Assessing Software Development Skills Among K-6 Learners in a Project-Based Workshop with Scratch

Francisco J. Gutierrez
Department of Computer Science
University of Chile
Santiago, Chile
frgutier@dcc.uchile.cl

Jocelyn Simmonds
Department of Computer Science
University of Chile
Santiago, Chile
jsimmond@dcc.uchile.cl

Nancy Hitschfeld
Department of Computer Science
University of Chile
Santiago, Chile
nancy@dcc.uchile.cl

Cecilia Casanova
Department of Computer Science
University of Chile
Santiago, Chile
ccasanov@dcc.uchile.cl

Cecilia Sotomayor
Graduate School, FCFM
University of Chile
Santiago, Chile
cecilia.sotomayor@u.uchile.cl

Vanessa Peña-Araya
Department of Computer Science
University of Chile
Santiago, Chile
vpena@dcc.uchile.cl

ABSTRACT

Recent literature reports a fair amount of initiatives on how to engage younger populations in achieving computational literacy. However, there is considerable less research on how to effectively deliver software development skills in a way that can be accepted and ultimately adopted by this user group. As a way to bridge this gap, we ran an extracurricular project-based workshop, targeting 10-12 years old learners with no prior coding experience, delivered over five days in the computer labs at the University of Chile. In this workshop, participants follow hands-on activities where they acquire the basics of computer programming and develop a small-scale software application using Scratch. These activities showcase that good software engineering practices can be taught to K-6 students, where these students are guided by experienced computer science undergraduate and graduate students. This paper presents a descriptive case study that focuses on assessing how K-6 learners assimilate and use these practices when developing their first computing application in a non-traditional learning experience. In order to do this, we designed and calibrated a rubric to evaluate the software products generated by the workshop participants. Our findings provide further evidence that it is indeed possible to teach initial notions of software engineering to this user group, structuring these constructs in a non-technical language that can be assimilated by novice developers. Furthermore, we did not observe significant differences in this matter according to gender and socio-economic status.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; **Student assessment**; **K-12 education**; *Computational thinking*;

KEYWORDS

Software engineering education; Software development; Rubric; K-6 learners; Scratch; Case study

ACM Reference Format:
Francisco J. Gutierrez, Jocelyn Simmonds, Nancy Hitschfeld, Cecilia Casanova, Cecilia Sotomayor, and Vanessa Peña-Araya. 2018. Assessing Software Development Skills Among K-6 Learners in a Project-Based Workshop with Scratch. In *ICSE-SEET'18: 40th International Conference on Software Engineering: Software Engineering Education and Training Track, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3183377.3183396

ICSE-SEET'18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5660-2/18/05.
https://doi.org/10.1145/3183377.3183396

1 INTRODUCTION

Software development has been increasingly reaching more diverse population, who do not necessarily fit into the traditional "computer scientist" or "software engineer" profiles. For instance, over the last few years there has been an explosion of computational thinking (CT) initiatives [16, 29] that look to introduce children and teenagers to coding, as a way to integrate engineering in K-12 education [17].

According to the K-12 Computer Science Framework [7], CT is a desirable set of skills among younger populations, aligning computer science, science, and engineering by promoting: (1) the development and use of abstractions, (2) the creation of new computational artifacts, and (3) testing and refining the created artifacts. These constructs resonate with the foundations of software engineering (SE), which according to Bollin et al. [4] is a useful drive to instantiate CT as a global framework for problem-solving, and not merely as a way to write source code.

Acknowledging that the main goal of software development is to create, produce, and deploy pieces of software considered acceptable by end-users [27], education and training in SE must encompass a broad array of knowledge and competencies that anyone who will eventually be involved in software production needs to embrace [5]. One of the challenges that SE research still faces in this matter is the need for fresh, empirical evidence [18], particularly in the case of education and training among younger populations.

At first sight, it may seem impossible to teach software development skills to young learners, given that this age group does

not necessarily have working knowledge of SE concepts, techniques, and methods, such as requirement analysis and elicitation, design and abstraction of software solutions to a specific problem, coding in a programming language, debugging, quality assurance, object-oriented programming, and design by patterns. Hermans and Aivaloglou [15], analyzing how young learners participated in a MOOC for learning to code with Scratch, suggest that software development skills, and particularly maintainability, can be effectively delivered to K-12 students. In that respect, Scratch[1] is a visual block-based programming language, translated to more than 70 languages and widely used in both formal and informal educational settings [24]. Likewise, Bollin and Sabitzer [5] studied how to deliver SE in formal educational scenarios (i.e., students enrolled in a course following classes with a fixed schedule and being instructed by a teacher in regular coursework), suggesting that teaching software engineering is indeed possible in high-school.

In this paper we focus on a particular subgroup of K-12 students. We claim that it is indeed possible to teach K-6 learners (i.e., aged between 10 and 12 years old) basic software development skills in extracurricular workshops following a project-based approach, which are usually seen as a complement to specific topics being delivered in formal education courses [7]. Students must pay to attend this workshop. The particularity of our workshop is that it is structured along a project that links both coding and core software development principles, presented in a non-technical language. The analysis of the conducted workshop follows the tradition of empirical SE in the form of a descriptive case study [30], hence extending prior results [15] and serving as fresh evidence taken from a different kind of non-traditional educational setting.

As context scenario, we run our workshop in Chile, where SE education and computer programming are not usually included in the formal course curricula [13]. Therefore, the challenge—and eventual benefits—of introducing software engineering practices to this particular age group are two-sided. On the one hand, it would be possible to directly impact in this target population and measure over time the acceptance and adoption of CT and SE practices—through controlled and longitudinal studies—in both formal and informal educational and training settings. On the other hand, it would be possible to study as future work the impact that the local culture has on the acceptance and adoption of CT and SE practices among younger populations. As such, this work provides a first insight on this matter, building upon and contrasting with the state-of-the-art, developed mostly in America and Western Europe.

As a result, the presented case study seeks to contribute to SE education and training by providing new evidence on how to deliver software development principles to younger populations. Furthermore, researchers and practitioners interested in designing and/or adapting interventions to promote SE in both formal and informal settings can reuse the workshop materials and assessment rubric to measure how software artifacts developed by 10-12 years old learners adhere to basic software development principles. This work represents a step towards understanding how to effectively design and deploy SE learning and training programs tailored to younger populations.

