# Computational Thinking and User Interfaces: A Systematic Review

Sara Rijo-García, Eduardo Segredo, *Member, IEEE*, and Coromoto León

***Abstract*—*Contribution:* This document presents a systematic bibliographic review that demonstrates the need to conduct research on how the user experience impacts the development of computational thinking.**

***Background:* In the field of computer science, computational thinking is defined as a method that enhances problem-solving skills, system design, and human behavior understanding. Over the last few decades, several tools have been proposed for the development of computational thinking skills; however, there is no area of study that evaluates the implications or the impact that these types of platforms have on users belonging to any knowledge area.**

***Research Question:* Do user interfaces influence the development of computational thinking skills?**

***Methodology:* To address this issue, a systematic review of the literature was conducted using the preferred reporting items for systematic reviews and meta-analyses (PRISMA) methodology for analyzing and evaluating scientific publications.**

***Findings:* The results show that despite the dearth of literature on the subject, the specific design of a user interface has a significant impact on the development of computational thinking. Bearing the above in mind, it is necessary to conduct research that delves more deeply into the effects caused by the technologies that are used to develop computational thinking, this being a line of research that is worthy of consideration.**

***Index Terms*—Computational thinking, human–computer interaction, preferred reporting items for systematic reviews and meta-analyses (PRISMA), survey, systematic review, usability, user experience, user interface, visual programming.**

## I. Introduction

THINKING is a cognitive process that relies on learning and previous experience, and is responsible for generating ideas and concepts or understanding situations. This process entails a set of skills that human beings develop throughout their lives and that can be promoted by applying different models of thought. In the context of computer science, there are different ways of thinking—mathematical thinking, critical thinking, algorithmic thinking, design thinking, computational thinking and others [1]—each one with cognitive processes that are not mutually exclusive and that can be used to solve a given problem.

Manuscript received 20 April 2021; revised 5 December 2021 and 28 February 2022; accepted 10 March 2022. Date of publication 25 March 2022; date of current version 28 October 2022. This work was supported in part by the Cabildo de Tenerife. Fundación General de la Universidad de La Laguna through the "Piens@ Computacion@ULLmente" project (REF 21120050). Programa educativo para el fomento del pensamiento computacional a través de la realizacón de actividades que permitan su desarrollo y su inclusión en el currículo" (Educational program to encourage computational thinking by engaging in activities that allow for its development and its inclusion in the curriculum). *(Corresponding author: Sara Rijo-García.)*

The authors are with the Departamento de Ingeniería Informática y de Sistemas, Universidad de La Laguna, 38200 San Cristóbal de La Laguna, Spain (e-mail: srijogar@ull.edu.es; esegredo@ull.edu.es; cleon@ull.edu.es).

Digital Object Identifier 10.1109/TE.2022.3159765

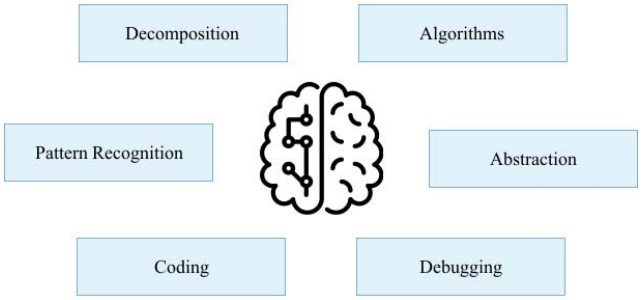


Fig. 1. Computational thinking skills.

Specifically, computational thinking shares certain similarities with other models of thinking, since it can be regarded as a technique that includes the skills involved in problem solving, systems design and understanding human behavior [2].

Based on these characteristics, Aho [3] proposed computational thinking as a research method to generate new, underlying computing models that are suitable for formulating problems. Elsewhere, Alan Perlis promoted the idea that programming could serve as a mental tool to understand and solve any problem at the university level [4]; while Papert, a pioneer of constructionist learning, introduced the use of technology in teaching children, arguing that programming could be taken as an active learning process based on understanding a problem in a practical way, providing students with a way of thinking based on their own learning [5]. But it was really at the beginning of this century that computational thinking came into its own. Wing [2] stressed the importance of developing computational thinking skills by focusing on the abilities of students, and not only on mastering certain programming languages. Later, Zapata-Ros [6], building on the concepts of Papert and Wing, consolidated the idea that using learning environments from the earliest educational stages fosters the skills involved in computational thinking. It may thus be said that computational thinking is divided into six skills (see Fig. 1) as follows.

