Building mathematical knowledge with programming: insights from the ScratchMaths project

Laura Benton[*], l.benton@ucl.ac.uk; **Celia Hoyles**[*], *c.hoyles@ioe.ac.uk;* **Ivan Kalas**[*^], *i.kalas@ioe.ac.uk; kalas@fmph.uniba.sk;* **Richard Noss**[*], *r.noss@ioe.ac.uk*
[*] London Knowledge Lab, UCL Institute of Education, 23-29 Emerald Street, London, UK
[^] Dept. of Informatics Education, Comenius University, Bratislava, Slovakia

Abstract

The ScratchMaths (SM) project sets out to exploit the recent commitment to programming in schools in England for the benefit of mathematics learning and reasoning. This design research project aims to introduce students (age 9-11 years) to computational thinking as a medium for exploring mathematics following a constructionist approach. This paper outlines the project and then focuses on two tensions related to (i) the tool and learning, and (ii) direction and discovery, which can arise within constructionist learning environments and describes how these tensions were addressed through the design of the SM curriculum.

Keywords

Programming; mathematics; Scratch; primary education; design research

Introduction

Computer programming is undergoing a renaissance in English schools. Recent policy and curriculum initiatives have resulted in ICT being replaced by computing across all ages from 6 to 16 years. These changes have been motivated by a concern about students leaving school with little understanding of computer science or the creative side of computing (Furber 2012). From September 2014, schools in England[1] have to teach the new National Computing Curriculum (DfE 2013), which requires students to learn about how computational systems work, to use technology safely and to design and build their own programs. At least at the policy level, computing is recognised as not just about programming *per se*, but programming as a modeling tool: a key component of thinking that allows ideas to be brought to life and explored in different subject areas and contexts. How far this will happen in practice is of course a complex matter shaped by schools, teachers and available resources (material and people) to support this work.

Much of the research in the field of programming within schools was conducted in the latter part of 20th century before the advent of the many new blocks-based programming environments developed specifically for young users (Weintrop & Wilensky 2015). One is Scratch, used by a huge number of young children in and out of school (with over 2 million registered users aged under 12 years). The popularity of this style of programming for use with novice programmers is in part due to its ease of readability, composition and browsability alongside its interactivity, and visual and dynamic outcomes (*ibid*).

In this paper, we introduce the ScratchMaths (SM) project, which aims to build mathematical knowledge through programming in Scratch during a 2-year intervention for students aged 9-11 years. We set out to exploit programming to support mathematical reasoning in pre-specified mathematical content areas and thus will explore among other areas, Papert's (1972) claim that learning to program in carefully designed ways should "make it easy to learn algebra and

[1] Within England a growing group of schools known as academies do not have to adhere to the national curriculum so can opt out of computing – an interesting dilemma

geometry" (p.4). We describe the design process leading to a constructionist curriculum and professional development program where students are able to exploit the powerful ideas of computational thinking and programming tools to engage in mathematical thinking. We borrowed from Brennan (2015) the identification of a number of tensions in supporting constructionist approaches[2], in this paper we focus on two: the tensions between (i) tool and learning, and (ii) direction and discovery. We describe how these tensions were addressed in the design and implementation of the SM curriculum.

Background

Tension between tool and learning

In the 1970s and 80s the earliest research in schools took place that explored the potential of learning mathematics through programming languages, (such as BASIC and Logo), (Hoyles & Noss 1992). As Logo became more integrated into schools, a perception grew that programming was too difficult to have any widespread impact on mathematics learning. An often-cited reason for this was the perception of programming as an *overhead* – something to be squeezed into an already-overcrowded curriculum. As Resnick et al. (2009) point out, the difficulty of mastering programming syntax, and the lack of specific skills/knowledge that was required by teachers to effectively guide or challenge students in capitalising on these early programming tools, posed problems in exploiting this potential. There are now growing concerns that this perception may come full circle with the 'floor' being lowered so far that novice programmers are discouraged – or at least, not encouraged – from engaging with the underlying concepts, which may in part be due to the implicit ways compilation errors are handled in tools such as Scratch. Thus, they can achieve a visually pleasing outcome on the screen almost 'by accident' with no desire to ask *why* it happened.

The accessibility of the programming *tool,* and the process of *learning* through programming using the tool, is identified as the first of Brennan's tensions. Brennan (2015) describes this as the necessity of achieving a balance "between knowledge about the tool and understanding of how to engage in creative design activities, using the computer for personal expression and problem solving" (a similar analogy in mathematics education is the focus on what is termed instrumentation) . Furthermore, there remains the critical challenge to exploit knowledge that has been gained within programming contexts to promote engagement with mathematical ideas and reasoning (Hoyles & Noss, 1992).

