Look What I Can Do: Acquisition of Programming Skills in the Context of Living Labs

Mazyar Seraj[1,2] Cornelia S. Große[1] Serge Autexier[2] Rolf Drechsler[1,2]
[1]Institute of Computer Science, University of Bremen, Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
{seraj, cornelia.grosse, drechsler}@uni-bremen.de
{mazyar.seraj, serge.autexier, rolf.drechsler}@dfki.de

***Abstract*—There is scientific knowledge about how to teach software programming, and the necessity to foster young learners' interest in computer science is broadly addressed. However, there is a lack of research on how to teach programming skills in a way that increases the learners' interest in the topic. We present a training session for young students, in order to support the acquisition of programming skills and, at the same time, a positive view towards computer science. The programming environment is based on a visual block-based application within a living lab. Thus, the abstract concept of software programming is presented within a real context and tightly connected to real experiences. In this training, the learners were introduced to a living lab and to programming concepts in order to acquire basics of programming. Two user studies with 44 7^{th} and 8^{th} grade students were conducted, specifically, the students' interest in computer science and their acquisition of programming skills were assessed. Two instructional interventions to support knowledge acquisition, namely worked examples and instructional procedures, were compared. The results did not strongly support one of these instructional interventions, thus, both seem to be appropriate in order to help learners to acquire basic programming skills. In sum, the results show that the tight connection of the training session to a real-world scenario can foster programming skills. This paper contributes by showing the potential of using visual block-based programming in the context of living labs in order to enable students to begin with programming activities.**

***Index Terms*—smart living lab, visual block-based programming, acquisition of programming skills, training sessions, software programming activities, K-12 education**

I. Introduction

Computer science and software programming are becoming more and more important in K-12 (Kindergarten (K) and 1^{st} through 12^{th} grade) education [1], [2]. Several studies aimed to introduce programming to young students in order to integrate computer science in K-12 education [1], [3]–[5]. These studies investigated how different programming environments affect K-12 students' computational thinking towards problem solving [1], [3], [6]. According to the K-12 Computer Science Framework [7], programming is one of the computer science core concepts [7] and core practices [5]. Thereby, programming is more than just coding [1], [3], but involves computer science concepts such as abstraction and debugging to solve problems [3], [7]. During programming, students are exposed to computational thinking aspects such as creativity, critical thinking and problem solving [3], [7], [8].

This work was partially funded by the German Federal Ministry for Education and Research (BMBF) within the project SMILE under grant number 01FP1613. The authors would like to thank for this support.

Acknowledging that the main goal of programming training for K-12 students is to minimize the distractions of "*thinking at multiple abstractions*" [3], [7], motivating students through software programming activities helps them to understand how computer science influences their daily life [9]. Thinking at multiple abstractions (e.g., students are enabled to find a piece of code that could be more efficiently implemented as a loop) and software programming activities can be considered to be fundamental for K-12 students in order to reduce complexity and increase efficiency of training [1], [3], [10]. In this way, one of the challenges that computer science research still faces is the lack of studies investigating empirical evidence in software engineering education and training among young learners [3], [7].

Young learners typically have difficulties to understand the requirements of designing, executing and debugging software programs [1], [11]. Likewise, they do not have experience in procedural and modular programming to solve programming issues. Thus, in the first place, training them to acquire programming skills seems to be difficult for school teachers and educators [1], [9], [12]. Furthermore, learning and recalling code syntax as well as assembling and manipulating code structures are error prone as they require a high level of concentration [13], [14]. Kalelioğlu [2] tried a new way of teaching programming skills to K-12 students in order to investigate the effect of teaching programming on reflective thinking skills towards problem solving. It was shown clearly that students learned computer science and programming concepts in the code.org platform which is based on Scratch while playing in an enjoyable way. In this respect, many block-based programming frameworks such as Scratch [15], Alice [16], Snap! [17] and Google Blockly [18] have been evolved in today's advancing technology and applications [3], [7] in order to reduce the complexity of programming. These frameworks use a visual programming language, where they represent a vision in which K-12 students engage in the concept of computer science and programming [7], [19]. Visual block-based programming is designed to allow the students to program without the obstacles of syntax and manipulation of code structure errors [7], [13], [14]. Young learners can learn programming through drag and snap blocks together in order to generate code syntax [3], [7]. They can take advantage of visual block-based programming in order to facilitate the design, the execution and the debugging process of programming, and thus, they are able to solve programming problems more easily.

We conducted two user studies in order to examine the acceptance of computational thinking through problem solving among 7^{th} and 8^{th} grade students (12 to 15 years old). We claim that it is possible to introduce new technologies which provide possibilities to connect computer science to reality and cover basic programming skills in one-day programming training sessions in order to enable students making programs and seeing their impacts in real-world environment. To this end, a visual block-based programming application is used in order to reduce the complexity of programming and facilitate it for the students in the context of a smart living lab. The main goal of the smart living lab is to provide exposure to software programming activities in the programming training sessions. An empirical evaluation of the programming training sessions with respect to the interest in computer science and the acquisition of programing skills was conducted. Specifically, based on instructional methods presented in [20]–[23], it was assessed whether the acquisition of programming skills in order to solve programming problems is better supported by worked examples or by instructional explanations.

Our training sessions were conducted in Germany, where the number of female graduates in the field of computer science is considerably lower than the number of male graduates [24], [25]. Thus, introducing software programming to our target users has two aspects. On the one hand, the acquisition of programming skills among inexperienced students and novices was measured in the context of a smart living lab using a visual programming paradigm. On the other hand, the acceptance of computer science and programming among young learners, especially girls, was assessed. It was a special feature of our training sessions that students were able to program a real-world environment. Collecting and analyzing data in this environment extended and refined prior results [1], [24] taken from different non-traditional education environments where students were able to produce and deploy pieces of code that can be applied to smart living labs.

This paper seeks to contribute to software engineering education through the development, implementation, and evaluation of a programming training session, with the goal to teach basic programming skills and principles to inexperienced students and novices, and with the goal to arouse their interest in computer science and programming.

The paper is structured as follows: In section II, background and related work are described. Our training sessions are described in section III. Section IV presents two user studies, the first one with 8^{th} grade students without prior programming experiences, the second one with 7^{th} grade students with little prior programming experience. The results are discussed in section V; the paper closes with conclusions in section VI.

II. Background and Related Work

Over the past few years, several block-based programming tools have emerged to provide visual environments [3], [26]–[28]. The environments aim to help K-12 students getting started with programming activities [13], [29]. These initiatives focus on promoting computational thinking among K-12 students [8], [30]. However, there have been very few studies with a special focus on promoting real-world environment programming activities among K-12 learners – both female and male – with different levels of prior programming experience [9].

