Paper submitted to EUD-2019: The 7th International Symposium on End-User Development, *Hertfordshire, UK, 10-12 July, 2019*

End-User Development goes to School: Collaborative Learning with Makerspaces in Subject Areas

Anders I. Mørch[1], Kristina Torine Litherland[1], Renate Andersen[2]

[1]Department of Education, University of Oslo, Norway
[2]Dept. of Primary and Secondary Teacher Education, Oslo Metropolitan University
Email: andersm@iped.uio.no, kristitl@iped.uio.no, renatea@oslomet.no

Abstract. Norwegian K-12 curriculum reform for 2020 aims to integrate programming in different subject areas, especially math, natural sciences, arts and crafts, and music. There are challenges and opportunities associated with this scenario. A challenge is that students need to learn two topics simultaneously, and an opportunity is that teachers can adopt computer science skills gradually by building on their domain expertise and the notion of different levels of modification since most teachers are not yet fluent in computer science. We present an exploratory case study to show that end-user development (EUD) is a possible solution for the Norwegian situation. The case study demonstrates evidence of collaborative learning with EUD in a makerspace in an advanced placement science classroom for a mixture of gifted underachievers and high-achievers.

Keywords: Collaborative Learning, Empirical Research, End-User Development, Makerspace, Programming (blocks vs. text), School, Qualitative Study.

Introduction

In many European countries, educational policy is under way to integrate computer science (programming, coding, computational thinking) in schools. The report "The Nordic approach to introducing computational thinking and programming in compulsory education" [3] reveals that in the last five years, programming or coding has emerged as a skill that young people should have. As a result, computational thinking [5] has emerged as a concept to prepare children for future challenges in an increasingly digital society [3].

EUD originated as an umbrella term for research and development in end-user tools for application development, such as spreadsheets [8] and design environments [4]. EUD researchers were inspired by easy-to-use programming languages and tools to improve computer applications, such as visual languages [15], scripting languages, and multiple representations [12]. EUD gained broader visibility and became a research topic with its own agenda in the European EUD-Net project (2002–2003), which defines EUD as "a set of methods, activities, techniques, and tools that allow people who are nonprofessional software developers, at some point to create or modify a software artifact" [14]. EUD researchers have developed tools and environments

and tested them in laboratories, organizations, and homes. We study EUD in educational institutions (schools).
The "gentle slope" to programming is a relevant concept when using EUD in education. Namely, to modify an application through its user interface, end users should only have to increase their knowledge by an amount in proportion to the complexity of the modification [9]. Furthermore, simple modifications should not require programming, and more complex modifications should be possible with user-oriented programming languages and higher-level building blocks [4, 9]. To ease the burden of programming for novices, two techniques are useful: "direct activation" and "different levels of tailoring." Direct activation means accessing tailoring tools from the ordinary user interface with a simple keyboard command. Different levels of tailoring refers to interfaces for making changes at different levels of complexity or to views of an application, ranging from editing attribute values of visual objects to creating new behavior by integrating high-level building blocks, to creating new behaviour by general-purpose programming [10].

The usefulness of EUD for educational purposes can be assessed with regards to the ways these environments balance programming and domain-orientation: on one hand, providing the right amount of flexibility for making changes to software artifacts possible, and on the other hand, ensuring usability in specific domains of teaching and learning. We address the following research questions:

1. How can EUD help end users in makerspaces create their own software artifacts?
2. How can EUD help teachers engage students in subject-related learning activities?
3. What theoretical frameworks help to integrate the learning activities (RQ1+2)?

We use empirical data and review previous work to address the research questions. The rest of the paper is organized as follows. In Section 2, we review related research. In Section 3, we describe the case study (method, participants, data analysis), and in Section 4, we discuss our results by answering the research questions. Finally, we summarize our findings and suggest some directions for further work.

