Learning to Program - Programming to Learn: Technology Supporting Digital, Physical and Social Learning in Schools

Kristina Litherland

University of Oslo, P.O Box 1092, Blindern, 0312 Oslo, Norway

Abstract

The purpose of this project is to provide a deeper understanding of programming pedagogic practices by studying two cases of programming in school, providing two different entry points to learning of and with computer programming. The cases represent two approaches to technology enhanced learning of programming, namely screencasts and so-called "makerspaces", but also how programming as a technology itself may enhance learning. Using qualitative research methods, my aim is to develop theory and practice related to programming pedagogy. Preliminary results show that both screencasts and makerspaces are potentially useful tools for learning programming, and that programming may be a useful learning tool in itself. However, these findings need to be explored and refined further.

Keywords 1

Computer programming, interdisciplinarity, screencasts, makerspaces, socio-cultural perspective

1. Introduction

The autumn of 2020 marked the starting point of the new national curriculum in Norwegian primary and secondary education (years 1 to 10), known as "the Renewal of Subjects" [1, author's translation]. One of the new aspects of the curriculum is the explicit inclusion of computer programming in several subjects, specifically mathematics, science, music, and arts and craft; all of which are mandatory subjects for all students. Computer programming has been an elective subject in Norwegian secondary schools since 2016, but with the new curriculum, all students are obliged to learn to program as part of their mathematics course so they can successfully use programming as a tool in both mathematics and other subjects. This provides several challenges, but also some opportunities. One such challenge is that teachers must learn both computer programming and how to integrate it into their subjects, even though there is little knowledge on how this is best done [2]. On the other hand, programming may provide the opportunity to engage students in interdisciplinary activities and problem solving in several subjects [3].

The Nordic approach to programming in school, where programming is integrated into other subjects [4], is fundamentally different to approaches seen in other Western countries' educational systems where programming is organised as separate subjects (see e.g. [5]). The new, Norwegian curriculum and the existing programming courses provide an opportunity to study programming *for* the subjects versus programming *as* a subject. One rationale for the importance of learning how to program at a basic level is the idea that all members of society need an understanding of the role of programming in the digital world that surrounds us (e.g. what is an algorithm and how can it be used to deliver personalised ads). However, not all students need professional knowledge on how to create industrial-strength computer programs. In the Nordic countries,

Proceedings of the Doctoral Consortium of Sixteenth European Conference on Technology Enhanced Learning, September 20–21, 2021, Bolzano, Italy (online).

EMAIL: Kristina.litherland@iped.uio.no
ORCID: 0000-0001-9694-1291

there is an emphasis on programming as a bridge between subject domains, e.g. mathematics and natural science, and statistics and social science.

The aim of this PhD-project is to provide a deeper understanding of programming pedagogic practices in Norwegian schools by studying two cases providing two entry points. The cases represent different approaches to technology enhanced learning of and with programming described in detail later in this paper. Note that my project concerns both learning of conceptual knowledge of programming and other subjects, and how technological tools can support this learning. I view programming skills themselves as technological learning tools.

The PhD-project overall is guided by the following research question with two sub-questions, which, when combined, will provide a basis for elaborating on the main research question. *How do computer programming classes and integrated subject/computing classes compare as interdisciplinary learning arenas?*

1. How does interactive screencast technology support digital and social learning practices in computer programming classes?
2. How are learning processes supported by programming as an intermediate tool between physical making and conceptual knowledge in a digital science classroom?

Using a qualitative, primarily bottom-up approach to explore my research questions, my contribution will be to improve the understanding of the two approaches to programming knowledge development in Norwegian schools. Hence, the aim of the project is not to make statistically generalizable claims, but to give reliable and valid perspectives of development processes observed within the cases at hand. I hope that the project will reveal both challenges and opportunities that are relevant for developing the field of programming pedagogy in school further, and how technical tools are involved in these processes.

2. Theoretical framework

The theoretical framework for the project is grounded in the sociocultural perspective on learning, which considers learning as fundamentally social [8]. A key concept of the sociocultural perspective is that tools mediate learning. According to Vygotsky [8], language itself is the most powerful mediating tool, and researchers should therefore give attention to language use when studying learning. However, language is not the only tool involved in learning computer programming; therefore, also other (computer mediated and non-computational) tools and artefacts will be included as objects for analysis. Computer programming is about creating code a computer can read, which is a technological artefact. However, humans also read, modify, use, and write code, often based on other people's code. I argue that this makes programming an inherently social activity, and that programming should be treated as such. This idea of sociality is in line with Vygotsky's view of learning.

