Principles to facilitate design-based learning environments for programming in secondary education while making learning visible in an authentic way

Bas van Zadelhoff
basvz17@gmail.com
Radboud University
The Netherlands

Ebrahim Rahimi (✉)
Ebrahim.Rahimi@ou.nl
Open University
The Netherlands

Erik Barendsen
Erik.Barendsen@ru.nl
Radboud University & Open University
The Netherlands

ABSTRACT

Design-based learning (DBL) environments seem a promising instructional approach to facilitating students' learning of programming concepts via applying them to develop various digital artifacts such as games, websites, robots, and software applications. However, there is a lack of theory-grounded principles to facilitate DBL environments for programming in K-12. In the absence of such principles, some essential elements of DBL such as the reciprocal link between "design" and "learning" processes might be easily neglected which can sacrifice "learning" in favor of "making" and degrade constructionist-based learning-by-making initiatives in programming to a form of "shallow constructionism" or doing for the sake of making with no significant impact on the students' programming knowledge.

In this study we formulated a set of principles to facilitate DBL environments for learning programming concepts in K-12 while making learning visible in an authentic way (i.e., without breaking the realistic design setting using separate tests), built upon related educational theories and concepts, including Constructionism, scaffolding, Context-based learning, Collaborative learning, and design-based learning. We instantiated these principles to develop a course for learning about algorithms for secondary school students. The developed course was implemented in five secondary schools (with 5 teachers and 87 students) and evaluated based on the experience of the participating teachers and students. The results of the evaluation helped us to reflect on the applicability of the principles and refine them.

CCS CONCEPTS

• **Computer systems education → Programming**; *Design-based learning*; K-12 programming education; Design principles; Learning-by-making.

KEYWORDS

Programming, K-12 programming education, design-based learning, design principles, course development, algorithms

Koli Calling '21, November 18–21, 2021, Joensuu, Finland

ACM ISBN 978-1-4503-8488-9/21/11.
https://doi.org/10.1145/3488042.3488067

ACM Reference Format:
Bas van Zadelhoff, Ebrahim Rahimi (✉), and Erik Barendsen. 2021. Principles to facilitate design-based learning environments for programming in secondary education while making learning visible in an authentic way. In *21st Koli Calling International Conference on Computing Education Research (Koli Calling '21), November 18–21, 2021, Joensuu, Finland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3488042.3488067

1 INTRODUCTION

The application of design-based learning (DBL) in K-12 programming education aims to create an engaging learning environment to help students learn programming and computational thinking skills through motivating and inspiring their interest and helping them create various computational artifacts such as games, apps, websites, and software applications [20]. There are strong arguments advocating the benefits of learning-by-making approaches such as DBL for children to learn programming concepts through application [23]. It has been argued that involving students in making activities help them learn how to use technological tools and through that discover new way of thinking [29]. From the perspective of constructionism, coding and making tangible digital artifacts allows students to have an "object-to-think-with" and learn about their own thinking [31].

Despite their promises, there are various issues facing the implementation of DBL environments for programming. The main issue concerns with addressing the link between *designing* and *learning* [29]. There is a reciprocal relation between design and conceptual learning so that students use concepts to support their design, and on the other hand, students' involvement in design activities, their encounter with design challenges, and their attempts to reason about and resolve these challenges can deepen their understanding of concepts underpinning the design [24]. Without carefully addressing this reciprocal link, any constructionist-based learning-by-making initiative in programming can easily decline to a form of "shallow constructionism" or doing for the sake of doing with no significant impact on the students' learning [34, 37]. As an example of "shallow constructionism", as reported by Meerbaum-Salant et al., only between 10-20% of the students' programming projects uploaded to the Scratch website, despite their design richness in terms of costumes and sounds of the sprites, use basic programming concepts such as conditional looping or variables [26]. Another issue with implementing DBL environments for programming concerns with the assessment and making students' learning visible in an authentic way. As asserted by Brennan and Resnick, in general, constructionist-based approaches to programming tend to merely

focus on final (working) design artifacts of students. Instead, they advocate an authentic process-oriented assessment approach to bringing the essential elements of the design and learning process, such as students' design practices and choices as well as their understanding or misconceptions of concepts, into the assessment [8].

The purpose of this study is to formulate a set of principles to facilitate the implementation of DBL environments for learning programming concepts while making students' learning visible in an authentic way (i.e., without breaking the design setting as a realistic context, in particular not through a separate test). To this end, we selected related educational theories and concepts based on some well-known perspectives on DBL and used these theoretical concepts to formulate an initial set of principles. Then, we instantiated these principles to develop a course for learning about algorithms, as the core part of programming, in secondary education. The course then was implemented in five schools and evaluated by the participating students and teachers to provide us with insights about the applicability of the principles and their needed refinement.

2 THEORETICAL BACKGROUND

We selected a set of educational theories and concepts to underpin principles for facilitating design-based learning environments for programming based on the following perspectives: First, A DBL environment for programming should allow the active creation of meaningful computational artifacts as a means for motivating and supportting students' learning and thinking [29]. This perspective suggests constructionism theory, design-based learning, and context-based learning concepts to be considered. Secondly, new perspectives to programming defines programming as a social and collaborative process [22]. Thirdly, the metacognitive perspectives on programming define it as a complex problem-solving skill, highlighting the importance of incorporating scaffolding prompts into programming education [25].

2.1 Constructionism

The theory of constructionism conceptualizes how and what students learn when they make something. Constructionist learning refers to learner's construction of her mental models to understand the world through "creative experimentation" and making social and meaningful objects [2, 30]. From the perspective of this theory, learning is the conscious process of building knowledge structures through constructing an artifact whether "a sand castle on the beach or a theory of the universe" [30] (p. 1). Constructionism states that learning takes place when learners engage in design challenges to develop a "public entity" or a personally meaningful artifact and "audience to share insights with" while working on their designs [30]. Some scholars discussed the difference and convergence between the constructivism and constructionism theories [1, 35]. Ackermann proposes Piaget's constructivism and Papert's constructionism as theories with similar goal (i.e., to understand who people learn), but with different means (e.g., their own cognitive tools and external realities) [1].