Literature Review: History, Scope, and Focus

Three lines of previous work have stimulated our research. First, domain-oriented, programmable design environments are software applications consisting of a domain-oriented user interface in the foreground and a programming environment in the background [4, 15]. Second, visual programming languages starting with the BLOX methodology [7] and Fabrik user interface [6] made it easier for non-expert and disabled users to learn textual programming languages (Pascal, Smalltalk, or C) by using direct manipulation (drag-and-drop program structures analogous to solving a jigsaw puzzle). Third, we were inspired by the line of research that started with Papert's work on Logo [13] and continued with Resnick, who developed new user-oriented languages and led the development of Scratch; pioneered connecting VPLs to physical components in construction kits [16]; and provided an online library of reusable applications, such as games and animations [17].

In recent years, there has been a growing interest in programming as part of a school context. This is based on the understanding that programming is a necessary competence for the 21st century. Consequently, to enable more people to learn programming, there has been an increased focus on programming languages that are more visual. Visual programming languages, and more recently block-based programming, enable young people to construct running programs by greatly simplifying the interface [1]. The environments or tools along with a repository of helpful examples available through the Internet are an important success factor in the continued interest in programming in schools.

We focus on visual programming that learners find interesting and teachers consider relevant. An example is a toy vehicle connected via a circuit board to motors and sensors that children use with other children during play, which is a type collaborative learning environment for combining discursive and hands-on activities. Programming in these environments allows the learners to control motors, sensors, lamps, and so on, and some of the languages are Scratch, MakeCode with Micro:bit, and Blockuino (Blockly with Arduino). We have used Blockuino and Micro:bit in this case. We argue that with access to a range of tailoring tools between user interfaces and program code, learners can modify software artifacts at different levels of abstraction to solve personally interesting problems and at the same time learn subject matter knowledge.

Bevan [2] argues that makerspaces in educational settings need better pedagogical foundation. According to Bevan, key aspects of such pedagogy are design failure and sense making, borrowing two terms from Papert's Constructionism [13]. Bevan suggests that "assembly activities" (step-by-step instructions) should precede "creative construction" (open-ended exploration), as the former are often a prerequisite for successful transitioning to the latter. Bevan says less about the role of the teacher in a makerspace classroom but argues that schools can participate in this important area. We argue that makerspaces must be aligned with teachers' practices, so that the two agendas meet (students' interest-driven learning and schools' responsibility for a shared curriculum).

Makerspace Case Study: Method, Participants, Data Analysis

Our study is an exploratory case study employing qualitative methods for data collection and analysis [18]. The aim of the study is for middle school pupils to learn about and engage with science, but the makerspace course is only loosely connected with the official curriculum in the science subjects. The pupils had all been identified as either gifted underachievers or high-achievers. Every second week, they were taken out of school to participate in the makerspace for four hours arranged at a nearby high school. The makerspace teachers are science and technology enthusiasts.

The participants (N = 19) were pupils in school years 7 to 10 (12–15 years old). We interviewed 17 of the pupils, using a semi-structured interview guide (246 min). We observed six makerspace sessions and wrote field notes using pen and paper. Two of these sessions were video-recorded (535 min, one shown in Fig. 1). To collect and manage the data (spoken utterances, video footage, and verbal interviews), each session and interview was stored in a separate file and transcribed. The pupils were assigned a reference number stored in a table separate from other data. Three researchers were working together to categorize the data: first, according to an open coding process (data-driven) and then informed by our research questions. We identified a number of data extracts according to a number of codes and we have reproduced two extracts below, representative of our findings, and formatted for a short paper.

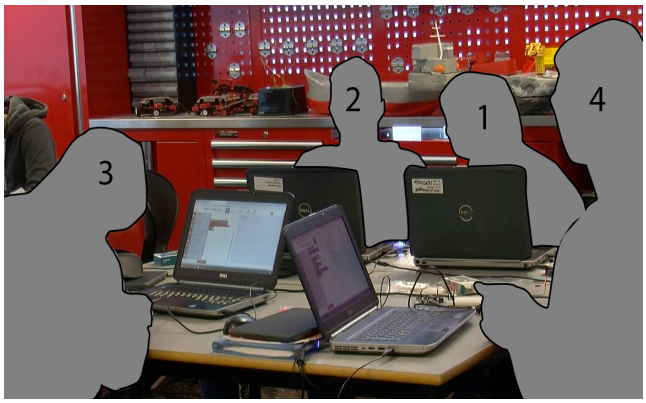

Fig. 1. Setting of makerspace case: Four pupils (aged 12–15) working to solve an assignment to program an Arduino board to light lamps and play music. The numbers refer to the informants.