[1] https://scratch.mit.edu

The rest of this paper is structured as follows. We discuss the related work in Section 2, and present our intervention in Section 3. Section 4 describes the case study design, specifying our research question, data sources, and how we evaluated the student projects. Section 5 describes the main results of our case study, which are then discussed in Section 6. Finally, we conclude and provide perspectives on future work in Section 7.

2 RELATED WORK

According to Bollin et al. [4], there are few research and practical initiatives in SE education particularly targeted to promoting software development practices among K-12 learners [4]. Conversely, several block-based programming environments [11, 20, 23], activities [2], and online courses [8] have recently emerged as alternatives to help young children get started with computer programming. While valuable, the focus of these initiatives still remains on promoting CT, rather than specifically focusing on SE.

Hermans and Aivaloglou [15] designed and manage a MOOC for introducing K-12 students to general programming and software development principles. The results obtained from this experience are quite encouraging, as the authors suggest that younger generations are able to learn SE principles under this kind of instructional scenario. For example, this MOOC raises awareness about code smells [1, 14], as a way to encourage software maintanability. Likewise, Robles et al. [26] are concerned about the "clone-and-own" reuse approach in Scratch and how it relates to knowledge acquisition and mastery in CT projects.

The literature also reports automated tools for assessing code production and software quality when developing Scratch-based projects, as a way of assisting both teachers and students in this endeavor. For instance, Hairball [3] and Dr. Scratch [22] can automatically identify and count code smells and other oddities in the development, such as duplicated sprites, issues with sprite naming, dead code identification, use of synchronization and parallelism, level of user interactivity, among others.

Finally, in terms of software production, two of the preferred software development approaches to empower novice programmers are disciplined methods (e.g., incremental) and agile processes [25]. While the former are structured and based on phases and roles, the latter are unstructured, focused on the product, and based on team self-organization. In particular, Robillard and Dulipovici [25] recommend that inexperienced developers should start out following a disciplined process, since these provide a structured recipe to assist students in accomplishing their endeavors.

Also, in the SE community there has been a recent spike in interest in better understanding the nuances of behavioral attitudes expressed by different user groups around software development, particularly millennials [9] and younger populations [15, 21]. Furthermore, related work reports several measures for assessing the quality of the produced software by young programmers. For instance, Tsan et al. [28] used a set of tasks, which are then objectively measured with a rubric by analyzing the generated code. In order to generate a result, all items are then weighted and consolidated according to quality measures, such as: requirements (i.e., ensuring correct and complete functionality), consistency, and usability.

Building upon the reviewed lines of research, we designed in 2012 —and iteratively refined over the years—an extracurricular workshop, linking both initial programming activities with project-based tasks. Through a dedicated set of workbooks that motivate student reflection as well as code design and analysis, we stimulated K-6 learners to acquire basic software development skills, without using excessively technical language and putting into practice disciplined methods.

3 INTERVENTION

We designed a 5-day workshop to encourage children between 10-12 years old to develop their own, small software project. The goal is to evaluate if children are able to understand certain good SE practices and use them during the development of a project. Children are not explicitly told that they are being taught good SE practices. At the beginning of each session they receive a workbook for the day, that states: (1) the goal of the class, (2) associated learning outcomes, (3) related CT and Scratch programming concepts, and (4) activities that are used to introduce SE practices and put into action the CT and programming concepts in a hands-on manner. Three types of activities are used in the workbooks: (1) explain what a code snippet does, (2) answer a question about a sample project, and (3) modify a sample project in some way and discuss the results of this modification.

Since 2012 we have run 1-2 of these workshops per year, during the winter and summer holidays. In this paper we analyze the 2016 winter workshop, because it is the result of extensive piloting conducting over the last four years. In particular, by the end of the workshop, most participants showed a positive attitude and effectively proved they acquired basic CT and programming skills [13].

Children are enrolled in the workshop by their parents, as these workshops have a monetary cost (used to pay instructors and tutors). We use Facebook and Twitter to advertise these workshops, and they are held at the computer labs of the Department of Computer Science at the University of Chile. Each of the five sessions is 3.5 hours long, with a 30 minute break. Participating students come from different public and private schools in the country, where children coming from private schools usually have broader access to current technology. They also usually benefit from more personalized interaction with their teachers, as class sizes are smaller in private schools. Workshop participants are selected so as to have a balanced cohort in terms of gender and socio-economic status, in order to encourage diversity among participants. No student had prior programming experience at the beginning of the intervention.

Approximately sixty students enroll in each workshop, and each instructor is in charge of at most 30 students, assisted by one tutor for each 5 students. These numbers have been determined empirically over the different pilots, as a way to work with the participants in an engaging way. The goal is to answer any student questions as quickly as possible, encourage active learning, and motivate students to work on their own projects. Half of the teaching staff are women, and all of the instructors and tutors are advanced undergraduate and graduate students of the University, with experience in working with K-6 learners. Each class starts with an introduction given by the instructor to motivate the topics that will be addressed during the session, followed by hands-on activities with the computer and project-based tasks. All workshop participants work individually on a computer, and we actively encourage participation and collaboration.

We now describe the topics covered over the different sessions, the expected learning outcomes, the associated activities and the SE practices showcased or reinforced in each session. Accompanying workbooks and scripts are available for download from our institutional repository[2]. Note that topics and sample projects are presented in growing order of complexity.

3.1 Day 1: Problem Solving

The learning outcomes are: (1) identify the data relevant to solve a problem, (2) propose a set of sequential steps to solve a problem, and (3) recognize the main elements of the Scratch programming environment. The main CT concept taught is *algorithm*, exemplified through everyday activities. Programming concepts are first introduced using a simple Scratch program that only includes event, look, and motion blocks, and later through a more complex animation program that adds control, operator and sensor blocks. Students are encouraged to execute each program and determine which block corresponds to a particular object action. Moreover, they are asked to modify these programs, and to test and explain what happens. Students are first asked to make superficial changes, like change the rate of movement, but are later asked to add new functionality.

The SE practices involved are: (1) source code reviewing, (2) reading and interpret functional requirements for designing an algorithm in a CS-Unplugged context [2], and (3) extending programs through the addition of new code. This last practice requires students to create abstractions, reuse existing code, identify where the change should be made, add new blocks and integrate them with the rest of the program, and to finally test the modified program.

3.2 Day 2: Animations with Scratch

The learning outcomes are: (1) understand simple animations and (2) create an animated story in Scratch. The notion of algorithm is reinforced by making students think about and write a storyboard in a CS-Unplugged context. The CT concepts are extended by introducing synchronization and message broadcasting. We ask students to read, understand, and execute two programs that animate the same short story: one synchronizes two sprites using time, while the other uses message broadcasting.