2.2 Scaffolding

Scaffolding, in education, refers to stepping stones that a student can use in order to learn something new. The term scaffolding is generally associated with the theory of Vygotsky asserting that it is not only important to reach a goal, but also how it is achieved [40]. In their conceptual model of scaffolding, Janneke et al. defined three main characteristics for scaffolding [41]. The first characteristic is contingency. Teachers should adjust their support to the current level of the students' development or competence. The second characteristic concerns "fading" or the "gradual withdrawal of the scaffolding" where the support a teacher provides to students decreases over time based on the current level of student's development. The third characteristic is the gradual transfer of responsibility to the students during the learning process by using formative feedback and reflection.

2.3 Collaborative learning

Both constructionism and scaffolding recognize the importance of group working to help students' co-learning and co-working as well as gradual transfer of the responsibility of learning from teachers to student via peers [3, 41]. Collaboration concerns with working together to accomplish shared goals. In collaborative learning, a classroom of students is split into smaller groups so they can learn new concepts together or accomplish shared goals such as co-creating design projects [38]. Several studies have shown the positive effects of collaboration and cooperative learning on student performance [9]. One of the crucial aspects of collaborative learning is debate and discussion. Discussions can help with conceptual reasoning and formative assessments. This, in turn, will give the teacher the ability to spot the underlying cause of any error a student makes more naturally. It is also possible that during debate, students will discover their own mistakes through interaction with their peers. Cohen stated that the level of discussion is improved with smaller groups instead of an entire classroom. Collaborative learning can also improve students' motivation and enjoyment of the subject [10].

2.4 Context-based learning

"Relevance" has been advocated as a determinant of what interests students in a subject [6]. Using authentic situations and contexts is a widely recognized and popular approach to creating effective learning environments through making students' learning process more relevant and meaningful as well as providing opportunities to apply concepts [14, 28]. Context provides a surroundings in which learning takes place. A meaningful situation, objective, or problem to motivate and direct the learning process [36]. Wieringa et al. define context as "a realistic situation from students' own lives, from society or from professional or scientific practices" [43] (p. 2439). Thus, the use of context has been propossed as the basis for curriculum design and classroom teaching and learning in science education [14].

While most of studies on context-based learning have been conducted in science education, research on the use of contexts in CS and programming education is still in its infancy. However, the importance and necessity of introducing context-based learning to programming education have been highlighted by CS researchers

[16, 28, 32]. According to Pasternak, "Programming is not an everyday life context for students, nor is it for their parents and friends. Therefore it is necessary to find contact positions in a context for programming" [32] (p. 659).

The concept-context window is a model that divides teaching materials into following four categories based on the way they use concepts and context [36]:

- The *illustrative context*: An illustrative context is always part of a larger design. The effect of the context is based on its place in this larger design, which means the context is chosen based on a specific concept. Teachers and students will see the concept as the main subject where the context is used to provide examples.
- The *connecting context*: A connecting context is a context that fits well with a predetermined group of concepts. The context determines the classification of these concepts.
- The *central context*: A central context forms the main thread through the course. The choice of concepts and their format is determined by what a student needs to understand about the central context.
- The *context at a distance*: A context at a distance is the subject of the course. Concepts are found to fit this context. The difference with a central context is that with context at a distance, concepts are discussed first, and context is only discussed after all the concepts have been explained.

The concept-context window can be used to develop and design new learning environments, including learning activities and course materials as well as to analyze and extend existing learning environments. Not all courses fall into these categories perfectly, however; in reality, it is possible to have courses that show properties of more than one category or even none of them.

2.5 Design-based learning

Design-based learning (DBL) is a sort of education where students try to design a tangible product and use design as a vehicle to learn its underpinning concepts [27]. Also, using (realistic) design assignments as the context of learning suggests DBL a subcategory of context-based learning. Various subjects in K-12 use DBL. Kolodner et al. proposed the Learning-by-design framework to introduce design-based learning to the middle-school science classrooms [24]. Based on this framework students learn how to explain their findings scientifically, how they can design fair tests and how to justify their results with evidence. The learning-by-design framework splits the learning process into two cycles: design/redesign and investigate/explore. In the first cycle, students use their current knowledge to design or redesign something. In the second cycle they use their current design and additional information in order to increase their knowledge and conceptual understanding.

3 RELATED WORK

3.1 Design-based learning in programming education

To create constructionist-based learning environments for programming, Papavlasopoulou et al. defined nine principles, including social interaction between students, appropriate design according to age groups, choosing right duration of the activity to have students personally, intelectually, and emotionally involved in the design, relevanct activity and meaningful content, physical and digital artifacts, and meaningful framework for the involvement of the instructors [29]. To ease learning of fundamental programming concepts, Meerbaum-Salant et al. developed learning materials for middle-school students utilizing the constructionist approach of Scratch. The learning materials were created based on four principles: focus on CS concepts rather than Scratch features, introducing programming constructs as needed, project-based presentation, and de-emphasizing appearance aspects such as the costumes, sounds, and visual effects. To assess students' conceptual understanding, they designed interim-test and post-test by combining the Bloom and the SOLO taxonomies. Their results showed that students could successfully learn concepts such as initialization and conditional execution while they experienced difficulties with learning concepts such as repeated execution and variables [26].

Despite its wide adoption in science education, the research about DBL environments is still in its infancy in programming education. A DBL model for computational thinking, developed and evaluated by S. Jun et al. [20], consists of four phases of design (i.e., the process of creating, interacting with, and shaping an idea through creative design engineering and problem-solving), personalization (i.e., the process of developing meaningful and personally relevant materials, including content and prototypes), collaboration (i.e., the process of co-developing new materials with other people), and reflection (i.e., the process of reviewing, thinking about, evaluating, and revising the developed materials). This DBL model has been shown effective in improving the programming ability, self-interest, and self-efficacy of elementary school students, and helping them learn how to transform their ideas into meaningful outcomes [20].

Most learning to program initiatives do not explicitly address scaffolding and problem-solving skills students need to succeed at ill-defined and open-ended problems in settings such as DBL environments [25]. By recognizing the necessity of teaching not only the syntax of program but also *how* to program, and considering design as a problem-solving process, Loksa et al. proposed an approach to imparting programming problem-solving skills based on scaffolding and metacognitive prompts. Their approach consists of four phases: (1) explicit instruction on goals and activities in programming problem-solving, (2) visualizing and monitoring progression through problem-solving stages, (3) explicit and on-demand prompts for learners to reflect on their problem-solving strategies when seeking help from instructors, and (4) providing context-sensitive help embedded in a code editor to reinforce the problem-solving strategies [25]. Their approach defines six stages for problem-solving, namely, reinterpret problem prompt, search for analogous problems, search for solutions, evaluate a potential solution, implement a solution, and evaluate the implemented solution [25].

