Informatics in Education, 2022, Vol. 21, No. 1, 33–54

DOI: 10.15388/infedu.2022.02

A Data-Driven Approach to Assess Computational Thinking Concepts Based on Learners' Artifacts

Adelmo ELOY*, Camila F. ACHUTTI, Cassia FERNANDEZ, Roseli de Deus LOPES

School of Engineering, Polytechnic School of the University of São Paulo
São Paulo, Brazil
e-mail: eloy.adelmo@gmail.com, achutti@ime.usp.br,
cassia.ofernandez@gmail.com, roseli.lopes@usp.br

Received: February 2021

Abstract. Integrating computational thinking into K-12 Education has been a widely explored topic in recent years. Particularly, effective assessment of computational thinking can support the understanding of how learners develop computational concepts and practices. Aiming to help advance research on this topic, we propose a data-driven approach to assess computational thinking concepts, based on the automatic analysis of data from learners' computational artifacts. As a proof of concept, the approach was applied to a Massive Open Online Course (MOOC) to investigate the course's effectiveness as well as to identify points for improvement. The data analyzed consists of over 3300 projects from the course participants, using the Scratch programming language. From that sample, we found patterns in how computational thinking manifests in projects, which can be used as evidence to guide opportunities for improving course design, as well as insights to support further research on the assessment of computational thinking.

Keywords: automatic assessment tools, computational thinking, computer science education, online learning.

1. Introduction

In 2013, former US President Barack Obama joined a campaign to encourage computer programming and stated:

> *"Do not just buy a new video game, make one. Do not just download the latest app, help design it. Do not just play on your phone, program*

*Corresponding author

> *it. No one's born a computer scientist, but with a little hard work, and some math and science, just about anyone can become one."* (Code.org, 2013).

Additional actions by the Obama administration, such as the Computer Science for All (The White House, 2016), exemplify the strength of the debate over the introduction of computer science into K-12 Education in the United States, particularly during the last decade. In the meantime, similar movements gained traction in other countries. For instance, Computing was included as a compulsory area of study in the national curriculum in England, through its four key stages (Department of Education, 2014; Brown *et al.*, 2014). At least 15 other European countries have already incorporated computer programming into their curriculum at different levels (Balanskat and Engelhardt, 2015). The new version of the Australian Curriculum includes Digital Technologies as one of its key learning areas, based on the use of computational thinking for the implementation of digital solutions (ACARA, 2014; Falkner *et al.*, 2014). In Brazil, the new National Common Curricular Base (BNCC, in the Portuguese acronym), highlights the importance of computational thinking for Mathematics Education (Brasil, 2018).

As with the traditional areas of STEM (acronym for Science, Technology, Engineering and Mathematics), one of the main motivations for introducing computing or programming in K-12 Education involves the growing demand for professionals with the skills to understand and produce digital technologies (Manyika *et al.*, 2017). However, the discussion on the theme is not recent and the motivations for children and youth to learn computer programming languages go far beyond the development of a technical skill: it is a strategy to foster metacognition and the creation of powerful ideas (Papert 1980; Tissenbaum, 2019), as well as other abilities, such as problem-solving to critical thinking (Popat and Starkey, 2019). More recently, computer programming has been framed as a key practice in computational thinking, based on Jeannette Wing's seminal paper (Wing, 2006), which provided a broader and more contextualized importance to the ability to create computer programs, especially for K-12 Education. Although there is no consensus about the definition of computational thinking, it has been a mandatory skill in national curricula in many countries across the globe and has fostered research in fields such as teacher professional development and assessment.

In this paper, we describe our findings in how to assess computational thinking using data collected from learners' computational artifacts. Here, it is contextualized in a Massive Open Online Course (MOOC) to serve as a qualified proof of concept because of the large volume of data the course provides. The whole analysis is based on data-driven approaches for assessment and decision-making. Before detailing our proposal, we present key research into relevant topics to this work: automatic strategies for assessing computational thinking in the context of the Scratch programming language, and how such strategies have been deployed to enhance learning experiences, particularly in online courses.

2. Background

2.1. *Computational Thinking and Scratch Programming Language*

In 2006, Wing proposed a wide description of computational thinking (CT), stating that it involved "solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (Wing, 2006), a skill that should not be limited to computer scientists. Since then, various researchers and organizations have proposed different ways to define and to operationalize CT; according to Barr & Stephenson, it "is an approach to solving problems in a way that can be implemented with a computer" (Barr and Stephenson, 2011), that includes the following concepts: data analysis, representation, and collection; problem decomposition; abstraction; algorithms & procedures; automation; parallelization; simulation. Likewise, Taslibeyaz *et al.* (2020) noticed the literature in CT suggests this skill is predominantly associated with analyzing and solving problems, combined with additional definitions such as systems design and computer programming. Similarly, Selby & Woolard (2013) state different definitions of CT at that time had a consensus on including three terms: the idea of a thought process and the concepts of abstraction and decomposition. Grover and Pea (2013), based on various definitions from researchers and organizations, propose additional elements that characterize CT, such as recursive thinking and debugging, besides highlighting the importance of computer programming for supporting the development of computational thinking. Using a different approach, Brennan and Resnick (2012) propose a definition of CT based on three dimensions: computational concepts, practices, and perspectives. Finally, Shute *et al.* (2017) present a categorization of CT in six facets: decomposition, abstraction, algorithm design, debugging, iteration, and generalization. In brief, although there is no single definition for CT, some of its features are widely accepted and guide different approaches to foster the development of this skill.

In addition to the ongoing research on how to frame CT, different approaches and tools have been used to foster its development. Among the existing programming languages and platforms to nurture computational concepts and practices, Scratch (Resnick *et al.*, 2009) is one of the most popular for that purpose, particularly in K-12 settings (Moreno-León *et al.*, 2017). In brief, Scratch is a visual or block-based programming language and online environment, publicly launched in 2007. It is a free service developed by MIT and has over 66 million users registered (Scratch, 2021), with a vast research literature on its use for developing programming and computational thinking skills. For instance, Papavlasopoulou *et al.* (2019) explored Scratch as a tool for constructionism-based experiences in coding, with participants aged 8–17 years old. Weng *et al.* (2010) used Scratch as a learning environment to introduce Boolean logic to ninth grade students. Topalli and Cagiltay (2018) used Scratch for introducing programming concepts for first-year Engineering students. Cardenas-Lobo *et al.* (2019) developed a Scratch extension to increase the effectiveness of Scratch use in Higher Education. Yadav *et al.* (2017) and Yurkofsky *et al.* (2019) presented relevant experiences into using Scratch to introduce computational thinking and programming for preservice teachers. Additionally, Scratch has been an effective tool to develop computational thinking

in online environments, via online courses for K-12 students (Basogain *et al.*, 2018) and teachers (Marcelino *et al.*, 2018). Although there are alternative programming languages and environments for introducing computational thinking to K-12 Education, such as Alice, Code.org and even unplugged methodologies, the ease of use and diversity of applications with Scratch supports its relevance in the field.

