Paper accepted to IS-EUD 2021 - 8th International Symposium on End-User Development, July 6-8, 2021.

Learning Domain Knowledge using Block-Based Programming: Design-Based Collaborative Learning

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

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

Abstract. Block-based programming languages have lowered the threshold to computer science (CS), providing a powerful (low threshold-high ceiling) arena for early CS education and engagement in STEM subjects. This paper presents results of an empirical study in three schools, involving 43 pupils aged 12–16 using MakeCode with Microbit (a microcontroller), basic physical objects, and Zoom video communication as a shared learning environment. Using design-based research (DBR) together with teachers, we created technology-rich learning materials and tasks in math, biology, and physics and organized a series of project-based learning activities wherein pupils met three hours per week for 16 weeks during two semesters. Recorded Zoom meetings serve as our data. We thematized and transcribed the video material of selected groups' online activities and used verbal interaction analysis and visual artefact analysis as our methods. Our results include a new analytical framework, design-based collaborative learning (DBCL), achieved by adopting concepts from computer-supported collaborative learning (CSCL) and end-user development (EUD), specifically domain-oriented design environments (DODE). Our empirical findings are: 1) block-based programming in a collaborative context, 2) block-based programming as part of a DODE, 3) block-based programming integrated with school subjects, and 4) block-based programming as an explorative design method.

Keywords: Block-Based Programming, Computer-Supported Collaborative Learning, Design-Based Collaborative Learning, Design-Based Research, Domain-Oriented Design Environment, End-User Development, Programming in School.

1 Introduction

In the Nordic countries, it has been suggested that programming and computational thinking (CT) in K–12 should not be taught as a separate subject but as part of a more comprehensive twenty-first century skills approach sometimes labelled as digital competence [3]. As of the autumn 2020, our country has implemented a curriculum where programming and CT have been integrated in other subjects, in particular mathematics, natural science, music, and arts and crafts. This unique situation calls for research on the implications of the new, technology-enriched curriculum to determine how it works in practice.

End-user development (EUD) is referred to as a set of methods and techniques that allow people who are nonprofessional software developers to create or modify a software artefact [18], as the opposite of technical development carried out by trained programmers and software engineers. It includes, among others, visual programming [23], domain-oriented design environments [10], and programmable applications [9].

The usefulness of these EUD environments can be assessed according to access, flexibility, and purpose. *Access* means the extent to which the EUD environment provides a gentle slope to modification and programming [21, 28]. *Flexibility* refers to the extent to which the tools are low threshold and high ceiling, allowing a large number of interesting artefacts to be created [9, 23, 24]. Furthermore, access and flexibility should be measured against *purpose* (e.g., solving technical problems, practicing computational thinking, or learning science, technology, engineering, and mathematics (STEM) topics) [10]. Our aim is to understand block-based programming as a tool for explorative learning of STEM topics. Therefore, we asked the following research question: What characterizes block-based programming as an explorative design space to learn STEM topics in an online collaborative setting?

The rest of the paper is organized as follows. First, empirical studies of block-based programming are discussed. Next, end-user development and collaborative learning are elaborated upon; this results in an analytical framework, which we use to analyze our empirical data. Finally, we discuss our results by comparing them with related work.

2 Related Work

This section presents a review of the literature related to end-user development and learning: block-based programming, domain-oriented design environments, and computer-supported collaborative learning.

2.1 Brief Review of Literature on Block-Based Programming

Block-based programming is a method of programming in which visual code blocks are combined to create and modify animations and games, and to interact with physical objects (sensors, buzzers, motors, lamps, LED displays, etc.). Popular block-based languages are Scratch, Blocky, Agentsheets, Alice, and MakeCode [23, 24]. The idea of blocks to encapsulate software programs goes back to structured programming with notions such as block structure and nesting (blocks inside blocks) to organize code in editors and compiled code in memory. Today's code blocks draw on children's visual metaphors (e.g., jigsaw puzzle; wooden blocks; lego bricks) and computational innovations like drag and drop interfaces and online repositories of code for reuse. They have lowered the threshold to programming and broadened participation to the entire school system. However, the success of block-based programming is hampered when children use block-based languages without understanding the programming concepts underlying the blocks, which may prevent computational thinking from being achieved.

Our own preliminary observation suggests that children refer to block types by color and talk about the “blue block” and “red block” and “red goes inside blue,” etc., instead of “loops” and “conditionals” and “conditionals are used in loops.” This is supported by affordance because blocks snap together to create syntactically correct programs. Researchers have found that primary school children use color, shape, and arguments (parameters in blocks) as visual cues when searching for blocks, and the children pick up both intended and false affordances when programming [8].

Lewis [17] compared the learning outcome of sixth graders’ coding of similar tasks in Logo and Scratch and found that the two languages were similar in terms of stimulating continued interest in programming, but differed in how affordances for programming varied for different programming constructs. In particular, Lewis [17] found that children understood loops better in Logo, but conditionals better in Scratch.