Computer science (CS) education is a broad field and includes CS education at all levels in the educational system: From elementary school to higher education. Nygaard [9] claims the term computer science is too narrow, as it places too much emphasis on the computer itself and does not cover all (e.g. social) aspects of the field. I choose to use the term *programming pedagogy*, as using a verb (programming) makes the term more process/action oriented, to cover the field of teaching and learning to program in a wide sense, including programming concepts, practices and perspectives [10].

This theoretical perspective will frame my analysis by providing a focal point on knowledge development over cognitive assessment. Potential findings relate to observed classroom episodes where the use of tools (e.g. language, gestures, and digital tools) are involved in this development. Since programming in Norwegian schools is a new phenomenon, there is a need to better understand what is happening during programming classes/classes with programming and what the potentials are.

3. Programming in school

The idea of using programming in school is not new and often dated to Seymour Papert's 1980 book *Mindstorms* [3] and his concept of "Turtle Geometry". At the time, Papert and his

team at Massachusetts Institute of Technology had recently developed a text-based programming language called Logo. Papert had grand ideas about how children could learn mathematics and geometry hands-on, but also how they could learn to think, by using Logo and constructing programs [11]. However, later research has criticised some of the claims by finding that a programmer's knowledge and experience does not always develop into cognitive/higher order skills (see e.g. [12]).

Mitch Resnick, one of Papert's students and leader of the team that developed the most well-known block-based programming language used in education, Scratch, is a champion for an interest-driven approach as a programming pedagogy [13]. Resnick's idea is that children can develop what is often referred to as 21st century skills, such as creativity and collaboration skills, through open ended programming activities, which involve very little upfront teaching. The success of this approach, according to Resnick, relies on the elimination of complicated programming syntax, which is the aim of block-based programming.

From Papert to Resnick the rationale has moved from being quite specific (mathematics and thinking) to talking about more general skills. The Nordic model of programming can be placed somewhere between the two, as programming is placed within subjects but are meant to develop both domain specific and general skills. Waite [2] mentions programming for the subject as a specific context for programming that needs a specific pedagogy. She uses as example the dilemma of how to help students both connect and differentiate between programming and the subject in question. One particular challenge in this regard is how symbols like punctuation marks or equals signs are used in specific ways in programming languages that are not necessarily compatible with other fields, such as mathematics.

In recent years, a growing number of researchers have studied programming pedagogy. The nominal paper by Wing [6] in 2006 is typically credited as the source of the current wave of programming in schools across the world. As a result of this wave, the field of programming in school has gotten an increasingly large mass of available tools and resources (see e.g. [14]). This is also symptomatic for the field of research. There is a high focus on programming languages and environments, but not on what concepts, ideas, or practices the learners are expected to know. In the Norwegian curriculum, concepts such as variables, loops and if-statements are mentioned explicitly, while a more basic concept such as sequencing is not. In addition, no practices, such as debugging, are included.

Lye and Koh [10] found that research on computational concepts dominated over computational practices (e.g. how students solve programming problems), which again dominated over computational perspectives (e.g. how students talk about what programming means to them or the society). Lye and Koh suggest that both teachers and researchers should focus more on practices and perspectives.

Interestingly, from a Nordic perspective, there is little research on what concepts across fields (including, but not limited to mathematics, natural science, arts and crafts, and music) that are suitable to combine with programming, or whether the integration of programming with these fields is more a question of practice integration.

Modern programming pedagogy is influenced by several, sometimes competing, approaches to the topic of how programming should be taught [2]. One of the main questions is how to structure programming classes. Sentance, Waite & Kallia [15] have identified that one of the most common ways is through traditional lecture style lessons, and also that there are several issues with this teaching style. Moving away from the lecture style approach gives way for more student-active approaches, where students can be encouraged to talk and use other tools.

In the programming industry, using spoken language to debug code was popularised under the term "rubber-duck debugging" back in 2000 [16]. Little research has been done in this field of "talking about code" and reading it aloud in professional and educational settings. Based on the premises of coding being a social activity [9] and that language is one of the most important tools for learning [8], this is a gap in the literature. Some work has been done, however, and several researchers point to the importance of using spoken language to bridge programming activities [10, 15, 17].

The few existing studies have promising results. In their research on what they call code phonology, Hermans, Swidan and Aivaloglou

[18] found that there was a correlation between a student's ability to read code consistently and accurately out load and their general programming knowledge. Kluge et al. [19] found that students could present their own code using screencasts and that the presentations provided a more detailed perspective of the students' understanding than the code would on its own.

Another student-active and interest driven approach is the use of makerspace methodology [20]. Makerspace methodology follows in the line of Papert's learning theory, where students are thought to learn through the construction of physical and digital objects.