2.2. *Automatic Assessment of Computational Thinking*

Assessing computational thinking concepts and practices is still a field with more questions than answers (e.g., see Balanskat and Engelhardt, 2015; Basso *et al.*, 2018; Grover and Pea, 2013; Haseski and Ilic, 2019). As possible strategies for assessment, tasks and challenges facilitate the comparison among large numbers of learners from different contexts (e.g., see de Araujo *et al.*, 2013; Berland *et al.*, 2014; Grover *et al.*, 2014; Izu *et al.*, 2015; Tsukamoto *et al.*, 2017). In turn, there are various studies which explore computational artifacts created by learners to evidence and to compare their development (Fields *et al.*, 2016; Seiter and Foreman, 2013). Particularly, computational artifacts represent a valuable resource for personalized and formative assessment, which can be enhanced by the design and implementation of automated analyses methods.

Additionally, automated analyses of computational artifacts can make the assessment of large amounts of productions possible, using comparable parameters at scale. Recent studies using different programming languages provide evidence that learning analytics techniques can be used to promote a better understanding of how learners evolve as they develop programming and CT skills (Alves *et al.*, 2019; Blikstein *et al.*, 2014; Dasgupta and Hill, 2017; Von Wangenheim *et al.*, 2018). In particular, different tools have been developed to analyze, assess, and give feedback to students working on the Scratch programming environment. Filvà *et al.* (2019) developed an approach using clickstream techniques to detect patterns in students' behavior in Scratch programming activities aiming to support teachers in evaluation and tutoring. Martin *et al.* (2016) and Brasiel *et al.* (2017) developed an automated analysis tool to help researchers studying the development of computational thinking using Scratch and used it to measure the computational thinking components of parallelism, logical thinking, synchronization, iterative and recursive thinking, and pattern generalization.

Among the existing approaches to assess computational thinking with Scratch, the web application Dr Scratch (Moreno-León *et al.*, 2015) provides a quantitative analysis and categorization based on seven CT concepts (abstraction and problem decomposition, parallelism, logical thinking, synchronization, algorithmic notions of flow control, user interactivity and data representation), grading projects from 0 to 21. For instance, the logical thinking concept is assessed as basic (1 point) if the project has an "if" block, as developing (2 points) if it has an "if-else" block and proficiency (3 points) if it has logic operators. Although it represents a remarkable approach to automatically assess projects, the tool makes narrow assumptions to categorize the level of a computational thinking concept or practice in a Scratch project, which can cause misleading feedback.

Building upon previous research, we propose a data-driven approach to assess computational thinking, based on automatic analysis of data from learners' computational artifacts. We aim to have a greater level of understanding on how data extracted from computational artifacts can support instructors and learners in developing different dimensions of computational thinking. Section 3 describes that approach, from its premises to coefficients that describe it, as well as aspects of implementation with the Scratch programming language. Section 4 details a proof of concept for this approach to investigate its feasibility, consisting of four editions from a massive online open course (MOOC) on introductory computer programming. Section 5 highlights the conclusions and recommendations for future research.

3. An Approach for Assessing Computational Thinking

This section describes the approach proposed to automatically assess computational thinking, especially the definition of CT used as a reference and the hypotheses to quantify computational concepts and convert them into numerical coefficients. In addition to that, it describes the algorithm used to extract coefficients from a Scratch project, including source code and application programming interfaces used, allowing replication by further research in the same sample of projects or in different samples.

3.1. *Premises for the Assessment Approach*

Building upon existing work on the automatic assessment of computational thinking, the approach designed herein is based on the following premises:

- The approach aims to assess the development of computational thinking concepts (Table 1). In the context of the Scratch programming language, the definition proposed by Brennan and Resnick (2012) was used as a reference, as it clearly defines how specific programming blocks connect with one of the seven computational concepts proposed by the authors. That also means that this approach does not target assessing computational practices and perspectives, which can be explored in future work.
- The approach explores the notion of artifact-based assessment, through the analysis of learners' computational artifacts created with Scratch. By that, it values the constructionist nature in programming with Scratch (Papert, 1980; Resnick *et al.*, 2009). Additionally, it assumes that learners master a concept as they apply it to an artifact, growing and cumulative repertoire that can be measured (Dasgupta and Hill, 2017).
- The approach aims to provide quantified measures for each computational concept, as Moreno-León *et al.* (2015), exploring data collection and manipulation techniques. As an alternative approac, it proposes coefficients that are more sensitive to code variation in Scratch projects. Additionally, it focuses on analyzing large volumes of data, applicable to both large groups, such as the summative assessment

of a final project in a massive course, and periodic collections from smaller groups, such as the formative assessment of projects as students work on it in a classroom.

3.2. *Coefficients for Computational Thinking Concepts*

This approach proposes numerical coefficients to measure the application of computational concepts, based on programming blocks and their connections in a Scratch project. For that, we build upon the detailed description for computational thinking concepts from Brennan and Resnick (2012), summarized in Table 1.

We identified which Scratch programming blocks and structures manifest the application of computational concepts, and defined ways to quantify them, considering not only the number of blocks and structures used, but also their variety. For instance, a project that uses the same loop block twice would have a different coefficient for Loops than a project which uses two different loop blocks, once each. Based on that notion, we defined seven coefficients, from C1 to C7, as follows:

- **C1. Sequences:** number of functional scripts in a project, that is, at least two blocks connected conditioned to an event.
- **C2. Events:** number of events blocks that start a script, known as "hat blocks", multiplied by how many kinds of events are applied to the project.
- **C3. Parallelism:** defined by how many times the same event is used to start at least two different scripts.
- **C4. Loops:** defined by the number of loop blocks ("repeat ()", "forever" and "repeat until ( )") in a project, multiplied by how many kinds of loop blocks are applied to the project.
- **C5. Conditionals:** defined by the number of conditional blocks ("if <> then", "if <> then, else", "repeat until <> and "wait until <>" ;) in a project, multiplied by how many kinds of conditional blocks are applied to the project.
- **C6. Operators:** defined by the number of operator blocks (all the blocks available in the Operator category in Scratch) in a project, multiplied by how many kinds of operator blocks are applied to the project.
- **C7. Data:** defined by the number of data blocks (all the blocks available in the Variables category in Scratch) in a project, multiplied by how many kinds of data blocks are applied to the project.

Table 1

Computational concepts proposed by Brennan and Resnick (2012)

Sequences	Identifying a series of steps for a task
Events	One thing causing another thing to happen
Parallelism	Making things happen at the same time
Loops	Running the same sequence multiple times
Conditionals	Making decisions based on conditions
Operators	Support for mathematical and logical expressions
Data	Storing, retrieving, and updating values

To make the coefficients more tangible, Table 2 presents the coefficients for a specific Scratch project (Fig. 1). In sum, we related the coefficients' value to the number of functional blocks on the project, using the description of computational concepts proposed by Brennan & Resnick (2012) to identify which blocks on Scratch would relate to each coefficient. Added to that, we multiplied the number of blocks by the number of different blocks used in C4, C5, C6 and C7, to value the diversity of blocks applied to the project.

The coefficients in our approach do not have a maximum value: as blocks are added to a project, their coefficients increase. Based on that, there is no single definition of what a low or high indicator for a computational concept would be; instead, such definitions are contextual and then customizable by instructors or learners. Simple numeric comparisons between coefficients are not relevant in this scenario, as each explores different metrics.