1) *Decomposition:* Divide the problem into smaller problems.
2) *Pattern Recognition:* Find similarities inside or outside the problem situation.

3) *Abstraction:* Simplify a complex problem by focusing only on the relevant details.
4) *Algorithm:* Define the steps in order needed to solve the problem.
5) *Coding:* Develop a program in a language that the computer understands based on the algorithm.
6) *Debugging:* Correct errors or improve the program.

Since then, multiple tools have been developed to promote and enhance computational thinking skills and algorithmic thinking. These tools offer a series of options that allow the user to carry out a plethora of actions to solve a given problem; as a result, this could at first lead to a steep learning curve due to the complexity of the interface. Consequently, having an adequately designed user interface allows the student to focus on the information and the task to be completed in order to satisfy their objectives [7], while a confusing and inefficient design makes it difficult for people to do work and leads to more mistakes, causing frustration and stress for the user [8]. Because of this, user interfaces should be user friendly, easy to understand, and they should satisfy the needs of users while making their tasks easier [7]. Usability [9], along with user experience, is the key point when designing and developing accessible interfaces.

The user interface is the means by which people can communicate and interact with a computer system. There are different categories of user interfaces; however, depending on how the user interacts with the system, a distinction can be made between command line interface, graphical user interface, natural user interface, voice user interface, tangible user interface, and others [10].

The design of the user interface must make the interaction between the user and system efficient and effective, which is why the design should be developed using quality standards so as to optimize usability and the user experience. It is thus highly recommended to follow user-centered design perspectives when developing interfaces [11], [12].

For these reasons, it is important that environments for learning programming exhibit a friendly interface that promotes aspects of computational thinking in students in early stages of education [13] since, as noted in the previous paragraph, if the interface requires a high cognitive load [14], users could have a negative experience with the activity, which could affect student learning [15].

This article seeks to present a systematic study of the literature in order to undertake research aimed at determining how the user experience, as it relates to the interfaces used, impacts the development of the skills that computational thinking provides.

## II. Objective

Over the last few decades, a large amount of research has been conducted aimed primarily at defining computational thinking, developing different tools to promote these skills, and describing methodologies to apply in curricula at various educational stages. However, there are still large empirical gaps involving the usability and efficiency of the platforms used to develop and promote computational thinking skills.

TABLE I
Selected Keywords

| Importance | Keywords | Abbrev. |
|---|---|---|
| First-order keywords | Computational Thinking | CT |
| | User Interface | UI |
| | User Experience | UX |
| | Usability | |
| | Human-Computer Interaction | HCI |
| Second-order keywords | Thinking | |
| | Models of Thinking | |
| | Critical Thinking | |
| | Algorithmic Thinking | |
| | Design Thinking | |
| | STEM Education | |
| | Visual Programming Languages | VPL |
| | Visual Programming Environment | VPE |
| | Initial Learning Environments | ILE |

Thus, the main objective of this document is to conduct a systematic analysis of the literature to see if there are articles that show how the different user interfaces of these tools influence the development of computational thinking skills. The authors' intention is to identify the results that exist in the field of study that combines computer science, human–computer interaction and education, in order to determine the advantages and shortcomings in the design and development of platforms in the field of computational thinking, and establish a line of research in this field.

## III. Methodology

To ascertain this state of the art, a systematic review was conducted that relied on the "preferred reporting items for systematic reviews and meta-analyses" (PRISMA) [16] methodology for analyzing and evaluating scientific publications. The first step was to establish the research question in order to frame the field of study. The goal is to study the impact of user interfaces and how they affect the experience of users who employ technology to develop the skills that are provided by computational thinking. This allowed us to posit the following question:

> Do user interfaces influence the development of computational thinking skills?

Table I defines the search terms, as well as their possible synonyms and abbreviations. Once the set of keywords was determined, the databases that were best suited to this study were selected, which were as follows.