The extracts are named "programming (blocks vs. text)" and "subject area relevance." In the first extract, the pupils were given a brief introduction to Arduino and Blockuino, a block-based programming language made from Blockly, which generates C code and communicates with an Arduino circuit board to control lamps and sounds. The pupils have created code to make the lamps flash in different patterns and start to create light and sound patterns when we enter here (95 min into the session).

Extract 1: Programming (Blocks vs. Text)

Pupil 3: It seems tedious to write down all the commands as text.
Pupil 1: It is much better to write code in text than to use blocks.
Pupil 4: Yes, it is indeed a lot of work [supporting Pupil 3].
Pupil 1: I think it is hard to find the blocks and how they fit together . . .
Pupil 2: I think blocks are the better alternative.
Pupil 1: That's because you don't know how to write code; when you do, you'll find it's better.
Pupil 3: Nooooo, it's not!
Pupil 1: Okay, right.

The four pupils in Extract 1 debate what is better: textual or block-based programming (see Fig. 2). Pupil 1 argues for text-based programming, whereas the other three prefer blocks. Pupil 1's argument against blocks is that it is difficult to find the blocks in the inventory and combine them into working assemblies; he replies to Pupil 3 that the latter does not know how to program if he does not write program text, but Pupil 3 is not convinced, and Pupil 1 eventually stops arguing.

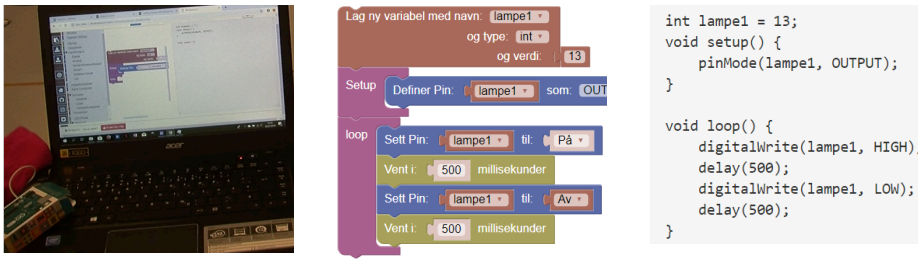

Fig. 2. Bridging the gap between hardware and software by EUD with interfaces at different levels of abstraction: Arduino board (wiring physical components), Blockuino (software blocks to program Arduino in Norwegian), and C (textual code generated from blocks assembly).

The second extract is an interview with a pupil who was working on a Micro:bit project concerning the measurement of velocity of egg in free fall without breaking it.

Extract 2: Subject Area Relevance

Researcher: Is there anything you don't like in the makerspace?

Pupil 14: I am not sure. It could be that we are not learning enough about subjects, in a way. We do not stick to a topic over time. It's like . . . You now . . . I know what we do is based on math and natural science or sciences in general [referring to the latest assignment to compute the velocity of an egg in free fall with techniques to lower acceleration], but we have so much freedom to explore . . . [seven minutes later in the transcript]

Researcher: . . . can you come up with any suggestions for how teaching could be different?

Pupil 14: It would be good to have subject-related knowledge better integrated with our exploratory activities. For example, the project we do now, we could also be learning about velocity and acceleration and similar things: how to compute values and set up formulas, or the way Micro:bit works out its computations, or whatever is happening with what we are doing.

Pupil 14 is one of a few we have interviewed who explicitly identifies a shortcoming in the advanced placement course, as most of the other pupils are more than happy to engage in explorative learning assignments based on their own interests and pace.

However, we found this criticism relevant to bring up, as it echoes a concern shared by many science teachers who are not makerspace enthusiasts. Pupil 14 would like a better integration of subject area knowledge and explorative makerspace learning activities to constrain the space of possibilities, which for many seem endless. Measures can be taken to address the issues, as revealed above: teachers pointing out relevant concepts and formulas when there is an opportunity for engaging pupils in deepened understanding, but it requires common background knowledge, which was not the case in our study as the pupils came from school years 7 to 10.