Table 2

Coefficients for Scratch project in Fig. 1

Coefficients	Value	Rationale
C1	4	4 five different scripts (sets of blocks)
C2	4	4 events * 1 type of event ("when 'green flag' clicked")
C3	1	1 type of block used in parallel ("when 'green flag' clicked")
C4	8	4 loop blocks * 2 types of loop blocks ("forever" and "repeat ( )")
C5	6	6 conditional blocks * 1 type ("if <> then")
C6	24	8 operator blocks * 3 types ("() < ( )", "( ) > ( )", "<> and <>")
C7	45	15 data blocks * 3 types ("variable", "set [] to ()" and "change [] by ( )")

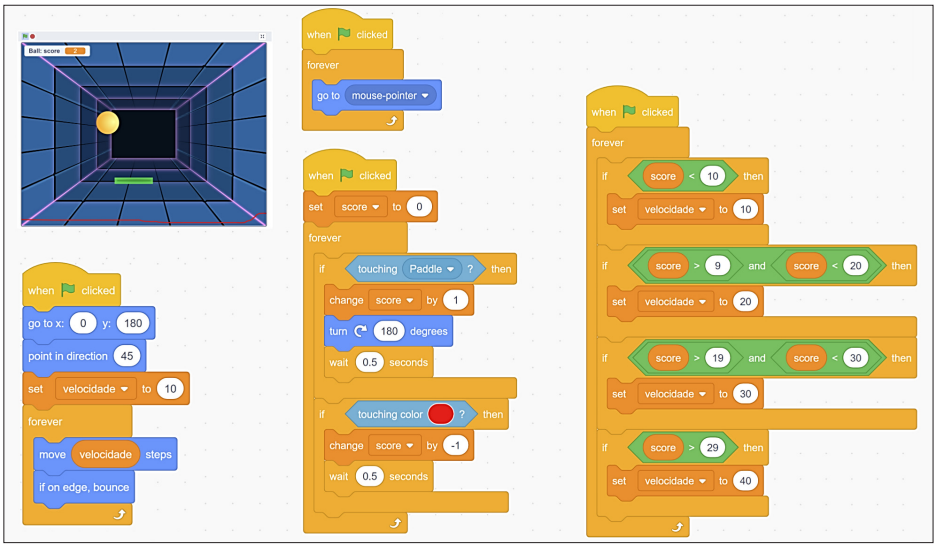

Fig. 1. Example of a Scratch project interface (Pong game) and its corresponding programming blocks.

3.3. *From Data Collection to Visualization*

Data from Scratch projects is collected, organized, and visualized by specific routines developed with Python programming language. Application programming interfaces (APIs) for data extraction and libraries for data visualization were key factors for defining the programming language. Fig. 2 illustrates the key steps in that process, which are detailed in the following paragraphs.

Data is collected from individual Scratch projects, identifiable by a uniform resource locator ("url"). This project can be exported into a JSON file (acronym for JavaScript Object Notation), with an API available in Python (step 1 in Fig. 2). The JSON file for a Scratch project details, among other information, the blocks that compose it as well as how they connect with each other. Fig. 3 exemplifies how a set of Scratch blocks is described in a JSON file.

Information from the JSON file is used to determine the coefficients for computational thinking coefficients, described in the previous section (step 2 in Fig. 2). From that, it is possible to generate visual representations for a project (step 3 in Fig. 2) and use them for comparison, such as how the project evolves over time or contrasting it with other projects. Moreover, information from a specific project can be stored in a database

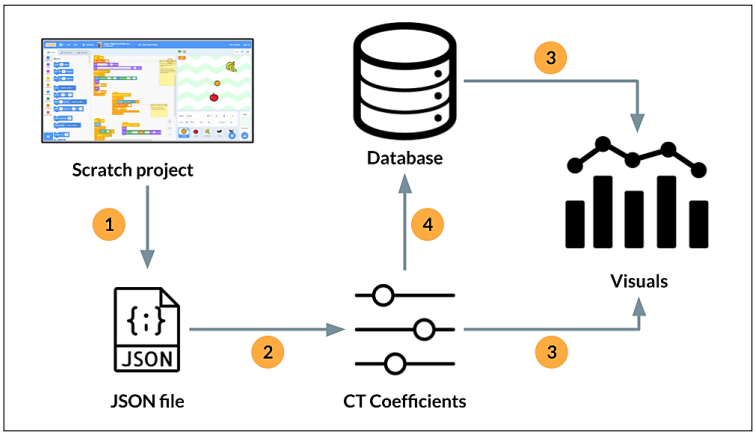

Fig. 2. Key steps for collecting, analyzing, and visualizing data from a Scratch project.

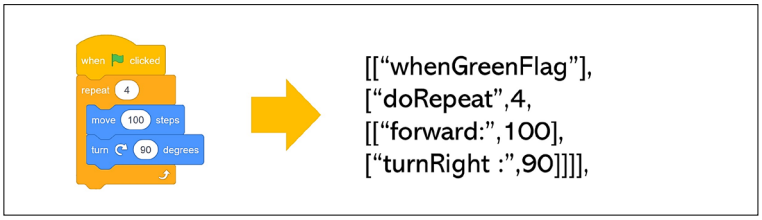

Fig. 3. Example of how information from a Scratch script is represented in a JSON file.

for analyzing a sample of projects (step 4 in Fig. 2); that includes, for example, multiple collections of one project over time. An alternative strategy includes comparing different projects from a given context, such as the final project of a course. A similar scenario is used as a proof of concept for this approach, which is described in the next section.

4. Proof of Concept with a Massive Online Open Course

This section describes learning outcomes from applying the approach to automatically assess computational thinking in a Massive Online Open Course (MOOC), as a proof of concept for our proposal. First, we justify why we chose a MOOC as an environment for a proof of concept, as well as general information about the course chosen. Second, we describe adaptations made to make the approach applicable to such an environment. Finally, we share insights from this experience, contextualized for teaching and learning in one specific course, but which can contribute to research in similar contexts.

4.1. *Choosing a MOOC for Applying the Approach*

There are different reasons for choosing a MOOC to run a proof of concept for the proposed tool. In our case, a feature of the programming MOOCs is the presence of several different solutions for the proposed projects; in most of these courses, project submission and tasks involve lines of code. Moreover, the analysis of those projects can explore different approaches, such as peer-to-peer analysis, in which students provide peer feedback on the project. However, assessment based on individual code submissions can be an arduous process, either because the work performed grows linearly at scale or becomes almost unworkable in open and online courses, such as MOOCs. Different scholars have explored the use of automated methods to analyze and assess learners' artifacts at scale, including in the field of computer science education (e.g., see Head *et al.*, 2017; Nguyen *et al.*, 2014; Wang *et al.*, 2018), which fits with the requirements of massive courses. Those approaches tend to develop an underlying objective simulation and similarity between students' responses in the educational process of computer science (Hovemeyer *et al.*, 2016) and not computational thinking competences. This is the contribution of this proof of concept with a MOOC.

In this proof of concept, we analyzed the productions from participants in the online course "*CodeIot* – Learning to code", offered by Code IoT platform, which also includes other courses about basic electronics, robotics, Android apps development, IoT concepts and IoT solutions development. Also, it is free of any charges and available in Portuguese, English, and Spanish. The "*CodeIot* – Learning to code" is an introductory course, which aims to enable participants to take the first steps into coding, by exploring the Scratch programming language. The course had its first edition in 2017 and three others in 2018, with more than 1000 participants in total and 3331 artifacts created and submitted by them. Other editions have been offered in 2019 and 2020 but were not considered in this study.

