Computers in Human Behavior xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Computers in Human Behavior

journal homepage: www.elsevier.com/locate/comphumbeh

Full length article

Can computational thinking be improved by using a methodology based on metaphors and scratch to teach computer programming to children?

Diana Pérez-Marín[a,*], Raquel Hijón-Neira[a], Adrián Bacelo[a], Celeste Pizarro[b]

[a] Rey Juan Carlos University, Computer Science Department, Móstoles, Madrid, Spain
[b] Rey Juan Carlos University, Applied Mathematics Department, Móstoles, Madrid, Spain

ARTICLE INFO

Keywords:
Computational thinking
Primary education
Programming
Methodology
Metaphor

ABSTRACT

Computational thinking (CT) is a key skill in the 21st century. However, it is not clear which is the most effective way to acquire and improve CT. Big research efforts are made to determine which pedagogical means should be used. One research trend is based on the idea that teaching programming since Primary Education suffices to improve CT. In our previous work, we proposed and validated a methodology based on metaphors and used of Scratch (MECOPROG) to teach basic programming concepts to children. It is our hypothesis H that by applying MECOPROG, students will develop their CT. To check H, we carried out an experiment with 132 Primary Education Students (9–12 years in age). At the beginning of the experiment, all students were asked to fill in a programming concepts test and two tests to measure their CT. During the sessions, all students were taught according to MECOPROG. Finally, they took the three tests again. A significant increase in the results on all the tests has been measured, supporting the use of metaphors and Scratch to teach computer programming concepts to Primary Education students to develop their CT.

1. Introduction

Computational Thinking (CT) can be defined as the skill of solving problems, designing systems, and understanding human behavior based on computer science concepts (Wing, 2006). CT is a key skill for children in the 21st century (Wing, 2016). However, it is unclear how CT can be developed in the most effective way in children. Currently, different pedagogical methodologies that can be used to develop CT are being researched.

In the last years, some authors have claimed that CT can be acquired and developed by teaching programming to children. In addition, it has been claimed that this should be done as early as possible (Heintz, Mannila, & Färnqvist, 2016; Kazakoff, Sullivan, & Bers, 2013; McCartney, 2015; McCartney & Tenenberg, 2014; Papadakis, Kalogiannakis, & Zaranis, 2016; Strawhacker, Portelance, Lee, & Bers, 2015).

It is possible that CT can be acquired by other means such as Educational Robotics (Bers et al., 2010), storytelling (Lee et al., 2011), unplugged activities (Brackmann, Barone, Casali, Boucinha, & Muñoz-Hernández, 2016), Scratch Jr (Papadakis et al., 2016) or even in Ethics lessons (Seoane-Pardo, 2018). Although this paper focuses on programming to foster CT, learning how to program is worthwhile not only for that reason, but also because of the real need for programmers in our digital society (Margulieux, Catrambone, & Guzdial, 2016), as well as other advantages such as the improvement of higher cognitive skills (Pea & Kurland, 1984).

Many countries implemented Computer Science as a subject in Primary Education to train students into creators of computer programs (Heintz et al., 2016). A common approach to teaching Computer Science to children is Scratch, defined as an authoring environment - developed by the Lifelong Kindergarten research group at the MIT Media Lab - to design interactive media by snapping together programming-instruction blocks (Resnick et al., 2009; Ouahbi, Kaddari, Darhmaoui, Elachqar, & Lahmine, 2015). Other approaches focus on using Makey Makey, where students can interact with the computer by means of fruits or Play-Doh rather than using the traditional mouse (Lee, Kafai, Vasudevan, & Davis, 2014); using Lego WeDo or Mindstorms EV3 robots (Sović, Jagušt, & Seršić, 2014), and (producing) making games (Campe & Denner, 2015). Another possibility is to follow unplugged approaches using storytelling or free exercises from Code.org. This is particularly useful in countries with limited resources, but also in developed countries, where Computer Science is considered interesting, but there is a lack of trained teachers and/or resources (Brackmann et al., 2016).

The results of these approaches have not yet been properly evaluated, and their effectiveness is still unclear (Kalelioğlu, 2015).

* Corresponding author.
E-mail address: diana.perez@urjc.es (D. Pérez-Marín).

https://doi.org/10.1016/j.chb.2018.12.027
Received 28 December 2017; Received in revised form 6 August 2018; Accepted 16 December 2018

Moreover, no methodology or particular resources have been identified as the most adequate to teach programming to children.

There are difficulties in teaching children even basic concepts such as program construction (Lahtinen, Ala-Mutka, & Järvinen, 2005), loops (Ginat, 2004), structures control, and algorithms (Seppälä, Malmi, & Korhonen, 2006). These difficulties arise because of poor teacher training or a lack of a proper teaching methodology (Barker, McDowell, & Kalahar, 2009; Coull & Duncan, 2011). It has become evident that teachers need guidance to approach this task adequately (Brackmann et al., 2016; Jovanov, Stankov, Mihova, Ristov, & Gusev, 2016; Yadav, Gretter, Hambrusch, & Sands, 2016).

In our previous work, we proposed and validated the use of metaphors to introduce children to basic concepts of programming according to the methodology MECOPROG (Pérez-Marín et al., 2018). For instance, we proposed using the metaphors of a Thermomix® recipe as a program (and sequence), pantry as memory, and boxes as variables. We also illustrated the possibility of applying these metaphors to any resource available to the teacher, such as Scratch.

The reason for using metaphors is the widely reported usefulness of metaphors as powerful educational tools. Metaphors focus on concepts and facilitate students' organization of ideas and clearer, more straightforward thinking (Rodríguez Diéguez, 1988). Using metaphors does not require special equipment and helps teachers turn abstract concepts into simple ideas and images. Students need clear and careful, well-focused thinking to correctly write computer programs (Heintz et al., 2016).

This research paper asks the following question: Can computational thinking be improved by using a methodology based on metaphors and Scratch to teach computer programming to children? It is our hypothesis (H) that the answer is yes. For this study, we asked 132 Primary Education students (aged 9 to 12) to follow MECOPROG for six weeks. There were two objectives: 1) to teach students the basic concepts of computer science programming; and, 2) to develop students' CT by teaching them those concepts using metaphors and Scratch. The results derived from this study show that using metaphors and Scratch can significantly develop students' CT, but also that students are able to learn basic programming concepts.

The paper is organised as follows: Section 2 reviews background literature on computational thinking and teaching programming in Primary Education; Section 3 outlines the materials and methods of the experiment carried out so that this study can be reproduced elsewhere; Section 4 presents the results of the experiment; and Section 5, summarises the main conclusions and suggests future lines of work.

2. Background

Computational Thinking (CT) is not a new term. It dates back to 1950s, when it was referred to as “algorithmic thinking”. It was defined as a way to use algorithms to produce appropriate output to a given input (Denning, 2009). In 2006, Wing relaunched interest in the topic and defined CT as follows: “it involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science” (Wing, 2006). Given the generic nature of that definition, there has recently been several unsuccessful attempts to make it more specific (Aho, 2012; Brennan & Resnick, 2012; CSTA & ISTE, 2011; Google for Education, 2018; Wing, 2008).