General Discussion

We discuss our results in terms of the three research questions we asked in the beginning of the paper and by comparing our results with those reported in the literature.

How can EUD Help End Users in Makerspaces Create Their Own Software Artifacts?

The tasks required the pupils to create different types of software artifacts and they were highly motivated to do so. However, the pupils' background and skills in programming varied as they came from different schools and class levels. EUD can help their entrance to programming by multiple representations or levels of abstraction [10]. For example, Extract 1 illustrates how students modified software by composing blocks (integration) and textually (extension) to program an Arduino board. When the technology provides interfaces at different levels, i.e., hardware and software and between block-based and textual programming, it makes the tasks easier for novice users. The notion of mixed textual/graphical interfaces was introduced in the BLOX program methodology [7] but it did not attract a large user population of its time. Our informants were all novice programmers and the majority of them (especially the youngest) preferred block-based programming. Some of the older pupils, such as Pupil 1 in Extract 1, preferred to write code, but the quality of the written code varied among the more experienced pupils.

How can EUD Help Teachers Engage Students in Subject-Related Learning Activities?

The organization of teaching was not optimized for engaging the pupils in curricular learning. They are middle school students who were taught outside the curriculum at a nearby high school. EUD can help to bridge the gap between these pupils' interests and curricular learning with the notion of "domain orientation," which means the artifacts the users interact with should resemble both the artifacts associated with a subject area (e.g. electric motors) and learners' prior experiences and interests (e.g. toy vehicles). Many of the artifacts the pupils interacted with revealed such connections or the potential for making connection. One example is the electric motor of a robot car, which uses the theory of electromagnetism to oscillate. Extract 2 shows that scaffolding could have helped the pupils to connect subject area knowledge and technological artifacts. Pupil 14 gave an example of how the teacher could intervene to constrain exploratory learning in the makerspace by providing scientific formula and

concepts to help the pupils better understand the principles of what they were making and measuring (in Extract 2, Pupil 14 refers to the concepts of velocity and acceleration).

What Theoretical Frameworks Help to Integrate the Learning Activities (RQ1+2)?

Constructivism and sociocultural theory are two theories with concepts for understanding the different aspects of learning with technology we have observed. Constructivism proposes that knowledge is not passively received but actively built on an individual's prior experiences. It also considers the main function of cognition as adaptive in order to organize and make sense of the experiential world. Constructionism developed by Papert and colleagues [13, 16] builds on Piaget's constructivist theory and emphasizes the active role that students can take in constructing their own learning through hands-on activity with physical or visual objects. An aim is to create "objects to think with," which are computational artifacts embedding culture, knowledge, and personal identity [13, 16].

The sociocultural perspective on learning is concerned with learning by social interaction and scaffolding and originated with Vygotsky. The "tool and sign" concept [19] is especially interesting in this regard. On the one hand, in human development, there is a focus on tools and actions involving them as part of practical work, and on the other hand, there are (intellectual) concepts or verbal means to make sense of the actions without using the tools.

Building on the tool-sign dichotomy, the evolving artifacts framework (EAF) conceptualizes learning with technology as a combination of technology adaptation (tool development), and knowledge adaptation as sign/concept development [11]. The former is synonymous with EUD, and the latter is development of shared knowledge in a small group of collaborators, which is mainly a verbal activity but includes the use of EUD tools. Whereas EAF focuses on novice learners and their transitioning to experienced learners by contributing to the evolution of two types of objects (technology and shared knowledge), Constructionism suggests that one learns by building objects-to-think-with, which we consider integrated objects and a goal of evolving artifacts. This is currently a hypothesis that will be investigated in more detail in further work.

Conclusions and Directions for Further Work

We have reported from a case study in end-user development in an educational setting, a makerspace with learning activities involving computational technology at different levels of abstraction, from hardware to software, and using block-based and textual programming, with a mixture of gifted underachievers and high-achievers. We have collected video data of pupils' interaction with each other and with the technology, and we conducted interviews afterward. We have found tentative evidence that intermediate representations (e.g. block-based programming) can aid learning of more advanced programming and that pupils prefer it when they can see the relevance of

what they do in school to their actual lives. Future work includes a more rigorous longitudinal study at different schools with an observation protocol based on theoretical frameworks for coding verbal data in collaborative learning activities in makerspaces, taking EUD and collaborative learning into account, as separate processes and as integrated process.