While the model and approach proposed by S. Jun et al. [20] and Loksa et al. [25] provide important implications for facilitating DBL environments for programming, they seem to be silent about two essential requirements of DBL, namely, addressing the reciprocal link between "learning" of concepts and "making" processes in DBL [24] as well as a process-oriented approach to making students'

designing and learning visible [8]. On the other hand, while the study by Meerbaum-Salant et al. [26] explicitly addresses students' learning of programming concepts in a constructionist-based environment (i.e., Scratch), the assessment of students' learning is conducted in a non-authentic way by breaking the design setting as a realistic context using the interim-test and post-test.

To make an explicit link between "making" and "learning" processes in programming, as the essential element of DBL, the *making-learning cycles* framework was propossed by [34]. This framework splits the program development process into four iterative phases of problem analysis, devising a solution in terms of an algorithm and evaluating it, coding, and testing and revising the code. Further, it defines four tangible intermediate design/learning products (namely, a conceptual solution, an algorithm in terms of a flowchart, code, and test report) as the tangible outputs of these phases to fusion and embody the interconnection between making and learning processes. These intermediate design products are meant to be used as authentic process-oriented assessment means through capturing snapshots of students' design and learning choices and outcomes along the programming problem-solving process. In this framework, transferring between making and learning cycles happen via *need-to-learn* and *need-to-do* steps: students first start by the making cycle and when they need to learn something in order to support their design they jump into the learning cycle. After learning the concepts they need they come back to the making cycle to continue their design with what they have learned.

3.2 Algorithms in K-12 programming education

Understanding algorithms and algorithmic thinking are indispensable parts of learning to program. Accordingly, we are witnessing ever-increasing initiatives worldwide to introduce learning about algorithms into their K-12 programming curriculum [4, 19, 42, 45]. Algorithmic thinking refers to a set of competencies a student needs in order to devise a solution to solve a problem independent of a specific programing environment. These competencies consist of analyzing a problem, precisely specifying a problem, finding or defining the steps necessary to solve the problem, constructing an appropriate algorithm using the necessary steps, thinking about all possible special and regular cases of a problem, and improving the efficiency of the algorithm [13].

Approaches to learning about algorithmic concepts can be divided into two broad categories of unplugged and plugged activities. The former includes initiatives like Bebras [11] and CS Unplugged (http://csunplugged.org/) that allows students understand algorithmic concepts without using a computer by involving them in common practices of computing or everyday life activities such as decomposing a daily task (e.g., brushingteeth or following cooking recipes) into steps [15, 44]. The latter includes applying the constructionist-based approaches such as "use-modify-create" progression model to help students' tinkering and creation of digital artifacts. These plugged approaches vastly use curricular activities such as game design and robotics to motivate and engage students, introduce them to computational thinking concepts, and facilitate an "iterative exploration" of computational thinking and algorithms. To this end, visual programming environments such as Scratch, Alice, Game Maker, and Greenfoot as well as robotics kits and Arduino are utilized for learning programming and algorithms in K-12 through creating computational artifacts [15].

Flowcharts have been used to learn about programming and algorithms since the introduction of computers in 1940s. The main application of flowcharts in programming education is to ease learning programming by providing a pictorial representations of algorithms and programs' components, flow of control and logic, and documenting their outcomes [7, 12, 18]. The benefits of flowcharts to aid learning about algorithms and programming stem from various features, including their simplicity of syntax, top-down presentation, two-dimensional display, compactness, and language independence [18]. Flowcharts have been suggested as effective visual tools to enhance the problem-solving and algorithmic thinking competencies by promoting a "thinking first" approach to programming for the novice programmers [39]. Also, it has been shown that establishing a direct connection between flowcharts and their equivalent program can improve the students' understanding of computer programming [12].

4 STUDY SETTING

To address the objective of this study we formulated the following research questions:

Research question 1: *What principles can be derived from the aforementioned educational theories to underpin a design-based environment for learning fundamental concepts of programming in secondary education while making students' conceptual learning visible in an authentic way?*

To answer this question, we utilized the educational theories and concepts explained in the previous sections to formulate a set of initial principles. The main specifications of design-based learning environments for programming, including personalization and collaboration [20], contextualization of learning, linking the learning and design processes [34], and scaffolding [25] informed the formulation of these principles.

To instantiate the formulated principles we used them to develop an algorithms course, consisting of learning activities, content, and a software tool to facilitate learning of fundamental concepts of algorithms via making a software application by secondary school students. Then, we implemented this course in four classrooms and evaluated it by answering the following research question:

Research question 2: *How is the course, built-upon the derived principles, perceived by participating teachers and students?*

We used the results out of answering this question to evaluate the course and its underpinning principles based on the experience and perceptions of the participating teacher and students about the course. We utilized these results to reflect on the initial list of principles, refine them and provide improvement suggestions for the course.

Evaluation

Participants

The participants of this research were five Dutch secondary school computer science teachers and 87 students (68 male, 19 female) aged 15-16. This research was a part of a larger study

about formative assessment in design-based programming education where these schools volunteered to take part. All students in this project had basic knowledge of programming with either Python or PHP through their previous courses, but no prior knowledge and experience about algorithms and flowcharts as separate concepts. The project lasted eight weeks.

Data collection

We used semi-structured interviews as the main research instrument to collect data needed to answer the second research question. In total 15 interviews were conducted (5 teachers: 3 male, 2 female, and 10 students: 7 male and 3 female). Interviews took place in the participating schools and lasted around 45 minutes each. The interview questions were designed to explore, obtain detailed information, and clarify ambiguities about the interviewees' perceived usefulness of course components and their underpinning principles. For the teachers, the interview questions were focused on exploring various aspects of their teaching experience with the course, their observations of students' learning, misunderstanding, faced difficulties and challenges, and reactions, their perception about the usefulness of each course' component for teaching and learning algorithms via design and its reasons, their faced difficulties with the teaching process, and also improvement suggestions. For students, the interview questions were targeted at exploring various aspects of their learning experience with the course, their satisfaction/dissatisfaction, perceived usefulness for learning algorithms, faced difficulties, and improvement suggestions.