Visual programming environments were designed in a variety of approaches (e.g., [2], [6], [12], [26]) for introducing K-12 students to general programming features and principles. The literature reports that programming is more accessible to young learners and novices using these environments [3], and results show that young learners are able to learn programming in visual block-based programming scenarios. In addition, according to [4], [9], researchers are interested in the "*robot programming*" approach in end-user programming tools. Huang et al. [26] took advantage of Google Blockly to develop a rapid block-based programming tool called "*CustomPrograms*" for end users. Paramasivam et al. [9] used this tool to design an end-user application which enables K-12 students to program Clearpath Turtlebot, capable of delivering items, interacting with people and autonomously navigating its environment. With this regard, twelfth-grade high-school students with disabilities (e.g., deafness, low vision or blindness, Aspergers Syndrome) attended a robot programming workshop. The results obtained from this experience are encouraging, as the authors were successful to establish the confidence that robot programming is both accessible and interesting. However, no information was provided whether this procedure also inspired interest in computer science in general.

The literature also reports block-based programming tools used for stimulating interest in computer science among K-12 learners [3], [4], [9], [12], [27]. In terms of teaching programming, two of the preferred block-based visual programming frameworks are Scratch and Google Blockly. For instance, Scratch – one of the most successful visual programming frameworks – is used to present MOOC (Massive Open Online Courses) [27] programming courses in which researchers teach elementary programming concepts to inexperienced high [4] and elementary [12] school students. The results showed that the majority of the participants had a positive attitude towards programming and stated that they plan to continue programming in the future. Nevertheless, teaching elementary programming features to inexperienced young learners and novices in the context of smart environments using visual programming tools is still an active area of research.

However, the use of visual block-based programming frameworks (e.g., Scratch or Google Blockly) alone may not be sufficient for K-12 learners to gain a deep understanding of computer science concepts and programming skills. Although these frameworks are able to reduce the complexity of programming for young learners, they may not enable the learners to develop a well-grounded connection between computer science and its impacts in their daily life. Granting access to a smart living lab may be particularly useful in order to show the learners how a real-world environment can react to their program when it is following the programming principles and structures. To this end, we designed and implemented one-day programming training sessions to provide learning opportunities for young learners in order to program a smart living lab in the German Research Center for Artificial intelligence (DFKI). Thus, learners can participate and experience new technologies which are adapted to technical equipments in order to help elderlies and people with disabilities. In other words, by using

the smart living lab and letting learners to program it, we aim to motivate them to learn programming and to understand influences of computer science in their daily life.

III. Training Sessions

Training sessions were held in premises of the University of Bremen. All used equipments and the smart environment – computers and the smart living lab – which were used were provided by the DFKI. Three instructors were involved in each session. The participants were introduced to the living lab by one instructor. The other two instructors were in charge of the teaching sessions. One instructor was female and all of them had a computer science background with experience in working with K-12 students. The goal of the training was to enable inexperienced students and novices to acquire primary programming skills and to use these skills in the context of a smart living lab.

At the beginning of each session, the students received a pre-questionnaire. Likewise, at the end of each session a post-questionnaire was given to them in order to assess the learners' view on computer science and programming. Apart from this, each session was divided into three parts: (1) introduction to the smart living lab, (2) introduction to programming structures and principles with the block-based programming application, and (3) performing two programming tasks. The questionnaires and the three parts of the training sessions are described in the following.

A. Questionnaires

The questionnaires were designed in order to evaluate the learners' view on computer science and programming. The learners were asked to indicate their prior programming experience in a pre-questionnaire at the beginning of each session. In addition to demographical questions (e.g., age, gender), students were asked to indicate their view on computer science and on programming, specifically with respect to the block-based programming application, in a post-questionnaire at the end of the sessions.

B. Introduction to the Smart Living Lab

Our smart environment is an approximately 60 m^2 smart ambient assisted living lab includes different smart appliances such as a sink adjusting automatically to a person's height and an intelligent wardrobe suggesting outfits (see Fig. 1). Furthermore, the living lab contains various actuators (e.g., dimmable lamps, doors, and lights) and sensors (e.g., lighting, temperature, and heating sensors). Two RGB lamps are available with an HTTP interface for color setting. The smart ambient assisted living lab's main educational use in the programming application is allowing students to program different appliances in different conditions. For instance, in the bedroom, students can turn certain lights on and change the color of an RGB light every 10 seconds for the duration of 3 minutes, as soon as the bedroom door is open.

During each session, the students were divided into two groups in order to enable an introduction to the living lab in smaller groups. All objects and their functionalities in the smart living lab were explained to each group in order to (1) enable students to understand how computer science can influence real-world environments, and (2) identify different smart items which are used in the living lab.

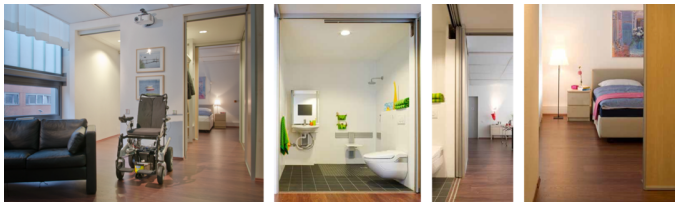

Fig. 1: A view of the smart ambient assisted living lab.

C. Introduction to Programming Structures and Principles

In this paper, a web-based programming application was used to enable inexperienced students and novices to make programs in the context of a smart living lab (see Fig. 2). This application is based on BEESM [28] which is primarily designed for inexperienced learners and novices to learn and write programs. BEESM takes advantage of Google Blockly [18] to design a visual block-based tool being applicable for smart objects and environments. Furthermore, different kinds of behaviors of smart objects are encapsulated in different functions and libraries [28]. In that respect, apart from pre-defined functions and libraries, different programming language features like variables, data types, conditionals, loops, functions and operators are included in this tool. Likewise, we followed the BEESM user interface (see Fig. 2) to enable learners using four different panels to have a full vision of the blocks (Block panel), code syntax (Code panel), output of the code (Output panel), and a 2D view of the smart living lab (2D Graphical panel).

All students were introduced to programming and how to use the programming application. They learned general aspects of using the application, (1) how to identify the blocks relevant to solve the programming tasks, and (2) how to recognize the main elements and panels of the programming application.

The main computational thinking concept which was taught is "*Programming Structures and Principles*", exemplified through visual block-based programming samples. With this regard, programming features such as variables, data types, conditional statements, loops, and logical operators, as well as pre-defined functions were taught. Programming concepts were also introduced using simple block-based programs that include variables, iterative logics and conditional statements, and later through more complex examples that add loops, logical operators and pre-defined functions in order to make different applications in the smart living lab. Students are encouraged to load pre-defined examples and execute them in order to recognize which block corresponds to a particular action.