With activities distributed along six weeks, the course invites participants to create various projects with Scratch, supported by videos, tutorials, and exercises, in addition to peer reviews for the projects built. It is organized into weekly activities, with a total workload of 20 hours. Table 3 presents the course outline from its most recent edition (since there were adjustments after the first one): we focus our analysis into the exercises (five in total), as the projects created on them are used to assess participants' computational thinking skills.

The percentage of participants that submitted activities decreased significantly along the course (Table 4), as other learning experiences with MOOCs, which is below 15% (Onah *et al.*, 2014; Rothkrantz, 2016). The most significant reduction occurred between Exercises 4 and 5, which indicates the need for greater attention to this stage of the course.

Table 3

CodeIoT – Course outline

Week	Course description
1	Course presentation
2	Theme: animations and stories Scratch programming language: user interface, online community, microworlds Computational concepts: sequences, loops, and parallelism Scratch feature: Messages Exercise 1: personal presentation with an animation on Scratch
3	Theme: interactive projects with Scratch Computational concepts: events and conditionals Scratch feature: pen Exercise 2: debug and add interaction to a project in Scratch
4	Theme: Creating games with Scratch Computational concepts: variables, logical and mathematical operators Scratch feature: random numbers Exercise 3: design a game with Scratch
5	Theme: adding features to games with Scratch Computational concepts: functions and procedures Scratch features: clones, sensors, functions, stages in a game Exercise 4: keep working on games with Scratch
6	Exercise 5: Creating a free project Course evaluation

Table 4

Number of exercises submitted by participants per edition

Course edition	Exercise 1	Exercise 2	Exercise 3	Exercise 4	Exercise 5
1st	663	481	398	279	133
2nd	284	227	185	142	78
3rd	62	47	39	32	16
4th	96	64	51	40	14
Total	1105	819 (74%)	673 (61%)	493 (45%)	241 (22%)

4.2. *Adapting the Approach for Assessment to a MOOC Course*

To apply the approach to assess computational thinking to the *CodeIot* Online course, we made a few adaptations in the algorithm for data collection and analysis, which are mainly described through the following steps:

1. Organizing the *CodeIot* platform database from the last four editions of the course.
2. Extracting and analyzing data from Scratch projects on the online Scratch platform.
3. Parameterizing the coefficients based on maximum and minimum values from the sample of projects used on the analysis.

In step (1), the data from the exercises along the course was extracted and organized in a .csv file, available on an online repository[1], in which each row refers to an exercise submission with an associated URL to a Scratch project. In addition to the project URL, other relevant information extracted from the database was: a user code identifier, the date/time of submission, a grade from 0 to 10 and qualitative feedback, both based on peer evaluation.

In step (2), we extracted data from Scratch projects to calculate their respective coefficients, as described in Table 2 and detailed in Section 3.B. From that, we were able to calculate the coefficients of computational thinking concepts, C1 to C7, for each project submitted to the course and those values to the database of projects. Fig. 4 describes the algorithm for extracting and analyzing the data from our database of Scratch projects

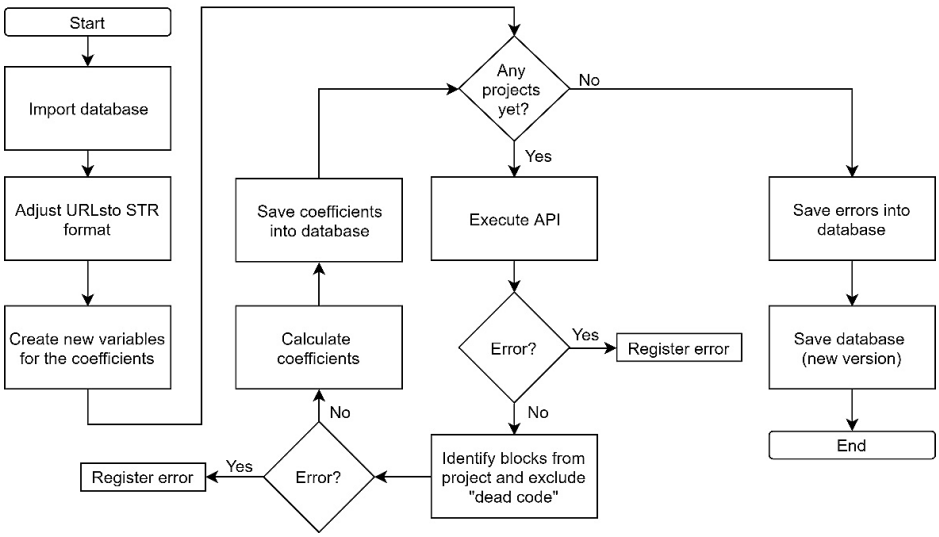

Fig. 4. Algorithm for extracting and analyzing data from a sample of Scratch projects, resulting in their coefficients for computational concepts.

[1] Online repository available at: `http://bit.ly/3aq9aS0`

Table 5
Maximum absolute value and maximum sum of median values for the coefficient

Coefficients		Maximum absolute value[1]	Maximum sum of median values[2]	Parameters
C1	Sequences	1010	78	1.28
C2	Events	5050	288.5	0.35
C3	Parallelism	7	11	44083
C4	Loops	852	52.5	1.90
C5	Conditionals	1408	65.5	1.53
C6	Operators	12690	196	0.51
C7	Data	10150	134	0.75

[1] Maximum absolute value (for a project, considering all editions).
[2] Maximum sum of median values, from Exercises 1 to 5 (considering all the editions).

submitted along the course editions. The algorithm was implemented using the Python programming language, which is detailed in another work by the same author (Eloy, 2019) and available on an online repository (`http://bit.ly/3aq9aS0`).

For each edition of the course, we applied steps (1) and (2). In step (3), the coefficients from different courses were parameterized to improve their comparison, especially with radar graphs. That is, maximum values registered in all the editions were considered as 100% for each axis. Table 5 presents the values for the seven coefficients: instead of using the maximum absolute value for parameterization, this study considered the maximum sum of median values, from Exercises 1 to 5, in all the editions. Also, to represent a learners' progression and cumulative domain of each computational concept, we opted for representing the sum of coefficients of different artifacts created over time.

When describing an average project in specific circumstances, such an exercise from an edition – which we called typical projects, we gave priority to the median over average values, as the sample has a high standard deviation (relative standard deviation higher than 50% for all the coefficients, considering different editions). Besides, by using median values, we could find projects in our database to exemplify a specific combination of coefficients. Based on that, the coefficients from all the editions were multiplied by their corresponding parameter, presented in Table 5.

As a limitation, the parametrization used mean values from the sample of projects and so the graphs could not represent individual projects with coefficients higher than "100" in the same axes; in those cases, we had to adjust the axes to represent them graphically. New editions of the course may have samples with median values higher than those registered so far, which will demand a new parametrization of the coefficients.

5. Results and Discussion

We defined the questions below to investigate the outcomes from applying the approach to automatically assess computational thinking to the MOOC "Learning to Code" from the *CodeIot* platform.