Data analysis

We recorded all the interviews with a voice recorder and made additional notes on paper. All interviews were transcribed and imported into Atlas.ti software. We followed a hybrid deductive-inductive qualitative content analysis procedure: First, the initial set of principles were used as the main categories to direct the analysis process. Within each category, we followed a flexible and open approach, including identifying, coding, re-coding, grouping, regrouping, and reporting the collected data. We discussed possible alternative interpretations of the identified codes within the research team until a consensus was reached.

5 INITIAL PRINCIPLES

We derived seven principles (D1, D2, ..., D7) built upon the described learning theories and concepts. Figure 1 presents these principles and their underpinning theories.

D1: Make students' learning visible via intermediate design products: we formulated the first principle (hereafter D1), drawing upon the *Context-based learning* to choose appropriate programming concepts to be learned by students. Further, the Loksa et al.'s scaffolding prompt of visualizing and monitoring progress [25] and the concept of intermediate design products (i.e., problem analysis, algorithm in terms of a flowchart, code, and test report) proposed by the Making-Learning Cycles framework [34] were adopted to facilitate and visualize the students' conceptual understanding of programming concepts via design.

D2: Contextualize learning: the second principle (hereafter D2), built upon the context-based learning, aims to introduce a meaningful context to the learning process (see next section for the implementation details of each design principle, including the selected context).

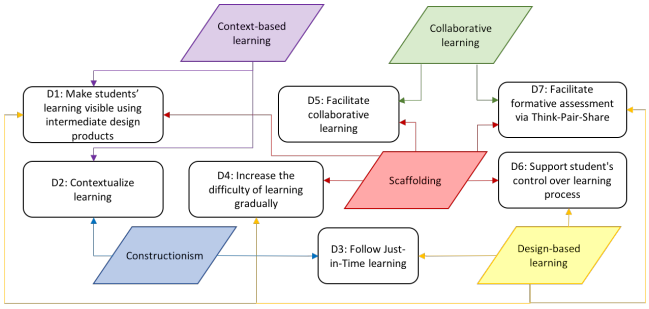

Figure 1: The design principles and their corresponding learning theories

D3: Follow Just-in-Time learning: the just-in-time approach of *design-based learning* represented by *Need-to-know* and *Need-to-do* phases [24, 34] and the authentic learning-by-making nature of *constructionism* informed the formulation of principle 3 (hereafter D3).

D4: Increase the difficulty of learning gradually: principle 4 (hereafter D4) draws upon the step-wised and gradual development approach of design-based learning as well as the contingency and fading characteristics of *scaffolding* [25, 34].

D5: facilitate collaborative learning: the fifth principle (hereafter D5) is informed by the *collaboration* component of design-based learning environments for programming proposed by [20], new perspectives on computational participation [22], and the scaffolding role of peers as supporters in the learning process to supplement or replace the teacher's support [41].

D6: Support student's control over learning process: the sixth principle (hereafter D6) draws upon the inherent characteristic of design-based learning, namely, dealing with uncertainties, through providing learning choices where students need to make decision. In real-world situations, there are open-ended problems where there exist multiple ways of working to solve them and multiple solutions. Further, this principle aligns with the *personalization* aspect of S.Jun et al.' DBL model [20]. From the scaffolding perspective, such a student-centered decision-making process is crucial to exert students' control over the learning process and the gradual transfer of learning responsibility from teacher to students [41].

D7: Facilitate formative assessment by generating and sharing intermediate design products via Think-Pair-Share: the seventh principle (hereafter D7) is informed by the necessity of bringing students' learning and design decisions and outcomes along with the process into assessment [8]. To this end, the design products co-generated by students during the process of design-based learning seem promising to lend themselves to process-oriented assessment approaches such as formative assessment [33]. These intermediate products seem benefitial to facilitate formative feedback and reflection characterizing the scaffolding process [41] as well as the utilize the role of peers, group-, and class-wide discussion in spotting and resolving students' misunderstanding of a topic [10]. Think-Pair-Share learning strategy [21] can be used to facilitate

formative assessment of students' understanding of programming concepts at the individual, group, and class levels around these student-generated design products.

6 INSTANTIATING THE PRINCIPLES IN A COURSE ABOUT ALGORITHMS

In this section, we instantiate the derived principles and explain how we used them to develop a course for learning about fundamental constructs of algorithms via design for secondary school students.

Informed by the first design principle (i.e., make learning visible), we developed a textbook to cover various aspects of algorithms, including the fundamental constructs of algorithms (i.e., sequence, condition, and iteration), flowcharts, arrays, strings, functions, sub-algorithms, and trace tables. An active approach to learning the provided concepts was followed, including applying the learned concepts to complete their design assignments, working in group to generate four types of intermediate design products, sharing and discussing their intermediate design products with the whole class, and assessing their learning using provided questions.

The second principle (i.e., contextualize learning) led us to provide context and define proper interaction between the context and the concepts within the course materials. We determined that the appropriate concept-context window for our documents would be a central context. The central context of our course materials is digital text processing and analysis. We will place this context in a real-world setting through defining realistic design assignments as shown in Table 1. Choosing digital text analysis as the central context is meant to promote and motivate students' learning of algorithmic concepts through developing and implementing meaningful and realistic working programs for their design assignments.

Table 1: List of design assignments

Description: *Assume that you are working as a data scientist at the BOL.com company. Your responsibility is to analyze a large body of digital text to find specific patterns required by different business processes of BOL.com. Your assignments include designing and implementing an algorithm:*
Design assignment 1: ... *that finds patterns of numbers in a text*
Design assignment 2: ... that finds the most frequently used word (s) in a text
Design assignment 3: ... *to calculate the similarity percentage of two text*
Design assignment 4: ... to perform sentiment analysis on a text
Design assignment 5: ... to find the repeating patterns of words in text
Design assignment 6: ... to find valid URL links in a text
Design assignment 7: ... to find valid email addresses in a text

Informed by the third principle (i.e., Just-in-Time learning), we decided to provide students with two design and content documents. Students were asked to follow a just-in-time approach to learning so that they first try to complete a design assignment and search for missing information and learn required concepts only if it is needed. Also, no predefined and planned teaching moment was included. Concerning the fourth principle (i.e., gradual learning), we implement an scaffolding process consisting of the following steps to help students develop their design assignments and learn the associated concept:

- Dividing the whole design process into four steps of problem analysis, devising algorithm, coding, and evaluating and asking students to follow these steps to generate related intermediate design products
- Initially providing activities or assignments about simple concepts (e.g., 1-dimension arrays), and then working out more advanced concepts (e.g., 2-dimension arrays)

Regarding the fifth principle (i.e., collaborative learning), two levels of group and class collaboration were defined. Students were grouped in teams of 4 members to complete their chosen design assignment. For difficult concepts, faced by several groups, a class-wide discussion and sharing around that concept orchestrated by the teacher was defined. No specific grouping strategy was chosen and teachers were left on their own to choose and implement a grouping strategy.