The SE practices introduced during the first class are reinforced. In addition, we introduce two synchronization software patterns, showing the advantages of message passing over concurrent time-based synchronization. During the last hour of the session, students are challenged to start working on their own software projects.

3.3 Day 3: Videogames in Scratch

The learning outcomes are: (1) understand the code for small videogames in Scratch, (2) modify these small videogames, and (3) create their own videogame. The CT concepts included in this session include the definition of control-flow statements, such as conditions and looping. Students are asked to execute and understand the control blocks used in partial versions of well-known games (pacman, pong, etc.) that are available as example projects in Scratch

[2]http://bit.ly/scratch-dcc-uchile

and were slightly modified to fit with the scope and activities designed in the class workbook (e.g., by illustrating the application of SE good practices). We call these examples "sample projects" in the rest of this paper. We also introduce the concept of variables, which are used to keep score in the sample games. As these games are interactive, students must learn how to move sprites and use sensors.

During the first two hours, participants are asked to modify and add simple functionalities and sprites to the sample projects. During the last hour, they are encouraged to continue the development of their own project. We reinforce the idea of remixing what they have learned during the previous days with the new instruction blocks taught that day. The SE practices strengthened in this class are: (1) event-based interaction, (2) pattern-driven design (e.g., collision detection and motion), (3) reading and writing code documentation, and (4) the use of meaningful variable and sprite names.

3.4 Day 4: Data Representation

The learning outcomes in this session are: (1) understand how the computer represents discrete and continuous data, such as numbers, text, and images, and (2) reinforce software development skills during the completion of the student projects. The first part of this session consists of CS-Unplugged activities that explain how information is internally represented in a computer. They learn that images are discrete, that colors are represented using triadic codes, and that numbers are stored using a binary system [2].

During the second part of the session, students continue working on their software projects, guided by their tutors. This time, students develop a finer notion of testing and code inspection on their own projects.

3.5 Day 5: Data Algorithms

The learning outcomes for the final session are: (1) understand how the computer internally searches for and sorts data, and (2) complete the project and present it to other students and their parents. The final day of the workshop is used to introduce search and sort algorithms in a CS-Unplugged context [2]. They also work on their projects during an hour.

The final activity for the day is a presentation session, where students show and explain their projects to their classmates and their parents in a closing social event. We estimate that students spend around 6 hours in total working on their own projects, which corresponds to about a third of the time allotted to the workshop.

4 METHODOLOGY

The objective of this work is to understand the impact that the early introduction of SE practices had in an extracurricular learning experience. These practices are usually introduced at a much later stage, when students are formally enrolled in computer science or software engineering programs, and have already had some experience developing programs. In particular, we wanted to understand whether students this young could assimilate and use these practices. Concretely, we focused on the following research questions:

RQ1: How are the SE practices included in the workshop assimilated and used?

RQ2: How does gender affect the assimilation and use of these practices?

RQ3: How does socioeconomic level affect the assimilation and use of these practices?

Given the nature of these research questions, we decided to conduct a descriptive case study [30], where the unit of analysis is the workshop presented in the previous section. Since we only study the 2016 cohort, it is a single case study.

4.1 Data sources

Given our research questions, the following data sources were considered relevant for our case study: (1) basic demographic information, (2) workshop entry and exit surveys, (3) workshop materials, and (4) the projects created by the students. We now discuss each data source, indicating what data was collected and how.

4.1.1 Basic demographic information about the 2016 cohort (e.g., age, sex, public/private school, etc.). The academic unit in charge of the workshop collected data like the age and sex of the participants, as well as the name of the school where they study. One of the co-authors of this paper looked up the classification of these schools in a Ministry of Education database. The data is kept anonymized, students are identified using a numeric id.

The sample for this case study consists of 55 students (26 girls and 29 boys), aged 10 to 12 years old, all with no prior programming experience. 68.2% of the workshop participants attended public schools, while 31.8% attended private schools. This information was obtained in a self-report questionnaire applied to all students during enrollment. This breakdown is the opposite of country averages, as reported by UNICEF[3]: 38% of Chilean students attend public schools and 60% private schools.

4.1.2 Entry and exit surveys applied to workshop participants. These surveys are structured questionnaires designed by two of the co-authors of this paper. These surveys try to gauge what the students understand by "computer" and "computing", and have been applied to participants of this workshop since 2013. These questionnaires have both open and closed questions and students are identified using the previously assigned numeric id.

In order to analyze this dataset, the questionnaires were transcribed and tabulated by five co-authors of this paper. Data analysis was structured in a systematic way around a grounded theory approach, consisting on iterative open, axial, and selective coding, and then theory framing using affinity diagrams and triangulation with the collected socio-demographic data.

4.1.3 Workshop materials: workbooks and sample projects. Each day of the workshop has its own workbook, which lists the learning objectives for that day, explains some concepts relevant to the day's activities, as well as the activities themselves. These materials were designed by three co-authors of this paper, who are participant-observers of this case study, as they designed and have taught several editions of this workshop.

4.1.4 Projects created by the students. Students are told from the beginning that they are expected to work on a final project using what they have learned in the workshop, and are given formal

[3] http://unicef.cl/web/educacion/

Table 1: Rubric for evaluating student projects

	None (0)	Beginner (1)	Developing (2)	Competent (3)
(A) % of sprites that only control themselves	[0%, 10%]	(10%, 40%]	(40%, 70%]	(70%, 100%]
(B) % of blocks that have a single purpose	[0%, 10%]	(10%, 40%]	(40%, 70%]	(70%, 100%]
(C) documentation	0 comments	1-2 comments	3-6 comments	> 6 comments
(D) % of reachable blocks	[0%, 10%]	(10%, 40%]	(40%, 70%]	(70%, 100%]
(E) % of items with appropriate names	[0%, 10%]	(10%, 40%]	(40%, 70%]	(70%, 100%]
(F) % of superficial changes w.r.t sample projects	(70%, 100%]	(40%, 70%]	(10%, 40%]	[0%, 10%]
(G) project novelty	the expected behavior is not clear	quite similar to a sample project	extends a sample project in a novel way	quite different from the sample projects, or integrates 2 or more sample projects

instructions on this task by the end of day 2. Students were free to pick the subject matter and of their projects, and could either start from scratch or choose to extend one or more of the sample projects provided during the workshop.