- Can the approach identify similarities and differences in students' creations in different course editions?
- Can we use the approach to characterize what is expected from a project for a specific exercise?
- Can we use the approach to identify outliers that contrast typical projects?
- Can the assessment of computational concepts help predict students' dropout?

5.1. *Can the Approach Identify Similarities and Differences in Students' Creations in Different Course Editions?*

Fig. 6 helps us reflect on that question by showing the typical behavior (median values from the sample) of projects from each exercise in the different editions of the course. It represents a cumulative repertoire: the green curve, for example, represents the total sum of coefficients from Exercises 1 to 3. The parametrization presented in Section 4.2 took the four editions of the course into account; as observed, the maximum cumulative repertoire for all the coefficients was reached in the fourth edition.

Fig. 5 also illustrates that all the editions have similar cumulative graphs for the first three exercises but are different for the last two exercises. For instance, the graphs for Exercise 1 are similar, although the scale of the graph makes it harder to compare. The similarity remains in Exercises 2 (which as expected, given that its proposal is based on remixing a project) and Exercise 3. There is a significant difference in Exercise 4, with

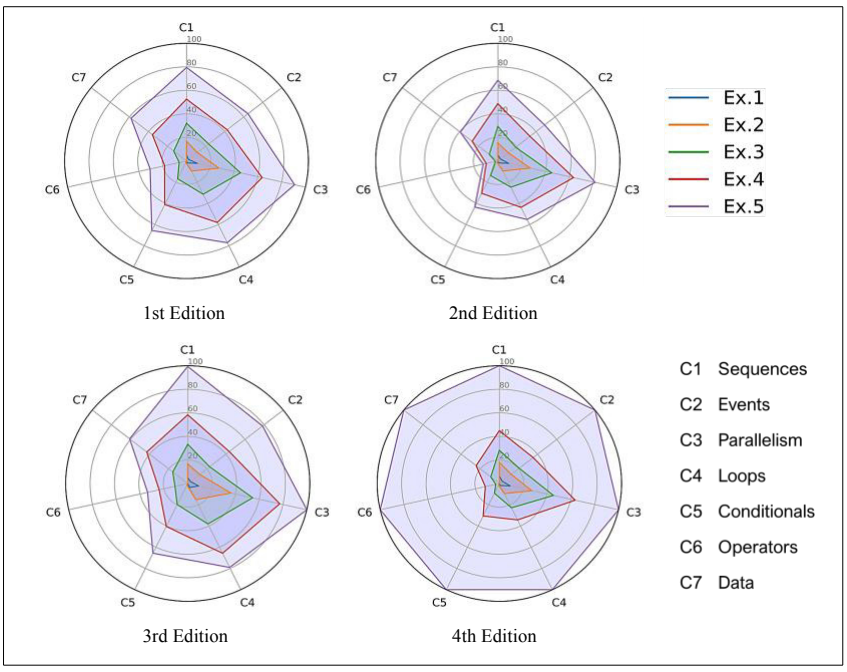

Fig. 5. Progression of coefficients throughout exercises (cumulative), for all the editions.

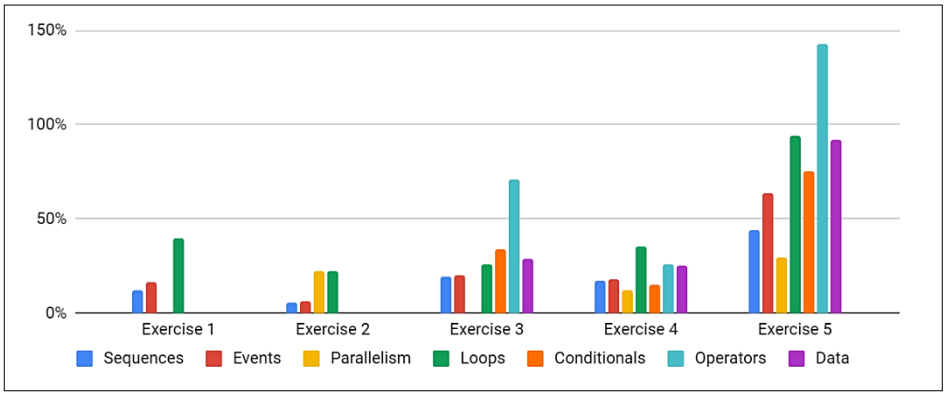

Fig. 6. Coefficients of variation for each coefficient in different exercises.

greater similarity between the first and the third editions, and between the second and the fourth editions. The largest variation occurs in Exercise 5, a free project, which indicates that deeper analyses among the different editions must be performed to understand the results observed.

From another perspective, Fig. 6 presents the coefficients of variation (CVs) for each coefficient, per exercise, considering the four editions of the course; CVs are defined by the ratio of the standard deviation to the mean value for each coefficient; they are used in this study to express the variability in relation to the mean of a population. There is no coefficient of variation for C5, C6 and C7 in Exercise 1 and C6 and C7 in Exercise 2, as the mean in those cases is equal to zero. The other null values in Fig. 6 correspond to CV equal to zero. From Fig. 6, the CVs from exercises 1 to 4 are low, having only one coefficient, C4, with CV over 30%. In Exercise 5, however, the CVs are considerably higher, most of them over 50%. That was a expected behavior, as the exercise prompted learners to develop free projects.

5.2. *Can we Use the Approach to Characterize what is Expected from a Project for a Specific Exercise?*

We defined typical projects as the ones identical or very similar to what was expected from learners to create for each project; in other words, they would have coefficient values equal to the mean values in the sample of submissions. By using a nearest neighbor search algorithm in k-d tree, available in Python, we could identify projects that were the closest to the typical value for a given exercise, in terms of their computational concepts.

To illustrate that approach, Fig. 7 represents project “Typ. 1”, whose coefficients are the closest to a typical project in Exercise 4 for the first edition of the course. This project includes all the aspects expected from it: besides being a game (as proposed by Exercise 4), the project explores resources and concepts presented in the week of the course, such as variables for recording the player’s score and random numbers for positioning the

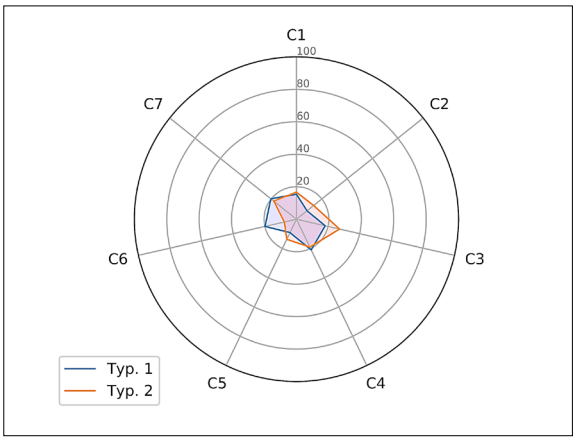

Fig. 7. Comparison between typical projects from the first (Typ. 1) and the fourth edition (Typ. 2), in Exercise 4.

fruits on the screen. This example illustrates how the approach for assessment could help identify typical projects for a given exercise.