Acknowledgements

The authors are grateful to Research Council of Norway (RFF) for funding GT-Make project. We thank Elena Biuso for helping to collect and transcribe data, and Ellen Egeland Flø and Louise Mifsud for comments on an earlier version of this paper.

References

1. Armoni, M., Meerbaum-Salant, O., Ben-Ari, M.: From Scratch to "real" programming. Trans. Comput. Educ. 14(4), Article 25 (2015).
2. Bevan, B.: The promise and the promises of making in science education. Studies in Science Education 53(1), 75-103 (2017).
3. Bocconi, S., Chioccariello, A., Earp, J.: The Nordic approach to introducing Computational Thinking and programming in compulsory education. Report prepared for the Nordic@Bett2018 Steering Group, http://www.itd.cnr.it/doc/CompuThinkNordic.pdf (2018).
4. Fischer, G., Girgensohn, A.: End-user modifiability in design environments. In: Proceedings CHI'90, pp. 183-192. ACM, New York, NY (1990).
5. Grover, S., Pea, R.: Computational thinking in K–12: A review of the state of the field. Educational Researcher, 42(1), 38-43 (2013).
6. Ingalls, D., Wallace, S., Chow, Y., Ludolph, F., Doyle, K.: Fabrik: A visual programming environment. In: Proceedings OOPSLA'88, pp. 176-190. ACM, New York, NY (1988).
7. Kopache, M.E., Glinert, E.P.: C2: A mixed textual/graphical environment for C. In Proceedings Workshop on Visual Languages, pp. 231-238. IEEE Press, Los Alamitos, CA (1988).
8. Lewis, C.: NoPumpG: Creating interactive graphics with spreadsheet machinery. University of Colorado, Department of Computer Science, Technical Report CU-CS-372-8 (1987).
9. MacLean, A., Carter, K., Lövstrand, L., Moran, T.: User-tailorable systems: Pressing the issues with buttons. In: Proceedings of CHI'90, pp. 175-182. ACM, New York, NY (1990).
10. Mørch, A.: Three levels of end-user tailoring: Customization, integration, and extension. In: Computers and Design in Context, pp. 51–76. MIT Press, Cambridge, MA (1997).
11. Mørch, A.I., Caruso, V., Hartley, M.D.: End-user development and learning in Second Life: The evolving artifacts framework with application. In: Paterno, F., Wulf, V. (eds.) New Perspectives in End-User Development, pp. 333-358. Springer, Berlin (2017).

12. Mørch, A.I., Mehandjiev, N.D.: Tailoring as collaboration: The mediating role of multiple representations and application units. Computer Supported Cooperative Work 9(1), 75-100 (2000).
13. Papert, S., Harel, I.: Constructionism. Ablex Publishing Corporation, Norwood, NJ (1991).
14. Paterno, F., Klann, M., Wulf, V.: Research Agenda and Roadmap for EUD. Technical report. IST-2001-37470, EUD-Net Network of Excellence, December (2003).
15. Repenning, A., Sumner, T.: Agentsheets: A medium for creating domain-oriented visual languages. IEEE Computer 28(3), 17-25 (1995).
16. Resnick, M., Martin, F., Sargent, R., Silverman, B.: Programmable bricks: Toys to think with. IBM Systems Journal 35(3), 443-452 (1996).
17. Roque, R., Rusk, N., Resnick, M.: Supporting diverse and creative collaboration in the Scratch online community. In: Cress, U. et al. (eds.) Mass Collaboration and Education, pp. 241-256. Springer, Berlin Heidelberg (2016).
18. Silverman, D.: Doing qualitative research: A practical handbook. Sage, London (2005).
19. Vygotsky, L.S., Luria, A.R.: Tool and symbol in child development. In: Valsiner, J., van der Veer, R. (eds.) The Vygotsky Reader, pp. 99-175. Blackwell, Oxford, UK (1994).