1) Association for Computing Machinery (ACM) Digital Library (https://dl.acm.org/).
2) Digital Bibliography and Library Project (DBLP) Computer Science Bibliography (https://dblp.org/).
3) Institute of Electrical and Electronics Engineers (IEEE) Xplore (https://ieeexplore.ieee.org/Xplore/home.jsp).
4) ScienceDirect (https://www.sciencedirect.com/).
5) Scopus (https://www.scopus.com/home.uri).
6) Web of Science (https://www.webofscience.com/wos/alldb/basic-search).

The next step was to define the query to carry out in the different databases

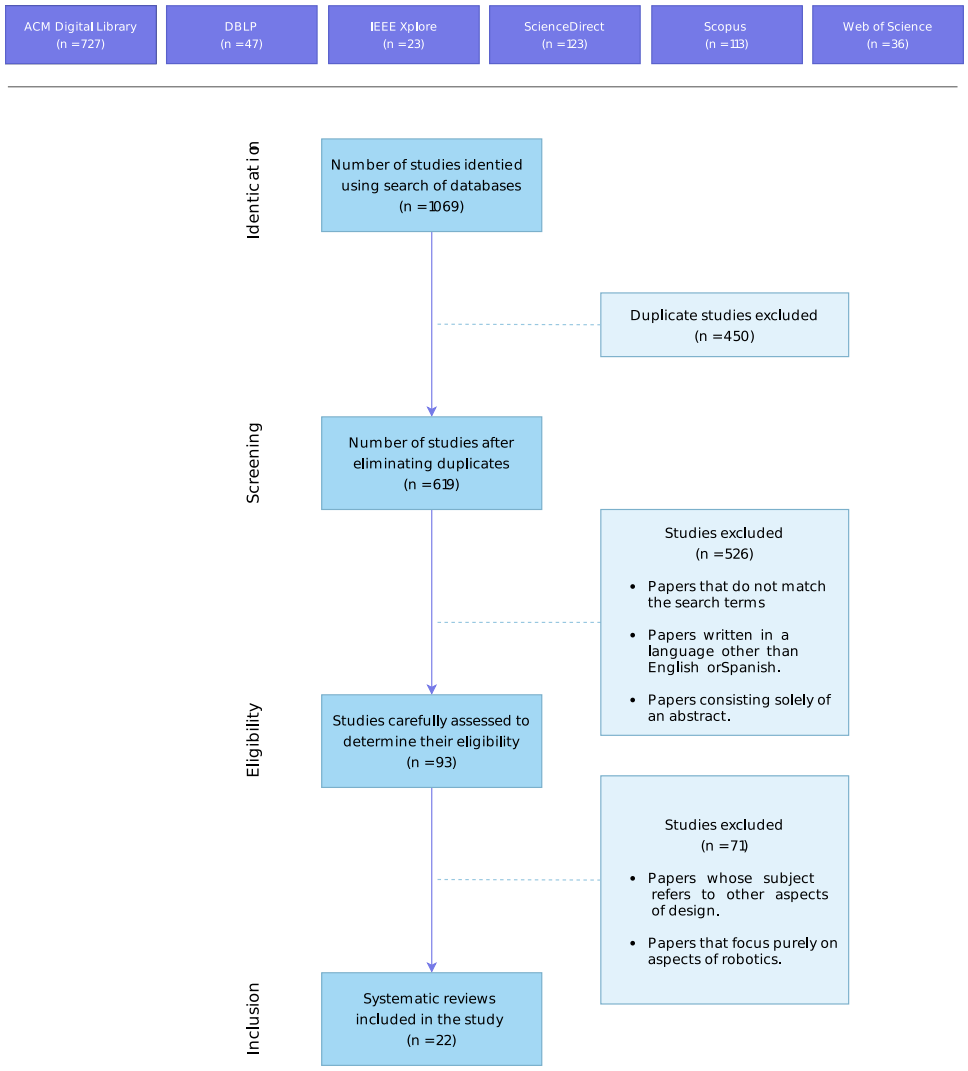


Fig. 2. Flow diagram of the PRISMA methodology.

```
("Models of thinking" OR "Computational
    Thinking") AND ("User Interface" OR "UI"
    OR "User Experience" OR "UX") AND ("Visual
    Programming Languages" OR "VPL" OR "STEM"
     OR "Education" OR "Human Computer
    Interaction" OR "HCI" OR *Thinking).
```

Finally, exclusion and inclusion criteria were established to determine the quality of the articles to be studied.