Based on a series of examples from the Scratch online community, Brennan and Resnick [4] argued that visual programming is more than writing code and leads children to socializing with peers. The authors suggested that computational practices and computational perspectives should supplement computational concepts (e.g., loops and conditionals) toward an understanding of computational thinking as a mass collaboration phenomenon. The authors found that some students were able to engage in computational practice and reuse code to create a new project without fully understanding the underlying computational concepts. The current study builds on this finding.

Based on empirical studies of middle and high school students’ practices with coding in and out of school, Lee et al. [16] suggested a model of computational practice, use-modify-create, which can scaffold children’s acquisition of CT concepts. The model entails children going through a progression toward learning computational thinking. In the use stage, they test existing solutions (e.g., playing a game created by someone else). In the modify stage, they begin to modify the game at different levels of complexity. In the create stage, they reuse the acquired understanding and apply it to create something new.

2.2 Domain-Oriented Design Environments: Visual language vs. Components

Researchers in EUD have been concerned with bringing programming closer to the domain expert users’ practices and needs for more than 30 years, along several dimensions. One dimension is general purpose vs. domain orientated technology support, which includes work in EUD that spans visual programming languages [e.g., 6, 23] to component-based design and end-user tailoring [e.g., 21, 27]. This line of work was reviewed in [20] and is here summarized as a conceptual framework, domain-oriented design environment (DODE). In a DODE, a programming language and software components are combined. A DODE is a software application consisting of a domain-oriented user interface that supports design activities within a particular domain [10], and a programming environment in the background that allows end-user developers to modify and further develop the DODE, such as to introduce new components and rules [11]. Component-based design is accomplished by users when they interact with components in visual builders to select, modify, and connect components using high-level operations rather than writing program code [21, 27]. In a DODE this is accomplished by the end users when they select components from a palette of parts, which are further edited in a work area, stored in a catalog of examples, and incorporated in future applications by reuse and redesign. Block-based programming languages combine aspects of a high-level programming language and component-based design not unlike a DODE. Therefore, we used DODE in a conceptual framework to understand block-based programming in practical use with research methods from the social sciences.

2.3 Group Cognition Review of Computer-Supported Collaborative Learning

Computer-supported collaborative learning (CSCL) is an interdisciplinary research field in which two or more people learn or attempt to learn something together [7]. The underlying premises in CSCL are that people learn together with the help of a computer and that what they learn together is more than what they could have learned independently [26]. The final results of CSCL are shared knowledge and expertise developed by collaborating through the creation of shared artefacts, which can enhance skills in social interaction and collaboration [19]. Thus, the core features of CSCL are: 1) interaction between learners, 2) information sharing, 3) joint meaning making based on negotiation within the group, and 4) developing common artefacts [1, 26]. CSCL research encompasses both co-located and distributed contexts. An empirical study of distributed CSCL referred to as collaborative knowledge creation [1], identified mutual development processes in online collaborative learning. "Mutual development" was first coined in the context of EUD as a joint collaboration process between different stakeholders when they co-create a shared artefact [1, 2]. We use CSCL in order to understand how pupils learn together when working in small groups to create code and physical artifacts to solve subject-specific tasks.

3 Analytical Framework: Design-Based Collaborative Learning

We created an analytical framework derived from central terms in CSCL and EUD. We refer to the combination of EUD and CSCL as design-based collaborative learning (DBCL). Our goal was to understand collaborative learning of knowledge and skills in specific STEM domains (e.g., math, biology, and physics) and to use block-based programming as a method toward that end. From CSCL we focused on the collaborative learning process of creating common artifacts. From EUD we drew on DODEs. This combination of concepts provided a group interaction perspective on domain-oriented design environments, which is new. The concepts in our analytical framework are displayed in Table 1.

Table 1. Concepts of analytical framework

Analytic concepts from CSCL	Analytic concepts from DODEs
Information sharing (from me to you)	Design unit (DU, separate building blocks to be combined with other DUs)
Negotiation (you and I decide what to focus on)	Rule (knowledge of the relations between two or more DUs to form a more complex DU)

Group cognition (shared meaning)	Argumentation base (a priori shared knowledge)
Scaffolding (help from teacher/senior peer)	Example (a previous solution for reuse)

Information sharing is a central element in collaborative learning as it starts all the other processes of meaning making [26]. *Negotiation* is defined as establishing a shared meaning, in which the individual group members have to negotiate multiple personal perspectives to create one that can be shared and to affirm that the meaning is shared [25]. *Group cognition* is a goal of collaborative learning and entails multiple people participating in coherent interactions that achieve cognitive accomplishments that are best analyzed, at least in part, at the group level, rather than attributing contributions and agency entirely to individual minds [25]. *Scaffolding* involves a metaphor in the collaborative learning context describing how teachers and more senior peers support learners by providing feedback and support [12].

From DODE [10], we adopted the following components: *Design units* (DUs) are defined as the basic objects in the design environment. They model specific domain objects as standalone components as well as more technical objects. *Rules* define domain knowledge and consist of desired relations between domain objects. The rules are triggered by actions and events in the environment. Critics are automated scaffolding based on rules, but this component is outside the scope of our work. *Argumentation base* is the design rationale associated with domain knowledge in the form of arguments for and against adding, removing, or replacing design units. *Examples* are previous solutions created by other users in the design environment, which provide ideas and starting points for new users to reuse and redesign. We used the two sets of analytic concepts in an integrated effort as thematic codes to make sense of empirical data in our analysis.