When using the same algorithm with the fourth edition of the course (most recent), we located project "Typ. 2", also shown in Fig. 7. Although the projects have distinct features, from the theme they explore to their interface, "Typ. 1" and "Typ. 2" can be compared by using the coefficients for computational concepts, as shown in Fig. 7: based on them, the projects are very similar, except for C3 and C6. Although more systematic analysis is required to investigate if this is a common behavior for the whole sample of projects, it illustrates how computational coefficients could help identifying typical projects, without limiting the diversity of interests and goals that are characteristic of projects built with Scratch.

5.3. *Can we Use the Approach to Identify Outliers that Contrast Typical Projects?*

We used the same algorithm described in 5.2 to identify projects whose coefficients were very different from what would be considered typical for that exercise, particularly higher – what we called outliers. To illustrate cases as these, the project "Typical", in Fig. 8, is an example of what would be typical for Exercise 1 for the first edition of the course. That project contrasts with "Outlier", whose coefficients are the highest submitted for Exercise 1 for the first edition of the course. Although the project meets the requirements for Exercise 1 (creating a personal presentation with Scratch), "Outlier" exceeds the expectation towards the fist exercise of the course.

Outliers as those shown in Fig. 8 help illustrate their potential in the course. By identifying participants who excel in their creations throughout the exercises, it is possible, for example, to engage them in different roles, such as tutors, or to propose more appropriate challenges to the skills demonstrated. Conversely, outliers with low value for the coefficients can be used to identify participants that deserve a higher level of support.

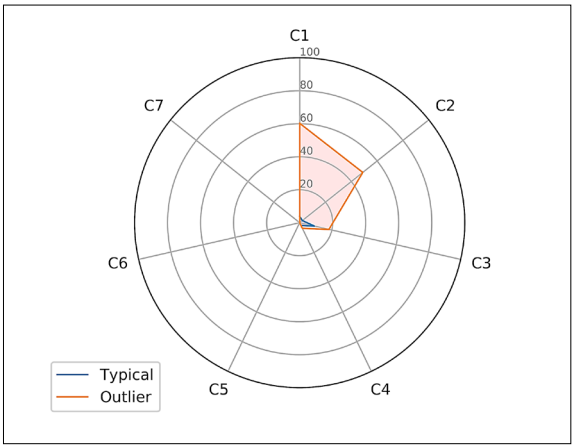

Fig. 8. Comparison between the typical project and the outlier for Exercise 1.

5.4. *Can the Assessment of Computational Concepts Help Predict Students' Dropout?*

Given that participants' dropout is one of the greatest challenges for massive online courses, we investigated if the computational coefficients used along the first projects created by the course's participants could be used to describe their inclination to drop out. For this, the following scenarios were explored: (1) project profile of participants that submitted Exercise 1 and then dropped out; and (2) project profile of participants who submitted Exercises 1 and 2 and then dropped out. The first scenario was chosen to analyze dropout as soon as it significantly occurs in the course, as shown in Fig. 1. The second scenario analyzed whether characteristics of Exercise 1 could help to identify them, among those who dropped out after Exercise 2.

For analyzing the scenarios, Fig. 9 compares the coefficients for Exercise 1 in four cases: scenarios 1 and 2 for dropout (called "Drop 1" and "Drop 2", respectively), the typical project for that exercise ("Typical"), and the typical project of participants who completed the course ("Completed"). The first edition was chosen for that comparison, as it has the largest sample of projects.

From Fig. 9, the curve for "Drop 1" has a value in C1 slightly lower than the others; also, the curve for typical projects is the same as "Drop 2"; it cannot thus be seen in the figure. In turn, the values for C4 in "Complete" (those who concluded the project) are significantly higher; the same characteristic was observed in the third and fourth editions (especially the latter), but not in the second one. We do not have a strong hypothesis for this behavior; although the use of loops (C4) is not required for developing simple animations as required in Exercise 1, it is presented in some of examples shared with participants. In any case, this is a topic worth investigating in future work.

Additionally, there are no significant differences between the two profiles of participants who drop out (orange and green). As there is not a clear pattern in those charac-

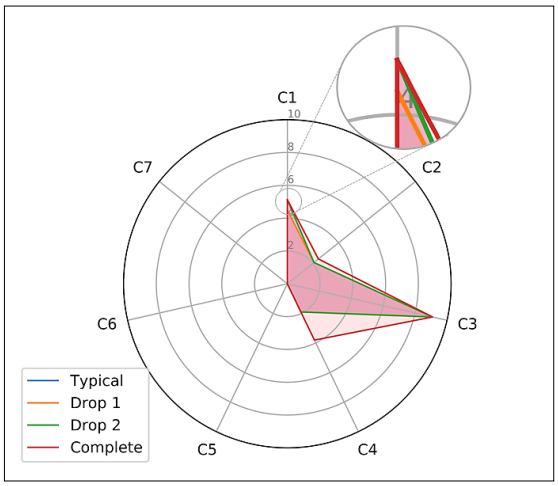

Fig. 9. Comparison of projects among participants that have dropped out and that have completed it.

teristics, data from more editions of the course could contribute to strengthen the idea of using C4 as a predictor for dropping out. If so, strategies for enhancing its use in Exercise 1 could be tested to keep learners engaged.

5.5. *Overall Findings*

Our results provided evidence that our approach has the potential to represent the diversity of creations and progression of participants and their projects, as well as to verify if the set of exercises guide learners in exploring all computational concepts.

From the analysis of each exercise proposed in the course, we could identify typical projects and outliers (both those that go beyond the expected and those that have greater dropout risks). In addition, a first approach to predict participants' dropout was possible, by using computational concepts for their description. As is common in massively open online courses, the course completion rate is low (about 13%) and better analyses of learning outcomes applying that approach can be a promising tool to plan and to implement actions to prevent dropouts. As more editions are offered, the projects characterization may become more specific.

Finally, we could describe learners' progression using a new approach to characterize computational thinking manifested in the Scratch project, by using larger samples of data than other strategies available. In that way, this approach provides a set of coefficients capable of identifying subtle differences in samples of computational artifacts. On the other hand, any of the propositions made on this paper are hints on how to describe learners' behaviors o the MOOC course, from typical projects to outliers to students' dropout. At the same time further research is required to strength and validate such assumptions, the use of automatic methods only will probably not be enough to answer, which gives room to their integration with qualitative methods of assessment.

6. Conclusion and Future Work

With the aim of investigating novel ways to assess computational thinking, this study proposed a data-driven approach to assess learners' manifestation of computational concepts, based on the automatic analysis of data from computational artifacts created programming languages, as they explore and master computational ideas. As an initial proof of concept, this approach was applied to a Massive Open Online Course (MOOC) entitled "*CodeIot* – Learning to Code". Given students on that course were required to design and submit various computational artifacts developed with the Scratch programming language, it provided a suitable context for the assessment approach described in this paper. At the same time, analyzing learners' productions from a CT perspective could provide a way to systematically investigate computing ideas they developed as they created projects with Scratch. From that application, we could not only have a better understanding of how learners progress as they create computational artifacts with Scratch but gather relevant evidence to support the improvement in instruction for further editions of the course.