D. Programming Tasks

During each session and before each task, the students were introduced to the protocols of the programming tasks, specifically (1) that they have only 20 minutes to finish each task, and (2) that they need to use their own supplementary document which comes along with each computer. During the sessions, each computer comes with one of the two types of supplementary documents – namely worked examples and

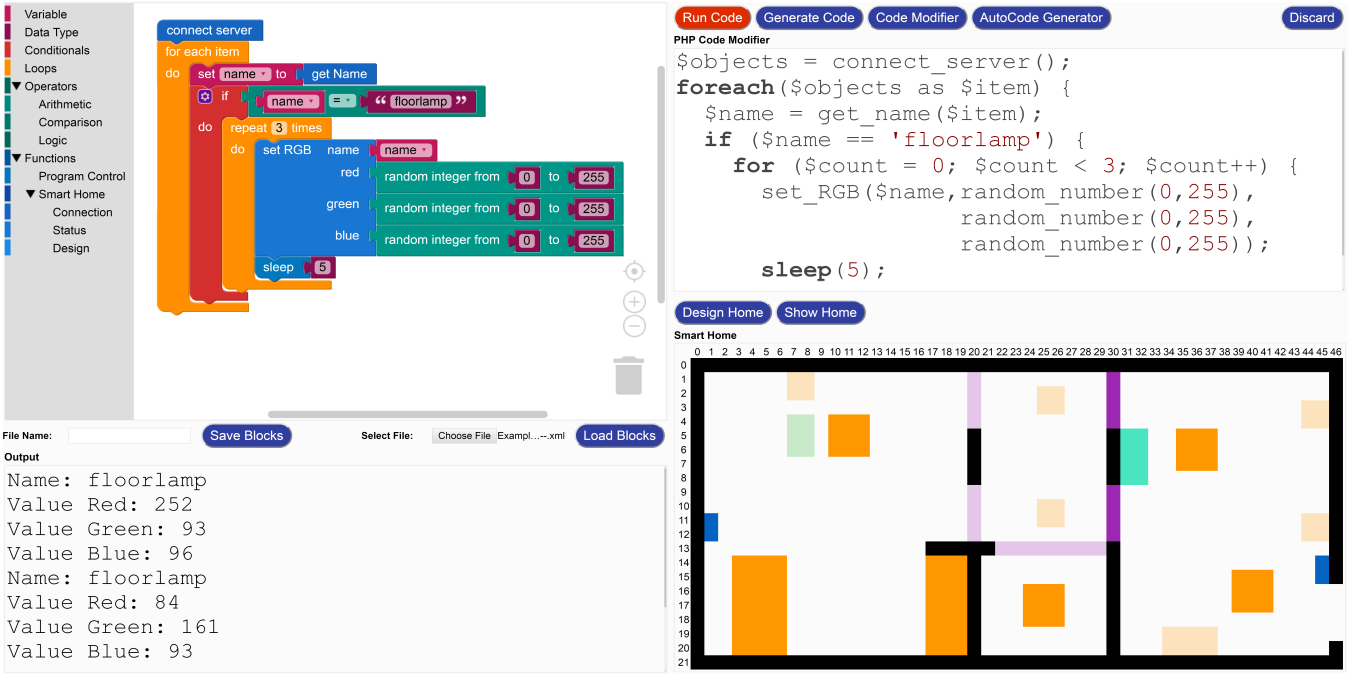

Fig. 2: A sample execution of the programming application.

instructional procedures – in order to enable students to solve the programming tasks. A worked example for each task was attached to half of the computers in paper form. The document contained a visual block-based representation (blocks are snapped together) of the task, along with an explanation of each block and of the code which is generated by the blocks. To the other half of the computers, an instructional procedure for each task was attached in paper form. This document contained a visual block-based representation (blocks are not snapped together) of the task, along with an explanation of each block. The output of the task was also included in both documents. In the first programming task, one of the blocks was used incorrectly in the documents which was highlighted by a different color. The students needed to identify the incorrect block and to replace it by the correct one in order to correctly perform the task. In the second programming task, one whole loop was used in an incorrect format which was also highlighted by a different color. The students needed to identify the incorrect set of blocks and replace them by the correct blocks in order to correctly perform the task. Furthermore, they were encouraged to perform the task using a simple example as an introduction to the particular task. The two programming tasks were:

(1) Showing the name and status of each object in the living lab. The task helps students to learn variables and iterative logics. Students are required to connect to the living lab's server, iterate through the list of objects (*foreach* loop), and fetch the name and status of each object. Then, they assign object name and status to two variables and show them in the *Output* panel.

(2) Changing the status of an RGB light. The task helps students to learn differences between operators as well as working with loops (*for* loops) and conditional statements (*if* statements). Students are required to connect to the server, iterate through the list of objects and find the corresponding RGB light. Then, they change the status of the light using random integers for 3 times, with 5 seconds delay between the changes.

In this respect, the computational thinking concepts are extended by introducing these tasks to the students. We require students to use the supplementary documents in order to identify the issue and where the change should be made, remove extra blocks and add new blocks, integrate them with the rest of the program, and finally test the program.

IV. User Studies

Two user studies were conducted in order to understand the impact of visual block-based programming on young learners' programming skills in the context of smart living labs. The training sessions were conducted with students without prior programming experience (Experiment 1) and with novices who already had minor experiences in visual and text-based programming (Experiment 2). Specifically, we wanted to understand whether the students could assimilate and use supplementary documents – namely worked examples and instructional procedures – including an issue in order to perform programming tasks. In that respect, the students were divided into two groups which were supported either with worked examples (Example Group) or with instructional procedures (Instruction Group). Concretely, the user studies addressed the following research questions:

1) When learning how to program with a visual block-based programming environment embedded in a smart living lab, is it more effective to present learners worked examples compared to instructional procedures? Does this effect depend on gender?

2) When learning how to program with a visual block-based programming environment embedded in a smart living

lab, is interest in computer science and programming fostered more when learners are presented worked examples compared to instructional procedures? Does this effect depend on gender?

In both studies, data with respect to the acquisition of programming skills and with respect to interest in computer science and programming were collected.

In the following, we describe the sample, the training session, the collected data, and the results for both studies.

A. Experiment 1

Sample and Design. A total of 22 8^{th} grade students of a German secondary school (12 girls, 10 boys, age: $M = 13.80$, $SD = 0.56$) participated in the study. The students were randomly assigned to two experimental groups, 6 girls and 4 boys to the Example Group (they revived a worked example for each task), and 6 girls and 6 boys to the Instruction Group (they received an instructional procedure for each task), respectively.

Procedure of the Training Session. The duration of training for each student was 3 hours, with a 15 minutes break. Students were randomly assigned to the two experimental groups at the beginning of the session. The Example Group students were asked to answer the questions of a pre-questionnaire, while the Instruction Group was introduced to the smart living lab; afterwards, the Example Group was introduced to the smart living lab, while the Instruction Group answered the pre-questionnaire (see Fig. 3). All objects and their functionalities in the smart living lab were explained for 20 minutes per group. Then, as gender effects were considered, pairs of two students (2 boys or 2 girls) were assigned to one computer. Each computer showed a real-time full vision of the smart living lab during the session using three IP cameras. All students were introduced together to programming features, structures and principles as well as how to use the programming application for one hour. Before working on each task, students were presented a program introducing the corresponding task. The two programs respectively were (1) demonstrating the name of all available objects in the smart living lab, and (2) changing the status of a dimmable light for one time. At the end of the second task, each group of students went to the living lab to see the changes in reality. All students were asked to complete a post-questionnaire at the end of the session.

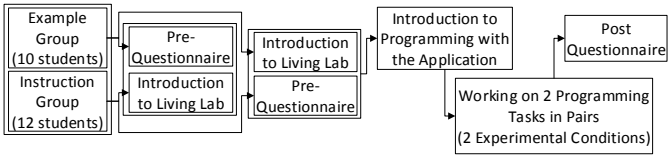

Fig. 3: Procedure of the first experiment.