These projects were downloaded for analysis at the end of the workshop. The projects were identified using the numeric id assigned to each student.

4.2 Evaluating Student Projects

As a first step to measure the degree of acquisition of the software development skills addressed in the workshop, we evaluated the final projects with two assessment tools: Dr. Scratch[4] and a project evaluation rubric designed by the authors of this paper.

Dr. Scratch is an automatic assessment tool, specifically tailored to measure CT skills in Scratch projects showing a high degree of correlation between human and automatic evaluations [22]. This instrument measures specific CT dimensions, based on the structure of Scratch applications: (1) logical thinking, (2) data representation, (3) user interactivity, (4) flow control, (5) abstraction and problem decomposition, (6) parallelism, and (7) synchronization. Each dimension has 4 mastery levels: None, denoting that little or no mastery; Beginner, denoting a basic mastery; Developing, denoting that there is some mastery, and Competent, denoting competency. Being competent assigns the highest score (3), whereas None does not assign any points (0).

In order to complement the evaluation yielded by Dr. Scratch, we designed a project evaluation rubric, based on the specific software development skills delivered in the workshop. After a calibration step, every project from the 2016 cohort was evaluated using this rubric by one of the 4 co-authors of this paper. We now describe these steps in more detail.

4.2.1 Rubric design. We followed an iterative process to create the rubric shown in Tab. 1, that assesses the level of mastery that the students achieve with regard to several dimensions related to the SE practices mentioned in Sect. 3. Two of the authors of this paper were involved in this process: one of such is also a co-author of the workbooks, and the other is an external expert in software engineering education. Both reviewers individually and independently identified a set of SE practices motivated in the activities as explicit or implicit prompts. Once these lists were completed, both authors discussed and reached agreement on the final set of SE practices to be evaluated in the rubric. We list these practices here to facilitate the discussion:

(1) review existing and own source code
(2) read and understand functional requirements
(3) extend programs through the addition of new code
(4) adequate use of names (e.g., sprites and variables)
(5) pattern-based design and implementation
(6) testing
(7) read and write documentation

Some of these practices can be directly measured, and have corresponding dimensions in our rubric: this is the case of practices (4) and (7), which correspond to dimensions (E) and (C), respectively. Practice (5) can be indirectly measured by Dr. Scratch, by combining the scores along the logical thinking, flow control, parallelism, and synchronization dimensions.

We cannot evaluate practices (1), (2) and (6) directly in the final projects, so we assess the use of these practices by proxy, defining several dimensions that measure the quality of the project code. We posit that students who follow these practices produce code of better quality, specifically with respect to cohesion (dimension (A)), functional decomposition (dimension (B)), and amount of reachable code (dimension (D)). Note that the descriptions of these dimensions in Tab. 1 show how we have adapted these dimensions to the Scratch.

Finally, dimensions (F) and (G) evaluate practice (3). On the one hand, we wanted to see how much of the sample projects were being incorporated into the final projects (dimension (F)), and on the other hand, we wanted to gauge if the participants could use what they learned in the course to extend the sample projects in new and interesting ways (dimension (G)).

Just like Dr. Scratch, we defined 4 levels of mastery for each dimension: None, Beginner, Developing, and Competent. Competent assigns 3 points to the evaluation score, whereas None does not assign any points. As there is an even number of levels, the middle level does not become a "catch-all" category.

Since more that one researcher was going to apply the rubric, we explicitly quantified the cross between dimensions and mastery levels, in order to allow a more objective application of the rubric. The exception is dimension (G), where we described the expected products for each mastery level.

4.2.2 Rubric calibration. We ran a pilot evaluation of the rubric in order to both calibrate the instrument and assess its applicability

[4]http://www.drscratch.org

Table 2: Inter-rater agreement (κ) for rubric calibration

	Stories	Light remix	Strong remix
Raters A–B	0.684	0.565	0.800
Raters A–C	0.845	0.557	0.654
Raters B–C	0.836	0.673	0.673
Mean	0.788	0.598	0.709
SD	0.090	0.065	0.079

by the research team. The goal of such evaluation was to measure the reliability of the instrument and to detect possible improvements in the formulation of the rubric prompts.

In order to assess the reliability of the rubric, three co-authors of this paper individually and independently applied the rubric to the same three projects in the same order. A manual inspection of all student projects revealed that they could be grouped into three groups: (1) interactive stories, (2) lightly-remixed videogames, and (3) strongly-remixed videogames, all inspired by the examples provided in the workshop. Therefore, we randomly selected one project per category, in order to have a broad coverage of student realizations for calibrating the rubric. Table 2 reports the Cohen's kappa (κ) for inter-rater agreement for the three evaluators under each kind of project.

Given these results, we consider the rubric to be reliable for assessing software development skills in the sample of student projects.

4.2.3 Rubric application. As a result, 4 co-authors of this paper independently evaluated the student projects using the rubric following the same protocol. This protocol was structured in three stages: (1) evaluators interacted with the program to get a first impression on its main functionality, (2) evaluators conducted black-box tests in order to contrast the offered functionality with that expected by an expert, and (3) evaluators carefully inspected the code for 2-3 minutes and filled in the rubric. In order to reduce potential biases on this assessment, projects were anonymized before the evaluation.

5 CASE STUDY

In this section we describe the findings of our case study. We start by showing the distribution of the project evaluation scores produced by both our rubric and Dr. Scratch. We also compare the total scores produced by both evaluation rubrics. We then discuss how the definition of "computing" changes over time, as expressed by the participants in their own words in the entry and exit surveys. Finally, we then discuss project scores by gender, by socio-economic level and by project type.