According to Grover and Pea (2013), CT includes decomposition, pattern generalisations and location, abstraction and algorithms among other Computer Science resources such as debugging and systematic error detection, iterative, parallel and recursive thinking, control flow and use of symbols.

Brennan, Balch, and Chung (2014) also explored CT in terms of programming, more specifically using Scratch based on a 3-D CT classification into concepts, practices, and computational perspectives (Brennan & Resnick, 2012). See Table 1.

Table 1
Summary table of the 3-D CT dimensions model (source: Brennan & Resnick, 2012).

- Concepts:
  - Sequences
  - Loops
  - Parallelism
  - Events
  - Conditionals
  - Operators
  - Data
- Practices:
  - Incremental & iterative development
  - Test & debugging
  - Mix & reuse
  - Abstract & encapsulate
- Perspectives:
  - To express
  - To connect
  - To question

The goal is not to replace creative and critical thinking or other competences, but to add the skill of using computers and algorithms to solve problems (Cuny, Snyder, & Wing, 2010; Wing, 2011; CSTA & ISTE, 2011; Furber, 2012; Espino, Soledad, & González, 2015). Many governments emphasise the need for children to be fluent in the digital language rather than making them mere users of computer software (García-Peñalvo, 2016). There are already certain resources available in the specialised literature for this (Balanskat & Engelhardt, 2015; Wing, 2008).

Learning how to program can induce changes in the way that people think (Papert, 1980; Resnick, 1996). This is likely because of the analytical component of CT, which is quite similar to mathematical thinking (i.e. problem solving), engineering thinking (design and evaluation of processes) and scientific thinking (systematic analysis).

CT can be useful not only for students or professionals of Computer Science, but for any other person (Wing, 2006). Starting CT training as early as possible is of particular interest, and it has been shown that children aged as young as four can understand programming concepts and even build simple robots which can move and interact with the environment (Bers, Ponte, Juelich, Viera, & Schenker, 2002; Bers et al., 2006).

This is why teaching Computer Science programming has been included in the Primary Education curricular in many countries (Heintz et al., 2016; see Table 2). A commonly used approach to teach Computer Science to children is using Scratch (Resnick et al., 2009). While interacting with Scratch, students learn basic concepts such as sequences, loops, parallelism, events, conditionals, operators and data (Brennan & Resnick, 2012; Ouahbi et al., 2015).

Other approaches include making an own program (Campe & Denner, 2015), using Lego WeDo or Mindstorms EV3 robots (Sović et al., 2014). With regard to unplugged approaches: these are common in countries with limited resources, but also in developed countries, which consider Computer Science an interesting option but lack trained teachers and/or Internet connections (Brackmann et al., 2016).

In unplugged approaches, the concepts of Computer Science are transmitted through storytelling or free exercises available on Code.org. It bears mentioning here that there is no established method to evaluate the effectiveness of these approaches; therefore, their validity is still unclear (Kalelioğlu, 2015).

A previous study by Pérez-Marín et al. (2018) contributed to the debate by introducing metaphors as an alternative approach to teaching programming. Metaphorical language is employed in real, everyday life, and is considered a crucial component of thinking (Lakoff & Johnson, 2008). In particular, conceptual metaphors (i.e., cognitive mechanisms that project from a source domain to a target domain in order to facilitate understanding of a concept in the target domain) are of great interest in educational environments (Sanford, Tietz, Farooq, Guyer, & Shapiro, 2014).

Metaphors have been used to teach Biology (Paris & Glynn, 2004), Chemistry (Thomas & McRobbie, 2001), and Mathematics (Boero, Bazzini, & Garuti, 2001). The use of metaphors to teach Computer Science is common at college level and has been the subject of research interest (Putnam, Sleeman, Baxter, & Kuspa, 1986; Sanford et al.,

Table 2
Interest in teaching programming (based on Heinz, Mannila & Färnqvist. et al., 2016).

Country	Content	Form	Primary	Secondary
Australia	Digital Technologies	Own subject and integrated	Compulsory	Compulsory
England	Computing	Replaces existing subject	Compulsory	
Estonia	Programming	Integrated	Compulsory	Compulsory
Finland	Programming	Integrated	Compulsory	
New Zealand	Programming and Computer Science	Own subject		Opcional
Norway	Programming	Own subject		Opcional
Sweden	Programming and Digital Competence	Integrated	Compulsory	Opcional
South Korea	Informatics	Own subject	Compulsory	Opcional
Finland	Computer Science	Own subject	Compulsory	Compulsory
USA	Computer Science	Own subject		Opcional
Macedonia	Computers and basics of programing	Own subject	Compulsory	

2014). There are studies on specific metaphors, such as the locker memory to teach dynamic memory (Jiménez-Peris, Pareja-Flores, Patiño-Martínez, & Velázquez-Iturbide, 1997), or matrixes for event-handling in JAVA (Milner, 2010). However, the use of metaphorical language as a tool to teach basic concepts of Computer Science for Primary Education, and the minimum age at which it can be used, have not yet been researched in detail. Therefore, we proposed and validated a methodology called MECOPROG (see description in Section 3) to teach programming to Primary Education students, using metaphors. The purpose of the experiment described in this paper is to analyse whether MECOPROG has an impact on the students' programming knowledge and whether it can improve computational thinking in students.

3. Method

3.1. Participants

132 Spanish Primary Education students (56.1% male and 43.9% femal, aged 9 to 12), recruited in two parts, were asked to take part in the experiment in order to assess whether their Computational Thinking (CT) improved after teaching programming using the MECOPROG methodology based on metaphors.

The reason for having two different parts is that programming is not compulsory in Spanish schools. Therefore, only a few students, usually in Private schools, have the opportunity to attend programming lessons. We asked several Private schools that offered programming to collaborate in the study; one school agreed because their programming teacher was on sick leave and they required a skilled temporary teacher for 4th, 5th and 6th grades. We were given permission to teach using the MECOPROG method during the six weeks the teacher needed to convalesce.

In addition, and in order to provide other children with the opportunity to attend programming classes and to ensure a more heterogeneous sample, we offered other schools in Madrid and the City Council of Fuenlabrada (where the authors live) a cost-free programming camp to be held on three consecutive Saturdays, which children aged 10 to 12 could attend. This camp also used the MECOPROG methodology.

50% of our participants attended Private schools and the rest attended 32 different public schools and were recruited as they participated in the programming camp. Fig. 1 shows the distribution of the students per grade (in Spain, 4th grade corresponds to students aged 9–10, 5th grade corresponds to students aged 10–11, and 6th grade corresponds to students aged 11–12). 18.2% were 4th grade students, 38.6% were 5th grade students, and 45.2% were 6th grade students.

3.2. Design

The research model followed a longitudinal pretest-posttest quasi-experimental design, because the Head-Master of the Private School did not provide a control group and we could not randomly assign students to each group.

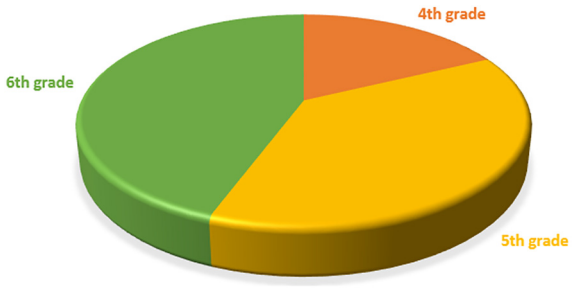

Fig. 1. Distribution of the participants in grades.