1) *Exclusion Criteria:*
   a) Duplicate papers.
   b) Papers that do not match the search terms.
   c) Papers whose subject refers to other aspects of design.
   d) Papers that focus purely on aspects of robotics.
   e) Papers written in a language other than English or Spanish.
   f) Papers consisting solely of an abstract.
2) *Inclusion Criteria:*
   a) No time restriction is applied.
   b) Papers written in Spanish or English.
   c) Peer-reviewed papers.
   d) Papers from journals.
   e) Papers that may describe an experiment or study with proven results.
   f) Papers whose subject is related to computational thinking and user interfaces.

### *A. Search, Screening, and Selection Process Carried Out in the Different Stages*

After performing different searches in the databases mentioned above, a total of 1069 records were obtained, of which 450 were duplicate studies. This resulted in a total of 619 studies after eliminating duplicates. During the screening phase, 407 papers were excluded based on the title and keywords, and 119 based on the abstract. In the eligibility phase, of the remaining 93 studies, a total of 71 studies that did not satisfy the inclusion criteria were excluded. In the end, 22 papers were found that were relevant to this study (see Fig. 2).

## IV. RESULTS

Once the screening phase was completed, 22 papers were selected in order to conduct a more in-depth analysis. The

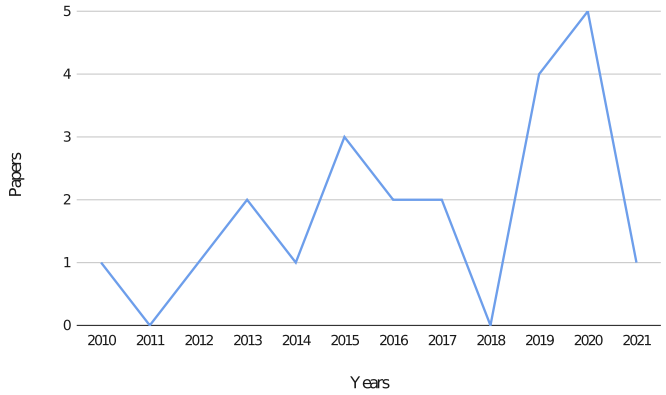


Fig. 3. Graph showing the selected papers grouped by year of publication.

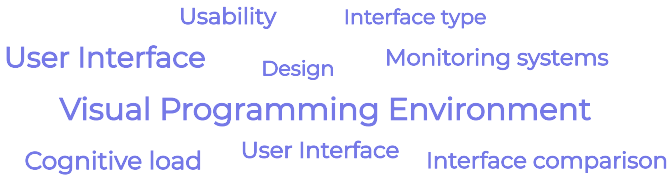


Fig. 4. Word cloud of the categories into which the papers analyzed have been classified.

resulting study period ranged from 2010 to 2021, and can thus be regarded as involving novel and current areas of research. It should be noted that the number of papers on the subject has been increasing over the years, and most of the articles selected for the study are from recent years (see Fig. 3).

The papers selected are those that best adapt to the research question that the authors are interested in solving, and they can be classified into various categories, which are not mutually exclusive, and where the same article could be included in several categories (see Fig. 4).

The results of this analysis are presented in the sections that follow.

*Machines for Thinking [13]:* The article focuses on how learning environments for programming should be appealing and interesting in order to promote aspects of computational thinking in students in the early stages of education.

It lays out a background by taking a tour of the first learning tools for programming, which later transformed into environments designed, especially, for young students. The precursor was provided by Papert and the Logo language.

The authors believe that this type of platform incorporates both an intellectual relationship with the learning that takes place, and a social relationship with the users, where each system is capable of presenting a theory for teaching programming.

These types of tools offer great advantages due to the ubiquitousness of the computers present in many of the classrooms at a large number of schools compared to other, higher cost alternatives, as well as due to the communities that have been created to share the educational experiences of students and teachers alike.

With the above in mind, it could be said that environments for learning programming should be appealing and interesting so as to encourage aspects of computational thinking. In this case, an environment is deemed to be appealing and interesting based on its user interface. If said interface is not intuitive or eye-catching, the environment may not be viewed in the same light, which could detract from the positive aspects related to the user experience on the platform.