Figure 1a shows the distribution of the evaluation scores assigned by Dr. Scratch to the student projects. Here we see that three dimensions have a low level of dispersion: flow control, abstraction and problem decomposition, and user interactivity. This is due to the fact that Dr. Scratch scores projects on the types of instructions that are used, and in the case of these three dimensions, neither the workbooks nor the sample projects included the types of instructions needed to obtain more points in these dimensions. In the case of data representation, synchronization and logical thinking, we see a little more dispersion. In the case of data representation,

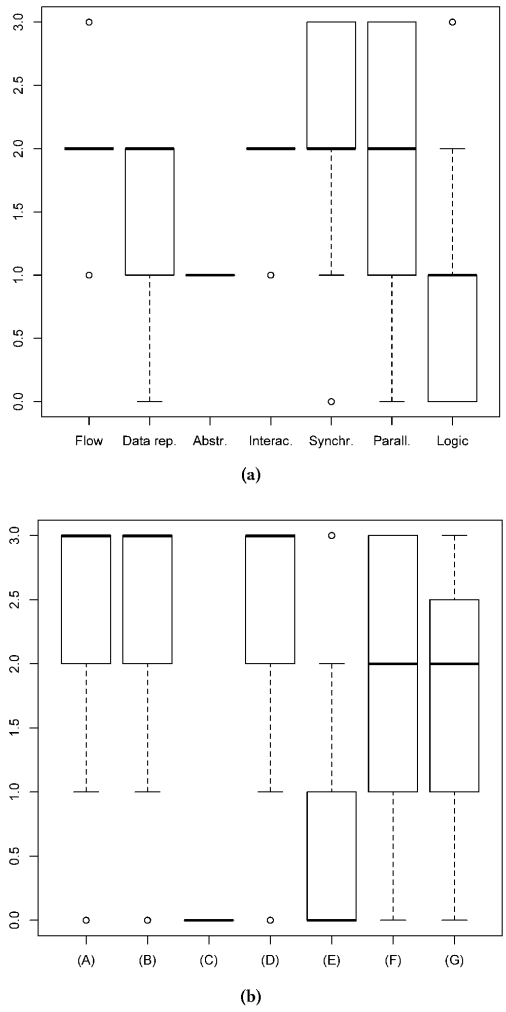

Figure 1: Project assessment: (a) Dr. Scratch scores and (a) our Rubric scores.

almost all projects modified object properties (1 point), but not everyone used variables (2 points). We see the opposite in the case of synchronization, since all three behaviors are explicitly included in the workshop material for day 2. In the case of logical thinking, most projects have no conditional statements (0 points) or if statements (1 point), but there are some projects that make use of if-else statements (2 points) and some that even make use of logical operators (3 points). Finally, in the case of parallelism we see the largest amount of dispersion, which had to do with the type of sample project that the participants used as a base for their projects. Like with synchronization, projects based on the material for day 3 had high scores for parallelism.

The distribution of the scores generated using our rubric are shown in Fig 1b. Here we see that cohesion (dimension (A)), functional decomposition (dimension (B)), and amount of reachable

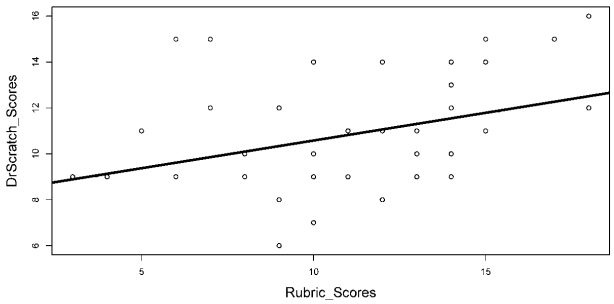

Figure 2: Relation between Rubric and Dr. Scratch scores (r = 0.33, indicating a weak positive correlation).

code (dimension (D)) are distributed similarly, meaning that a large amount of the sprites produced by the students control only themselves, most blocks have a single purpose and most of the blocks are reachable. On the other hand, we see that no student projects were documented (dimension (C)), and a large amount of projects had poorly named sprites, variables and messages (dimension (E)). The final two dimensions, (E) and (F), have a higher level of dispersion than the rest of the dimensions. In the case of (E), we see reuse of the sample projects, with varying levels of changes. Finally, in the case of (F), we see that there are some outliers that add new behaviors, but most are quite conservative in what they add.

We wanted to understand how the scores computed using our rubric compared to those generated by Dr. Scratch. Both rubrics consider 7 dimensions with 4 mastery levels, producing a maximum score of 21. Figure 2 shows a scatter plot of the resulting evaluation scores. We removed from this representation 6 projects that Dr. Scratch was unable to assess due to technical difficulties with the online application. Here Pearson's correlation (r) = 0.3300173, indicating weak positive correlation between scores along both scales for the set of student projects. Note that Dr. Scratch evaluates the use of Scratch instructions and existence of code smells, while our rubric focuses more on the development process.

Table 3 shows how the participants' perceptions of what "computing" is changes between the beginning and end of the intervention. At the beginning, students associate computing with the physical computer, and it the most frequently used word. After the workshop is over, the same amount of participants use it to describe computing, but is is now the fourth most popular word. The word `games` retains it spot in the list, increasing in frequency. This may be due to the fact that all three sample projects for day 3 are games. The word `program` jumps from third to first place, more than doubling in frequency, as participants now associate the action of programming with "computing". Finally, we see that there is a slight increase in the number of participants that mention `create` and `make`, with `create` dropping one spot in the ranking and `make` rising two.

We ran a Mann-Whitney U test to verify whether the median values of each rubric dimension differ or not across gender. According to the results presented in Tab. 4, we did not find statistically significant differences in performance according to gender.

Table 3: Definition of "computing" – top 5 list of words used before and after the workshop.

Words (before)	Frequency	Words (after)	Frequency
`computer`	12	`program`	28
`games`	11	`games`	19
`program`	11	`make`	16
`create`	10	`computer`	12
`make`	10	`create`	11

Table 4: Mann-Whitney U Test: Rubric Score (RS) vs Gender

	n	(A)	(B)	(C)	(D)	(E)	(F)	(G)	RS
F	26	3	3	0	3	1	1	2	13
M	29	3	3	0	3	0	2	2	13
U		316.5	315.0	377.0	466.0	369.0	336.5	369.0	
p-value		0.195	0.232	N/A	1	0.094	0.896	0.481	0.899

Table 5: Mann-Whitney U Test: Rubric Score vs Socioeconomic Status

	n	(A)	(B)	(C)	(D)	(E)	(F)	(G)	RS
Private	14	3	3	0	3	0	2	2	13
Public	30	3	3	0	3	0	1.5	2	12
U		222.5	198.0	210.0	218.5	185.5	146.5	212.5	180.0
p-value		0.637	0.739	N/A	0.817	0.479	0.101	0.958	

We also ran a Mann-Whitney U test to verify whether the median values of each rubric dimension differ or not across socio-economic status. Students from private schools in Chile tend to have a better socioeconomic status than those that go to public schools. In this case, we only consider the participants that indicated the name of the school they attend during the workshop enrollment process. According to the results shown in Tab. 5, we did not find statistically significant differences in performance according to socioeconomic status.