Similarly, we could not have a control group in the programming camp or randomly assign students to each group, because one of the City Council's conditions to providing us with a class-room was that all students must receive the same teaching. Moreover, as rooms were only available from 10:00 a.m. to 2:00 p.m., we had to divide students; thus, students in the 5th grade attended from 10:00 a.m. to midday, whereas those in the 6th grade attended from midday to 2:00 p.m. It was not possible to recruit 4th grade students from the City Council.

According to Cook & Campbell (1986), we could measure the impact of our intervention using MECOPROG by following the quasi-experimental design outlined below. No rewards were offered.

3.3. Materials

3.3.1. MECOPROG

Our main resource was MECOPROG (Pérez-Marín et al., 2018). Table 3 summarises the metaphors used in MECOPROG grouped into three blocks: 1 – Program, sequence, variable and input and output instructions, 2 - Conditional instructions and 3 - Loops.

The process for each block was (1) introduce the concept by using metaphors, and then (2), practice with Scratch. See Fig. 2 for a global overview of MECOPROG.

Block 1 (2 h): First, the concept of programming was explained by using the Thermomix® (Tx) cooking recipe metaphor. Children were explained that – just likewhen they are following steps in a Tx recipe -

Table 3
Metaphors used in MECOPROG for programming concepts.

Block	Concept	Metaphor
1	Program, sequence, memory and variable	Thermomix® (Tx) recipe, pantry and box
	Input and Output	Mouth and rectum (beginning and end of the digestive system)
2	Conditional	Intelligent fridge
3	Loop	Hand mixer

Fig. 2. Overview of MECOPROG blocks: up (first) concept (second) Scratch. From left to right: Sample script to work with Input/Output (block 1), sample script to work with Conditionals (Block 2), sample script to work with Loops (Block 3).

the computer is able to execute the instructions of a program one step at a time. No ambiguity is allowed as the computer does not understand instructions that are not precise. Additionally, just as the goal of following a recipe is to produce good food, a program always has a specific output. The first program that they are asked to execute in Scratch is to say "Hello!".

Scratch's "say" and "ask" instructions help to teach basic input/output programming concepts. Whenever students do not understand them, the metaphor of input as eating something with their mouth, and how their digestive system processes it until it passes through the rectum (output) is used. That way, students understand that you can enter data into the computer (input), execute a program and produce a result (output).

The concept of sequence is also explained here, as the sentences in a Tx recipe/program must be executed one after another. Students are also asked whether they think that a computer has a memory. Surprisingly, not all students think they do. Therefore, we help them understand the concept of computer memory by using the pantry metaphor. The concept of data was explained by comparing the ingredients they need to carry out a Tx recipe with the data a computer needs to executes a program. Moreover, the pantry metaphor can also be used to explain the concept of variables as a boxes metaphor. Just like food is organised in the kitchen: eggs in their box, fruit in their fruit-bowl, etc. so the computer organises data into boxes inside its memory as variables. To illustrate the concept, we created the Scratch program shown in Fig. 2 left.

Block 2 (2 h): An intelligent fridge was used as a metaphor to explain conditionals. Students were told that the fridge has a sensor to detect how many pieces of food it contains. For instance, we told children to imagine that they were in charge of serving dessert to their family at dinner. An intelligent fridge would know how many family members there are and thus how many pieces of fruit they would need. If they were four family members, they would need at least four pieces of fruit. If there were fewer than four pieces contained in the fridge, the intelligent fridge will connect to Internet to buy more fruit. To illustrate the concept, we created the Scratch program shown in Fig. 2 center.

Block 3 (2 h): For loops, a hand mixer metaphor was used. Just like a hand mixer repeats the same movement over and over again, a loop repeats the same command over and over again until a condition is fulfilled. Students are told that the condition in the case of the hand mixer is to whip the eggs five times. To illustrate the concept, we created the Scratch program shown in Fig. 2 right.

Following the 3D CT Model (see Table 1) for Concept Dimensions the methodology covers Sequences, Loops, Conditionals, Operators and Data (see Table 4 left). MECOPROG also covers the first three practices of the Practices Dimension (see Table 1). All students were encouraged to revisit their programs and incrementally improve them as they learnt new concepts. All the examples involved testing and debugging when simple coding was reused and incorporated into more complex ones (see Table 4 top right).

In regard to the Perspective Dimension, the metaphor methodology also allows questioning (giving solutions to a proposed guided methodology) and expressing (giving solution to a problem using the computer) (see Table 4 right bottom).

Table 4
Concepts, Practices and Perspectives covered by MECOPROG, highlighted in underlined, in the 3D CT Model proposed by Brennan and Resnick (2012).

- Concepts:
  - Sequences
  - Loops
  - Parallelism
  - Events
  - Conditionals
  - Operators
  - Data
- Practices:
  - Incremental & iterative development
  - Test & debugging
  - Mix & reuse
  - Abstract & encapsulate
- Perspectives:
  - To express
  - To connect
  - To question

During the experiment, cutting cross all three blocks, we occasionally used an application named CompThink App in our sessions (2018), developed ad-hoc for the improvement of children's computational thinking. The app works with seven aspects, all focused on improving the students' computational thinking (see Fig. 3):

1 Loops: Students select an element and set the number of times that the element will repeat the action. Then, they can watch an animation of the action being repeated the selected number of times.
2 Algorithms: Students establish the order of a finite set of steps to carry out a certain activity, such as cooking a recipe, or planting a tree.
3 Patterns: Students choose the different features of a face out of a variety of options to create a face. They choose between different types of hair, eyes or mouths, and the outcome is an animated gif with the selections they have made.
4 Conditionals: Students drag and drop images according to the options given in “if/else” branches. For instance: “if the weather is cold (scarfs, coat and boots), else (t-shirts, bathing suit and shorts)“.
5 Steps: Students select which part is missing from several possibilities in the picture. In Fig. 3, for example, Cream and a Cherry is needed to complete the cupcake.
6 Instructions: Students find the final position on a map, divided in squares, after following a set of movement instructions.
7 Automats: Students select the correct order to follow a path from one place to another with certain restrictions.

CompThink App (CompThink App, 2018) is a drag-and-drop visual interface for Android tablets or smartphones. Table 1 shows the 3-D CT dimensions model (Brennan & Resnick, 2012). Table 5 presents the concepts that each part of the App covers including those proposed by Brennan and Resnick (op. cit.).

3.3.2. Tests

Three tests have been used to measure the impact of our intervention that used MECOPROG to teach students programming concepts and skills and to improve their computational thinking.

We used the CONT[1] questionnaire to measure participants' knowledge of programming concepts (measures the CON variable as explained in Section 3.5). CONT tests the participants' knowledge of Programming, Sequence, Memory & Variables, Input & Output instructions, Conditionals, and Loops. The question formats are as follows: “What do you think a program is? Can you give an example?“, and seek to measure students' knowledge of those areas.