As limitations for this work, the approach is based on the definition of computational thinking proposed by Brennan & Resnick (2012) and adaptations should be made to apply it to other programming languages. Besides, it focuses on computational concepts, demanding complementary strategies for assessing computational practices and perspectives, as well as other aspects present in alternative definitions of CT, such as decomposition and abstraction. Regardless, we build on the definition of a system of assessments described in Basso *et al.* (2018) and reinforce the notion of using various techniques and tools combined to provide a more comprehensive assessment of computational thinking than any specific method. Additional limitations include the proof of concept itself, based on a specific online course with a low number of editions, and the need to parametrize the coefficients as new editions are available. Addressing those limitations will be important to generalize the results of this work and present possible paths for future research.

Finally, we believe this approach has the potential for further research on the assessment of computational thinking. Particularly, ideas that are worth exploring include:

- Integrating the approach into the online courses such as "*CodeIot* – Learning to Code", as a tool to help learners visualize and reflect on their progress, as well as a resource for mentors to be more effective in their instruction and support.
- Applying the approach to different learning environments, particularly to classroom scenarios, which can enable more in-depth investigation, such as monitoring the progress of individual artifacts as they are developed, and integration with complementary assessment techniques to analyze computational practices and perspectives.
- Adapting this approach with other programming languages and platforms to broaden its impact, which includes reviewing the concepts and how to identify them in different coding structures and adjusting the algorithms for data collection and analysis.

7. Acknowledgements

We would like to thank SAMSUNG and LSITEC for the access to the *CodeIot* – Learning to Code database used in this study. We would also like to thank Irene Karaguilla and Leandro Biazon for their technical support.

8. References

Alves, N.D.C., Von Wangenheim, C.G., Hauck, J.C. (2019). Approaches to assess computational thinking competences based on code analysis in K-12 education: A systematic mapping study. *Informatics in Education*, 18(1), 17. https://doi.org/10.15388/infedu.2019.02

Eloy, A. A. D. S. (2019). *Contribuições para aplicação de learning analytics no apoio à avaliação em atividades de introdução à programação com Scratch* (Doctoral dissertation, Universidade de São Paulo).

Australian Curriculum, Assessment and Reporting Authority [ACARA]. (2014). Foundation to year 10 curriculum: Language for interaction (ACELA1428). Retrieved February 15, 2021, from http://www.australiancurriculum.edu.au/english/curriculum/f-10?layout=1#cdcode=ACELA1428&level=F

Balanskat, A., Engelhardt, K. (2015). *Computing our Future: Computer Programming and Coding-Priorities, School Curricula and Initiatives across Europe*. European Schoolnet.

Barr, V., Stephenson, C. (2011). Bringing computational thinking to K-12: What is Involved and what is the role of the computer science education community? *Acm Inroads*, 2(1), 48–54. https://doi.org/10.1145/1929887.1929905

Basogain, X., Olabe, M. Á., Olabe, J. C., Rico, M. J. (2018). Computational Thinking in pre-university Blended Learning classrooms. *Computers in Human Behavior*, 80, 412–419. https://doi.org/10.1016/j.chb.2017.04.058

Basso, D., Fronza, I., Colombi, A., Pahl, C. (2018, November). Improving assessment of computational thinking through a comprehensive framework. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research* (pp. 1–5). https://doi.org/10.1145/3279720.3279735

Berland, M., Baker, R. S., Blikstein, P. (2014). Educational data mining and learning analytics: Applications to constructionist research. *Technology, Knowledge and Learning*, 19(1–2), 205–220. https://doi.org/10.1007/s10758-014-9223-7

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., Koller, D. (2014). Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, 23(4), 561–599. https://doi.org/10.1080/10508406.2014.954750

Brasiel, S., Close, K., Jeong, S., Lawanto, K., Janisiewicz, P., Martin, T. (2017). Measuring computational thinking development with the FUN! Tool. In *Emerging research, practice, and policy on computational thinking* (pp. 327–347). Springer, Cham. https://doi.org/10.1007/978-3-319-52691-1_20

Brasil (2018). Ministério da Educação. *Base Nacional Comum Curricular – Educação é a Base*. Brasília, DF.

Brennan, K., Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada* (Vol. 1, p. 25).

Brown, N. C., Sentance, S., Crick, T., Humphreys, S. (2014). Restart: The resurgence of computer science in UK schools. *ACM Transactions on Computing Education (TOCE)*, 14(2), 1–22. https://doi.org/10.1145/2602484

Cárdenas-Cobo, J., Puris, A., Novoa-Hernández, P., Galindo, J. A., Benavides, D. (2019). Recommender Systems and Scratch: An integrated approach for enhancing computer programming learning. *IEEE Transactions on Learning Technologies*, 13(2), 387–403. https://doi.org/10.1109/TLT.2019.2901457

Code.org. (2013). *President Obama asks America to learn computer science*. Retrieved February 15, 2021, from https://youtu.be/6XvmhE1J9PY

Dasgupta, S., Hill, B. M. (2017, April). Learning to code in localized programming languages. In *Proceedings of the fourth (2017) ACM conference on learning@ scale* (pp. 33–39). https://doi.org/10.1145/3051457.3051464

de Araújo, A. L. S. O., Scaico, P. D., de Paiva, L. F., de Morais Rabêlo, H., de Luna Santos, L., Pessoa, F. I. R., ... & dos Santos Costa, L. (2013). Aplicação da Taxonomia de Bloom no ensino de programação com Scratch. In *Anais do Workshop de Informática na Escola* (Vol. 1, No. 1, p. 31). https://doi.org/10.5753/CBIE.WIE.2013.31

Department for Education (2014). *The National Curriculum in England: Key Stages 3 and 4 framework document*. Retrieved February 15, 2021, from https://www.gov.uk/government/publications/national-curriculum-in-england-secondary-curriculum

Falkner, K., Vivian, R., Falkner, N. (2014, January). The Australian digital technologies curriculum: challenge and opportunity. In *Proceedings of the Sixteenth Australasian Computing Education Conference, Volume* 148 (pp. 3–12).

Fields, D. A., Quirke, L., Amely, J., Maughan, J. (2016, February). Combining big data and thick data analyses for understanding youth learning trajectories in a summer coding camp. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 150–155). https://doi.org/10.1145/2839509.2844631

Filvà, D. A., Forment, M. A., García-Peñalvo, F. J., Escudero, D. F., Casañ, M. J. (2019). Clickstream for learning analytics to assess students' behavior with Scratch. *Future Generation Computer Systems*, 93, 673–686. https://doi.org/10.1016/j.future.2018.10.057

Grover, S., Cooper, S., Pea, R. (2014, June). Assessing computational learning in K-12. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 57–62). https://doi.org/10.1145/2591708.2591713

Grover, S., Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher*, 42(1), 38–43. https://doi.org/10.3102%2F0013189X12463051

Haseski, H. I., Ilic, U. (2019). An investigation of the data collection instruments developed to measure computational thinking. *Informatics in Education*, 18(2), 297–319. https://doi.org/10.15388/infedu.2019.14

Head, A., Glassman, E., Soares, G., Suzuki, R., Figueredo, L., D'Antoni, L., Hartmann, B. (2017, April). Writing reusable code feedback at scale with mixed-initiative program synthesis. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale* (pp. 89–98). https://doi.org/10.1145/3051457.3051467

Hovemeyer, D., Hellas, A., Petersen, A., Spacco, J. (2016, August). Control-flow-only abstract syntax trees for analyzing students' programming progress. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (pp. 63–72). https://doi.org/10.1145/2960310.2960326

Izu, C., Mirolo, C., Settle, A., Mannila, L., Stupuriene, G. (2017). Exploring Bebras Tasks Content and Performance: A Multinational Study. *Informatics in Education*, 16(1), 39–59. https://doi.org/10.15388/infedu.2017.03

Manyika, J., Chui, M., Madgavkar, A., Lund, S. (2017). *Technology, jobs, and the future of work*. McKinsey Global Institute.