4 Methods

In this section, we describe our research design and techniques for data collection and analyses within the umbrella of design-based research (DBR).

Design-Based Research is an educational research tradition focused on both the development of pedagogical practice and the development of theory. In DBR, researchers, teachers, and other stakeholders typically design educational interventions collaboratively. The interventions are situated in authentic educational contexts and are tested and developed through several iterations [14]. Note that the use of design in DBR (design as intervention) is different from the use of design in DODE (design as creation).

Research Design and Participants. We developed and tested a series of technology-enriched classes for gifted pupils to learn block-based programming as an integral part of their course subject. Four iterations are planned over two years, with one intervention per semester. Based on evaluations and feedback, the DBR process iterates. In total up to 100 pupils aged 12-16 will participate, 50 each semester, divided into four classes. Each class is taught by a high school teacher, which means there are also four participating high school teachers in the research project. Each of the four interventions has a different focus concerning the course subject. All of the interventions have the same time perspective: a duration of eight weeks, with a three-hour class each week.

Materials and procedure. The data derived from the second intervention where the topics were mathematics, biology, and physics, aided by physical components and Microbit. Each week began with the teacher introducing the topic for the half-day and the topic to be learned and then the teacher divided the class into groups (breakout rooms in Zoom) consisting of three to four pupils. The assigned task was to learn a one or more domain concepts by using MakeCode as one of the tools. In order to collaborate with the use of different materials, the teachers encouraged the pupils to share their screens with the others to show their partial solutions. In this way, the pupils had a common ground for the discussions. The group was the unit of analysis in our research.

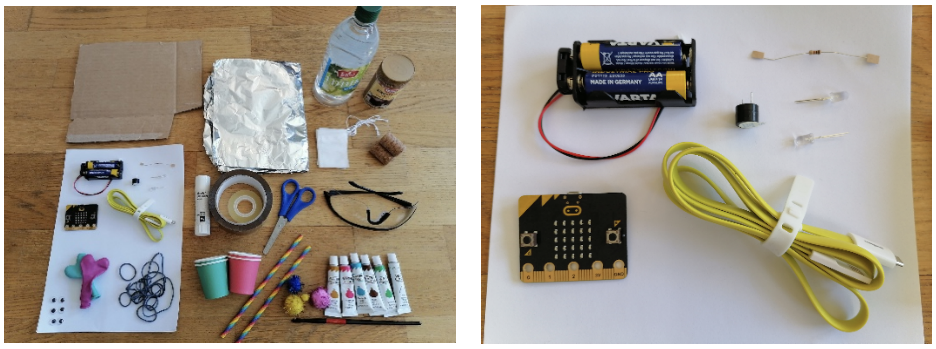


Figure 1. a) The physical materials the pupils had access to and used in some of the assignments. b) Microbit circuit board with electronic components. We referred to each item as a design unit.

Figure 1a) shows the materials the pupils were provided when participating in the second intervention. Figure 1b), is a picture of the Microbit microcontroller (a small computer) with some sensors that can be connected to it. MakeCode is the block-based programming environment used for writing the code that controls the Microbit.

*Selection of participants and context of study***.** The participants in the study were gifted children, who, through nomination from parents/teachers, applied for participation in the research project. The reason for choosing gifted pupils as participants was their need for differentiated learning. As the pupils were aged 12–16, the intervention covered competence goals aimed at high school pupils, that is, one to four academic years above their actual ages.

4.1 Data Collection

When collecting the empirical data, we used participant observation and video recordings of the participants' screens in Zoom. We collected our data in an online context due to the Covid-19 pandemic. Virtual participatory observation is a technique used in virtual ethnography. Virtual ethnography is a method for analyzing social interactions in an online context. Hine [13] stated that virtual ethnography is an ethnographic approach to the Internet. When conducting virtual ethnography, different techniques can be used, ranging from "fly on the wall" (non-participant observation) to engaging in the social interactions when collecting empirical data [13]. We used a participant observation approach wherein we sometimes asked the pupils to explain what they were doing while programming to initiate social interaction [15]. In total, from Intervention 2, we collected around 70 hours of screen recordings of interaction data. The pupils decided whether and when they shared their screens, and toggled their web cameras and microphones at will. Data privacy is an important concern in all research, therefore, we obtained written consent from all the participants' parents/guardians, and the data is stored in encrypted and secure data management servers. In addition, all the extracts presented in this article are anonymized using pseudonyms. The Norwegian center for research data approved this study, ensuring that we collect personal data legally and safely.

4.2 Data Analysis

To analyze the empirical data, the first two authors used a combination of thematic analysis [5] and interaction analysis [15] in a two-step manner. In using this approach, we applied a combination of inductive and deductive coding schemes, defined as an abductive approach [22]. Through collective analysis in data workshops, we focused on the participant's "voice," being inductive, and we also relied on our analytical framework, being deductive.