To instantiate the sixth principle (i.e., student's control) students were provided with several choices. First, they could choose their design assignments with a high flexibility in the design process. Secondly, they could decide to read which concepts from the content document based on their previous knowledge and learning needs. Thirdly, we developed a text processing tool to help students get an idea of the project and choose and use its built-in functions (e.g., substr(str, s1, s2)) as building blocks to generate their program. The tool was provided for PhP and Python programming languages to allow students choose one based on their preference or experience. Students were encouraged to do a free design assignment by define their design assignment and completing it with the building blocks in the text processing tool. Finally, in some classes students could choose their teammates.

Informed by the seventh principle (i.e., formative assessment), in order to provide feedback to students, multiple ways for capturing evidence of students' learning or misconception of the intended concepts were provided, including exit questions at the end of some lessons and students-generated design products (e.g., conceptual solutions, flowcharts, code). Further, we included some specific questions in the course materials which asked students to provide argumentation for their answers or explain their design choices. These questions could also be used by teachers to spot the difficult concepts for students and provide feedback. These assignments were meant to have students to reflect on their own answers. We embedded the Think-Pair-Share strategy in the course to have students answer the questions or generate the intermediate design products, first individually, then discuss it with their groups, and finally share and discuss the results with the whole class. This strategy was meant to tap into the potential of individual thinking and group and class-wide discussions for revealing some students' misconceptions by the teachers.

7 PARTICIPANTS' PERCEPTIONS OF THE COURSE

D1: Make students' learning visible using intermediate design products

Student-generated intermediate design products were perceived as very useful in showcasing students' use and understanding of algorithmic concepts. Figure 2 shows two samples of these intermediate design products. Condition, iteration and strings were concepts that students in general found easy to grasp. Arrays of strings, on

the other hand, appeared to be a new and challenging concept for many students, as echoed below:

Here, there are two types of arrays: an array with strings, and the string itself is also an array. The step from one to the other is difficult. (Teacher 1)

Making flowcharts appeared to be a challenging activity for many students:

By making flowcharts they can understand the Python code better, and they are already walking on their tiptoes because they find that flowchart making difficult. (Teacher 3)

However, students and teachers found flowcharts useful means to divide a problem into sub-problems, make the code more readable and clarify its primary purposes. Sub-routines were perceived by both teachers and students very helpful concept. Some students, as echoed below, asserted using sub-routines has equipped them with a new way of thinking to solve a problem by dividing it into smaller sub-problems:

You can use this technique to address daily-basis problems. It can be used as a new way of thinking. Indeed, in my future assignments, most probably I will divide the problem into sub-problems as it is handy. (Student 3, class 1)

Students and some teachers had never used trace tables which was not a problem until they faced a large trace table:

The first trace table was clear to me and easy, but the second trace table assignment that asked us to create a trace table for our own flowchart and test it was very difficult. (Student 2, class 1)

The trace table is experienced as very difficult. Maybe there can be an example with a slightly simpler flowchart. (Teacher 2)

D2: Contextualize learning

The use of digital text processing to solve realistic problems was well received by students. Students said these contextualized features gave a more realistic taste to the project and made it more enjoyable as it created the feeling that they are completing it for something real instead of merely as an exercise given by the teacher. These points are well-reflected by the following quotes:

I really liked the first part of the problem, where it was like you are an employee at bol.com or NASA, and you have to find weird emails or something. And I really liked that because you can see the use of it in the actual world. (Student 8, class 1)

I do not think the realistic context changes the nature of problems or solutions. But because of their realistic situations (e.g., bol.com), they seem useful to us. So this usefulness attracts us more than just exercises we have been told to do by the teacher. (Student 3, class 4)

Some teachers asserted that the questions and examples within the content document were not as realistic and contextualized as the design assignments.

D3: Follow Just-in-Time learning

We aimed to implement a just-in-time-learning by separating the learning (reading content and grasping concepts) and design (applying the concepts) aspects and encouraging students to first start by the design side and follow the authentic *need–to-know* and *need–to-do* phases when needed. Some students followed this strategy to focus on the design assignments first and use the content document as a reference point for tips and tricks. On the contrary, many students first started by reading the content document and jumped into the design assignments after reading the whole content document, as echoed in the following quote:

So they are reading it for 15 minutes, then the next day, they come and continue reading. But, then they are missing what they actually were doing [last time] and forgetting about things. (Teacher 4).

You have to read a lot because I am not good at computer science [concepts] and I still do not understand them. (Student 3, class 2)

According to the teachers, this was due to two reasons: the existence of a high diversity between students' programming preknowledge and experience, and a large gap between content and design documents. To shorten this gap, they suggested to include the necessary information on the required concepts within the design document. Further, given the students' difficulty with concepts of flowcharts and arrays of strings, teachers suggested creating a short starting module in which these concepts are explained before students begin with the design assignments.

D4: Increase the difficulty of the learning process gradually

Both teachers and students found dividing the design process into four phases of problem analysis, design algorithm, coding, and testing very useful:

These phases served as milestone for students' projects and helped them to get an overview of the whole design process and make sense of it. (Teacher 4)

In general, students were able to keep up with the gradual increase in the difficulty of the design assignments with more challenge at the begin, as echoed below.

In the beginning, it was difficult because we just learned the basics of PHP. But after doing some assignments, it seemed OK, and now we have been doing the assignments for four hours. (student 6, class 1)

Some teachers raised their concern about the steep jumps between some design examples and assignments and the need for smoothing them.

You could build the flowcharts [module] separately, the introduction to algorithms [module] separately, and then a module PHP with arrays and everything students need before those design assignments. (Teacher 5)

D5: Facilitate collaborative learning

Most students participated in this research worked in teams of 3-4 members to complete their design assignments. In addition to helping the design part, group working was perceived a useful means to seek information about the concepts students required to complete their design assignments. In this regard, some students said that in the case of having difficulty in understanding a concept or assignment they first asked someone in their groups or other students before posing the question to the teacher. This observation confirms the role of peers and group in supplementing or replacing teacher's support and scaffolding. In this regard, the teachers found the collaborative learning a useful means to get insights about students' learning or misunderstanding through observing and listening to their discussions.