Throughout the past decades, we have seen several ideas about what students can learn through programming. They include thinking skills, subject specific and general skills, as well as to teach students about our "digital world". However, most of the research on programming is based on programming for the sake of programming, i.e. to educate professional developers. The Nordic approach assumes that programming can contribute to the learning of other subjects. As is the case with many programming pedagogical topics in school contexts, also the field of programming for the subjects is "underinvestigated" [17, p. 42]. One of the most known cases of such research is on Logo and mathematics [21], but there are some more recent examples.

The project ScratchMaths has shown promising results in using Scratch to teach primary school children basic mathematics skills [22]. In their approach, mathematical and programming concepts were taught "simultaneously", using subtle colour coding to help students differentiate between the two subjects and help them see the connections. This is an important point, as Mørch and colleagues [20] found that students do not automatically connect programming concepts with the relevant school subject(s) if this is not explicitly pointed out to them.

As presented in this section, the programming literature has several interesting lines of research. Since I am applying a qualitative, explorative approach in this project, and I am still at an early stage of my project, I prefer to keep an open mind as to what lines I will pursue later based on the affordances of my data.

4. Research design and method

This qualitative research project is based on data from two cases that represent different approaches to programming in Norwegian schools. See Table 1 for reference. Both cases involve the empirical study of programming interventions in Norwegian schools, and follow design-based research methodology [23].

The first case is situated in the elective programming subjects in Norwegian secondary and upper secondary school, and the purpose of the case to explore the first and main research questions. We employed a digital tool called Scrimba, which is an instructional tool, a code editor, a screen recording tool, and a learning management system, and, in our case, a research data collection tool.

Students and teachers from six schools participated in the intervention. We explore the making and use of screencasts (screen recordings) in different ways, for example to structure lessons and in assessment. The screencasts capture the students' programming activities as a process, including how the students describe and discuss their code.

The second case involves underachieving gifted/talented students attending a natural science class intervention where they incorporate programming and making in science. The aim of this case is to explore the second and main research questions. Potential participants are tested using the Wechsler Intelligence Scale for Children (WISC) test, to identify students who can be defined as underachieving gifted/talented students, but this is not emphasised in my PhD project.

During the intervention, the students are invited to make digital and physical programmed artefacts with the aim of developing understanding of natural science concepts. Approximately 40 students participated in the first iteration, and more are recruited for the second iteration, which is starting during the autumn of 2021.

As both research projects are design based projects, I aim to contribute to both theory development and the development of pedagogical practices that are more "hands on" useful for the practice community.

Table 1
Case comparison

	Case 1	Case 2
Student age	13-19	12-16
Programming rationale	Programming as subject	Programming as learning tool
Context	Elective course in school	Elective course for gifted students across schools
Main pedagogical tools	Interactive screencast technology	Makerspace technology
Data collection	Video/audio recordings in classrooms, semi-structured interviews, screencasts from screencasting software	Video/audio recordings in classrooms, semi-structures interviews, screen recordings from digital classroom environment
N (students)	134	~200

4.1. Data collection

Data from both cases is/was collected using participant observation, screen recordings and interviews. Observations are collected using field notes (meta-data), video cameras, microphones, and screen recording software. This will enable me to capture both what the students are saying, with whom they are talking, how they use their bodies/gestures to communicate, what digital and physical objects they are interacting with as well as what they are constructing. It is vital that the students are encouraged to interact and work together in order to capture these conversations. The student assignments are designed for working in pairs to assure that I may collect interaction data, but in the first case, there are also students who have worked alone and have recorded their own, individual screencast explanations.

In both cases, we used (or intend to use) a voice- and tool-focused approach to video recordings, informed by our theoretical perspective. This is achieved by a particular focus on the relative placement of video and audio recording hardware in the classroom, where cameras are placed so that we capture events on the students' screens and the shared physical space between the students and their persons, enabling us to capture e.g. gestures and how the students potentially move the shared laptop computer or other physical tools between them. A table microphone ensures good quality voice recordings.

Interviews held individually and/or in groups using a semi-structured approach, may provide a meta-cognitive perspective.

The first case is formally concluded, meaning no more data is collected. Data collection in the second case started during the autumn of 2020, and there is available data from the pilot project that is relevant [20]. Because of the Covid-19 pandemic, the 2020/2021 academic year interventions in the second case were conducted digitally, providing considerable challenges forcing all case participants to adapt. This has also affected my project and research questions. We have started conducting the next iteration in a physically co-located classroom, which may provide opportunities for comparing the iterations and cases on even more conceptual levels, which I have not started exploring as of now.