*Thematic Analysis***.** Thematic analysis is a method for systematically identifying, organizing and offering insight into patterns of meanings (themes) across a dataset [5]. We started by using thematic analysis to comb through our empirical data seeking themes and creating thematic codes that came across as interesting and relevant to the research question. Examples of themes that emerged were block-based programming, collaboration, and programming integrated in the subjects.

Interaction Analysis. Next, we selected empirical data extracts that reflected the themes we wanted to focus on in detail and used interaction analysis to scrutinize the utterances [15]. Interaction analysis is a method for empirical investigation of social interaction between humans and the objects in their environment, including speech, non-verbal actions and the use of artefacts [15].

5 Data and Analysis

In this section, we present empirical data that illustrates how the pupils used blocks to experiment and explore a subject area topic within mathematics, biology and physics. Each section is divided into four subsections: 1) contextualizing the extract, 2) raw data (informants' voices), 3) technology object (software and hardware being composed) and 4) discursive object (a sequence of utterances advancing the groups' common understanding). The transcript notations we used include these symbols: ((*text*)) participants' actions, [*text*] researchers' clarifying comments, (.) long pause in interaction and … pause in interaction. During all the data extracts one of the pupils shared his/her screen so that the other pupils could see what he/she was working on.

5.1 Verbal Data Extract 1: Simulating a Die

Contextualizing the data extract: In the first data extract, the pupils discussed the task of the first assignment, which was to determine how to use the Microbit as a die as part of an exercise related to probability in mathematics. Olivia was sharing her screen.

Table 3. Simulating a die

Turn	Person	Utterance	Analytic Concepts
1.1	Sophia	It's just like… Can the Microbit be used as a die?	Argumen-tation base
1.2	Olivia	Nothing more than that?	
1.3	Sophia	No, that's all! ...	
1.4	Olivia	Should we try making a die then, or what?	Negotiation
1.5	Liam	Okay. Mm. We could actually just do what we did earlier.	Example (idea reuse);
1.6	Olivia	Like… On [Microbit] shake… ((pics purple block from pallet)) Then...	Design unit (1)
1.7	Liam	Then you can add "show number"… ((Olivia picks blue block and puts inside the purple block)) and a... And maths of course. ((Olivia goes to the maths pal-let and searches)) Then we have a random number ((Olivia finds the "pick random" block set to default values 0 and 10)) from 1 to 6 or 1 to 10 ((Liam sees the number 10)), but 1 to 6 ((Olivia changes the value from 10 to 6 in the innermost block)). And this should work like a die.	Rule; Information sharing; Design unit (3)
1.8	Olivia	((Tests Microbit by clicking the "shake" button 5 times)) It works! ((tone of voice is happy))	Design unit (4); Group cognition

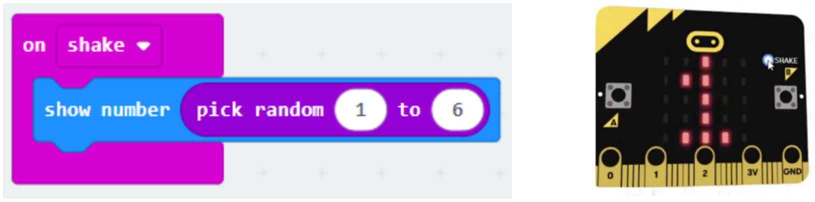

```
on shake ▾
    show number  pick random 1 to 6
```

Figure 2. Visual artefacts for extract 1: A program composed of four design units that simulates a die when the user clicks the shake button. *Left*: Code after turn 1.7. *Right*: Output of program.

Extract 1 illustrates what we mean by two trajectories of participation: technology object development (Microbit and its code) and discursive object development (collab-orative learning conversation). We show how these objects develop in next paragraphs.

Technology Object: When developing the program code, Liam told Olivia what code blocks to choose and she put three design units into the configuration shown in Figure 2 (left image). MakeCode automated the task of creating a syntactically correct program with the "snap" operation (domain general rule). Liam explained the name of two nested blocks of importance for domain knowledge, "show number" and "pick random" (domain specific rules). Olivia changed the default numbers to match the features of a die (ranging from one to six), using another domain-specific rule. Finally, Olivia tested whether the simulated die worked, now consisting of four design units, as shown in Figure 2 (left and right). It is clear that the pupils did not meet any big challenges creating the code, but they revealed great joy of accomplishment based on tone of voice.

Discursive Object: The main aim of the tasks was to learn domain knowledge as part of their group work, in this case probability and random numbers. In collaborative learning, an essential part of the process is that individual group members share information in order to demonstrate shared understanding [26]. Sophia thought the problem was easy and stated the common task (argumentation base). Olivia reformatted the task into a question for the group to start their collaborative inquiry. Liam suggested that they could reuse the code they had used in a previous assignment and Olivia shared her MakeCode screen. Olivia talked with Liam while she composed the program. Important information was shared and negotiated by the two group members, such as when Liam stated that the block "show number" should show a random number from 1 to 6, analogous to throwing dice, thus connecting with domain knowledge. In turn 1.7, Olivia said "It works," referring to the shared visual artifact, a simulated die using Microbit (Figure 2, right image), which indicated that Liam and Olivia, and perhaps Sophia, had collaboratively created shared knowledge.