*App Inventor for Android (Report From a Summer Camp [17]):* This article details the experience of using App Inventor in a summer camp with high school students. The author provides a detailed description of the design of these camps, including the process of selecting and configuring an Android device. Furthermore, an evaluation process was conducted with the students through surveys to determine the effects of this type of tool compared to other, less intangible, tools. In addition to the experience with the platform, the interesting thing about this article is that it presents the advantages of this tool compared to others, as well as the improvements that the platform would need to increase its usability and improve the user experience.

Elsewhere, Roy highlighted as advantages of App Inventor its fusion of the best tangible and intangible interfaces, it complements the learning experience of users who have already used Scratch or Alice, and for computer science students, it can serve as a basis for the later study of programming languages such as Java.

Improvements include social interaction and sharing between users, developing an easily accessible media library like Scratch, improving the design of the user interface with a single window so that elements are always visible, and improving the usability of the implementation.

*Interfaces for Thinkers (Computer Input Capabilities That Support Inferential Reasoning [14]):* This article considers the implications of designing effective computational thinking tools. One of the fundamental goals of this research is to determine if inferential thinking deteriorates or improves depending on the cognitive load of the interface used.

To achieve an optimal design of the interfaces of educational tools, one where learning prevails, it is essential for said interface to have a low cognitive load. To this end, Sharon Oviatt, the author of this research, did a comparative study contrasting the inference that results from using different interfaces.

This study showed that inferential precision is affected by interfaces that facilitate greater distractions, since the user cannot focus on solving the problem at hand.

Currently, the use of digital tools is widespread in all areas of society. It is thus of great importance to understand how the design of these tools impacts certain aspects of human cognition. The long-term goal of this research is to design computer interfaces that act as thinking tools to improve learning.

To conclude, the authors note that user interfaces that are not designed with the cognitive load they place on the student in mind make the user unable to focus on the solution of the problem at hand, thus potentially frustrating the learning process.

*TUI, GUI, HUI (Is a Bimodal Interface Truly Worth the Sum of Its Parts? [18]):* This article conducts a comparative study of three types of interfaces dedicated to the development

of computational thinking: 1) tangible interfaces; 2) hybrid interfaces; and 3) graphic interfaces. The study was carried out on a sample of boys and girls between five and six years old. It yielded promising results for tangible interfaces, as they help to enhance computational thinking skills, although graphic interfaces obtained higher scores in the evaluation of repetition structures.

The statistics on the results show that the group of tangible interfaces had a higher proportion of correct answers in the majority of programming tasks, whereas the graphical interfaces obtained the highest score when evaluating repeating structures.

As a final finding, the strategies used by students to solve problems are closely related to the learning interface used, meaning it is essential to conduct research in this field in order to determine how these technologies impact computational thinking learning.

*Students' Experiences From the Use of MIT App Inventor in Classroom [19]:* In this article, Perdikuri described how the App Inventor works and its use as a means of developing computational skills among secondary school students.

In student evaluations of their experience with the tool, more than 50% of the students said that App Inventor provides a fairly usable development environment. In noting the significant aspects of this tool, students deemed it important to develop mobile applications that are simple and accessible to everyone, a Web-based platform and a user-friendly visual environment for the development of user interfaces. The students also noted the greater complexity of the block editor view.

Despite having used a small study sample, and in the absence of studies involving larger groups, the author was able to state that App Inventor is a good tool for developing computational thinking skills since, due to the visual nature of the platform, it lets users focus more on solving the problems and less on the language syntax, which minimizes the programming learning process.

As a line of future work, the author mentions conducting a comparative study of the App Inventor and other programming environments, which are used to introduce programming to high school students.

In this article, the author addresses the need to use visual programming environments with an interface that helps users develop computer and computational thinking skills, by making it possible for users to focus on solving the problem in question, and not just on the code.

*Entry (Visual Programming to Enhance Children's Computational Thinking [20]):* This article introduces Entry, a visual programming platform based on HTML5 that encourages the development of computational thinking in learners.

It features a simple and intuitive block-based interface. Through this friendly environment, it allows the user to engage in problem-based learning that spans the basic principles of programming. Furthermore, since it is Web-based, it is adaptive and can be easily used on mobile devices.

In future research, the authors propose conducting qualitative surveys to study the usability of the interface and apply specific measures to adapt the improvements to the user interface.

*Measuring the Usability and Capability of App Inventor to Create Mobile Applications [21]:* The goal of this study is to evaluate the usability of App Inventor to develop applications for mobile devices. To this end, the environment must be intuitive, usable and functional so that basic users can create simple applications, and higher level users can develop more advanced applications.