Despite recent innovations of programming tools, which aim to support a constructionist approach to learning whilst making the tool more accessible to a more diverse range of learners, the tension between content knowledge of the programming tool and pedagogical knowledge is still an important issue to address within the classroom. In their commentary on Brennan's paper Gash and McCloughlin highlight the close relationship between content and pedagogy, suggesting that teachers may find it easier to address the pedagogical issues. Furthermore through their observations and interviews with teachers (primary, secondary and university) during a series of Scratch workshops, Bustillo and Garaizar (2014) suggest that often students and teachers "have a limited, immediate, and concrete vision of using Scratch (e.g. a step-by-step guide to program a video game during a semester), instead of realizing the cross-curricular potential of computational thinking". They advocate a set of best practices, learning guides and curriculum models to help teachers and students encounter the richness of the pool of ideas embedded within Scratch. We concur but would go further: teachers need to appreciate the core goals of the programming activities, the power of computational thinking skills and the purpose of exploiting them in mathematics.

[2] The complete list of tensions include (i) tool and learning, (ii) direction and discovery, (iii) individual and group, (iv) expert and novice, and (v) actual and aspirational.

Tension between direction and discovery

Much research in this field has focused on extra-curricular activities that were either voluntary or involved specially selected students, with rather few studies in naturalistic classroom settings (Lye & Koh 2014). Israel et al. (2015) also note the lack of research examining how teachers implement school-wide computing initiatives at the elementary level and particularly highlight this as the case for diverse students (in terms of background and including those affected by poverty or disability). Bers et al. (2014) claim that one key factor in the successful implementation of a programming-based curriculum is to understand how to support individual teachers' needs, especially in terms of curriculum modifications, classroom management alternatives and forms of adult support. Furthermore one failure with early programming initiatives was a neglect to design explorations that linked to young learner's interests or experiences (Resnick et al. 2009).

Defining the role of the teacher and the details of the curriculum design also present a challenging dilemma that needs to be addressed. This was framed as the 'play paradox' by Noss and Hoyles (1996): the problem of designing a learning activity in a way that allows students to explore and construct ideas for themselves, but also ensuring that they encounter the powerful ideas embedded within the activities; thus balancing exploration and guidance. Brennan's second tension (2015) resonates with this paradox: i.e. the tension between "direction (providing resources in advance, anticipating and steering learner needs) and discovery (making resources available when they are needed, in response to learner needs)". We claim however that an overarching challenge for those whose interest lies in teaching mathematics more effectively is not only one of pedagogy, but of defining and elaborating new kinds of mathematical knowledge that can be expressed by programming. We now turn to the SM project, which has attempted to tackle this challenge.

The ScratchMaths project

Programming in schools has been shown to have the potential to develop higher levels of mathematical thinking in relation to aspects of number linked to multiplicative reasoning, mathematical abstraction including algebraic thinking as well as problem solving abilities (Clements 2000). More recently, attention has been paid to defining computational thinking (McMaster et al. 2010; Wing 2008), which is seen by Wing, for example, as part of a 'family' of different aspects of mathematics thinking (Wing 2008). This relationship helps to explain why programming and computer-based mathematical instruction have been found to have a positive effect on both student attitudes, and on attainment in mathematics (Clements 1999). But of course, such results depend fundamentally on the design of materials, support and implementation. The SM project aims to maximise the benefits of programming for students' mathematical thinking, reasoning and attainment. The overarching goal of the research is to iteratively design and evaluate, both quantitatively and qualitatively, materials for students and teachers that directly address the learning of computational thinking and its exploitation to enhance mathematical engagement and attainment.

ScratchMaths is a 3-year research project involving a 2-year intervention with students aged 9-11 years. The intervention is intended to comprise approximately 20 hours teaching time across each of these two school years, with the first year focusing on computational thinking with an implicit mathematical component, and the second year foregrounding explorations of key mathematical concepts using programing tools. The intervention has been subject to cycles of iterative design research with the final quantitative outcome measure being based on national standardised mathematics test scores, taken by all students at the end of primary school.