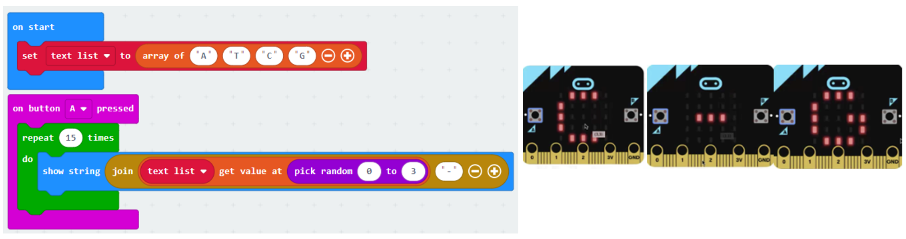
5.2 Verbal Data Extract 2: Programming the Microbit to Print Gene Codes

Contextualizing the extract: This extract is part of a biology topic wherein the pupils were given the task to print out 15 random gene codes. Felix shared his screen so the other group members could see his coding process, but only Matheo engaged with him.

Table 4. Programming gene sequences

Turn	Person	Utterance	Analytic Concepts
2.1	Felix	Okay, we're supposed to make a random sequence of 15 [genes]. ((Places the blue "on start" block. Sets a variable "text-list," to include an array. Modifies values in the array block to A, T, C, G.))	Design unit (3); Information sharing
2.2	Felix	[After 1 minute of group negotiating the nature of the task] Just try to… or? Yes, we'll try. Should we take [this block]? ((Felix picks a new top-level purple input block and places below start block; places a green loop block inside and sets parameter to 15; places a blue output text block inside)) … Mmmm… Yes. ((puts a variable "list" block as output, initialized with a "random value" block.))	Negotiation; Design unit (3+6); Rule
2.3	Matheo	This whole thing about genes and stuff is quite cool.	Design unit (10); Rule;

2.4	Felix	Yes, it’s really fun.	
2.5	Matheo	Hang on, I’m writing [code]… There, wait Felix, I’m still making… Felix, I think I also did it, but the thing is, if you go back to what you had [code]...	Design unit (10)
2.6	Felix	Mm ((Both pupils are working out their code separately.))	
2.7	Matheo	Ehmm then I have… or, you’re using the join thing [block].	Information sharing
2.8	Felix	Oh, it’s not really necessary… it’s just…	
2.9	Matheo	What does it do? Does it just make them [the blocks] stick together, like follow each other?	Negotiation
2.10	Felix	Yeah, actually, I don't think you need it. ((Starts to reflect in his own code.))	
2.11	Matheo	Yeah	Group cognition
2.12	Felix	No, you could really just have used (.) ((Tries to understand the construct.))	
2.13	Matheo	I just wrote [coded] show string…	
2.14	Felix	Yes, that is really all you need to do.	
2.15	Matheo	Then I don’t get those hyphens [to space out the letters output the Microbit] - I just get “ding ding ding” [the gene letters are written out one after the other without hyphens].	Rule
2.16	Felix	Eh, yeah, yeah, because you don’t really need… Yes, he [the teacher] just had hyphens in his code, I think. But you don’t really need the hyphens.	Information sharing
2.17	Matheo	Yeah, but can you try running it [the code]? May I see what it looks like?	Group cognition
2.18	Felix	With this? ((Referring to the join block with the hyphen.))	
2.19	Matheo	Umm, yes.	
2.20	Felix	It just goes like this. ((Points to Microbit simulation)) It just takes forever.	Design unit (11)



```
on start
  set text list ▾ to array of "A" "T" "C" "G" ⊖ ⊕

on button A ▾ pressed
  repeat 15 times
  do
    show string join text list ▾ get value at pick random 0 to 3 "-" ⊖ ⊕
```

Figure 3. Visual artefact table for Extract 2: The program composed of 11 design units displays a random sequence of 15 letter gene codes (here two letters, C & G, are shown in three snapshots).

Technology Object: In turn 2.2, Felix shared his screen and started the task by combining blocks in order to write out the letters A, T, C, and G (abbreviations of the four basic building blocks in an organism's DNA) by initializing an array block. He continued to create the program by searching for blocks to combine the letters randomly. At 10 blocks, he indicated that he had finished by asking: "Does it work now?" (Turn 2.4). The other pupils had worked with their own code, and in Turn 2.7, Matheo asked Felix if he really needed the "join block" because it was not used (or may have had an error). Felix pondered the question, agreed, and changed his code. This illustrates how the pupils helped each other to create the code, despite working on separate versions locally with one being shared as a technology object to talk about.

Discursive Object: When Felix started to create the program as shown in the previous paragraph and Matheo said that he found this way of working "quite cool" (Turn 2.3), it can be interpreted that the pupils perceived their main focus as the science course, using programming as a tool toward learning science. The two boys agreed that a certain block was not needed after some negotiation, with the effect that shared understanding (group cognition) was achieved after first establishing a shared understanding of the code. The utterances in Turns 2.17–2.20 can be interpreted as a collaborative learning process in which group cognition emerged [25]. Shared understanding was achieved during Turns 2.15-20 (producing 15 gene codes as a random combination of the four basic DNA letters) when Felix elaborated that this could be accomplished by using hyphens between the letters, a point emphasized by the teacher in a previous lecture. This achievement was visualized in the program execution in Turn 2.20.