The participating teachers followed three different methods for forming teams in their classrooms: letting the students create their own groups, creating groups of students with the same skill level to ensure that all members participate instead of only the smartest one, dividing students into three skill level categories and creating a group by picking students from each category. Some teachers highlighted the challenges related to grading the final projects developed by groups and the need for having a grading rubric consisting of both individual and group level metrics.

D6: Support student's control over the learning process

The provided learning choices such as choosing programming language (Php or Python), design assignment, or teammates was perceived positively by the interviewed students to allow them exert some sorts of control over the learning process. Besides these choices, the open and flexible characteristics of the design assignments and process was well-recognized by students. Students found that this approach gave them more flexibility in making the assignments and also more responsibility in taking decisions by themselves, which was overall well-received, as echoed below:

It is very open. It is scripted [step-wise], but still, there is a lot of open activity space for your own [choices]. (Student 6, class 3)

It is an open question, and we have to think about it ourselves. (Student 7, class 1)

Another challenge we faced, in comparison to other previous projects, is that here we have to write an algorithm that needs to consider all possible types of input to work well. (Student 5, class 4)

For teachers having flexibility and choices in the design assignments and learning process was perceived very important due to high diversity in students' programming knowledge and experience background:

Having design assignment choices fits very well with my class setting. We have such a variety of pupils in the classroom, from culture and society to nature and technique profiles. And this year I have relatively more students with poor mathematics and programming background, which of course makes a huge difference, so that asking all students to do the same [difficult] assignment always remains very challenging. (Teacher 1)

D7: Facilitate formative assessment by generating and sharing intermediate design products via Think-Pair-Share

Using the Think-Pair-Share strategy was well-received by the participant teachers and students as an effective means for activating students and keeping them engaged during the course. Some teachers embedded specific rituals into the Think-Pair-Share strategy, such as gallery walking and commenting on fellow students' intermediate design products. The process of Think-Pair-share around student-generated intermediate design products provided various possibilities for teachers to provide formative feedback to their students:

sharing, and discussing students' design products was useful to get insights on where they in their thought process made a mistake about a concept, very early in the design process before trying to extract it from the final code or even lose it there. (Teacher 4)

Figure 2 shows two misconceptions held by students about calculating the minimum of three numbers and the parallel execution of flowchart's steps, presented in their intermediate design products. Teachers also found using the exit questions very useful to spot a specific conceptual mistake held by individual students.

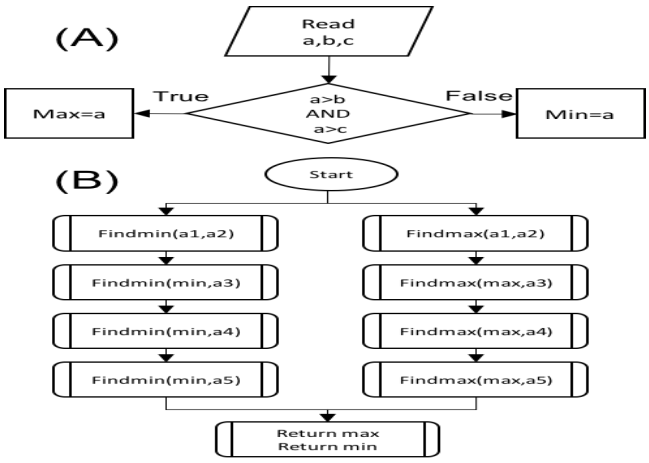

Figure 2: Two examples of captured misconceptions: (A) misconception about calculating the *minimum* of three numbers a,b,c, (B) misconception about the initialization and also sequence of steps in a flowchart (i.e., parallel steps)

8 DISCUSSION

The aim of this study was to propose a set of theory-grounded principles to facilitate design-based learning of programming in secondary education and make students' learning visible in an authentic way. We formulated an initial set of seven principles drawing upon related educational theories and frameworks, and instantiated them in a course for learning about algorithms for secondary school students.

The first principle concerns with making students' learning visible using their generated intermediate design products. The results show that the student-generated intermediate design products were useful to showcase and make visible their learning or difficulty with various algorithmic concepts, including condition, iteration, arrays, sub-routines, arrays of strings, and trace tables. One can see this principle as a concrete supplement to the Loksa et al.'s prompt of visualizing and monitoring progression through problem-solving stages [25]. Arguably, the introduction of the intermediate design products, in terms of conceptual solution, flowchart, code, test report, can facilitate DBL environments for programming and algorithms in three ways: (i) interlinking the "making" and "learning" processes as a mostly neglected aspect in many constructionist-based initiatives in programming [20, 24], (ii) supporting a gradual and iterative process of refinement and approximation of students' understanding of algorithms at different level of abstraction [42],

and (iii) visualizing students' mental representations or misunderstanding of problems, solutions, and code [25]. As a concrete example, Figure 2 shows the effectiveness of the language independence feature of flowcharts to detect some abstract misconceptions of programming, such as the parallel execution of a flowchart or program, which can be simply filtered out in programming languages because of their syntactical structure.

With regard to the second principle, *contextualize learning*, the use of digital text processing was well-received by students as motivating, realistic, and meaningful means for their learning. This observation concurs with Guzdial and Ericson pointing that students do not learn a topic for the sake of learning. Rather, they learn it to do something with it [17]. Moreover, it has been claimed that using recognizable context to introduce a new concept can contribute to understanding the concept by, first, eliminating the ambiguities in the meaning and purpose of the concept through providing more details, and, secondly, by connecting the new concept to other concepts used in the context [28].

Concerning the third principle, *Follow Just-in-Time learning*, despite the initial intention of starting learning concepts only when facing a knowledge deficit in the design cycle, it turns out that many students started by first reading the whole content document and then jumped into the design part. One possible reason for this observation might be the fact that designing a software is a complex process and demands knowledge and experience which is in general absent with secondary school students.

Regarding to the fourth principle, *increase the difficulty of the learning process gradually*, both teachers and students observed dividing the whole design and learning processes into four phases of problem analysis, devising an algorithm, coding, and testing as useful milestones to make sense of and demystify the design process. Thus, one may claim that dividing the design process into phases and specifying the tangible learning and design products for each phase can provide some scaffolding means for students. Investigating this claim suggests a line for future research.