Acquisition of Programming Skills. It was assessed whether the students were able to perform the tasks or not. Each task consisted of several steps. The performance was operationalized by the rate of steps made without errors. In this respect, at the end of each training session, the generated blocks were checked in each computer. Each block was labeled with a value and the whole program was rated based on the blocks which were correctly used and placed. Furthermore, the number of errors were counted and the type of errors was categorized as *major* or *minor* errors based on students' difficulties in introductory programming which are discussed in [11]. Minor errors are related to the students' syntactic knowledge; for example, missing variable names or typing errors. In contrast, major errors are mostly related to the students' conceptual and strategic knowledge; for example, using an *if* statement to check objects conditions outside of a *for* loop.

Interest in Computer Science and Programming. The subjective data regarding interest in computer science and programming was collected using a questionnaire. The students were asked to rate the items (1) "*is it easy to program with blocks?*", (2) "*do blocks help you to understand program better?*", and (3) "*do you think that it is helpful to be able to see directly in reality whether the program works as desired?*" using a 5 point Likert scale (with 1 being *no*, and 5 being *yes*). Furthermore, the students were asked about their preference of programming with blocks or directly with code using a 5 point Likert scale (with 1 being *definitely with code*, and 5 being *definitely with blocks*).

Finally, the students were asked before each session started in the pre-questionnaire and at the end of each session in the post-questionnaire to rate (1) "*do you think computer science is difficult to understand?*", and (2) "*would you like to learn how to program?*" on a 5 point Likert scale (with 1 "*no*", and 5 "*yes*").

Results. At the beginning of the training session, the students were asked to indicate whether they had programming experience. Only four students answered that they had programmed before: two boys and one girl in the Example Group, and one boy in the Instruction Group. Nevertheless, all four students indicated "*low*" or "*no*" prior experience, thus, the level of prior knowledge was not included in the further analyses.

The following analyses were computed as two-factorial analyses of variance, with the factors *example vs. instruction* and *gender*, respectively. For the questionnaire items, "*no*" was coded with 1, and "*yes*" was coded with 5, respectively. With respect to finding programming easy with blocks, on average, the students indicated a medium level (see Table I); no significant main or interaction effects occurred, all $F < 1$. The students indicated that blocks are helpful in order to understand programs (see Table I); no significant main or interaction effects occurred, all $F < 1$.

With respect to the helpfulness of seeing the impacts of their program in a real-world environment, neither the main effect *gender* nor the interaction effect reached the level of significance, both $F < 1$. The students in the Instruction Group rated the question higher in comparison to the students in the Example Group (see Table I); however, the main effect *example vs. instruction* was not significant, $F(1, 18) = 1.16$, $p = 0.30$.

Concerning the preference for programming with blocks or with code, "*code*" was coded with 1 and "*blocks*" were coded with 5. The students in both groups indicated an indecisive stance (see Table I), however, the Instruction Group indicated a slightly higher preference towards blocks compared to the Example Group, and the girls indicated a higher tendency

towards blocks compared to the boys. However, both main effects just barely missed the level of significance, both $F(1, 17) = 3.15$, $p = 0.09$, *partial* $\eta^2 = 0.16$. The interaction effect was also not significant, $F(1, 17) = 1.14$, $p = 0.30$.

With respect to the question whether they would like to learn how to program (see Table II), before the training session the boys showed broad approval, while the girls were undecided. Accordingly, the main effect *gender* yielded a significant result, $F(1, 18) = 8.89$, $p = 0.008$, *partial* $\eta^2 = 0.33$. However, neither the main effect *example vs. instruction* nor the interaction effect reached the level of significance, $F(1, 18) = 1.38$, $p = 0.26$, and $F < 1$, respectively. After the training session, the boys still indicated that they would like to learn how to program significantly more than the girls, $F(1, 18) = 7.46$, $p = 0.01$, *partial* $\eta^2 = 0.29$. Concerning the main effect *example vs. instruction* and the interaction effect, no significant results occurred, both $F < 1$. Descriptively, the boys and the girls indicated a higher level for liking to learn how to program before the training session compared to after the training session. In order to determine whether this decrease was significant, a regression analysis was performed. For the girls, the regression slope from the after-session score to the pre-session score was not significant, $B = 0.26$, $p = 0.59$ (constant: $B = 1.67$, $p = 0.30$). For the boys, the regression slope from the after-session score to the pre-session score was also not significant, $B = -0.28$, $p = 0.43$ (constant: $B = 5.24$, $p = 0.01$). Thus, the decrease with respect to liking to learn how to program was not significant, neither for the girls, nor for the boys.

With respect to computer science being difficult to understand, on average, the students indicated a medium level before the training session (see Table II); no significant main or interaction effects occurred, all $F < 1$. After the training session, the boys indicated a medium level of difficulty for computer science, while the girls opted more in the direction of "*difficult*". However, the effect *gender* was not significant, $F(1, 18) = 2.97$, $p = 0.10$. Neither the effect *example vs. instruction* nor the interaction effect were significant, both $F < 1$.

After the introduction to programming with the application, the students were asked to perform two programming tasks. Due to technical problems, the results of one group (two girls) in the Example Group were not saved, and thus, cannot be included in the following analyses. On account of the small sample size ($n = 10$ dyads), we decided not to perform analyses of variance with respect to these two tasks. Overall, the students performed 56% of task 1 and 49% of task 2 without errors (see Table III). Concerning both tasks, the students in the Instruction Group performed better than the students in the Example Group. As the students worked in dyads either girls with girls, or boys with boys, but not in mixed-gender groups, we were able to analyze the percentages of tasks solved correctly dependent on gender. In both tasks, the girls performed considerably better compared to the boys.

For both tasks, errors were categorized into "*major*" and "*minor*" errors. In order to take into account the different number of students in the two experimental groups, the number of errors was divided by the number of dyads: 2 girl dyads and 2 boy dyads in Example Group; 3 girl dyads and 3 boy dyads in Instruction Group (see Table III). Overall, in the first task, the number of major errors was higher than the number of minor errors. The students in the Example Group made more major errors than the students in the Instruction Group. Likewise, the students in the Example Group made more minor errors compared to the Instruction Group. In the second task, the number of major errors was higher than the number of minor errors. The students in the Instruction Group made more major errors and less minor errors than the students in the Example Group. Major errors occurred more often than minor errors, and both types of major and minor errors occurred more often among boys than among girls.

Across both tasks, the most common mistakes were setting different values to the same variable (minor error), using loops in an incorrect place (major error); for example, using a *for* loop to iterate through a list while the list is not defined yet, and placing blocks outside of loops and conditional statements where they should be placed within (major error).

B. Experiment 2

Sample and Design. A total of 22 7^{th} grade students of a German advanced secondary school (6 girls, 16 boys, age: $M = 12.45$, $SD = 0.60$) participated in the study. The students were randomly assigned to two experimental groups, 3 girls and 8 boys to each the Example Group and the Instruction Group, respectively.