In order to carry out this research, the applications from a sample of 5228 resources were categorized and filtered. The findings indicate that the interface is limited by the learning resources, and that there is thus a relationship between the usability of the interfaces and the tutorials proposed to carry out the activity.

*Floors and Flexibility (Designing a Programming Environment for 4th-6th Grade Classrooms [22]):* Several studies focused on Human–Computer Interaction show the differences between how children interact compared to adults. It is these differences that must therefore be kept in mind when designing visual tools for programming, since the development interface plays a fundamental role, as it aims to promote change in the students' roles as they transition from being mere users to developers. As a result, this paper argues for the need to have a tool that can be adapted to different educational levels, making the user focus on the project to be developed without becoming lost learning the interface.

To do this, the authors of this article developed an environment that can reveal sections of the interface as the student progresses in their learning, so as not to cognitively saturate the student. As an alternative, the environment can be made completely visible so that users can develop their creativity by engaging in unscripted projects.

In developing this platform, the authors established the following design principles.

1) Support multiple task types.
2) Require age-appropriate content.
3) Include an age-appropriate interface.
4) Aid the project developers.

These design principles rely on existing visual programming languages based on blocks, as well as on research during the pilot implementation phase. Likewise, the authors identified inconsistencies in the designs of programming environments, such as Scratch, ScratchJr, and Blockly. This led them to carry out a comparison applying the design principles outlined above, and which they also used to develop their own environment based on the strengths of each of the interfaces studied. With the above in mind, the design criteria established by the authors yielded an improved user experience and encouraged computational thinking, specifically in fourth- and sixth-grade students.

In the future, the authors plan to conduct further studies, including the benefits provided by a visual development environment and how learning in these environments affects educators, since many teachers do not usually have previous experience teaching computational thinking.

This article clearly shows the need to have programming environments that are tailored to cognitive skills, since many

block-based tools have features in their user interfaces that are too complex and not suited to the development of students ranging in age from nine to eleven.

*Using Computational Thinking Patterns to Scaffold Program Design in Introductory Programming Course [23]:* Chang, the author of this research, has created a system that monitors student development in a block-based programming environment. This system covers the deficiencies present in certain user interfaces by suggesting to the student a snippet of code that can help them, such that the user can progress in their learning without obsessing over the multitude of options that these interfaces offer.

This could be one possible solution in terms of the development of computational thinking by relying on block-based programming environments, since it would optimize the user interface while also reducing its cognitive load.

*Can Students Design Software? The Answer Is More Complex Than You Think [24]:* The author of this article defined the design as a series of software elements, as well as the interaction between them. Hu argued that software design should be an iterative process that improves the final design of a product.

Therefore, a quality design must be able to adapt to changes in the initial requirements, as well as to other possible modifications. This is why practitioners using agile software development methodologies may believe that design is not only highly iterative, but emergent. Then, only coding, test execution and code refactoring reveal the correct functioning of a good design.

In conclusion, the author points out that in order to obtain stable, adaptable and durable designs over time, the teaching of design must be improved, such that any student finishing a computer science program should be able to design optimal software.

*Employing Retention of Flow to Improve Online Tutorials [15]:* It is imperative that designers and developers create optimal platforms to promote learning in computer science, since if users have a negative experience with the activity, it could have effects that are counterproductive to the specified objective. In order to identify those parts where users normally leave the activity, methods are needed to identify these points so they can be subsequently improved.

This article maintains that the development of platforms has to adhere to design norms, such as, for example, the clarity and simplicity of the interface, in order to allow for a proper user–interface interaction in order to minimize user error. This thus promotes the learning of computer science since, if users have negative experiences with the activity in question, effects contrary to the defined objective could result. It is because of this that in this case, the researchers created a tool that can be used to measure the flow retention of students. This allows identifying the cognitive loads in order to improve the design of the user interface. Keeping the above in mind, using this type of tool could lead to qualitative improvements in both the user interface and user experience, enhancing computational thinking.

*The Impact of User Interface on Young Children's Computational Thinking [25]:* This article studies the impact of tangible interfaces versus graphics on the performance of computational thinking skills in boys and girls between the ages of five and seven.