Concerning the fifth principle, *facilitate collaborative learning*, groups and peers were perceived useful in providing help and guidance for the design activities and served as sources for seeking information about the algorithmic concepts. This observation highlights the importance of peers and learning groups in programming education to facilitate fading or the gradual withdrawal of teacher's scaffolding [41] through supplementing or replacing teacher's role by peers. Further, groups' discussions served as an assessment means for the teachers to spot what was going wrong with students' learning. This observation concurs with other studies reported the value of groups' discussions to help with conceptual reasoning and formative assessments [10]. According to the teachers, to be more effective, a well-thought rubric is needed to evaluate the performance of group members as the individual and group levels.

Regarding to the sixth principle, *support student's control over learning process*, the interesting finding was about the open and flexible aspects of the design assignments. Students found these aspects interesting as well as challenging. Openness and uncertainty are inherent characters of software design and development. Arguably, providing students with design choices represented by open design assignments can push and help them to acquire critical thinking and decision making skills required to deal with the uncertainty of program design. Further, having students to make design decision on their own can contribute to the personalization of learning [20], transferring the responsibility of learning to students, and ultimately improve their meta-cognitive programming skills [25]. Moreover, as asserted by the participating teachers, considering this principle to provide a variety of design assignments with different level of difficulty is needed in highly diversified K-12 programming classrooms in terms of students' programming knowledge and experience background.

Concerning the seventh principle, *facilitate formative assessment by generating and sharing intermediate design products via Think-Pair-Share*, it turns out that co-creating, discussing, and sharing intermediate design products, supplemented by individual exit questions, can facilitate a formative assessment strategy for spotting students' (mis)understanding of algorithmic concepts at the individual, group and class levels at the very early stage of design prior to coding (see Figure 2). Thus, one can argue that addressing this principle can supplement the constructionist-based approaches to programming by shifting their mere focus on the final (working) artifacts [8] toward a more process-oriented assessment approach that recognizes and taps into the assessment potential of generated intermediate design products along with the design and learning processes.

9 CONCLUSIONS

This study set out to formulate a set of principles, drawn upon appropriate educational theories and concepts, to facilitate design-based learning (DBL) environments for programming in secondary education while making students' learning visible in an authentic way. We evaluated these principles by instantiating them in a course about algorithms which was implemented and evaluated in five secondary schools. The overall evaluation of participating students and teachers about the course's components and their underpinning principles was positive. However, it turns out that principle 3 (i.e., Follow Just-in-Time learning) and its associated course's features were not adopted by the students and teachers.

According to Bell et al., design principles are "subject to refinement over time as others try to adapt them to their own experiences. In this sense, they are falsifiable; if they do not yield purchase in the design process, they will be debated, altered, and eventually dropped" [5] (p. 83). Following this approach to falsifying design principles, our observations led us to adjust the initial list of the principles by dropping principle 3, which was not well-adopted by the students and teachers. The refined list consists of six principles, being: (1) Make students' learning visible using intermediate design products, (2) Contextualize learning, (3) Increase the difficulty of learning gradually, (4) Facilitate collaborative learning, (5) Support student's control over learning process, and (6) Facilitate formative assessment by generating and sharing intermediate design products via Think-Pair-Share.

There are limitations with this research and its results: The participating students had basic knowledge and experience about programming in terms of syntactic understanding of commands and HTML, but not with algorithms and flowcharts. Moreover, using interviews did not allow us to verify some students' statements and claims about their conceptual learning of algorithmic concepts.

Further lines of research are suggested, including expert evaluation of the derived principles, adjusting and researching the course by addressing the identified shortages and improvement suggestions, applying the refined principles to develop courses for other groups of students in K-12 (e.g., primary school students) or students with more programming knowledge and experience, choosing another context such as video, audio, or image processing, instead of digital text processing, as the context for programming, and repeating the research with more data sources such as surveys or analysis of students' artifacts to allow deeper analysis of students' conceptual learning.

REFERENCES

[1] Edith Ackermann. 2001. Piaget's constructivism, Papert's constructionism: What's the difference. *Future of learning group publication* 5, 3 (2001), 438.

[2] Kathryn Alesandrini and Linda Larson. 2002. Teachers bridge to constructivism. *The clearing house* 75, 3 (2002), 118–121.

[3] Sasha A. Barab, Michael Barnett, Lisa Yamagata-Lynch, Kurt Squire, and Thomas Keating. 2002. Using activity theory to understand the systemic tensions characterizing a technology-rich introductory astronomy course. *Mind, Culture, and Activity* 9, 2 (2002), 76–107.

[4] Erik Barendsen, Nataša Grgurina, and Jos Tolboom. 2016. A New Informatics Curriculum for Secondary Education in The Netherlands. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer, 105–117.

[5] Philip Bell, Christopher M Hoadley, and Marcia C Linn. 2013. Design-based research in education. In *Internet environments for science education*. Routledge, 101–114.

[6] Judith Bennett and John Holman. 2002. Context-based approaches to the teaching of chemistry: What are they and what are their effects? In *Chemical education: Towards research-based practice*. Springer, 165–184.

[7] Dennis J Bouvier. 2003. Pilot study: living flowcharts in an introduction to programming course. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education*. 293–295.

[8] Karen Brennan and Mitchel Resnick. 2012. 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. 25.

[9] Michael Chau, Daniel Zeng, Hsinchun Chen, Michael Huang, and David Hendriawan. 2003. Design and evaluation of a multi-agent collaborative Web mining system. *Decision Support Systems* 35, 1 (2003), 167–183.

[10] Elizabeth G Cohen. 1994. Restructuring the classroom: Conditions for productive small groups. *Review of educational research* 64, 1 (1994), 1–35.

[11] Valentina Dagienė and Sue Sentance. 2016. It's computational thinking! Bebras tasks in the curriculum. In *International conference on informatics in schools: Situation, evolution, and perspectives*. Springer, 28–39.

[12] Emanuel de Jesus. 2011. Teaching computer programming with structured programming language and flowcharts. In *Proceedings of the 2011 Workshop on Open Source and Design of Communication*. 45–48.

[13] Gerald Futschek. 2006. Algorithmic thinking: the key for understanding computer science. In *International conference on informatics in secondary schools-evolution and perspectives*. Springer, 159–168.

[14] John K Gilbert. 2006. On the nature of "context" in chemical education. *International journal of science education* 28, 9 (2006), 957–976.