Procedure of the Training Session. The duration of the training session for each student was 3 hours, with a 15 minutes break. At the beginning of the session, students were randomly assigned to one of two groups (11 students per group) which were trained separately one after another. In the first training group, 6 students (3 girls and 3 boys) received a worked example for each task (Example Group), and 5 students (3 girls and 2 boys) received an instructional procedure for each task (Instruction Group). In the second training group, 6 students (all boys) received an instructional procedure for each task (Instruction Group), and 5 students (all boys) received a worked example for each task (Example Group). The first training group was asked to answer the questions of a pre-questionnaire, while the other group was introduced to the smart living lab; afterwards, students in the second training group were introduced to the smart living lab, while the first group answered the pre-questionnaire (see Fig. 4). All objects and their functionalities in the smart living lab were explained for 20 minutes per group. Then, each student was assigned to one computer. Each computer showed a real-time full vision of the smart living lab during each session using three IP cameras. All students were introduced together to programming features, structures and principles as well as how to use the programming application for one hour. Before working on each task, students were presented a program introducing the corresponding task. The two programs respectively were (1) demonstrating the name of all available objects in the smart living lab, and (2) changing the status of a dimmable light for one time. At the end of the second task, students went to the living lab to see the changes in reality. All students were asked to complete a post-questionnaire at the end of the session.

Acquisition of Programming Skills. It was assessed whether the students were able to perform the tasks or not.

TABLE I: Subjective Data on the Ease of Use

Questions	Example Group			Instruction Group		
	All Users M (SD)	*Female M (SD)*	*Male M (SD)*	*All Users M (SD)*	*Female M (SD)*	*Male M (SD)*
Is it easy to program with blocks?	3.10 (1.52)	3.00 (1.67)	3.25 (1.50)	3.42 (1.24)	3.33 (1.03)	3.50 (1.52)
Do blocks help you to easily understand programs?	3.90 (0.99)	3.83 (0.98)	4.00 (1.15)	4.27 (0.90)	4.20 (1.10)	4.33 (0.82)
Do you think that it is helpful to be able to see directly in reality whether the program works as desired?	3.70 (1.06)	3.50 (1.05)	4.00 (1.15)	4.33 (1.30)	4.33 (1.63)	4.33 (1.03)
Do you prefer to program with block or directly with code syntax?	2.90 (0.74)	3.00 (0.89)	2.75 (0.50)	3.45 (0.93)	4.00 (1.00)	3.00 (0.63)

M: Mean **SD**: Standard Deviation

TABLE II: Subjective Data on Students' Interest

Pre-Questionnaire						
	Example Group			Instruction Group		
Questions	*All Users M (SD)*	*Female M (SD)*	*Male M (SD)*	*All Users M (SD)*	*Female M (SD)*	*Male M (SD)*
Do you think computer science is difficult to understand?	3.30 (0.67)	3.33 (0.52)	3.25 (0.95)	3.58 (0.90)	3.67 (0.82)	3.50 (1.04)
Would you like to learn how to program?	4.00 (0.94)	3.50 (0.84)	4.75 (0.50)	3.58 (1.44)	2.83 (0.75)	4.33 (1.63)
Post-Questionnaire						
	Example Group			Instruction Group		
Questions	*All Users M (SD)*	*Female M (SD)*	*Male M (SD)*	*All Users M (SD)*	*Female M (SD)*	*Male M (SD)*
Do you think computer science is difficult to understand?	3.50 (0.97)	3.83 (0.98)	3.00 (0.82)	3.33 (1.07)	3.67 (1.03)	3.00 (1.09)
Would you like to learn how to program?	3.20 (1.32)	2.50 (1.05)	4.25 (0.96)	3.17 (1.59)	2.50 (1.52)	3.83 (1.47)

M: Mean **SD**: Standard Deviation

TABLE III: Students' Performance

Tasks		Example Group			Instruction Group		
		All Users (per dyad)	*Female (per dyad)*	*Male (per dyad)*	*All Users (per dyad)*	*Female (per dyad)*	*Male (per dyad)*
Task 1	Number of Major errors	8 (2.00)	4 (2.00)	4 (2.00)	11 (1.83)	4 (1.33)	7 (2.33)
	Number of Minor errors	7 (1.75)	5 (2.50)	2 (1.00)	2 (0.33)	0 (0.00)	2 (0.67)
	Rate of task completed without errors	50%	50%	50%	60%	75%	44%
Task 2	Number of Major errors	9 (2.25)	4 (2.00)	5 (2.50)	16 (2.67)	5 (1.67)	11 (3.67)
	Number of Minor errors	5 (1.25)	1 (0.50)	4 (2.00)	6 (1.00)	3 (1.00)	3 (1.00)
	Rate of task completed without errors	45%	59%	31%	51%	67%	36%

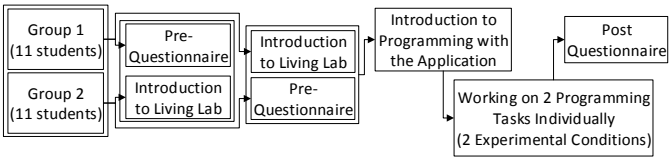

Fig. 4: Procedure of the second experiment.

The students' performance on the programming tasks were evaluated in the same way as in Experiment 1.

Interest in Computer Science and Programming. The subjective data regarding interest in computer science and programming was collected and analyzed in the same way as in Experiment 1.

Results. At the beginning of the training session, the students were asked to indicate whether they had programming experience. All but one (in the Example Group in the first training group) had already programmed before, and on a scale from 1 to 5 they indicated a medium level of prior programming experience ($M = 2.64$, $SD = 0.79$). With respect to this score, the students in the Instruction Group indicated a higher level of prior knowledge compared to the Example Group (Instruction Group: $M = 2.82$, $SD = 0.87$, Example Group: $M = 2.45$, $SD = 0.69$). A two-factorial analysis of variance with the factors *example vs. instruction* and *gender* revealed no significant main effect for *gender*, $F < 1$; however, the main effect *example vs. instruction* and the interaction effect just barely missed the level of significance, for both effects $F(1, 18) = 3.35$, $p = 0.08$, *partial* $\eta^2 = 1.57$, respectively. Thus, for the following analyses, this score was included as a covariate.

With respect to finding programming easy with blocks, neither the main effect *example vs. instruction* nor the interaction effect reached the level of significance, both $F < 1$ (see Table IV). However, the main effect *gender* was significant, $F(1, 17) = 4.75$, $p = 0.04$, *partial* $\eta^2 = 0.22$. Boys indicated a higher level towards easiness of programming with blocks compared to the girls. The effect of prior programming skills was not significant, $F < 1$. Likewise, students indicated that blocks are helpful in order to understand programs (see Table IV); no significant main or interaction effect occurred, main effect *example vs. instruction*: $F < 1$, main effect *gender*: $F(1, 17) = 2.46$, $p = 0.14$, interaction effect: $F < 1$. The effect of prior programming skills was also not significant, $F < 1$.