Two tests were used to measure computational thinking. The first test is called ROMT[2] (measures the ROM variable as explained in Section 3.5). ROMT is a validated test with 28 items that measures the Computational Thinking of children aged over 10 (Román-González, Pérez-González, & Jiménez-Fernández, 2017). Fig. 4 shows a sample question in ROMT. Questions are based on Scratch code blocks and cover certain CT areas.

Given that ROMT cannot evaluate Computational Thinking in children younger than 10, we also used a second validated test suitable for those students. However, as far as we know, no test has been validated to measure CT in children younger than 10. Therefore this study proposed using a new test to measure CT created for children of this age. It is called PCNT[3] and measures the PCN variable as explained in Section 3.5. Fig. 5 shows an example PCNT question.

PCNT has 14 exercises grouped into the categories outlined in Section 3.3.1 for the CompThink App (2018), to cover the 3-D CT dimensions model created by Brennan and Resnick (2012) to assess computational thinking. Unplugged approaches use these types of exercises to develop CT (Brackmann et al., 2016).

We debated whether to use PCNT exclusively for all students regardless of their age. However, we decided against that idea as ROMT is a validated test, and we wanted to ensure that the results provided by both tests could be correlated.

3.4. Procedure

Fig. 6 shows the experimental procedure. At the start of the experiment, all students took three tests:

- ROMT: a validated test for children to measure computational thinking (Román-González et al., 2017).
- CONT: a concept test created ad-hoc for the experiment.
- PCNT: a new test to measure CT based on the field's literature.

Students from 4th to 6th grades in Spanish Primary Education (aged 9–12) attended classes that taught programming through the MECOPROG pedagogical method. Certain students, who had chosen Programming as one of their optional school subjects, attended as part of their schooling while the remaining students, who did not take Programming as a subject (as it is not compulsory in Spain), were grouped into a programming camp on Saturdays. All of them used Scratch.

After 6 weeks/1 h per week in the Private School and 3 2 h-sessions in the camp, all students took those same three tests again. We had previously decided to use the same tests again to guarantee that the post-tests had the same level of difficulty as the pre-tests.

In order to avoid student boredom of taking the same tests again, we waited a minimum of 3 weeks before asking them to take the post-test. In addition, during the first test session we did not resolve any questions they had from taking the pre-tests to avoid giving solutions to the post-test.

3.5. Measures

The variables were the scores achieved by the students in the CONT, ROMT and PCNT tests. Specifically, the following:

- CON: students' knowledge test (CONT) score
- ROM: students' validated CT (ROMT) score
- PCNT: students' new CTT test score (PCNT)

3.6. Data analysis

A comparative study using non-parametric tests to measure the hypothesis contrast between the pre-post PCN, CON and ROM values was performed. Non-parametric tests were used because when the data gathered was analysed, we saw that they had not come from a normal distribution, and we did not have enough data to assume normality.

4. Findings

4.1. Overall results

Table 6 shows the means, medians (more representative than the mean in asymmetric distribution), and standard deviation for pre-test and post-test of PCN, CON and ROM.

Without making distinctions per grade, Table 6 reveals a clear increase in the post-test results in the three variables, showing a greater improvement in CON variable, and a smaller improvement in ROM variable. Standard deviation slightly increases in all the variables, except in PCN, where it is more reduced in the post-test.

Fig. 7 shows box-plots for these three variables, both in the pre-test and post-test. Graphically, 50% of the central data are represented in

[1] https://tinyurl.com/mecoprogCT (in Spanish).
[2] https://tinyurl.com/mecoprogTRG (in Spanish).
[3] https://tinyurl.com/mecoprogCTL (in Spanish).

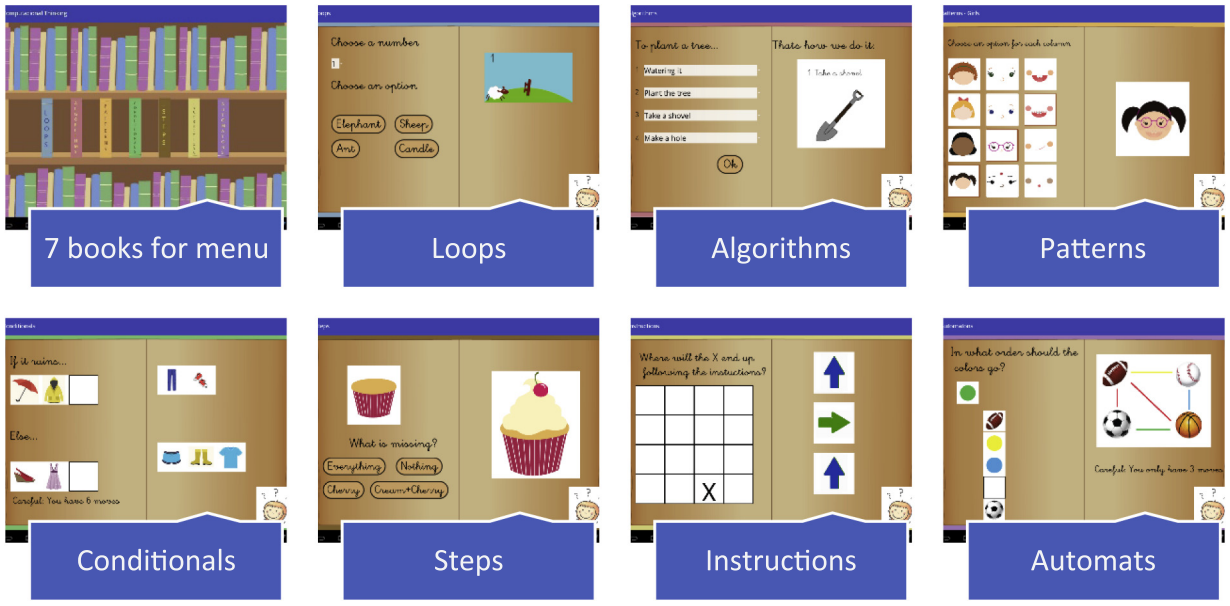

Fig. 3. CompThink App (2018), an example of the games for each of the seven options available.

Table 5
Concept Dimension Covered of 3-D CT dimensions model by CompThink APP.

CompThink App	Brennan and Resnick Concept Dimension on CT
Loops	Loops and Data
Algorithms	Sequences
Pattern	Sequences and Data
Conditionals	Conditionals
Steps	Sequences and Operators
Instructions	Sequences
Automats	Sequences and Conditionals

the box, and the median are marked with a line as representative measure. The highest and lowest values for each box-plot correspond to values which are not less than Q1-1.5·(Q3-Q1) and not greater than Q3 + 1.5·(Q3-Q1). Some outliers are marked with the case number.

After analysing the data using the Shapiro-Wilk test, we found that the distribution of the variables under study did not come from a normal distribution, except in POST_CON variable ($p = 0.292$) and POST_ROM ($p = 0.203$). Therefore, and without having a high enough number of population, nonparametric tests were chosen for the study to guarantee the robustness of the results.

Spearman's rank correlation coefficient shows a significant correlation ($p < 0.001$) between pre and post-tests in PCN, CON and ROM variables. The Wilcoxon signed-rank test is used to compare two related samples, in this case the pre and post-tests, and evaluate whether there is any statistically-significant difference in the pre and post-tests in the three variables studied.