Finally, we counted how many times the sample projects were used by the students. When a student project incorporated significant parts of two or more kinds of sample projects, each sample project was assigned an equal amount of partial credit. For example, a student project that combined whack-a-mole with a narrative story meant that we added 0.5 points to "whack-a-mole" and 0.5 to "stories". The counts were as follows: 3 "new", 10 "pacman", 9.5 "pong", 23.5 "stories" and 10 "whack-a-mole". Here we see that the most used sample project was "stories" from day 2, and that the three sample projects from day 3 appeared in a similar amount of projects. We also see that there is a "new" category, indicating that these projects had elements and behaviors that were not specified in the workbooks, nor in the sample projects.

6 DISCUSSION

We now discuss the results presented in the previous section.

6.1 Summary of Findings

We begin by answering the research questions stated in Sect. 4.

Table 6: SE Practices and their Level of Assimilation and Use. Note that in the case of practice (5), we directly refer to the corresponding Dr. Scratch dimensions.

SE Practice	Dimensions	Level
(1) Review existing and own source code	A, B, D	✓✓
(2) Read and understand functional requirements	A, B, D	✓✓
(3) Extend programs through addition of new code	F, G	✓
(4) Adequate use of names	E	X
(5) Pattern-based design and implementation	Logic, Flow, Parall, Synchr	~
(6) Testing	A, B, D	✓✓
(7) Read and write documentation	C	X

RQ1: How are the SE practices included in the workshop assimilated and used?

In the case of practices (1), (2) and (6), we see strong evidence that these SE practices are indeed assimilated and used by the workshop participants, as all three of the associated dimensions score Developing to Competent in Fig. 1b. We use a double checkmark (✓✓) to indicate this in Tab. 6. We also see relatively strong evidence for practice (3), since the median for both practices (F) and (G) in Fig. 1b is 2, so we have put a single checkmark in the corresponding row of Tab. 6.

In the case of practice (5), we use a ~ in Tab. 6 to show that the evidence for the assimilation and use of this practice, as defined, is not clear. Since the dimensions defined by Dr. Scratch focus on whether projects make use or not of specific Scratch language constructs, a project would need to make use of a variety of blocks and patterns in order to get an overall Developing to Competent score when grouping several Dr. Scratch dimensions, like we did in this case. Moreover, since several of the Competent-level blocks required by Dr. Scratch were not included in the workshop materials. Participants also only spent approximately 6 hours working on their projects, so there may have not been enough time to explore new instructions on their own.

If we divide practice (5) into two sub-practices, (5a) focusing on Logic and Flow, and (5b) focusing on Parall and Synchr, we can be more conclusive about their assimilation. Participants have not yet seemed to master more complex cases of control flow, like conditionals nested within looping blocks, but did not have much trouble understanding and using messages as a synchronization mechanism between sprites.

Finally, in the case of practices (4) and (7), we see little to no evidence of these practices in the student projects, which is indicated using a X in Tab. 6. In general, participants used default names for sprites and messages, which is not a problem at the moment, since these projects are relatively small, but this does not bode well for projects as they get larger [1]. The same goes for documentation: it is important to learn what needs to be documented, as we want to avoid the extreme cases of no comments vs. comments about every single instruction. Note that we are currently studying if the order and depth with which the topics are covered affects student assimilation of Scratch and SE practices.

RQ2: How does gender affect the assimilation and use of these practices?

According to the analysis of our data (c.f. Tab. 4), overall there is no significant difference in rubric scores by gender. This is a promising result. We could also attribute this result to the gender balance of the teaching staff. As mentioned in Sect. 4, we take special care to reach gender parity when selecting course tutors, since workshop participants spend a lot of time with them. These tutors many times become role models to the students, as they are closer in age to the participants than the instructors. Cheryan et al. [6] have shown that role models show significant effects on female students' self-confidence when pursuing careers in STEM fields. So, by targeting K-6 learners, we may be reaching girls with a positive and constructive STEM experience before they "self-select" out of STEM fields [10]. One might argue that the students that participate in these workshops are those who already have an interest in the computing field, but, as stated in Sect. 4, in our experience it is usually the parents that select the workshops, and not the students.

RQ3: How does socioeconomic level affect the assimilation and use of these practices?

As with gender, we did not detect a significant difference in rubric scores by socioeconomic status (c.f. Tab. 5). We did not expect to see this result, as students at private schools tend to have earlier and broader access to technology, including Internet access. However, when taking into account the words used to define "computing" (c.f. Tab. 3), we see that there is general misconception of what computing is, and students tend to focus on the physical computer. This is similar to what has been reported by other authors [12]. Even the inclusion of CS-Unplugged activities did not do much to change this perception. Similar to the trend we see in gender, we seem to be reaching students before the digital divide is evident by targeting K-6 students. This reinforces the idea that the earlier CT and SE are introduced in the standard curriculum, the better.

In order to understand if the answers to RQ2 and RQ3 can be generalized, we are now building the case study database for this year's workshop, as this year's cohort has a similar gender and socioeconomic distribution as the 2016 cohort.

6.2 Implications

The future needs software engineers from different cultures, gender and socioeconomic status, among other characteristics, but we are all aware of how hard it is to get younger people interested in CS and SE, especially women. The results of our extracurricular workshop show that K-6 learners are mature enough and know enough mathematics to start thinking about building their own small software projects. In the current SE curriculum, students gradually adopt good SE practices over time, through exposure to projects of different scope and complexity. Our main take-away from this case study is that we do not need to wait until students are in a formal CS or SE program to begin exposing them to good SE practices.

Another advantage of starting early is that students, in particular women, have not yet seemed to internalize socio-cultural biases about "who" works in computing. In our case study, we did not see any significant difference in project scores by gender or by

socioeconomic status. This is quite different from other programs targeting K-12, like the International Olympiad in Informatics (IOI). Recent statistics show that female participation in the IOI is even lower than the already low participation of women in computing, and their performance has been historically lower than that of their male peers [19].

We would also like to emphasize that the idea of this paper is not that everyone should become a software developer. What is important is that in this data-driven world, knowing how to manage your own data and make use of it is going to be an increasingly valuable life skill. We are not teaching K-6 learners good SE practices in order to convince them to become software engineers. We are doing this because we are convinced that everybody should learn the fundamental computing concepts that drive most of the applications we now use daily, and to give them the opportunity to explore computing.