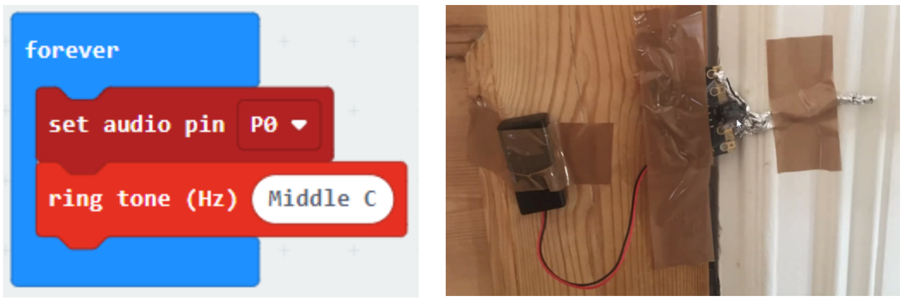
5.3 Verbal Data Extract 3: Microbit as a Burglar Alarm

Contextualizing the extract: This extract is part of a physics topic wherein the pupils were given the task to program the Microbit to be used as a burglar alarm. When doing this, the children were asked to use aluminum foil and connect it to the Microbit to explore how it conducts electricity. The extract below is from an end-of-class summing up session in which the pupils were invited by the teacher to present their projects.

Table 3. Microbit as a Burglar Alarm.

Turn	Person	Utterance	Analytic Concepts
3.1	Teacher	Ok, so what are you up to?	Scaffolding
3.2	David	Yeah, so, we, we have a more practical idea. We really just took the same program we made earlier and modified it, so really when…	Example (code reuse)
3.3	Teacher	Can you show us?	Scaffolding
3.4	David	Yes, it's a very simple program. There, this is our program ((shares screen)). Very simple.	Design unit (3)
3.5	Teacher	But how… I'm eager to hear how you stopped it from [ringing]…	Scaffolding
3.6	David	Since we have connected this buzzer, and sort of if both of the connectors to the buzzer are connected	Rule

		to the aluminum it will stop ringing. So, what we did on my door was, I put the aluminum sort of right next to it [door end] so when you open [the door], it moves away from the aluminum and then it starts ringing.	
3.7	Teacher	So basically—the connection from the Microbit to the buzzer is cut in two, is that what you did?	Scaffolding
3.8	David	Sort of. We took a ((looks at door)).	
3.9	Teacher	Can you film it [with the webcam]?	
3.10	David	It's stuck to my door now. Umm, yes, I'll just turn around the camera. I'll stop sharing [the computer screen]. Here you see it—we've named it "Tacky-Mech," and it starts ringing when it's not in contact with the aluminum ((buzzer sound))) and when it comes back again it stops ((buzzer stops ringing)).	Design unit (3 software and 4 hardware); Rule; Group cognition



```
forever
    set audio pin P0
    ring tone (Hz) Middle C
```

Figure 4. Visual artefact table for Extract 3: The code to the left reads signals from the Microbit to the right (attached to the door), which triggers a buzzer when two aluminum cords are not in contact.

Technology Object: David volunteered to show the solution that he created with a peer when prompted by the teacher. He started with three code blocks from a previous exercise. The code worked by sending a ring-signal to the buzzer when the door was opened. David stated that when the buzzer was connected to the aluminum, the burglar alarm would stop ringing. This is one example of a rule in the design environment, that is, aluminum as electrical conductor. Based on the code alone (Figure 4, left image), one may get the impression that the alarm will ring "forever." David said that the alarm would ring only when the door was opened as this would break the electrical circuit. In Figure 4, we see David's code (left) and the set-up of the Microbit with the buzzer taped to the door (right). This extract is interesting as it is an example of how the pupil used the Microbit as an integrated element of particle physics (electricity) by using aluminum as a conductor in a practical application.

Discursive Object: The teacher prompted the knowledge building that occurred in this conversation four times, by asking questions and scaffolding the pupils' learning process [12]. In response, David explained what he and his peer did in steps that gradually led to group cognition, sharing information about “a more practical idea” (Turn 3.2), with negotiation that led to a “very simple program,” (Turn 3.4) and finally shared understanding of basic circuitry and program logic (understanding why buzzer “starts ringing when it’s not in contact with the aluminum”). The latter (Turn 3.10), is an example of group cognition, as they named their hack “TackyMech.” An essential element in group cognition is when multiple people build on each other's statements and participate in coherent interactions to achieve something beyond their individual efforts [25]. In Extract 3, the interactions were scaffolded by the teacher to achieve group cognition. The teacher asked questions about what the burglar alarm looked like and how it worked, and the pupil explained and reflected upon the process. These questions brought the technology objects into focus.