Table 7 shows a significant improvement for all tests. The ROM p-value is much higher than the others. Therefore, CON variable is the most significant. Consequently, we could conclude that the population saw a significant improvement in the tests.

Some additional information to size the effect is the r value, introduced by Rosenthal in 1991. PCN had a value of $r = 0.15$ corresponding to a small effect, $r = 0.55$ for CON variable, corresponding to a large effect, and, finally, $r = 0.16$ for ROM variable, indicating a small effect.

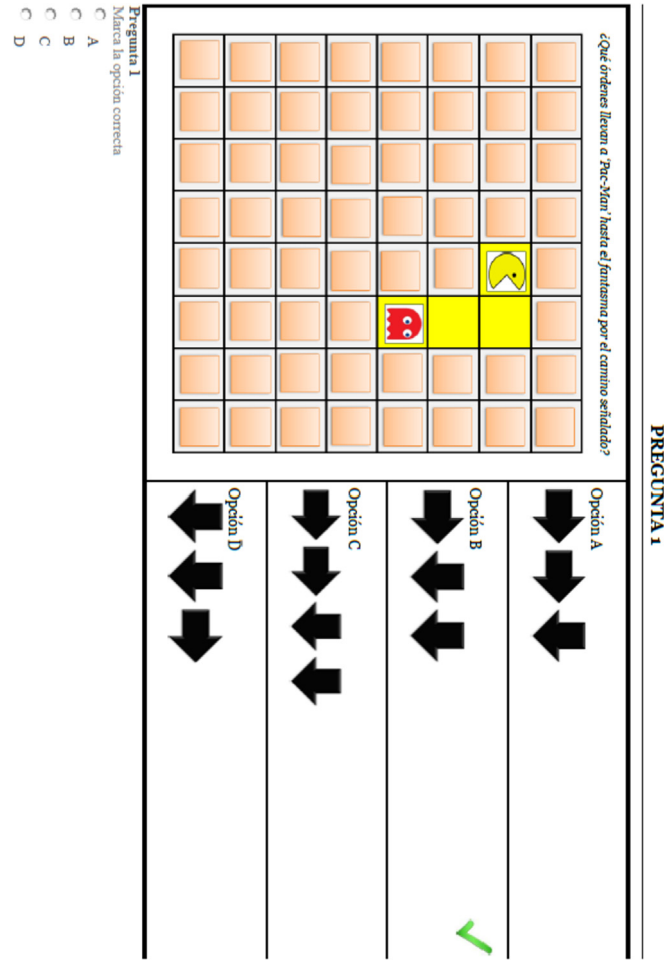

Fig. 4. An example of a ROMT question ("What instructions can you give Pac-Man to reach the ghost?" Choose from: a, b, c or d).

4.2. Results per grades

First, a descriptive analysis for each variable in the three grades is presented. Tables 8–10 show the median, mean and standard deviation.

CON variable shows a large increase for all grades, in addition to increasing data dispersion (see Table 9). In ROM variable, 5th and 6th show an increase for the median, as well as the standard deviation. Box-

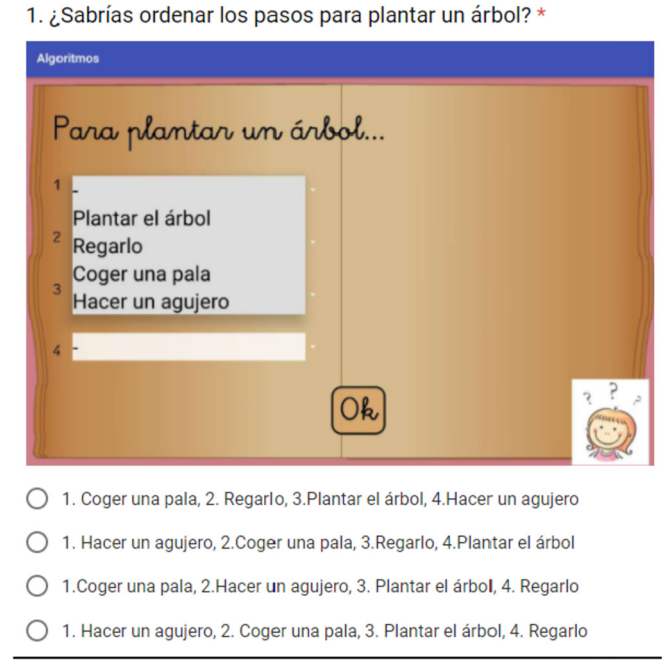

Fig. 5. Example of a PCNT question ("Do you know what steps you need to take to plant a tree? Place the following four actions into order: 1) Take a shovel, 2) Water the tree, 3) Plant the tree, 4) Dig a hole").

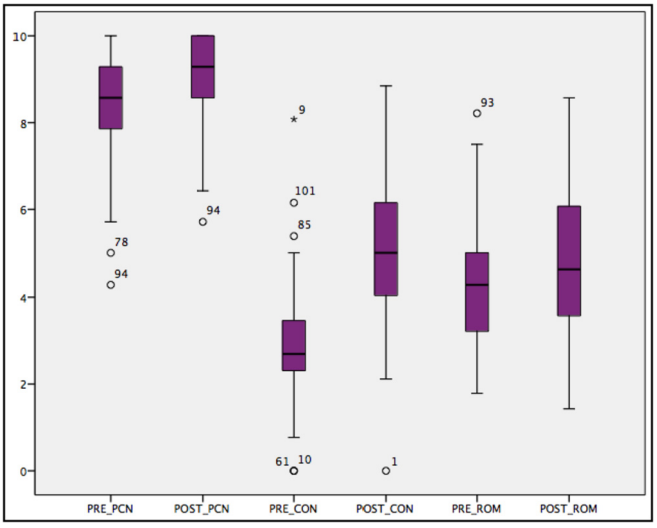

Fig. 7. Box-plot for variables PCN, CON and ROM in pre and post-test.

Again, Spearman's rank correlation coefficient shows a significant correlation ($p < 0.001$) between pre and post-tests in PCN and CON in all the grades, and in 5th and 6th grades in ROM. The Wilcoxon signed-rank test is used to compare two related samples, in this case pre and post-test, and evaluate whether there is any statistically significant

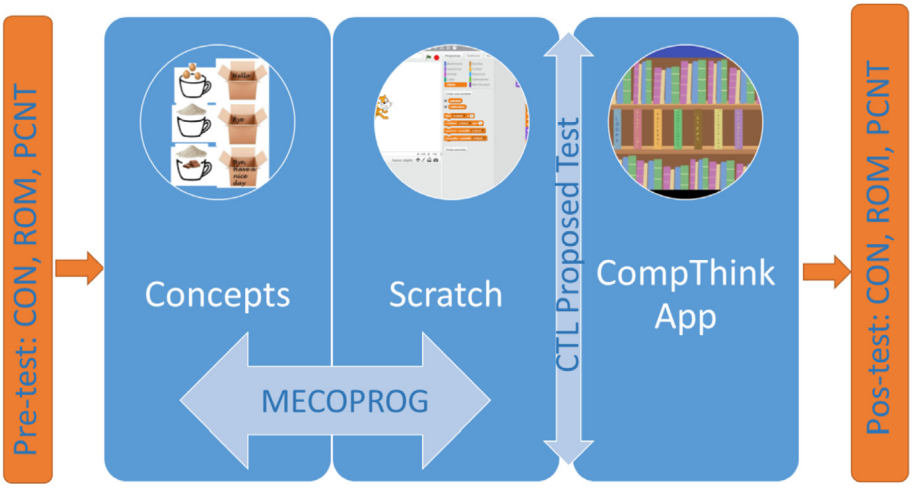

Fig. 6. Flow diagram of the experiment.