With respect to the helpfulness of seeing the impacts of their program in real-world environment (see Table IV), neither the main effect *gender* nor the main effect *example vs. instruction* reached the level of significance, both $F < 1$. The interaction effect and the effect of prior programming skills were also not significant, $F(1, 17) = 1.16$, $p = 0.30$ and $F(1, 17) = 2.12$, $p = 0.16$, respectively.

Concerning the preference for programming with blocks or with code, code was coded with 1 and blocks were coded with 5. On average, the students indicated an opinion strongly

towards blocks (see Table IV). The students in the Instruction Group indicated a higher preference for programming with blocks compared to the Example Group. Girls opted more for blocks compared to the boys. However, no significant main or interaction effects were obtained, main effect *example vs. instruction*: $F < 1$, main effect *gender*: $F(1, 17) = 1.33$, $p = 0.26$, interaction effect: $F(1, 17) = 1.06$, $p = 0.32$. The effect of prior programming skills was also not significant, $F < 1$.

With respect to the question whether they would like to learn how to program (see Table V), no significant main or interaction effects occurred before the training session, all $F < 1$. However, the effect of prior programming skills yielded a significant result, $F(1, 17) = 6.30$, $p = 0.02$, *partial* $\eta^2 = 0.27$. After the training session, no significant main or interaction effects were obtained, main effect *example vs. instruction*: $F < 1$, main effect *gender*: $F(1, 17) = 1.38$, $p = 0.26$, interaction effect: $F < 1$. The effect of prior programming skills was also not significant, $F(1, 17) = 2.65$, $p = 0.12$. Descriptively, the boys indicated a slightly higher level for liking to learn how to program before training session compared to after the training session. The girls showed a strong opinion for liking to learn how to program before the training session which remained the same after the training session.

With respect to computer science being difficult to understand, on average, the students indicated a medium level before the training session (see Table IV); no significant main or interaction effects occurred, all $F < 1$. The effect of prior programming skills was also not significant, $F < 1$. After the training session, the students indicated a lower level of difficulty for computer science. Neither significant main nor interaction effects occurred, all $F < 1$. The effect of prior programming skills was also not significant, $F < 1$.

After the introduction to programming with the application, the students were asked to perform two programming tasks. Due to technical issues, the results of two students (one girl in the Example Group and one boy in the Instruction Group) were not saved, and thus, cannot be included in the following analyses. Overall, the students performed 76% of task 1 and 58% of task 2 without errors (see Table VI). Concerning both tasks, the students in the Example Group performed better than the students in the Instruction Group. In task 1, the girls performed slightly better compared to the boys. In contrast, the boys performed slightly better in task 2 compared to the girls. Overall, in the first task, no significant main or interaction effects occurred, all $F < 1$. The effect of prior programming skills was also not significant, $F(1, 15) = 1.11$, $p = 0.31$. In the second task, no significant main or interaction effects were obtained, main effect *example vs. instruction*: $F < 1$, main effect *gender*: $F(1, 15) = 1.10$, $p = 0.31$, interaction effect: $F < 1$. The effect of prior programming skills was also not significant, $F(1, 15) = 3.11$, $p = 0.10$.

For both tasks, errors were categorized into "*major*" and "*minor*" errors. In order to take into account the different number of boys and girls in the two experimental groups, the number of errors was divided by the number of them in each group: 2 girls and 8 boys in Example Group; 3 girls and 7 boys in Instruction Group (see Table VI). Overall, major errors occurred more often than minor errors. There were no large differences between girls and boys, neither with respect to the number of major errors, nor with respect to the number of minor errors. In the first task, the students in the Instruction Group made more major errors than the students in the Example Group. Neither the main effects *gender* and *example vs. instruction* nor the interaction effect were significant, all $F < 1$. The influence of prior programming experience was also not significant, $F(1, 15) = 1.44$, $p = 0.25$. No minor errors occurred in task 1.

In the second task, the number of major errors was slightly higher than the number of minor errors (see Table VI). The students in the Instruction Group made more major errors and more minor errors compared to the students in the Example Group. With respect to major errors, no significant group differences were obtained, main effect *example vs. instruction* and interaction effect: $F < 1$, main effect *gender*: $F(1, 15) = 1.70$, $p = 0.21$. The influence of prior programming experience was also not significant, $F(1, 15) = 2.55$, $p = 0.13$. With respect to minor errors, no significant group differences were obtained, main effect *example vs. instruction*, main effect *gender* and interaction effect: all $F < 1$. The influence of prior programming experience was also not significant, $F(1, 15) = 1.97$, $p = 0.18$.

Across both tasks, the most common mistakes were typing errors (minor error) and placing blocks outside of loops and conditional statements where they should be placed within (major error).

V. Discussion

The results presented in the previous section are now discussed. In this respect, we begin with our findings based on the research questions presented in Section IV; followed by implications and future work.

A. Findings

Our findings are discussed in order to answer the research questions stated in Section IV.

1) *When learning how to program with a visual block-based programming environment embedded in a smart living lab, is it more effective to present learners worked examples compared to instructional procedures? Does this effect depend on gender?*

In general, both boys and girls showed a high tendency toward using visual block-based programming environments. The results in both experiments showed that girls preferred to program with blocks – compared to traditional code syntax – more than boys. Novice boys rated the easiness of programming with blocks significantly higher than novice girls.

With respect to the two programming tasks for the inexperienced learners in Experiment 1, the learners in the Instruction Group performed better than those in the Example Group in both programming tasks. Matching this result, the learners in the Instruction Group indicated that they found computer science less difficult at the end of the session compared to the students in the Example Group. In Experiment 1, concerning both programming tasks, the girls performed better, but still the boys indicated that they would like to learn how to program more than the girls, and the girls indicated (descriptively) higher values with respect to finding computer

TABLE IV: Subjective Data on the Ease of Use

Questions	Example Group			Instruction Group		
	All Users M (SD)	*Female M (SD)*	*Male M (SD)*	*All Users M (SD)*	*Female M (SD)*	*Male M (SD)*
Is it easy to program with blocks?	4.18 (1.08)	3.67 (1.15)	4.38 (1.06)	3.82 (0.75)	3.00 (0.00)	4.13 (0.64)
Do blocks help you to easily understand programs?	4.91 (0.30)	4.67 (0.58)	5.00 (0.00)	4.82 (0.40)	4.67 (0.58)	4.88 (0.35)
Do you think that it is helpful to be able to see directly in reality whether the program works as desired?	4.73 (0.47)	4.33 (0.58)	4.88 (0.35)	4.73 (0.65)	5.00 (0.00)	4.63 (0.74)
Do you prefer to program with block or directly with code syntax?	4.27 (1.10)	5.00 (0.00)	4.63 (0.74)	4.73 (0.47)	4.67 (0.58)	4.75 (0.46)