6.3 Lessons Learned

We now list some of the lessons we learned about our workshop:

Show participants real student projects from previous workshops in an early fashion. We believe that participants are now mainly basing their projects on the "stories" sample project, as this is the sample that is used the day that they start working on their projects. We know that some percentage of the students will change the focus of their project the next day, when they see more complex, game-like sample projects. However, what is not clear is how many participants keep working on their "stories"-style project because of the sunk cost. At the next workshop, we plan to show students more complex examples at the beginning of the workshop. These will probably be selected from previous student projects, so that students get a clearer idea of what they can achieve in the time assigned to the project during the workshop.

Reinforce the practices of appropriate names and documentation. Even though the workbooks and sample projects all made use of appropriately named variables, sprites and messages, participants usually resorted to default values, which usually makes it harder to manually trace through the code. In the case of documentation, this does not necessarily need to be in the form of code comments. We could ask participants to informally specify what their project is going to do. The idea is that this will help them keep track of what they are trying to do, and it would help us better evaluate the process, instead of just the final product. Ideally, participants could keep a blog, updating it every day to specify what functionality they tried to implement that day, what they managed to test, and they managed to implement correctly that day. Since K-6 students are not always accustomed to writing in a freeform style, we can give them prompts to guide the documentation process.

Incorporate new Scratch instructions in the material. Several Scratch instructions have been introduced in order to reduce sprite and code duplication, like custom blocks and instance clones. We need to evaluate if K-6 learners are mature enough to understand these concepts, and if it is feasible to include these instructions and the necessary exercises during an already busy week.

6.4 Threats to Validity

We have specified the protocol of our case study and the analysis of the resulting data in detail, in order to aid replicability. In order to allow the verification of the case study, the corresponding database (including student projects) will be made available upon formal request and approval from our Institutional Review Board.

6.4.1 Internal Validity. The workshop used for informing the conducted case study is the result of iterative refinement over the years. In particular, the first version of the workshops was conceived in 2012 and the data used for the empirical analysis correspond to that gathered on 2016. We acknowledge that the length of the workshop may affect our results, a follow-up study is required to understand how much students retain of what they learned. Also, we need to replicate the analysis with new cases, in order to ensure the consistency of the reported findings.

6.4.2 External Validity. Although the sample size for our analysis was $n = 55$, it is relatively small for generalizing our findings to the entire population. Also, the experience was carried out in a particular socio-demographical context in Latin America, so we cannot ensure that the case study findings could be generalizable, due to the lack of evidence on the influence of cross-cultural in the advancement of software development practices among younger populations.

6.4.3 Construct Validity. To avoid threats to construct validity, we defined a clear protocol for designing the evaluation rubric, and we also validated it through a calibration step that involved three of the co-authors of this paper and external sources of data. To avoid subjective evaluations, all data that could not be obtained in a straightforward way was coded or evaluated by 2 or more co-authors of this paper. Finally, the relationships between rubric dimensions and SE practices were specified *before* the student project scores were computed.

7 CONCLUSION

Through a descriptive case study with 55 students (26 girls and 29 boys), aged 10-12 years old, this paper presents the design of a one-week project-based workshop for promoting the achievement of software development skills among K-6 learners using Scratch. The promoted practices were: (1) review existing and own source code, (2) read and understand functional requirements, (3) extend programs through the addition of new code, (4) adequate use of names (e.g., sprites and variables), (5) pattern-based design and implementation, (6) testing, and (7) read and write documentation. These practices were empirically assessed with a rubric we designed and calibrated to measure the level of mastery that the students achieve with regard to several dimensions related to the SE practices. The results of this process show that the stated practices can be effectively assimilated, accepted, and used by workshop participants. Furthermore, we did not find significant differences that can be attributed to both gender and socio-economic status in the acquisition of the proposed SE development practices.

Although this kind of non-traditional software engineering training experience is short and conducted in an informal fashion (i.e., as an extracurricular initiative instead of a formal course), our results provide promising perspectives on the feasibility and effectiveness

of delivering introductory software engineering concepts to this age group. This acts both as a complement to the existing broad of knowledge in promoting computational thinking and project-based learning endeavors. In consequence, we do not need to wait until students are enrolled in a formal computer science program to begin their involvement with good SE practices.

As ongoing work, we are studying if the order and depth with which the different topics are covered affects how students assimilate CT concepts, as well as SE practices. In a new version of the workshop (2018), we gave the students a more compact introduction to Scratch, so that they could see all the sample projects before starting their own projects. We also gave them more time to work on their projects. We are now analyzing the results of this experience. We also want to study the impact that these workshops have had over the long run.

ACKNOWLEDGMENTS

We would like to thank Fernanda Ramírez, Jorge Romo, and Giselle Font for helping create the workshop activities. We would also like to thank the workshop tutors and participants, as well as Escuela de Verano, for all their support.

REFERENCES

[1] Efthimia Aivaloglou and Felienne Hermans. 2016. How kids code and how we know: an exploratory study on the Scratch repository. In *Proceedings of the ACM Conference on International Computing Education Research (ICER'16)*. ACM, New York, NY, USA, 53–61. https://doi.org/10.1145/2960310.2960325

[2] Tim Bell, Ian H. Witten, and Mike Fellows. 2015. CS Unplugged: an enrichment and extension programme for primary-aged students. (2015). Retrieved October 18, 2017 from http://csunplugged.org/wp-content/uploads/2015/03/CSUnplugged_OS_2015_v3.1.pdf

[3] Bryce Boe, Charlotte Hill, Michelle Len, Greg Deschler, Phillip Conrad, and Diana Franklin. 2013. Hairball: lint-inspired static analysis of scratch projects. In *Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE'13)*. ACM, New York, NY, USA, 215–220. https://doi.org/10.1145/2445196.2445265

[4] Andreas Bollin, Stefan Pasterk, Peter Antonitsch, and Barbara Sabitzer. 2016. Software engineering in primary and secondary schools - informatics education is more than programming. In *Proceedings of the IEEE International Conference on Software Engineering Education and Training (CSEE&T'16)*. IEEE, New York, NY, USA, 132–136. https://doi.org/10.1109/CSEET.2016.26

[5] Andreas Bollin and Barbara Sabitzer. 2015. Teaching software engineering in schools on the right time to introduce software engineering concepts. In *Proceedings of the IEEE Global Engineering Education Conference (EDUCON'15)*. IEEE, New York, NY, USA, 518–525. https://doi.org/10.1109/EDUCON.2015.7096019