Table 6
Median, mean and standard deviation in pre- and post-tests PCN, CON and ROM.

	PCN			CON			ROM		
	Mdn	M	SD	Mdn	M	SD	Mdn	M	SD
Pre	8.57	8.37	1.25	2.69	2.77	1.32	4.28	4.23	1.36
Post	9.28	8.99	1.05	5	5.08	1.59	4.64	4.77	1.56

plots confirm this statement (see Table 10). PCN variable shows a large increase in 5th grade, followed by 6th grade. Standard deviation is reduced. No improvement is observed for 4th grade students (see Table 8).

Box-plots confirm this reasoning and show the existence of several outliers related to low marks for the three grades, especially in 4th grade.

Table 7
Comparative study using Wilcoxon test.

	PCN	CON	ROM
Z	−2.830	−8.543	−2.294
p-value	0.005	0.000	0.022

difference in pre- and post-tests in the three variables studied for each grade.

Table 11 shows significant differences in different grades: there is a significant improvement in PCNT variable in 5th ($p = 0.008$), as well as ROMT variable, although with a higher p-value ($p = 0.023$). In the case of CON variable, the improvement is significant in all grades ($p = 0.000$).

Rosenthal r value quantifies the improvement where it happens. In 4th grade, a large improvement, close to very large, is observed in CON variable ($r = 0.62$). In 5th grade, there is a small increase, near to a

Table 8
Median, mean and standard deviation for pre-test and post-test for PCN, per grades.

	PCN								
	4th			5th			6th		
	Mdn	M	SD	Mdn	M	SD	Mdn	M	SD
Pre	9.28	8.60	1.42	7.85	8.26	1.12	8.57	8.45	1.34
Post	9.28	8.75	1.68	9.28	9.03	1.01	9.28	8.95	1.09

Table 9
Median, mean and standard deviation for pre-test and post-test for CON, per grades.

	CON								
	4th			5th			6th		
	Mdn	M	SD	Mdn	M	SD	Mdn	M	SD
Pre	2.69	2.50	1.01	2.31	2.74	1.67	2.69	2.80	0.99
Post	5.19	5.09	1.28	4.80	5.01	1.68	5.19	5.14	1.54

Table 10
Median, mean and standard deviation for pre-test and post-test for ROM per grade.

	ROM					
	5th			6th		
	Mdn	M	SD	Mdn	M	SD
Pre	4.28	4.43	1.54	4.11	4.08	1.21
Post	5.00	5.17	1.71	4.64	4.46	1.38

medium increase in PCN variable (r = 0.27). A large increase occurs in the CON variable (r = 0.57) and a small increase in the ROM variable (r = 0.23). Finally, in 6th grade, a large effect is found, with r = 0.55.

4.3. Synopsys table

Table 12 gathers the increase of the scores in the three tests taken by the students before and after MECOPROG.

Finally, in response to our question regarding the relationship between PCNT and ROMT, there is a low lineal correlation between them (Spearman r = 0,248 $p < 0.01$).

5. Discussion

This paper explored whether Primary Education students' CT can be improved and, to what extent Primary Education students are able to learn programming concepts. It included factors such as grade and used tests to measure children's' knowledge and computational thinking.

One important conclusion is that there is a statistic significant increase in children's post-test results both in knowledge (according to the CONT knowledge test), and CT values for all grades (according to PCNT and ROMT CT tests). This suggests that even in a short period of time it is possible to teach children basic computer programming concepts such as memory, programming, conditionals or loops, and improve their CT, with children as young as nine.

Table 12
Rosenthal r to quantify the improvement detected in the three tests.

	N	DPCN	DCON	DROM
4th	23	–	0.62	–
5th	38	0.27	0.51	0.23
6th	50	–	0.55	–
All	85	0.15	0.55	0.16

It is worth noting that although there is a general consensus regarding the need to foster children's CT (Román-González, 2015), and the results reported in this study significantly contribute to the literature in this sens, there is still much controversy surrounding the definition of the term CT and how and when to integrate it into the curriculum (Gouws, Bradshaw, & Wentworth, 2013).

Since this study required a practical definition of what comprises CT in order to work with children and anaylse what parts of CT could be improved and how, we chose the 3D CT Model (Brennan & Resnick, 2012; see Table 1). The reason for selecting that model was that it had been created by the authors of Scratch, a program that allows children to program.

As noted by Vico (2017) (translated from Spanish); “A child who does not learn how to program will have the same handicap as a Spanish child who is not able to understand English”. This is also why we wanted to foster an interest in CT, so that children can become programmers. Otherwise, it seems as though we have only taught our Pre-school children to read, but not how to write.

However, some Computer Science educators have argued that programming is not necessary to teach computational thinking (Lu & Fletcher, 2009; Yadav, Zhou, Mayfield, Hambrusch, & Korb, 2011). Some have even suggested that teaching programming to foster CT could deter students as some may find computer science and programming boring (Lu & Fletcher, 2009).

In light of our results, and from our experience with children age 9 to 12, learning how to program is engaging and helps them focus on problems. All children payed attention during the lessons regardless of their grade. In general, it is well known that children love computers and do not think that they are difficult or boring to use. This could be used as a base to start working with children, who are naturally fascinated by technology.

This study is particularly relevant for teachers and national curriculums as it shows that children can enjoy learning about programming. Until recently, it was unconceivable to think children could learn about programming. On the contrary, children were not taught these concepts until Secondary School, or even University, at an age that students begin to find these complex ideas difficult to understand, in contrast to younger children, who can easily absorb them when adapted to their age.

It has also become evident that teachers need guidance in their approach to this task. According to this study, students are able to learn programming concepts if they are taught with methodologies such as MECOPROG with Scratch and those that use metaphors. Teachers must be trained in those methodologies if we want to reach Primary

Table 11
Comparative study using Wilcoxon test.

	PCN			CON			ROM	
	4th	5th	6th	4th	5th	6th	5th	6th
Z	−0.515	−2.674	−1.362	−4.204	−4.845	−5.715	−2.274	−0.928
p-value	0.607	0.008	0.173	0.000	0.000	0.000	0.023	0.354

Education students. Therefore, and in line with the Digital Competence that teachers should develop, we should include this training in their pedagogical education (INTEF, 2017).

Measuring the progress of CT is also necessary. The tests used for this could differ depending on the definition of CT and the age of the students. For instance, the test created by Korkmaz, Çakir, and Özden (2017) is limited to the sub-skills comprising the ISTE (2015) definition. In addition, it is limited to associate students and older students.