Methodology

Design workshops

At the start of the project seven teachers from four London primary schools were recruited to act as 'design partners' for the SM intervention. The teachers were either class teachers or had responsibility for teaching computing for this age range, and had a variety of experience with using Scratch ranging from none to reasonably experienced. The teachers attended five ‘twilight’ design workshops at the university to help them to understand the main features of Scratch as a computational tool as well as to encourage them to work collaboratively with the project team to design, test and evaluate potential student activities. The teachers also shared some of their experiences of teaching computing and maths, with project team members visiting each of the schools to observe some lessons. This design phase established that the intervention needed to be appropriate for teachers with a range of experience in computing and in using Scratch as well as for students of wide attainment levels and support needs[3]. It was apparent that the materials needed to be clear as to ways the intervention could fit within an already full teaching schedule through making explicit links with the existing curriculum, as well as signpost critical teaching points and progression and suggest discussion opportunities to reinforce understanding.

The project team also conducted several intensive internal design workshops to set out a high-level overview of the entire SM curriculum guided by existing knowledge and experience from within the team’s prior research and the findings from the design workshops and school visits. This overview identified the key concepts to be introduced and an overarching structure. Each year would be structured into several modules, each with multiple investigations, and each of which comprising a series of activities, mixing hands-on and unplugged. The activities for the first investigation were then planned in detail so it could be trialled in the design schools.

Design research in schools

A design research approach was followed primarily to establish the suitability of materials for use within the current primary school context and how far the teacher was able to understand and communicate the key learning objectives of each activity as well as feel comfortable with the content. The activities needed to be: sufficiently adaptable so as to be accessible for all students while offering challenges for the higher attainers, and also presenting a balance between scaffolding of concepts and space for exploration. Further the structure of the content was modified so it could be taught in lessons of varying length and frequency. The design research was undertaken in four design schools over a year with one school progressing through the entire Y5 curriculum with three Y5 classes and the remaining three design schools testing a subset of the materials with Y5 students in their schools. During the lessons in which these materials were trialled, three researchers with a range of backgrounds, which included expertise in computer science, mathematics teaching and primary school research, conducted observations. This involved the researchers writing extensive field notes during these observations and speaking with both teachers and children to gauge the appropriateness of the different materials. After each lesson observation the researchers met to discuss what worked well and where improvements could be made to the materials. Minor changes were agreed amongst this sub-team, with more significant changes discussed with the wider project team. The materials were then trialled again with a different class (or in the case of substantial modifications the same class) to check the appropriateness of these changes.

Addressing the challenges through design

One key outcome of the curriculum design process described above was a ‘framework for action’ (DiSessa & Cobb 2004), which we named the “5Es”[4]. This framework (consisting of five unordered constructs) was clearly framed by a host of research into good practice in teaching mathematics but also emerged from the early design workshops and was refined through the design research in schools. It has been developed to provide guidance on the pedagogical strategies teachers may adopt to successfully implement different aspects of the SM

[3] The SM project is charged with ”narrowing the attainment gap”, which requires as many students as possible to be successfully engaged

[4] This is different from, but clearly intersects with, the BSCS 5E Instructional model, which includes five phases: Engage, Explore, Explain, Elaborate and Evaluate, and is primarily targeted at science education (Bybee et al. 2006)

intervention. This framework is described in more detail below in relation to the two of the tensions identified by Brennan.

Explore

Papert (1980) believes that children should use computers to explore their thinking processes, suggesting with respect to Logo that the primary learning experience is about "getting to know the Turtle, exploring what a Turtle can and cannot do". Constructionist approaches value learning in this way through design activities (Brennan 2015), which provide opportunities to explore ways to deal with different constraints and ambiguity through employing skills such as iterative thinking, problem solving and creativity. Therefore this first construct suggests the importance of developing and supporting activities that allow learners to investigate ideas, try things out *for themselves* and debug conceptual and technical errors where necessary. Part of this endeavour is to shift students towards 'taking control of their own learning' and to seek out the reasons behind different outcomes.

The activities designed around this construct addressed two of the tensions that were highlighted by Brennan, tool and learning as well as direction and discovery. During the trials in design schools it was noted that the children's previous experience and knowledge of the tool had a major influence on their approach to SM activities. Students with previous experience of using Scratch would often use the blocks and strategies with which they were already familiar, whereas students with no experience of Scratch were more cautious about trying things out that they had not been explicitly told to by the teacher. Early in the SM curriculum students are introduced to tools for exploration within the Scratch environment. For example the use of what Blackwell (2002) describes as direct manipulation - a single action with a single visible effect. Within the SM curriculum this term is seen as encompassing the manipulation of all objects both physical (i.e. BeeBots) and digital, and therefore we use the term 'direct drive' to refer just to digital objects. Direct drive is used to firstly explore the blocks on their own (Fig. 1 left), by clicking them and observing the reaction, an important precursor we claim to using them to build more complex scripts (Fig. 1 right). This gradual progression from direct drive of blocks to planning and building behaviours built into scripts provides a structured approach to exploration encouraged throughout the SM curriculum.