M: Mean **SD**: Standard Deviation

TABLE V: Subjective Data on Students' Interest

Pre-Questionnaire						
	Example Group			Instruction Group		
Questions	*All Users M (SD)*	*Female M (SD)*	*Male M (SD)*	*All Users M (SD)*	*Female M (SD)*	*Male M (SD)*
Do you think computer science is difficult to understand?	3.10 (0.94)	3.00 (1.00)	3.13 (0.99)	3.10 (0.94)	3.00 (1.00)	3.13 (0.99)
Would you like to learn how to Program?	4.73 (0.47)	4.67 (0.58)	4.75 (0.46)	4.73 (0.65)	5.00 (0.00)	4.63 (0.74)
Post-Questionnaire						
	Example Group			Instruction Group		
Questions	*All Users M (SD)*	*Female M (SD)*	*Male M (SD)*	*All Users M (SD)*	*Female M (SD)*	*Male M (SD)*
Do you think computer science is difficult to understand?	2.91 (1.14)	3.00 (1.00)	2.88 (1.25)	3.00 (0.63)	2.67 (0.58)	3.13 (0.64)
Would you like to learn how to Program?	4.45 (0.52)	4.67 (0.58)	4.37 (0.52)	4.73 (0.65)	5.00 (0.00)	4.63 (0.74)

M: Mean **SD**: Standard Deviation

TABLE VI: Students' Performance

Tasks		Example Group			Instruction Group		
		All Users (per student)	*Female (per student)*	*Male (per student)*	*All Users (per student)*	*Female (per student)*	*Male (per student)*
Task 1	Number of Major errors	12 (1.20)	1 (0.50)	11 (1.38)	15 (1.50)	5 (1.67)	10 (1.43)
	Number of Minor errors	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)
	Rate of task completed without errors	78%	94%	74%	73%	71%	74%
Task 2	Number of Major errors	20 (2.00)	5 (2.50)	15 (1.87)	21 (2.10)	7 (2.33)	14 (2.00)
	Number of Minor errors	15 (1.50)	3 (1.50)	12 (1.50)	19 (1.90)	5 (1.67)	14 (2.00)
	Rate of task completed without errors	60%	53%	61%	56%	54%	56%

science difficult. Thus, the self-perception of the girls does not seem to match the objective achievement.

With respect to the number of errors in the two programming tasks for the inexperienced students (first experiment), major errors occurred considerably more often than minor errors, and boys made more major and minor errors than girls. Thus, the better learning outcomes for girls are also reflected in the lower number of errors. Furthermore, more major and less minor errors occurred in the Instruction Group compared to Example Group.

Concerning the type of errors, on the one hand, the students in the Instruction Group had more issues with the structure of the program, and it seemed that they did not follow the instructions strictly. On the other hand, at least some students in the Instruction Group were able to follow the instruction and solved the programming issue with a minimum number of errors and a maximum performance.

With respect to the two programming tasks for the learners with some prior programming experience (second experiment), the girls performed better in the first task, and they indicated higher willingness to learn how to program compared to the boys. In the second task the boys performed slightly better than the girls, and the learners in the Example Group performed better than the learners in the Instruction Group. Thus, the novice girls with some prior experience in computer science and programming had a better self-perception which matches their objective achievements. Furthermore, the learners in the Example Group performed better than the learners in the Instruction Group across the two programming tasks.

With respect to the number of errors across the two programming tasks, in Experiment 2, major errors occurred considerably more often than minor errors. In the first task, boys made more major errors than girls, and in the second task, girls made more major errors than boys. Thus, the learning outcome for girls in the first and second task is reflected by the number of errors. Furthermore, more major and minor errors occurred in the Instruction Group compared to the Example Group. Concerning the type of errors, on the one hand, the learners in the Instruction Group had more issues with the placing of blocks in their corresponding loops, and it seems that they did not strictly follow the instruction which was given to them. On the other hand, the performance and the number of errors fluctuated among the students in the Instruction Group. One might argue that the students who participated in Experiment 2 already attended a computer science course, and thus, it is not clear why they still made errors related to the program structure. However, they tried different ways to solve the programming issue. Thus, it seems that they made more errors while trying to solve the issue.

2) *When learning how to program with a visual block-based programming environment embedded in a smart living lab, is interest in computer science and programming fostered more when learners are presented worked examples compared to instructional procedures? Does this effect depend on gender?*

According to the analysis of our pre-questionnaire data, overall there is a significant difference in learners' willingness to learn programming by gender. This result is important as it is observed only among learners without programming

experience and repeated in the post-questionnaire where the boys indicated that they would like to learn programming significantly more than the girls. With respect to the students' programming experience, an interesting result shows that the students' willingness to learn programming dropped among inexperienced learners towards the end of the training session. In contrast, the level of perceived difficulty for computer science decreased among inexperienced learners towards the end of the training session. Thus, it can be concluded that programming experience has an important influence on students' view on learning programming and on the perceived difficulty of computer science. The type of supplementary documents did not have a significant influence in this respect in both experiments.

Even if the learners in Experiment 2 showed a positive attitude toward working with a visual programming environment, a one-day programming workshop may have a negative influence on both inexperienced boys and girls. The influence of supplementary documents is not clear as they did not differ significantly in terms of the interest in computer science, student's willingness to learn programming, and the perceived difficulty for computer science. Furthermore, novices indicated an opinion strongly towards using blocks and smart living labs for programming purposes. The students found computer science easier to understand after the training session. However, this short programming workshop had also a negative influence on their willingness to learn programming; thus, further studies should assess effects in a long-term perspective.

B. Implications

The future needs computer scientists and programmers from different gender. However, we are aware that getting young students interested in computer science and programming, especially female learners, is difficult. The results of our training sessions show that students are able to start building their own programs which can be applied to the smart living lab. In this respect, visual programming environments as well as supplementary documents can be helpful to simplify programming for learners and to provide computational support for them. Our main take-home message from these user studies is that visual block-based programming within a smart environment is suitable in order to improve self-confidence among young learners to begin with software programming activities. Furthermore, another advantage of beginning to expose young learners to software programming activities is that learners can realize that computer science can be presented in a way which is not necessarily difficult to understand. The results obtained from our experience are promising, as inexperienced girls were able to perform successfully two programming tasks with the provided programming environment. This is supporting the results presented in [1] that no significant difference was observed in software-based project scores by gender. However, our studies showed that inexperienced students' interest in learning how to program can be decreased, especially among female students. This result is in contrast to results from other programs targeting young female learners in Germany like [24] which showed that by providing opportunities for K-6 female students to have positive experiences in STEM fields, we may have them in the future in STEM professions. While it is not possible to trace our result back to specific features of the training session, it might be possible that the short duration of only one day had a negative influence. Thus, it remains an open point for future research to assess effects of the duration of trainings on attitudes towards computer science and programming.

We would like to emphasize that our results might be affected by the nature and number of programming tasks as well as by the length of the training sessions. Follow-up studies are required in order to understand how young learners react to extracurricular programming workshops running for several days in different contexts within smart living labs such as programming robots and microcontrollers. There, we can find out the sustainability of programming skills that they learned in order to transfer them to other contexts. Another question for future work is to ascertain when K-12 learners can move from visual block-based programming environments to pure programming Integrated Development Environments (IDEs) using traditional text-based code syntax. Although our sample included 44 7^{th} and 8^{th} grade students, this sample size is still too small to generalize the findings on a large scale. Furthermore, studies in different countries and with learners of different socio-economic status might as well shed light on effects of visual block-based programming environments and smart living labs on young learners' knowledge acquisition and interest in computer science and programming.