In sum, in the data we have shown, we identified two types of “objects” that increase in complexity: technology objects increase in the number of design units, that is, parts that can be combined in different ways [10], whereas discursive objects increase in level abstraction, from personal to shared perspectives [25, 26]. Both are important and interdependent.

6 Discussion and Conclusions

The research question asked in this study was: What characterizes block-based programming as an explorative design space to learn STEM topics in an online collaborative setting? We have addressed the question by analyzing three examples in Section 5, using an analytical framework obtained from theories in EUD and CSCL. Application of the framework led to the following empirical results:

- Block-based programming in a collaborative context.
- Block-based programming as part of a domain-oriented design environment reduces the gap between the problem domain and the human-computer interface.
- Block-based programming can be integrated in school subjects.
- Block-based programming as explorative design method for solving domain specific tasks.

Block-based programming in a collaborative context. Our unit of analysis was the group. However, the programs created by the group members were made locally (each pupil had their own MakeCode environment and the physical components kit). To bridge the gap between individual artefacts and group work, we draw on CSCL. Collaborative learning is the discursive element of the analysis, including key collaborative processes such as “sharing information,” “negotiation” and “group cognition” [25, 26].

Block-based programming to reduce the semantic gap. The main aim of a DODE is to move the human-computer interface from resembling interaction with a programming language to interacting with higher-level abstractions, which ends with the problem domains [10]. Our empirical findings turn this model upside-down, starting with the problem domain as given by an assignment and ending with individual technology objects and collaboratively-created knowledge (group cognition). For example, in data

Extract 1, the domain-specific context was mathematics and the problem to be solved was basic understanding of probability (random numbers), which was materialized in the MakeCode/Microbit die.

Block-based programming can be integrated in school subjects. We found across the three examples that pupils could solve the tasks without advanced programming knowledge. We believe a reason for this is the intuitiveness of block-based programming and the automation of programming language syntax. Our findings indicate that pupils, if not explicitly taught programming knowledge, would choose blocks according to color, shape and parameters [8]. The use-modify-create model [16] suggested in the literature applied to the programming part in the three examples, but it did not apply to collaborative learning because it failed to consider domain orientation outside programming. To consider arbitrary (STEM) subject domains, the model could be prefixed with "concepts," whereby one or more domain specific concepts are introduced by a teacher before the pupils start programming with, for example, use-modify-create.

Block-based programming as exploratory method for solving domain specific tasks. Our data illustrates how programming can be used as a method for collaborative learning of school subjects in math and science (STEM) topics. Our research indicates a method to learn STEM subjects. In Extract 3, we see how a pupil solved a physics task by combining knowledge of metal conductivity and programming physical objects. The programming serves as an intermediate stage between the mastery of a curricular goal in physics and the actual physical objects. This can be understood in two ways (bottom up and top down): 1) the physical objects served as stepping-stones to gradually understand the higher-level physics concepts through the medium of programming or 2) the concepts as first taught were then "lowered" to a more concrete level of abstraction by hands-on exploration to ground the concepts. In the creation of a burglar alarm, we observed both top-down and bottom-up understanding at work, which varied among the groups according to preferences in following teachers' instructions (top down) versus doing interesting things themselves (bottom up).

A dilemma of integrating programming as part of the subjects is that the teachers need to split their time in two, that is, teaching and scaffolding the subject and programming. We observed this phenomenon from the point of view of the pupils learning activities by identifying two types of artefacts that developed in parallel during the group work: 1) technical objects and 2) discursive objects. These objects developed in different directions as one is concrete and the other abstract (design complexity in the number of building blocks vs. abstraction of knowledge in terms of the degree of shared understanding held by a group). The design complexity and abstraction of knowledge developed as the teachers gave the pupils tasks involving physical objects related to domain specific topics, and followed it up by scaffolding.

The usefulness of EUD can be assessed according to *access*, *flexibility* and *purpose*. Our empirical studies of block-based programming combine the three criteria. We have leveraged previous work in EUD by low-threshold interfaces supporting component-based design [21, 27] and previous research in high-ceiling block-based programming environments [23, 24]. We contributed by combining access and flexibility with a purpose beyond interest by connecting block-based programming with domain-specific learning activities [10]. Further work includes increasing the accessibility of computational tools and materials by defining them within the same programmable design environment, such as Minecraft, in which specific program constructs can be taught by practical problem solving in the immersive game environment. For educational applications, the use-modify-create model [16] can benefit an initial phase of setting a goal.

Finally, we have developed a new conceptual framework for analyzing collaborative learning in block-based programming by combining concepts from domain-oriented design environments (DODEs) and computer supported collaborative learning (CSCL) based on group cognition. We have named the new analytical framework design-based collaborative learning (DBCL), providing a set of analytic concepts to be used as an observation protocol in empirical research. Further research will harness the protocol.

Acknowledgments

This research is part of the research project “Programming and Making in School” funded by Regional Research Fund (RFF) in Norway. We thank Ellen Egeland Flø for helping in organization of the educational activities presented here.