Marcelino, M. J., Pessoa, T., Vieira, C., Salvador, T., Mendes, A. J. (2018). Learning computational thinking and scratch at distance. *Computers in Human Behavior*, 80, 470–477. https://doi.org/10.1016/j.chb.2017.09.025

Martin, T., Brasiel, S., Jeong, S., Close, K., Lawanto, K., Janisciewcz, P. (2016, April). Macro Data for Micro Learning: Developing the FUN! Tool for Automated Assessment of Learning. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale* (pp. 233–236). https://doi.org/10.1145/2876034.2893422

Moreno-León, J., Robles, G., Román-González, M. (2015). Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, (46), 1–23.

Nguyen, A., Piech, C., Huang, J., Guibas, L. (2014, April). Codewebs: scalable homework search for massive open online programming courses. In *Proceedings of the 23rd international conference on World wide web* (pp. 491–502). https://doi.org/10.1145/2566486.2568023

Onah, D. F., Sinclair, J., Boyatt, R. (2014). Dropout rates of massive open online courses: behavioural patterns. *EDULEARN14 proceedings*, 1, 5825–5834.

Papavlasopoulou, S., Giannakos, M. N., Jaccheri, L. (2019). Exploring children's learning experience in constructionism-based coding activities through design-based research. *Computers in Human Behavior*, 99, 415–427. https://doi.org/10.1016/j.chb.2019.01.008

Papert, S. A. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic books.

Popat, S., Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education*, 128, 365–376. https://doi.org/10.1016/j.compedu.2018.10.005

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11), 60–67. http://doi.acm.org/10.1145/1592761.1592779

Rothkrantz, L. (2016, April). Dropout rates of regular courses and MOOCs. In *International Conference on Computer Supported Education* (pp. 25–46). Springer, Cham. https://doi.org/10.1007/978-3-319-63184-4_3

Scratch. (2021). *Scratch Statistics – Imagine, Program, Share*. Retrieved February 15, 2021, from https://scratch.mit.edu/statistics/

Seiter, L., Foreman, B. (2013, August). Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 59–66). https://doi.org/10.1145/2493394.2493403

Selby, C., Woollard, J. (2013). Computational thinking: the developing definition. *University of Southampton (E-prints) 6pp*.

Shute, V. J., Sun, C., Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, 22, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Taslibeyaz, E., Kursun, E., Karaman, S. (2020). How to Develop Computational Thinking: A Systematic Review of Empirical Studies. *Informatics in Education*, 19(4), 701–719. https://doi.org/10.15388/infedu.2020.30

The White House. (2016). *FACT SHEET: President Obama Announces Computer Science For All Initiative.* Retrieved February 15, 2021, from https://obamawhitehouse.archives.gov/the-press-office/2016/01/30/fact-sheet-president-obama-announces-computer-science-all-initiative-0

Tissenbaum, M., Sheldon, J., Abelson, H. (2019). From computational thinking to computational action. *Communications of the ACM*, 62(3), 34–36. https://doi.org/10.1145/3265747

Topalli, D., Cagiltay, N. E. (2018). Improving programming skills in engineering education through problem-based game projects with Scratch. *Computers & Education*, 120, 64–74. https://doi.org/10.1016/j.compedu.2018.01.011

Tsukamoto, H., Oomori, Y., Nagumo, H., Takemura, Y., Monden, A., Matsumoto, K. I. (2017, October). Evaluating algorithmic thinking ability of primary schoolchildren who learn computer programming. In *2017 IEEE Frontiers in Education Conference (FIE)* (pp. 1–8). IEEE. https://doi.org/10.1109/FIE.2017.8190609

Von Wangenheim, C. G., Hauck, J. C., Demetrio, M. F., Pelle, R., da Cruz Alves, N., Barbosa, H., Azevedo, L. F. (2018). CodeMaster – Automatic Assessment and Grading of App Inventor and Snap! Programs. *Informatics in Education*, 17(1), 117–150. https://doi.org/10.15388/infedu.2018.08

Wang, K., Singh, R., Su, Z. (2018, June). Search, align, and repair: data-driven feedback generation for introductory programming exercises. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 481–495). https://doi.org/10.1145/3192366.3192384

Weng, J. F., Tseng, S. S., Lee, T. J. (2010). Teaching boolean logic through game Rule tuning. *IEEE transactions on learning technologies*, 3(4), 319–328. https://doi.org/10.1109/TLT.2010.33

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. https://doi.org/10.1145/1118178.1118215

Yadav, A., Gretter, S., Good, J., McLean, T. (2017). Computational thinking in teacher education. In *Emerging research, practice, and policy on computational thinking* (pp. 205–220). Springer, Cham. https://doi.org/10.1007/978-3-319-52691-1_13

Yurkofsky, M. M., Blum-Smith, S., Brennan, K. (2019). Expanding outcomes: Exploring varied conceptions of teacher learning in an online professional development experience. *Teaching and Teacher Education*, 82, 1–13. https://doi.org/10.1016/j.tate.2019.03.002

A. Eloy is a Research Fellow with the TLT Lab with a focus on curriculum design in K-12 Computer Science and STEM fields. He holds a Master's degree in Learning, Design and Technology from Stanford University. Prior to that, he worked with the design of professional development programs in CS Education, in partnership with Departments of Education and local Universities in Brazil. Adelmo also holds a Master's Degree in Electrical Engineering and Bachelor's Degree in Mechatronics Engineering from the Universidade de São Paulo (USP).

C.F. Achutti is a professor at INSPER, an innovative engineering college in São Paulo. She received a bachelor's degree in computer science (2013) and a Master of Sciences in Computer Science, both from the University of São Paulo. Currently, she is a PhD candidate at the School of Engineering of the Universidade de São Paulo. She is also a founder of Mastertech and SOMAS. Both organizations that share the mission to use technology and data to improve education.

C. Fernandez is a Research Fellow with the TLT Lab, where she conducts research regarding science learning, curriculum design and teacher professional development. She holds a master's degree in Electrical Engineering from the Universidade de São Paulo (USP) and has worked in the Interdisciplinary Center for Interactive Technologies at USP developing tools for kids to learn coding, electronics and science in meaningful and creative ways. She is also a PhD student at USP; her research focuses on designing tools and approaches to connect programming and science learning in k-12 education.

R.D. Lopes is an Associate Professor at the Electronic Systems Department from the School of Engineering of the Universidade of São Paulo (USP). She received the undergraduate, master, doctorate and post-doctorate degrees in Electrical Engineering from USP. She is the vice-chair of the Instrumentation Center of Interactive Technologies at USP (CITI-USP). She was vice-chair (2006–2008) and director (2008–feb.2010) of Estação Ciência, a Center for Scientific, Technological and Cultural Dissemination of USP. She has been a researcher at the Laboratory for Integrated Systems (LSI) since 1988, where she is a principal investigator of the Interactive Electronic Media research group.