4.2. Data analysis

The data will be analysed using a qualitative approach. I will look at interactions themselves (i.e. the contents and organisation of conversations and other social acts) using interaction analysis (IA) [24]. Typically, this means to look for recurring and/or exceptional "episodes" and sequences of turn taking contributing to meaning making, and organising them into themes that conceptualise the events in the episode [25]. However, as the students are interacting with digital and physical tools and may be using gestures (both physically and digitally) to communicate, these actions are also considered parts of the interaction to analyse. This is in line with a Vygotskyan view on mediational tools as essential parts of learning processes.

The primary data therefore consists of the video observations and screen recordings, as these best capture the complex processes we are studying. The interviews are a secondary data source that may support or challenge what we observe in the classrooms.

As the two cases include relatively large amounts of data (tens of hours of video data), it will be necessary to reduce the data to those that are most relevant for the research questions. This means that I will focus on data were the students are actively engaged in programming, over episodes that are e.g. mainly teacher oriented or where the students are engaged in other types of activities.

Both the cases are parts of larger research projects where other researchers employ several analytical tools and data sources to answer different research questions. My project differs in that I employ the same analytical tools across the two cases.

4.3. Research quality

Although there is some overlap, the cases have distinct takes on programming in school. Instead of viewing this as mainly a challenge, the cases provide an opportunity to investigate contrasting approaches to programming pedagogy.

One challenge, particularly about generalisation to the general population of students who are expected to learn programming within the mandatory subjects following the new curriculum, is that the participants do not represent "typical students" in the Norwegian school, as they have all opted in to take part in the elective programming subjects. Furthermore, all the students in the second case belong to the group of underachieving gifted/talented students. This brings about some methodological challenges, but also the opportunity to study programming with students that are likely to be motivated. It is possible to assume the challenges we might experience with the participants can be even bigger when programming is implemented in mandatory education for everyone.

In the second case, the coronavirus pandemic had a big impact on the first iteration of the interventions. This has provided an opportunity to study the learning of science concepts using digital tools such as "Microbits" and programming, in a digital classroom, but there are challenges on how the data from the digital iteration will compare with the second round.

One way we ensure the research quality in the complex case contexts, is by developing codes and then viewing data separately as researchers to ensure a level of inter-coder reliability.

5. Preliminary results and discussion

In this section, I will briefly describe my preliminary findings and discuss these and the current state of the project. I will frame this discussion using the research questions, starting with the sub-questions and moving on to the main research question.

Sub-question 1: How does interactive screencast technology support digital and social learning practices in computer programming classes?

In the first case, we are exploring affordances of different modes of using integrated screencast technology [19]. The most promising results include how making screencast code presentations may create new learning opportunities for the students, as presented in our short-paper [26]. We have observed episodes where students work collaboratively on developing code and how switching to a screencast recording "mode" of working, e.g. creating a screencast as cultural tool, changed how they talked, edited and tested code. Recording a screencast is not simply a representation of a learning process, but is connected to particular cultural practices. This interrelationship between activity framing, talk, code changes and other development actions will be explored further, and is especially interesting for comparison with the case where another level of abstraction is added, namely the explicit goal of subject learning through programming.

Sub-question 2: How are learning processes supported by programming as an intermediate tool between physical making and conceptual knowledge in a digital science classroom?

Although the digital classroom of the Covid-19 pandemic has caused several problems such as technical difficulties, students dropping out, and changes to the activities in the intervention, we have seen signs of how programming may be a bridge between the individual, concrete, physical artefacts the students made, and the social and digital classrooms where interactions and teaching took place. The students could not manipulate other students' physical artefacts or work together on creating common physical artefacts as they would in a physical classroom,

but they could share and manipulate code in the online classroom environment [27]. I will continue to explore the role of programming and screen sharing practices as tools for supporting the students' learning.

Main research question: How do computer programming classes and integrated subject/programming classes compare as learning arenas?

With this research question, I intend to compare the two approaches to programming (traditional approach, and Nordic approach), and explore in what ways they differ and how the interdisciplinarity of the Nordic approach is expressed through the students' learning processes, and how this differs from the traditional approach.

In some respects, the pandemic made the cases more similar, as the collaboration activities in both cases were, in large, mediated by what the students saw and did on the screen.

Currently, data from the two cases are being analysed separately, but I intend to do a comparative analysis once I am more familiar with the separate data sets.

Preliminary findings are mostly empirical, but with deeper analysis, I hope to develop these into more refined models or theories, that may contribute both to the research field of learning to program and programming to learn, but also the practice of how and why.