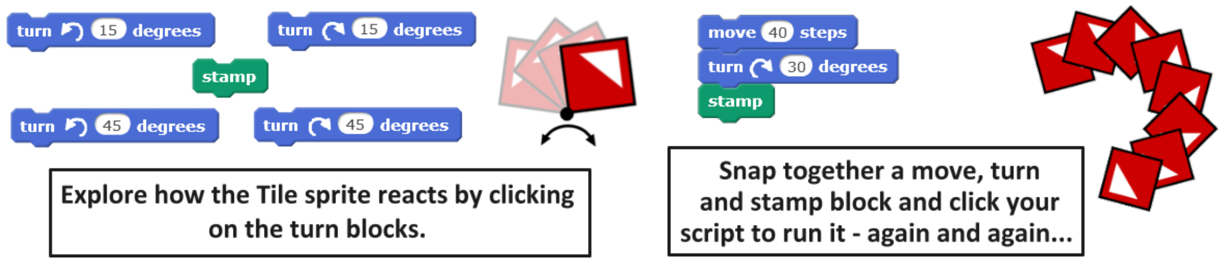
turn 15 degrees
turn 15 degrees
stamp
turn 45 degrees
turn 45 degrees
Explore how the Tile sprite reacts by clicking on the turn blocks.
move 40 steps
turn 30 degrees
stamp
Snap together a move, turn and stamp block and click your script to run it - again and again...

Figure 1. Example direct drive activity (left) progressing on to building simple scripts (right)

Explain

A crucial aspect of understanding ideas is being able to explain what has been learned and articulating the reasons behind a chosen approach (using different modes of communication). This helps clarify ideas, by expressing them explicitly as well as in answering questions from peers. Several theorists have highlighted the cognitive benefit of generating verbal explanations (Harel & Papert 1990). For example, Brown (1988) has shown that being encouraged to explain and represent knowledge in multiple ways can increase motivation and levels of understanding as well as subject mastery. In relation to mathematics, Hoyles (1985) discusses how language can facilitate reflection and internal regulation, and how part of this process is the identification of which parts of the mathematical idea are important and which are not. This reflection, or thinking about one's own thinking is a key component of the constructionist approach (Han & Bhattacharya 2001), with the programming language itself becoming the tool "to think with". This second construct suggests the importance of incorporating reflective questions and opportunities for discussion with peers as well whole-class interactions orchestrated by the teacher.

The activities designed around this construct seek to address the tension between the tool and learning. Once students are familiar with the tool, it was observed that the ease of building scripts may encourage students to create extremely long scripts which then appeared to have status as demonstrating a lot of 'work'! However, it is difficult to understand and predict what these scripts would do when clicked. Another key idea of the SM curriculum is definitions, an under-used component of Scratch but which helps to reduce complexity and aids the readability of scripts. The SM curriculum promotes the use of definitions through the process of building a script and then giving it a meaningful name (e.g. Fig. 2). This in turn supports students in explaining step by step what their script is doing and what outcome they intend to happen.

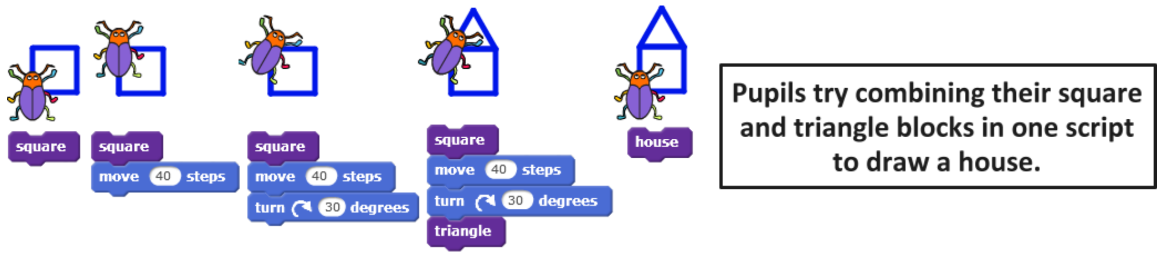

Figure 2. Example activity where students are encouraged to define blocks that then help them explain how they have drawn their houses