The researchers focused specifically on ScratchJr as a graphical interface, and the KIBO Robot as the tangible interface. The results showed significant differences between the two types of interface; however, for this sample, the group that worked with the KIBO robot obtained better scores than the group that worked with ScratchJr, especially in the sequencing and debugging activities. This may be due to the nature of the interfaces proposed. Since the KIBO Robot is a tangible interface, it allows children to use their own hands to understand how the actions associated with the blocks are translated into the robot's movements in a physical space. By contrast, ScratchJr, which is a graphical interface shown on a screen, could pose some additional complexities or increase the level of distractions, especially when one is unfamiliar with the development environment.

In conclusion, this study provides evidence on the impact that both interface types have on student learning, which shows that the type of interface presented to the student influences, in one way or another, the experiences of the children and, thus, on the development of skills associated with computational thinking. Because of this, as the authors of the research assert, it is important to implement technologies that are tailored to the different educational stages so as to promote the proper development of computational thinking skills.

*Exploring Factors Influencing the Acceptance of Visual Programming Environment Among Boys and Girls in Primary Schools [26]:* This study adapts the "Technology Acceptance Model" (TAM), by Davis, Bagozzi and Warshaw, to study the effect that visual programming environments have on primary school students. The model is based on four principles: perceived utility, perceived ease of use, attitude and behavioral intention.

The results of the research show that students tended to perceive the utility of the environment rather than the ease of use. It also determined that outside assistance is necessary for the perceived ease of use. These results thus indicate that there is a link between external assistance and the ease of use perceived by students.

*The Computational Puzzle Design Framework (A Design Guide for Games Teaching Computational Thinking [27]):* After a systematic investigation, the authors of this study conclude that there is nothing in the literature on how to design and develop games to teach computational thinking.

This article tries to address the unknown that arises when developing this type of platform in an efficient and effective way, and presents a design framework for game development, which the authors of this research put into practice by applying said framework to redesign a game that focuses on learning computational thinking.

As a result, a common framework should be established for the design and development of games that teaches computational thinking and that also allows the researcher to measure the effect that these interface have on its development, and determine the objective improvements that could be made.

As future work, the researchers propose testing this design and development framework.

*Fostering Computational Thinking Through Collaborative Game-Based Learning [28]:* This study tries to determine if the design of a game system can be used to help teach computational thinking in a way that is fun and entertaining. To argue this hypothesis, the researchers in this study developed a tangible programming environment using mobile devices.

The most significant difference with other environments is that the system developed had a greater degree of abstraction compared to other environments, such as Scratch. The experiment yielded positive results, suggesting future research where certain improvements will be applied.

*Computational Thinking With the Web Crowd Using CodeMapper [29]:* In this article, a new programming platform, called CodeMapper, is presented. It can build computational logic in separate modules and aggregate them to create complex modules, so that students can focus first on programming logic and then on code development.

CodeMapper offers a visual interface that allows students to stay in a conceptual realm, similar to block-based programming environments, and also to write software in a language, such as Java, C++, or Python.

The platform is developed in ASP.NET Core 2.0 and React due to their flexibility, as it allows for incremental design. The result is a dynamic user experience and interaction.

In conclusion, the authors emphasize that CodeMapper is part of an overall system called MindReader, but can stand alone as a smart Web application with great potential for teaching computational thinking.

*A Natural User Interface Implementation for an Interactive Learning Environment [30]:* This article shows a development environment called Create and Play based on natural user interfaces. The TAM was used to study user impressions. In addition, the Interface Style was included as an external variable of the research because, according to previous studies, the use of a given system can be affected by the style of the interface used, thus influencing the users' attitude when using the platform.

After conducting an empirical study, the authors show that the type of interface influences the ease of use of the technology and is related to the usefulness and enjoyment of the platform. Therefore, the authors claim that the use of natural user interfaces in computational thinking provides additional motivation for secondary school students.

*BlocklyScript (Design and Pilot Evaluation of an RPG Platform Game for Cultivating Computational Thinking Skills to Young Students [31]):* Karakasis and Xinogalos, the authors of this article, present a new educational game called BlocklyScript, which aims to teach basic programming concepts, algorithm design and error correction, skills that are intrinsic to computational thinking.

To provide an improved gaming experience for the user, the authors relied on design norms focused on usability, multimodality and entertainment. This translates into an implementation of a gaming environment that is easy to use and navigate and is compatible with various browsers.

