. Karakus, and S. W. Turner. Implementing it0/cs0 with scratch, app inventor for android, and lego mindstorms. In *SIGITE*, 2011.

[34] I. Utting, S. Cooper, M. Kölling, J. Maloney, and M. Resnick. Alice, greenfoot, and scratch – a discussion. *Trans. Comput. Educ.*, 10(4):17:1–17:11, Nov. 2010.

[35] B. Vöcking, H. Alt, M. Dietzfelbinger, R. Reischuk, C. Scheideler, H. Vollmer, and D. Wagner, editors. *Algorithms Unplugged.* Springer, 2011.

[36] H.-G. Weigand and T. Weth. *Computer im Mathematikunterricht. Neue Wege zu alten Zielen.* Spektrum Verlag, 2002.

[37] L. A. Zavala, S. C. H. Gallardo, and M. A. García-Ruíz. Designing interactive activities within scratch 2.0 for improving abilities to identify numerical sequences. In *IDC*, 2013.

[38] J. Ziegenbalg. Informatik-affine Themen in der Didaktik der Mathematik. *GDM-Mitteilungen*, 96:7–14, 2014.