[6] Sapna Cheryan, John Oliver Siy, Marissa Vichayapai, Benjamin J. Drury, and Saenam Kim. 2011. Do female and male role models who embody STEM stereotypes hinder women's anticipated success in STEM? *Social Psychological and Personality Science* 2, 6 (2011), 656–664. https://doi.org/10.1177/1948550611405218 arXiv:https://doi.org/10.1177/1948550611405218

[7] K-12 Computer Science Framework Steering Committee. 2016. K-12 Computer Science Framework. (2016). Retrieved August 28, 2017 from http://www.k12cs.org

[8] Inés Friss de Kereki and Areti Manataki. 2016. "Code Yourself" and "A Programar": a bilingual MOOC for teaching computer science to teenagers. In *Proceedings of the IEEE Frontiers in Education Conference (FIE'16)*. IEEE, New York, NY, USA. https://doi.org/10.1109/FIE.2016.7757569

[9] Hakan Erdogmus and Cécile Péraire. 2017. International workshop on software engineering curricula for millennials (at ICSE 2017). (2017). Retrieved October 19, 2017 from http://secm2017.se-edu.org/wp/

[10] Bernhard Ertl, Silke Luttenberger, and Manuela Paechter. 2017. The Impact of Gender Stereotypes on the Self-Concept of Female Students in STEM Subjects with an Under-Representation of Females. *Frontiers in Psychology* 8 (2017), 703. https://doi.org/10.3389/fpsyg.2017.00703

[11] Louise P. Flannery, Brian Silverman, Elizabeth R. Kazakoff, Marina Umaschi Bers, Paula Bontá, and Mitchel Resnick. 2010. Designing ScratchJr: support for early childhood learning through computer programming. In *Proceedings of the International Conference on Interaction Design and Children (IDC'10)*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/2485760.2485785

[12] Shuchi Grover, Daisy Rutstein, and Eric Snow. 2016. "What is A computer": what do secondary school students think?. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*. ACM, New York, NY, USA, 564–569. https://doi.org/10.1145/2839509.2844579

[13] Francisco J. Gutierrez, Jocelyn Simmonds, Cecilia Casanova, Cecilia Sotomayor, and Nancy Hitschfeld. 2018. Coding or hacking? exploring inaccurate views on computing and computer scientists among K-6 learners in Chile. In *Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE'18)*. ACM, New York, NY, USA. https://doi.org/10.1145/3159450.3159598

[14] Felienne Hermans and Efthimia Aivaloglou. 2016. Do code smells hamper novice programming? A controlled experiment on Scratch programs. In *Proceedings of the IEEE International Conference on Program Comprehension (ICPC'16)*. IEEE, New York, NY, USA, 1–10. https://doi.org/10.1109/ICPC.2016.7503706

[15] Felienne Hermans and Efthimia Aivaloglou. 2017. Teaching software engineering principles to K-12 students: a MOOC on Scratch. In *Proceedings of the International Conference on Software Engineering: Software Engineering, Education, and Training Track (SEET'17)*. IEEE, Piscataway, NJ, USA, 13–22. https://doi.org/10.1109/ICSE-SEET.2017.13

[16] Yasmin B. Kafai. 2016. From computational thinking to computational participation in K-12 education. *Commun. ACM* 59, 8 (Aug. 2016), 26–27. https://doi.org/10.1145/2955114

[17] Linda Katehi, Greg Pearson, and Michael Feder. 2009. *Engineering in K-12 Education: Understanding the Status and Improving the Prospects.* National Academy of Engineering and National Research Council.

[18] Timothy C. Lethbridge, Jorge Diaz-Herrera, Richard J. Jr. LeBlanc, and J. Barrie Thompson. 2007. Improving software practice through education: challenges and future trends. In *Proceedings of the Conference on the Future of Software Engineering (FOSE'07)*. IEEE, New York, NY, USA, 12–28. https://doi.org/10.1109/CSEET.2016.26

[19] Stefano Maggiolo. 2015. An update on the female presence at the IOI. *Olympiads in Informatics* 9, 127 (2015), 127–137. https://doi.org/10.15388/ioi.2015.10

[20] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch programming language and environment. *ACM Transactions on Computing Education* 10, 4 (Nov. 2010), article 16. https://doi.org/10.1145/1868358.1868363

[21] Marcello Missiroli, Daniel Russo, and Paolo Ciancarini. 2016. Learning agile software development in high school: an investigation. In *Proceedings of the International Conference on Software Engineering: Software Engineering, Education, and Training Track (SEET'16)*. ACM, New York, NY, USA, 239–302. https://doi.org/10.1145/2889160.2889180

[22] Jesús Moreno-León, Marcos Román-González, Casper Harteveld, and Gregorio Robles. 2017. On the automatic assessment of computational thinking skills: a comparison with human experts. In *Extended Abstracts of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'17)*. ACM, New York, NY, USA, 2788–2795. https://doi.org/10.1145/3027063.3053216

[23] Rachel S. Phillips and Benjamin PC Brooks. 2017. The Hour of Code: Impact on Attitudes Towards and Self-Efficacy with Computer Science. (2017). Retrieved October 19, 2017 from https://code.org/files/HourOfCodeImpactStudy_Jan2017.pdf

[24] Mitchell Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (Nov. 2009), 60–67. https://doi.org/10.1145/1592761.1592779

[25] Pierre N. Robillard and Mihaela Dulipovici. 2008. Teaching agile versus disciplined processes. *International Journal of Engineering Education* 24, 4 (July 2008), 671–680.

[26] Gregorio Robles, Jesús Moreno-León, Efthimia Aivaloglou, and Felienne Hermans. 2017. Software clones in scratch projects: on the presence of copy-and-paste in computational thinking learning. In *Proceedings of the IEEE International Workshop on Software Clones (IWSC'17)*. IEEE, New York, NY, USA, 31–37. https://doi.org/10.1109/IWSC.2017.7880506

[27] Ian Sommerville. 2015. *Software Engineering* (10th ed.). Pearson Education.

[28] Jennifer Tsan, Kristy Elizabeth Boyer, and Collin F. Lynch. 2016. How early does the CS gender gap emerge?: a study of collaborative problem solving in 5th grade computer science. In *Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE'16)*. ACM, New York, NY, USA, 388–393. https://doi.org/10.1145/2839509.2844605

[29] Jeannette M. Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (March 2006), 33–35. https://doi.org/10.1145/1118178.1118215

[30] Robert K. Yin. 2013. *Case Study Research: Design and Methods* (5th ed.). SAGE Publications.