Only one validated test was found to measure CT according to the 3D CT Model and to be useful for young children. The authors of this test kindly allowed us to use it (Román-González et al., 2017) for this study. However, the test (ROMT) is only validated for children older than 10. For younger students, we tried a new test (PCNT), as explained above, which was also in line with the published 3D CT Model.

The results gathered both from ROMT and PNCT proved that CT can be improved by using MECOPROG. However, given the low correlation found between PCNT and ROMT, more studies should be carried out focusing on how to assess CT for young students, particularly those aged under 9.

5.1. Limitations and future work

We are aware that these measures would change if we use a different CT definition, and that the results may change if we use a different model, and/or a different computer program other than Scratch.

Furthermore, MECOPROG can be used with different resources, and may thus produce different results in each case. The core metaphors used in this paper are based on cooking. Different metaphors can be used such as car metaphors: the door as input/output for the car, junctions for conditionals, roundabouts for loops and so on. During class, teachers are able to select the most adequate metaphor from MECOPROG, as published (Pérez-Marín et al., 2018).

The experiment has been described in great detail so that other researchers are interested in repeating it with a different sample or to test more advanced programming concepts, can do so easily. This study experiment focused on basic introductory computer programming concepts because it was the first contact with those particular students and the project only had a limited amount of time.

The authors are currently also carrying out a multifactorial study to determine whether other factors such as sex, motivation or effort might have an impact on students' CT test scores. We would also like to continue with the validation of PCNT, given the low correlation found between PCNT and ROMT.

6. Conclusions

The findings of this longitudinal pre- and post-test quasi-experiment carried out with 132 Primary Education students (aged 9 to 12) positively confirmed the formulated research question: Can computational thinking be improved using a methodology based on metaphors and using Scratch to teach computer programming to children?

Table 12 gathers the main results of the research study. Here, both the knowledge programming concept test and the CT tests (ROMT and PCNT) found better post-test scores, when data were analysed for all the grades. These results suggest that using metaphors and Scratch is useful for teaching computer programming concepts to Primary Education students, and for improving students' CT, providing a positive answer to the research question. It contributes to the area of study exploring how to develop Computational Thinking by covering gaps in methodologies, and uses, for the first time, metaphors to teach basic programming concepts to Primary Education Students, together with Scratch.

When the analysis is carried out per grade, it shows that 4th grade students can understand programming concepts. In fact, it seems to show that these students are actually able to learn more about programming concepts, as their increase in CONT knowledge improves more than any other group. Because we could not apply the ROMT test to them as they were aged younger than 10, no conclusion could be drawn from this test. When the results of the PCNT are analysed, no significant results could be drawn either, indicating the need for more studies to quantify the increase (if any) of the development of CT in students aged younger than 10 when they learn programming concepts.

We cannot end without highlighting the fact that 5th grade students improved their performance in all tests. Not only did they learn more programming concepts through MECOPROG, but their scores in both CT-measuring tests increased significantly. Finally, 6th and 4th grade students' knowledge of programming improved, but no significant improvement was found in ROMT or PCNT scores. This may indicate either that these students need more time to improve their CT, or that the MECOPROG metaphors methodology is more applicable to students aged 10–11.

Acknowledgments

Research funded by the projects TIN 2015-66731-C2-1-R and S2013/ICE-2715.

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832–835.

Balanskat, A., & Engelhardt, K. (2015). *Computing our future. Computer programming and coding Priorities, school curricula and initiatives across Europe.* Brussels, Belgium: European Schoolnet2015.

Barker, L. J., McDowell, C., & Kalahar, K. (2009). Exploring factors that influence computer science introductory course students to persist in the major. *ACM SIGCSE bulletin: Vol. 41*, (pp. 153–157). ACM No. 1.

Bers, M. U. (2010). The TangibleK Robotics program: Applied computational thinking for young children. *Early Childhood Research & Practice, 12*(2), 2.

Bers, M. U., Ponte, I., Juelich, C., Viera, A., & Schenker, J. (2002). Teachers as designers: Integrating robotics in early childhood education. *Information Technology in Childhood Education Annual, 1*(1), 123–145.

Bers, M., Rogers, C., Beals, L., Portsmore, M., Staszowski, K., Cejka, E., et al. (2006). Innovative session: Early childhood robotics for learning. *Proceedings of ISTE, C. 2011. Computational thinking in K–12 education leadership toolkit*.

Boero, P., Bazzini, L., & Garuti, R. (2001). Metaphors in teaching and learning mathematics: A case study concerning inequalities. *Pme conference: Vol. 2*, (pp. 2–185).

Brackmann, C., Barone, D., Casali, A., Boucinha, R., & Muñoz-Hernández, S. (2016). Computational thinking: Panorama of the americas. *Computers in education (SIIE), international symposium on IEE* (pp. 1–6). .

Brennan, K., Balch, C., & Chung, M. (2014). *Creative computing. Cambridge [masachussets]*. Harvard Graduate School of Educationhttp://scratched.gse.harvard.edu/guide/files/CreativeComputing20141015.pdf [Consulta: 30/05/2017].

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

Campe, S., & Denner, J. (2015). Programming games for learning: A research synthesis. *Paper presented at the annual meeting of the American educational research association, Chicago, IL*.

Cook, T. D., & Campbell, D. T. (1986). The causal assumptions of quasi-experimental practice. *Synthese, 68*(1), 141–180.

CompThink App (2018). http://www.lite.etsii.urjc.es/tools/compthink-app/.

Coull, N. J., & Duncan, I. M. (2011). Emergent requirements for supporting introductory programming. *Innovation in Teaching and Learning in Information and Computer Sciences, 10*(1), 78–85.

CSTA, & ISTE (2011). *Operational definition of computational thinking for K–12 education.* Retrieved from http://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf.

Cuny, J., Snyder, L., & Wing, J. M. (2010). *Demystifying computational thinking for non-computer scientists.* Unpublished manuscript in progress, [On line]. Available: referenced in http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.

Denning, P. J. (2009). The profession of IT. Beyond computational thinking. *Communications of the ACM, 52*(8), 28–30.

Espino, E., Soledad, C., & González, C. (2015). Estudio sobre diferencias de género en las competencias y las estrategias educativas para el desarrollo del pensamiento computacional. *Revista de Educación a Distancia, 46*.

Furber, S. (2012). *Shut down or restart: The way forward for computing in UK schools.* [On line]. Available: Retrieved from http://royalsociety.org/education/policy/computing-in-schools/report/.

García-Peñalvo, F. J. (2016). A brief introduction to TACCLE 3—coding european project. *Computers in education (SIIE), international symposium on IEEE* (pp. 1–4). .

Ginat, D. (2004). On novice loop boundaries and range conceptions. *Computer Science Education, 14*(3), 165–181.

Google for Education (2018). *Exploring computational thinking.* Retrieved from https://www.google.com/edu/resources/programs/exploring-computational-thinking/.

Gouws, L. A., Bradshaw, K., & Wentworth, P. (2013). Computational thinking in

educational activities: An evaluation of the educational game light-bot. *Proceedings of the 18th ACM conference on innovation and technology in computer science education* (pp. 10–15). .

Grover, S., & Pea, R. (2013). Computational thinking in K–12. A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

Heintz, F., Mannila, L., & Färnqvist, T. (2016). A review of models for introducing computational thinking, computer science and computing in K-12 education. *Frontiers in education Conference (FIE), 2016* (pp. 1–9). IEEE.