VI. Conclusions

This paper presents a training session developed for inexperienced students and novices, in order to support the acquisition of programming skills and in order to support a positive view towards computer science and programming. The programming environment is based on a visual block-based programming application within a smart ambient assisted living lab. Thus, the more abstract concept of programming is presented within a real context and tightly connected to real experiences for the learners.

The results show that students are able to build their own programs which can be applied to the smart living lab. Furthermore, the results indicate the importance of supporting and strengthening the learners' motivation in learning programming and their interest in computer science over a longer period.

Two prominent instructional interventions to support knowledge acquisition, namely worked examples and instructional procedures, were compared. As the results did not strongly support one of them, it can be concluded that both are appropriate in order to help learners to acquire basic programming skills.

Future work should investigate how long-term training sessions affect the learners' interest in computer science and programming as well as acquisition of programming skills. In addition, it should be investigated how to adapt training sessions to individual prior knowledge and learning processes in order to support motivation and knowledge acquisition in an optimal way. This study shows the potential of using visual block-based programming environments in the context of smart living labs in order to foster young students' programming skills, and to enable them to begin with software programming activities.

REFERENCES

[1] F. J. Gutierrez, J. Simmonds, N. Hitschfeld, C. Casanova, C. Sotomayor, and V. Peña-Araya, "Assessing software development skills among k-6 learners in a project-based workshop with scratch," in *Proc. of the 40th International Conference on Software Engineering: Software Engineering Education and Training*. ACM, 2018, pp. 98–107.

[2] F. Kalelioğlu, "A new way of teaching programming skills to k-12 students: Code. org," *Computers in Human Behavior*, vol. 52, pp. 200–210, 2015.

[3] S. Y. Lye and J. H. L. Koh, "Review on teaching and learning of computational thinking through programming: What is next for k-12?" *Computers in Human Behavior*, vol. 41, pp. 51–61, 2014.

[4] I. F. de Kereki and A. Manataki, "code yourself and a programar: a bilingual mooc for teaching computer science to teenagers," in *Frontiers in Education Conference (FIE), 2016 IEEE*. IEEE, 2016, pp. 1–9.

[5] V. Barr and C. Stephenson, "Bringing computational thinking to k-12: what is involved and what is the role of the computer science education community?" *Acm Inroads*, vol. 2, no. 1, pp. 48–54, 2011.

[6] L. P. Flannery, B. Silverman, E. R. Kazakoff, M. U. Bers, P. Bontá, and M. Resnick, "Designing scratchjr: support for early childhood learning through computer programming," in *Proc. of the 12th International Conference on Interaction Design and Children*. ACM, 2013, pp. 1–10.

[7] C. S. F. S. Committee, "K-12 computer science framework," 2016, retrieved September 15, 2018 from http://www.k12cs.org.

[8] Y. B. Kafai and Q. Burke, "The social turn in k-12 programming: moving from computational thinking to computational participation," in *Proc. of the 44th ACM technical symposium on computer science education*. ACM, 2013, pp. 603–608.

[9] V. Paramasivam, J. Huang, S. Elliott, and M. Cakmak, "Computer science outreach with end-user robot-programming tools," in *Proc. of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 2017, pp. 447–452.

[10] J. W. Coffey, "A study of the use of a reflective activity to improve students' software design capabilities," in *Proc. of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 2017, pp. 129–134.

[11] Y. Qian and J. Lehman, "Students misconceptions and other difficulties in introductory programming: a literature review," *ACM Transactions on Computing Education (TOCE)*, vol. 18, no. 1, p. 1, 2017.

[12] F. Hermans and E. Aivaloglou, "Teaching software engineering principles to k-12 students: a mooc on scratch," in *Proc. of the 39th International Conference on Software Engineering: Software Engineering and Education Track*. IEEE Press, 2017, pp. 13–22.

[13] D. Bau, "Droplet, a blocks-based editor for text code," *Journal of Computing Sciences in Colleges*, vol. 30, no. 6, pp. 138–144, 2015.

[14] D. Bau, D. A. Bau, M. Dawson, and C. Pickens, "Pencil code: block code for a text world," in *Proc. of the 14th International Conference on Interaction Design and Children*. ACM, 2015, pp. 445–448.

[15] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[16] S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-d tool for introductory programming concepts," in *Journal of Computing Sciences in Colleges*, vol. 15, no. 5. Consortium for Computing Sciences in Colleges, 2000, pp. 107–116.

[17] B. Harvey and J. Mönig, "Bringing no ceiling to scratch: Can one language serve kids and computer scientists," *Proc. Constructionism*, pp. 1–10, 2010.

[18] N. Fraser, "Google blockly-a visual programming editor," *URL: http://code.google.com/p/blockly. Accessed Sep*, 2014, now available at https://developers.google.com/blockly/; accessed 10-September-2018.

[19] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, p. 16, 2010.

[20] J. Wittwer and A. Renkl, "Why instructional explanations often do not work: A framework for understanding the effectiveness of instructional explanations," *Educational Psychologist*, vol. 43, no. 1, pp. 49–64, 2008.

[21] A. Destrebecqz and A. Cleeremans, "Can sequence learning be implicit? new evidence with the process dissociation procedure," *Psychonomic bulletin & review*, vol. 8, no. 2, pp. 343–350, 2001.

[22] R. K. Atkinson, S. J. Derry, A. Renkl, and D. Wortham, "Learning from examples: Instructional principles from the worked examples research," *Review of educational research*, vol. 70, no. 2, pp. 181–214, 2000.

[23] B. M. McLaren, T. van Gog, C. Ganoe, D. Yaron, and M. Karabinos, "Exploring the assistance dilemma: Comparing instructional support in examples and problems," in *International Conference on Intelligent Tutoring Systems*. Springer, 2014, pp. 354–361.

[24] B. Ertl, S. Luttenberger, and M. Paechter, "The impact of gender stereotypes on the self-concept of female students in stem subjects with an under-representation of females," *Frontiers in psychology*, vol. 8, p. 703, 2017.

[25] M. Corporation, "Why europe's girls aren't studying stem," 2017, retrieved September 15, 2018 from http://hdl.voced.edu.au/10707/427011.

[26] J. Huang, T. Lau, and M. Cakmak, "Design and evaluation of a rapid programming system for service robots," in *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*. IEEE Press, 2016, pp. 295–302.

[27] N. Spyropoulou, G. Demopoulou, C. Pierrakeas, I. Koutsonikos, and A. Kameas, "Developing a computer programming mooc," *Procedia Computer Science*, vol. 65, pp. 182–191, 2015.

[28] M. Seraj, S. Autexier, and J. Janssen, "Beesm, a block-based educational programming tool for end users," in *Proc. of the 10th Nordic Conference on Human-Computer Interaction*. ACM, 2018.

[29] MakeCode, "Makecode homepage," 2018, accessed 03-September-2018. [Online]. Available: https://www.microsoft.com/en-us/makecode.

[30] Y. B. Kafai, "From computational thinking to computational participation in k–12 education," *Communications of the ACM*, vol. 59, no. 8, pp. 26–27, 2016.