6. References

[1] Norwegian directorate of Education, the, Nye læreplaner – grunnskolen og gjennomgående fag vgo, 2019. URL: https://www.udir.no/laring-og-trivsel/lareplanverket/Nye-lareplaner-i-grunnskolen-og-gjennomgaende-fag-vgo.

[2] J. Waite, Pedagogy in teaching Computer Science in Schools: A Literature Review, 2018. URL: https://royalsociety.org/~/media/policy/projects/computing-education/literature-review-pedagogy-in-teaching.pdf

[3] S. Papert, Mindstorms: Children, computers, and powerful ideas. Basic Books Inc, 1980.

[4] S. Bocconi, A. Chioccariello, J. Earp, The Nordic approach to introducing Computational Thinking and programming in compulsory education. Report prepared for the Nordic@ BETT2018 Steering Group (2018) doi:10.17471/54007.

[5] N. C. Brown, S. Sentance, T. Crick, S. Humphreys, Restart: The resurgence of computer science in UK schools. ACM Transactions on Computing Education 14 (2014): 1–22.

[6] J. M. Wing, Computational thinking. Communications of the ACM 49 (2006): 33–35.

[7] A. V. Aho. Computation and Computational Thinking. Ubiquity symposium (2011) doi: https://doi.org/10.1145/1922681.1922682

[8] L. S. Vygotsky, Mind in society: The development of higher psychological processes. Harvard university press, 1980.

[9] K. Nygaard, Program development as a social activity. In IFIP Congress (1986): 189–198.

[10] S. Y. Lye, J. H. L. Koh, Review on teaching and learning of computational thinking through programming: What is next for K-12?. Computers in Human Behavior, 41 (2014): 51–61.

[11] I. E. Harel, S. E. Papert, Constructionism. Ablex Publishing, 1991.

[12] R. E. Mayer, J. L. Dyck, W. Vilberg, Learning to program and learning to think: what's the connection?. Communications of the ACM 29 (1986): 605–610.

[13] M. Resnick, Lifelong kindergarten: Cultivating creativity through projects, passion, peers, and play. MIT press, 2017.

[14] F. J. García-Peñalvo, J. Hughes, A. Rees, I. Jormanainen, T. Toivonen, D. Reimann, M. Tuul, M. Virnes, Evaluation of existing resources (study/analysis). Belgium: TACCLE3 Consortium. 2016 doi:10.5281/zenodo.163112

[15] S. Sentance, J. Waite, M. Kallia, Teaching computer programming with PRIMM: a sociocultural perspective. Computer Science Education 29 (2019): 136–176.

[16] A. Hunt, D. Thomas, The Pragmatic Programmer. Boston: Addison-Wesley, 2000.

[17] S. Grover, R. Pea, Computational thinking in K–12: A review of the state of the field. Educational researcher 42 (2013): 38–43.

[18] F. Hermans, A. Swidan, E. Aivaloglou, Code Phonology: an exploration into the vocalization of code, in: Proceedings of the 26th Conference on Program

Comprehension, 2018, pp. 308–311. ACM.
[19] A. Kluge, K. T. Litherland, P. H. Borgen, G. O. Langslet, Combining programming with audio explanations, in: Proceedings of the 11th International Conference on Education Technology and Computers, 2019, pp. 155–159.
[20] A. I. Mørch, K. T. Litherland, R. Andersen, End-User Development Goes to School: Collaborative Learning with Makerspaces in Subject Areas. In International Symposium on End User Development, 2019, pp. 200–208.
[21] I. E. Harel, S. E. Papert, Software design as a learning environment, Interactive learning environments 1 (1990): 1–32.
[22] L. Benton, P. Saunders, I. Kalas, C. Hoyles, R. Noss, Designing for learning mathematics through programming: A case study of pupils engaging with place value. International journal of child-computer interaction 16 (2018): 68–76.
[23] S. Barab, K. Squire, Design-based research: Putting a stake in the ground. The journal of the learning sciences 13 (2004): 1–14.
[24] B. Jordan, A. Henderson, Interaction analysis: Foundations and practice, The journal of the learning sciences 4 (1995): 39–103.
[25] V. Braun, V. Clarke, Using thematic analysis in psychology. Qualitative research in psychology 3 (2006): 77–101.
[26] K. Litherland, A. Kluge, A. I. Mørch, Interactive Screencasts as Learning Tools in Introductory Programming, in: Proceedings of the 16th European Conference on Technology Enhanced Learning, 2021, pp. 342–346.
[27] R. Andersen, A.I. Mørch, K. T. Litherland, Learning Domain Knowledge using Block-Based Programming: Design-Based Collaborative Learning, in: Proceedings of the 8th International Symposium on End-User Development, 2021, pp. 119–135.