Envisage

It is important to have a goal in mind when building a computer program and to predict what the outcome might be *before trying it out.* Papert (1980) describes the role of building computer programs in facilitating reflection on intuitive expectations and knowledge, and highlights that the link between the idea and the child's intuitive knowledge is seen as key in understanding the power of the idea (Papert 2000). However, Rader et al. (1997) found that in their work using a children's programming environment, children can easily create programs without "much knowledge of the underlying program mechanisms" (as mentioned above). As often novice programming tools can now manage much of the syntactic error handling, students are only required to 'debug' when they have a clear goal, in other words it is quite straightforward to generate *an* outcome but not necessarily a *specific* outcome decided upon in advance. Therefore to truly understand an idea it is necessary to take time to predict the outcome *before* building the program and then compare the actual outcome with this prediction. This enables the establishment of whether the original intuition was correct or whether this knowledge needs to be remodelled (Papert 1980). This third construct suggests a need for some learning activities to be conducted prior to exploration with the programming tool, to provide learners with the opportunity to consider the program goal and to predict the potential outcomes of using different strategies. It is important this construct is balanced with explore, providing exploration opportunities that allow discoveries to be made but also occasions to envisage the outcome first.

The activities designed around this construct intend to address the tension between the tool and learning. Many students observed during the trial were very happy and excited with any outcome that resulted in an attractive pattern being stamped on the screen or a fun animation being played out, which they were able to produce without necessarily fully understanding how they created it or even having a clear goal in what they were aiming to achieve. The SM curriculum thus includes a series of unplugged activities that require students to work off the computer, encouraging them to practice prediction, reflection and debugging skills before they test things on the computer, and to reflect on how to engage with similar activities in other areas of the curriculum (see bridgE below). It also promotes body syntoncity (Watt 1998) by encouraging both teachers and students to envisage themselves as the sprite through activities which require them to *act out* the scripts or through physical objects such as paper cut-outs and toys.

Exchange

Collaborating and sharing is a powerful way to learn, with constructionist approaches advocating the development of ideas through interactions with others (Han & Bhattacharya 2001). This allows you to 'decentre', while trying to see a problem from another's perspective as well as defend your own approach and compare it with others. Furthermore Hoyles (1985) suggests that others' ideas can potentially result in modifications to an individual's thought processes, particularly helpful in clarifying predictions or explaining ideas that are not yet fully formed. Bruckman (1998) has also undertaken research demonstrating the cognitive, social and psychological benefits that undertaking constructionist activities as part of an online community can provide. However, children are still developing their collaboration skills and may need help to work together, resolve disagreements and question one another (Hoyles 1985). Therefore the fourth construct requires the inclusion of meaningful opportunities to share and build on others' ideas.

The activities designed around this construct intend to address the tension between direction and discovery. Collaborative learning offers the potential to promote less directed exploration and discovery. During the trial in schools it was observed that pair work could encourage discussions, requiring students to explain their strategies and discoveries to their partner. Some teachers arranged mixed ability pairing encouraging the more able students to support less able students by 'teaching' them what they had already discovered for themselves. Individual discoveries were also observed quickly spreading around the whole class without teacher intervention through the students monitoring what their peers were working on.

bridgE

Powerful ideas should be embedded in any well-designed constructionist activity (Bers et al. 2014), and ideas are seen as powerful partly through their connections with other disciplines, such as mathematics (Papert 2000), and partly by virtue of the language in which they are expressed. In order to develop these connections the ideas need to be re-contextualised and re-built in the language of the other discipline. Therefore the final construct requires that activities or teacher moves be suggested to make explicit links to another context (in our case school mathematics).

The activities designed around this construct look to address the tension between the tool and learning. In one of the classes during an activity, which required circular repeated patterns to be stamped, students were observed calculating the value of the repeat block by dividing 360 by any chosen value in the turn block. Sometimes this resulted in a decimal number, e.g. 5.5, which they then inputted into the repeat block. Scratch automatically treats the input to repeat by rounding it prior to running (as it is not possible to stamp .5), which in this case was done by rounding up. To ensure these important mathematics learning opportunities are not overlooked due to the behaviour of the tool, explicit links with the mathematics curriculum are made and suggested teacher discussion starter questions are provided within the materials. The unplugged activities are also intended to make further consolidate these links away from the computer.

Conclusion

The ScratchMaths project is still in progress and here we primarily focus on describing the design process adopted to develop the curriculum materials. We have also provided glimpses of the myriad of challenges in implementation. In our presentation we will provide some interim results along with more examples to give a greater feel for the different activities and the progression we envisage.