INTEF (2017). *Marco común de Competencia digital docente.* https://intef.es/Blog/marco-comun-de-competencia-digital-docente-septiembre-2017/.

ISTE (2015). *CT leadership toolkit.* Available at: https://www.iste.org/docs/ctdocuments/ct-leadership-toolkit.pdf%3fsfvrsn%bc;4.

Jiménez-Peris, R., Pareja-Flores, C., Patiño-Martínez, M., & Velázquez-Iturbide, J.Á. (1997). The locker metaphor to teach dynamic memory. *ACM SIGCSE bulletin: Vol. 29*, (pp. 169–173). ACM No. 1.

Jovanov, M., Stankov, E., Mihova, M., Ristov, S., & Gusev, M. (2016). Computing as a new compulsory subject in the Macedonian primary schools curriculum. *Global engineering education conference (EDUCON), 2016 IEEE* (pp. 680–685). IEEE.

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code. Org. *Computers in Human Behavior, 52*, 200–210.

Kazakoff, E. R., Sullivan, A., & Bers, M. U. (2013). The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education, 41*, 245–255.

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the Computational Thinking Scales (CTS). *Computers in Human Behavior, 72*, 558–569.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005). A study of the difficulties of novice programmers. *ACM SIGCSE bulletin: Vol. 37*, (pp. 14–18). no. 3.

Lakoff, G., & Johnson, M. (2008). *Metaphors we live by.* University of Chicago press.

Lee, E., Kafai, Y. B., Vasudevan, V., & Davis, R. L. (2014). Playing in the arcade: Designing tangible interfaces with MaKey MaKey for scratch games. *Playful user interfaces* (pp. 277–292). Springer Singapore.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J.,., & Werner, L. (2011). Computational thinking for youth in practice. *Acm Inroads, 2*(1), 32–37.

Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletings, 41*(1), 260e264.

Margulieux, L. E., Catrambone, R., & Guzdial, M. (2016). Employing subgoals in computer programming education. *Computer Science Education, 26*(1), 44–67. https://doi.org/10.1080/08993408.2016.1144429.

Special issue II on computer science education in K-12 schools. R. McCartney (Ed.). *Transactions on Computing Education, ACM, 14*, 2.

Special issue on computing education in (K-12) schools. Transactions on computing education. R. McCartney, & J. Tenenberg (Eds.). *ACM, 14*, 2.

Milner, W. W. (2010). A broken metaphor in Java. *ACM SIGCSE Bulletings, 41*(4), 76–77.

Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., & Lahmine, S. (2015). Learning basic programming concepts by creating games with scratch programming environment. *Procedia-Social and Behavioral Sciences, 191*, 1479–1482.

Papadakis, S., Kalogiannakis, M., & Zaranis, N. (2016). Developing fundamental programming concepts and computational thinking with ScratchJr in preschool education: A case study. *International Journal of Mobile Learning and Organisation, 10*(3), 187–202.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* New York, NY: Basic Books.

Paris, N. A., & Glynn, S. M. (2004). Elaborate analogies in science text: Tools for enhancing preservice teachers' knowledge and attitudes. *Contemporary Educational Psychology, 29*(3), 230–247.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*(2), 137–168.

Pérez-Marín, D., Hijón-Neira, R., & Martín-Lope, M. (2018). A Methodology proposal based on metaphors to teach programming to children. *IEEE Revista Iberoamericana de Tecnologías del Aprendizaje, 13*(1), 46–53.

Putnam, R. T., Sleeman, D., Baxter, J. A., & Kuspa, L. K. (1986). A summary of misconceptions of high school Basic programmers. *Journal of Educational Computing Research, 2*(4), 459–472.

Resnick, M. (1996). New paradigms for computing, new paradigms for thinking. In Y. InKafai, & M. Resnick (Eds.). *Constructionism in practice: Designing, thinking, and learning in a digital world.* Mahwah, NJ: Erlbaum.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60–67.

Rodríguez Diéguez, J. L. (1988). Las metáforas en la enseñanza. Enseñanza & Teaching. *Revista interuniversitaria de didáctica* (pp. 223–240). Universidad de Salamanca 6.

Román-González, M. (2015). Computational thinking test: Design guidelines and content validation. *Proceedings of EDULEARN15 conference* (pp. 2436–2444). .

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. *Computers in Human Behavior, 72*, 678–691. https://doi.org/10.1016/j.chb.2016.08.047.

Rosenthal, R. (1991). *Meta-analytic procedures for social research* (2nd ed.). Newbury Park, CA: Sage.

Sanford, J. P., Tietz, A., Farooq, S., Guyer, S., & Shapiro, R. B. (2014). Metaphors we teach by. *Proceedings of the 45th ACM technical symposium on Computer science education* (pp. 585–590). ACM.

Seoane-Pardo, A. M. (2018). Computational thinking between philosophy and STEM. *Programming decision making applied to the behaviour of "moral machines" in ethical values classroom*IEEE-RITAhttps://doi.org/10.1109/RITA.2018.2809940.

Seppälä, O., Malmi, L., & Korhonen, A. (2006). "Observations on student misconceptions—a case study of the Build–Heap Algorithm". *Computer Science Education, 16*(3), 241–255.

Sović, A., Jagušt, T., & Seršić, D. (2014). How to teach basic university-level programming concepts to first graders? *Integrated STEM education conference (ISEC), 2014 IEEE* (pp. 1–6). IEEE.

Strawhacker, A., Portelance, D., Lee, M., & Bers, M. (2015). *Designing tools for developing minds: The role of child development in educational technology. IDC 2015 workshop.* Available on-line at: http://everychildacoder.org.uk/wp-content/uploads/2015/05/Strawhacker_et_al_final.pdf Last visit: November 23rd, 2017 .

Thomas, G. P., & McRobbie, C. J. (2001). Using a metaphor for learning to improve students' metacognition in the chemistry classroom. *Journal of Research in Science Teaching, 38*(2), 222–259.

Vico, F. (2017). *El niño que no programe tendrá un hándicap como hoy lo tiene el que no entiende inglés. Entrevistas Educación 3.0.* https://www.educaciontrespuntocero.com/entrevistas/francisco-j-vico-programacion/59063.html.

Wing, J. M. (2006a). Computational thinking. *Communications of the ACM, 49*(3), 33–35.

Wing, J. M. (2006b). Computational thinking. *Communications of the ACM, 49*(3), 33–35.

Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of Mathematical Physical and Engineering Sciences, 366*, 3717–3725.

Wing, J. (2011). Research notebook: Computational thinking— what and why. [On line]. Available: *The link magazine, Spring*Pittsburgh: Carnegie Mellon University. Retrieved from http://link.cs.cmu.edu/article.php?a=600.

Wing, J. M. (2016). *Computational thinking, 10 years later.* http://www.microsoft.com/en-us/research/blog/computational-thinking-10-years-later.

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2016). Expanding computer science education in schools: Understanding teacher experiences and challenges. *Computer Science Education,* 1–20.

Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., & Korb, J. T. (2011). Introducing computational thinking in education courses. *Proceedings of ACM special interest group on computer science education, dallas, TX*.