References

1. Andersen, R.: Mutual development in online collaborative processes. Three case studies of artifact co-creation at different levels of participation PhD dissertation, Faculty of Educational Sciences, University of Oslo, Norway (2019).
2. Andersen, R., Mørch, A.I.: Mutual development: A case study in customer-initiated software product development. In: Pipek, V., Rosson, M.B., de Ruyter, B., Wulf V. (eds.) End-User Development. IS-EUD 2009. Lecture Notes in Computer Science, vol. 5435, pp. 31-49. Springer, Berlin, Heidelberg (2009).
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, pp. 397–400 (2018).
4. Brennan, K., Resnick, M.: New frameworks for studying and assessing the development of computational thinking. In: Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vancouver, Canada, vol. 1, pp. 1-25 (2012).
5. Clarke, V., Braun, V.: Thematic analysis. In H. Cooper (ed.) APA Handbook of Research Methods in Psychology: Vol. 2 Research Designs, pp. 57–71. American Psychological Association, Washington, DC (2015).
6. Costabile, M-F., Fogli, D., Letondal, C., Mussio, P., & Piccinno, A.: Domain-Expert Users and their Needs of Software Development. In Proc. HCI 2003 EUD Session, Crete (2003).
7. Dillenbourg, P.: Collaborative Learning: Cognitive and Computational Approaches. Advances in learning and instruction series. Elsevier Science, Inc., New York (1999).
8. Dwyer, H., Hill, C., Hansen, A., Iveland, A., Franklin, D., Harlow, D.: Fourth grade students reading block-based programs: Predictions, visual cues, and affordances. In Proceedings of the 11th Annual International Conference on International Computing Education Research (ICER '15), pp. 111–119. ACM (2015).
9. Eisenberg, M.: Programmable applications: Interpreter meets interface. SIGCHI Bulletin. 27(2), 68–93 (1995).

10. Fischer, G.: Domain-oriented design environments. Automated Software Engineering, 1(2), 177–203 (1994).
11. Fischer, G., Girgensohn, A.: End-user modifiability in design environments. In: Proceedings CHI'90, pp. 183-192. ACM, New York, NY (1990).
12. Hammond, J., Gibbons, P.: What is scaffolding? Teachers' Voices 8, 8–16 (2005).
13. Hine, C.: Virtual Ethnography. Sage Publications, London (2000).
14. Hoadley, C. P.: Creating context: Design-based research in creating and understanding. In: Proceedings of Computer Support for Cooperative Learning (CSCL), Boulder, USA (2002).
15. Jordan, B., Henderson, A.: Interaction analysis: Foundations and practice. The Journal of the Learning Sciences, 4(1), 39–103 (1995).
16. Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., Werner, L.: Computational thinking for youth in practice. ACM Inroads, 2(1), 32–37 (2011).
17. Lewis, C.M.: How programming environment shapes perception, learning and goals: Logo vs. Scratch. In Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE'10), pp. 346–350. ACM, New York (2010).
18. Lieberman, H., Paternò, F., Klann, M., Wulf, V.: End-user development: An emerging paradigm. In Lieberman, H., Paternò, F., Wulf, V. (eds.) End-User Development, pp. 1–7. Springer, Heidelberg (2006).
19. Lipponen, L., Hakkarainen, K., Paavola, S.: Practices and orientations of CSCL. In Strijbos, J.W., Kirschner, P.A., Martens, R.L. (eds.) What We Know about CSCL, vol. 3, pp. 31–50. Springer, Dordrecht, NL (2004).
20. Mørch A.I., Litherland K.T., Andersen R.: End-user development goes to school: Collaborative learning with makerspaces in subject areas. In: Proceedings IS-EUD 2019, pp. 200-208. Springer, Cham, Switzerland (2019).
21. Mørch A.I., Zhu L.: Component-based design and software readymades. In: Dittrich, Y., Burnett. M., Mørch, A., Redmiles, D. (eds.) End-User Development. IS-EUD 2013. Lecture Notes in Computer Science, vol. 7897. Springer, Berlin, Heidelberg (2013).
22. Reichertz, J.: Induction, deduction, abduction. In: Flick, U. (ed.) The SAGE Handbook of Qualitative Data Analysis, pp. 123–135. SAGE Publications, London (2014).
23. Repenning, A.: Agentsheets: A tool for building domain-oriented visual programming environments. In: Proceedings of the INTERACT '93 and CHI '93, pp. 142–143. Association for Computing Machinery, New York (1993).
24. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y.: Scratch: Programming for all. Communications of the ACM, 52(11), 60–67 (2009).
25. Stahl, G.: From intersubjectivity to group cognition. Computer Supported Cooperative Work, 25(4), 355–384 (2016).
26. Stahl, G., Koschmann, T., Suthers, D.: Computer-supported collaborative learning: An historical perspective. In Sawyer, R.K. (ed.) Cambridge Handbook of the Learning Sciences, pp. 409–426. Cambridge University Press, Cambridge, UK (2006).
27. Won, M., Stiemerling, O., Wulf, V.: Component-based approaches to tailorable systems. In Lieberman, H. et al. (eds.). End-User Development, pp. 115-141. Springer, Berlin (2006).
28. Wulf, V., Golombek, B.: Direct activation. A concept to encourage tailoring activities. Behaviour & Information Technology, 20(4), 249–263 (2001).