Acknowledgments

We would like to thank the Education Endowment Foundation for funding this work. We also thank all of the teachers and students from our four design schools for their invaluable contributions to the design and development of the SM intervention.

References

Bers, M.U. et al., 2014. Computational thinking and tinkering: Exploration of an early children robotics curriculum. *Computers & Education*, 72, pp.145–157.
Blackwell, A.F., 2002. What is Programming? In *14th Workshop of the Psychology of Programming Interest Group*. pp. 204–218.
Brennan, K., 2015. Beyond Technocentrism: Supporting Constructionism in the Classroom. *Constructivist Foundations*, 10(3), pp.289–296.
Brown, A.L., 1988. Motivation to learn and understand: On taking charge of one's own learning. *Cognition and Instruction*, 5, pp.311–322.
Bruckman, A., 1998. Community Support for Constructionist Learning. *CSCW (Computer Supported Collaborative Work: The Journal of Collaborative Computing)*, 7, pp.47–86.
Bustillo, J. & Garaizar, P., 2014. Scratching the surface of digital literacy...but do we need to go deeper. In *IEEE Frontiers in Education Conference (FIE)*. pp. 1–4.
Bybee, R.W. et al., 2006. *The BSCS 5E instructional model: Origins and effectiveness*, Colorado Springs, CO: BSCS, 5, pp. 88-98.
Clements, D., 2000. From exercises and tasks to problems and projects unique contributions of computers to innovation mathematics education. *Journal of Mathematical Beh.*, 19(1), pp.9–47.
Clements, D.H., 1999. The future of educational computing research: The case of computer programming. *Information Technology in Childhood Education Annual*, 1, pp.147–179.
DfE, 2013. *Computing Programmes of Study: Key Stages 1 and 2*, DfE. Available at: https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study.
DiSessa, A.A. & Cobb, P., 2004. Ontological Innovation and the Role of Theory in Design Experiments. *Journal of the Learning Sciences*, 13(1), pp.77–103.
Furber, S., 2012. *Shut Down or Restart? The way forward for Computing in UK schools*,
Han, S. & Bhattacharya, K., 2001. Constructionism, Learning by Design, and Project Based Learning. In *Emerging perspectives on learning, teaching, and technology*.
Harel, I. & Papert, S., 1990. Software design as a learning environment. *Interactive Learning Environments*, 1(1), pp.1–32.
Hoyles, C., 1985. What is the point of group discussion in mathematics? *Educational studies in mathematics*, 16(2), pp.205–214.
Hoyles, C. & Noss, R., 1992. *Learning Mathematics & Logo*, Cambridge, MA, USA: MIT Press.
Israel, M. et al., 2015. Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers & Education*, 82, pp.263–279.
Lye, S.Y. & Koh, J.H.L., 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, pp.51–61.
McMaster, K., Rague, B. & Anderson, N., 2010. Integrating mathematical thinking, abstract thinking and computational thinking. In *ASEE/IEEE Frontiers in Ed. Conf*. Washington, DC.
Noss, R. & Hoyles, C., 1996. *Windows on mathematical meanings: Learning cultures and computers*, Springer Science & Business Media.
Papert, S., 1980. *Mindstorms: Children, computers, and powerful ideas*, Basic Books, Inc.
Papert, S., 1972. Teaching Children to Be Mathematicians vs. Teaching About Mathematics. *International Journal of Mathematics Education and Science Technology*, 3, pp.249–262.
Papert, S., 2000. What's the big idea? Toward a pedagogy of idea power. *IBM Systems Journal*, 39(3.4), pp.720–729.
Rader, C., Brand, C. & Lewis, C., 1997. Degrees of comprehension: children's understanding of a visual programming environment. In *Proc. CHI '9*. ACM, pp. 351–358.
Resnick, M. et al., 2009. Scratch: Programming for all. *Comms of the ACM*, 52(11), pp.60–67.
Watt, S., 1998. Syntonicity and the psychology of programming. In *Proceedings of 10th Annual Meeting of the Psychology of Programming Interest Group*. pp. 75–86.
Weintrop, D. & Wilensky, U., 2015. To block or not to block, that is the question: students' perceptions of blocks-based progrmming. In *Proc. IDC '15*. ACM, pp. 199–208.
Wing, J.M., 2008. Computational thinking and thinking about computing. *Philos. Trans. Roy. Soc. London Ser. A: Math., Phys. and Eng. Sci.*, 366(1881), pp.3717–3725.