[15] Shuchi Grover and Roy Pea. 2013. Computational thinking in K–12: A review of the state of the field. *Educational researcher* 42, 1 (2013), 38–43.

[16] Mark Guzdial. 2010. Does contextualized computing education help? *ACM Inroads* 1, 4 (2010), 4–6.

[17] Mark Guzdial and Barbara Ericson. 2016. *Introduction to computing and programming in python*. Pearson.

[18] Richard E Haskell, David E Boddy, and Glenn A Jackson. 1976. Use of structured flowcharts in the undergraduate Computer Science curriculum. *ACM SIGCSE Bulletin* 8, 3 (1976), 67–74.

[19] Peter Hubwieser, Michail N Giannakos, Marc Berges, Torsten Brinda, Ira Diethelm, Johannes Magenheim, Yogendra Pal, Jana Jackova, and Egle Jasute. 2015. A global snapshot of computer science education in K-12 schools. In *Proceedings of the 2015 ITiCSE on working group reports*. 65–83.

[20] SooJin Jun, SeonKwan Han, and SooHwan Kim. 2017. Effect of design-based learning on improving computational thinking. *Behaviour & Information Technology* 36, 1 (2017), 43–53.

[21] Mahmoud Kaddoura. 2013. Think pair share: A teaching learning strategy to enhance students' critical thinking. *Educational Research Quarterly* 36, 4 (2013), 3–24.

[22] Yasmin B Kafai. 2016. From computational thinking to computational participation in K–12 education. *Commun. ACM* 59, 8 (2016), 26–27.

[23] Yasmin B Kafai and Veena Vasudevan. 2015. Constructionist gaming beyond the screen: Middle school students' crafting and computing of touchpads, board games, and controllers. In *Proceedings of the workshop in primary and secondary computing education*. 49–54.

[24] Janet L Kolodner, Paul J Camp, David Crismond, Barbara Fasse, Jackie Gray, Jennifer Holbrook, Sadhana Puntambekar, and Mike Ryan. 2003. Problem-based learning meets case-based reasoning in the middle-school science classroom: Putting learning by design (tm) into practice. *The journal of the learning sciences* 12, 4 (2003), 495–547.

[25] Dastyni Loksa, Andrew J Ko, Will Jernigan, Alannah Oleson, Christopher J Mendez, and Margaret M Burnett. 2016. Programming, problem solving, and self-awareness: Effects of explicit guidance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 1449–1461.

[26] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. 2013. Learning computer science concepts with scratch. *Computer Science Education* 23, 3 (2013), 239–264.

[27] Matthew M Mehalik, Yaron Doppelt, and Christian D Schuun. 2008. Middle-school science through design-based learning versus scripted inquiry: Better overall science concept learning and equity gap reduction. *Journal of engineering education* 97, 1 (2008), 71–85.

[28] Jacqueline Nijenhuis-Voogt, Durdane Bayram-Jacobs, Paulien C Meijer, and Erik Barendsen. 2020. Omnipresent yet elusive: Teachers' views on contexts for teaching algorithms in secondary education. *Computer Science Education* (2020), 1–30.

[29] Sofia Papavlasopoulou, Michail N Giannakos, and Letizia Jaccheri. 2019. Exploring children's learning experience in constructionism-based coding activities through design-based research. *Computers in Human Behavior* 99 (2019), 415–427.

[30] Seymour Papert and Idit Harel. 1991. Situating constructionism. *Constructionism* 36 (1991), 1–11.

[31] Seymour A Papert. 2020. *Mindstorms: Children, computers, and powerful ideas*. Basic books.

[32] Arno Pasternak. 2016. Contextualized Teaching in the Lower Secondary EducationLong-term Evaluation of a CS Course from Grade 6 to 10. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 657–662.

[33] Ebrahim Rahimi, Erik Barendsen, and Ineke Henze. 2016. Typifying Informatics Teachers' PCK of Designing Digital Artefacts in Dutch Upper Secondary Education. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer, 65–77.

[34] Ebrahim Rahimi, Erik Barendsen, and Ineke Henze. 2018. An instructional model to link designing and conceptual understanding in secondary computer science education. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education*. 1–4.

[35] Jonathan D Raskin. 2002. Constructivism in psychology: Personal construct psychology, radical constructivism, and social constructionism. *American communication journal* 5, 3 (2002), 1–25.

[36] G Sanders, M Pieters, H Schalk, F Carelsen, and J Kortland. 2016. Het implementeren van contexten in onderwijsmateriaal: Een ontwerp-en analyse-instrument voor de natuurwetenschappelijke vakken.

[37] M Scardamalia and C Bereiter. 2006. Knowledge building: theory, pedagogy, and technology (pp. 97-119).

[38] RE Slavin. 1987. Cooperative Learning: Review of Educational Research. *Needham Heights, Massachusetts: Allyn &Bacon* (1987).

[39] Renske Smetsers-Weeda. 2016. *Think... then act() Flowcharts as a tool for novice programmers to enhance their problem solving skills*. Master's thesis. TU Delft, the Netherlands.

[40] C Addison Stone. 1998. The metaphor of scaffolding: Its utility for the field of learning disabilities. *Journal of learning disabilities* 31, 4 (1998), 344–364.

[41] Janneke Van de Pol, Monique Volman, and Jos Beishuizen. 2010. Scaffolding in teacher–student interaction: A decade of research. *Educational psychology review* 22, 3 (2010), 271–296.

[42] Jane Waite, Paul Curzon, William Marsh, and Sue Sentence. 2020. Difficulties with design: The challenges of teaching design in K-5 programming. *Computers & Education* (2020), 103838.

[43] Nienke Wieringa, Fred JJM Janssen, and Jan H Van Driel. 2011. Biology teachers designing context-based lessons for their classroom practice—the importance of rules-of-thumb. *International Journal of Science Education* 33, 17 (2011), 2437–2462.

[44] Aman Yadav, Hai Hong, and Chris Stephenson. 2016. Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in K-12 classrooms. *TechTrends* 60, 6 (2016), 565–568.

[45] Iris Zur Bargury, Bruria Haberman, Avi Cohen, Orna Muller, Doron Zohar, Dalit Levy, and Reuven Hotoveli. 2012. Implementing a new computer science curriculum for middle school in Israel. In *2012 Frontiers in Education Conference Proceedings*. IEEE, 1–6.