According to the surveys conducted during the study, the game is usable, the design of the game is appealing, the colors are representative and the game provides guided learning. These are important characteristics in the design and development of user interfaces. As for the user experience, parameters, such as confidence, challenge, satisfaction, engagement, and relevance were evaluated.

On the other hand, that students are expected to easily grasp the basic concepts of computational thinking as they play with the BlocklyScript platform. However, due to the current COVID-19 pandemic, new teaching models have been born and as future research, the authors propose improving the design and features of BlocklyScript so that it can be used in nonface-to-face education.

Finally, there is evidence that the user interface does affect the outcome of these abilities, since this platform was designed taking into consideration user-centered design norms. As a result, the authors were able to develop a platform with an appealing design that focuses on usability while also promoting computational thinking skills.

*Pixasso (A Development Stage-Based Learning Application for Children [32]):* In this article, Nandan *et al.* presented a tool to teach programming by coloring the pixels of an image. In terms of the user interface and user experience, this application was developed employing the user-centered design paradigm (in this case, child-centered design), which aims to improve the proposed skills in computational thinking and encourage computer science education at an early age.

The Pixasso application was designed to be adaptive and to let students select the difficulty level for the various skills. To validate the design of the application, the authors relied on the Developmentally Situated Design (DSD) card, adding cards applicable to the computational thinking use case, i.e., problem solving, attention and instruction. They also applied the Touchscreen Interaction Design Recommendations for Children framework to properly develop the application.

As is apparent, this article provides a clear example of how the user interfaces of tools for learning programming can affect the cognitive development of the user when attempting to solve a problem. In this case, the researchers employed a user-centered design that was validated using DSD cards. By doing so, the authors were able to identify potential problems in the tool and apply solutions to improve their design.

*Enhancing Computational Thinking Capability of Preschool Children by Game-based Smart Toys [33]:* This study proposes a system based on tangible user interfaces, using Arduino as the basis for assembling robot cars and colored cards for developing applications. The aim of the study is to investigate different teaching approaches for learning computational logic and programming concepts.

The article builds on previous research that showed that there are no major differences in learning usage between graphical and tangible interfaces; however, this study is based on the hypothesis that users will engage more actively with a tangible interface.

The learning difficulties were reduced and learner interest was increased thanks to facilitated entertainment scenarios and user-friendly interfaces, promoting the improvement of

computational thinking skills. As a result, the cognitive load associated with learning is lower, as students quickly become accustomed to the interactive interface.

*Comparing TUIs and GUIs for Primary School Programming [34]:* This study is part of broader research that aims to analyze how different user interfaces can support the development of computer skills in primary school students, which is why the authors of this article focus on comparing the use of graphical user interfaces and tangible user interfaces.

The results suggest that graphical user interfaces can be more effective for teaching the basic concepts of programming, while tangible user interfaces can be more effective for teaching computer science to younger students. Thus, both interfaces can be beneficial depending on the educational objective.

In this article, the authors consider the difference that exists between tangible and graphical user interfaces when assimilating programming concepts, with tangible interfaces being more popular among younger children, and graphical interfaces being easier for older children to use.

The results also show that the users of graphical interfaces exhibited significantly greater improvements than the users of tangible interfaces in terms of learning.

As a result, the type of interface used when teaching programming can influence the development of computational thinking skills.

*EEG-Based Cognitive Load Assessment in MATLAB GUI and Impact on Learning System [35]:* This research focuses on reducing the cognitive load for computational thinking-based learning. The aim is to build a graphical user interface for MATLAB software in order to instantly measure cognitive load based on the electroencephalograms of individuals. The user interface is easy to use, since it does not require programming knowledge in order to use it, and its objective is to determine the cognitive load experienced by the user when solving a given problem. According to the authors, this research could be applied in the field of education to determine the learning status of students.

## V. Discussion

Most of the results studied have been presented at international conferences and congresses, and can be categorized into three areas: 1) Computer Science; 2) Human–Computer Interaction; and 3) Education. The field of study of this research is a combination of these three areas.

In turn, this article presents a classification of the papers based on their content and the area of study that is considered herein, which were grouped into: visual programming environment, user interface, user experience, usability, cognitive load, comparison of interfaces, type of interface, design, and monitoring systems. The classification is shown in Fig. 5.

Despite the long history of the study of computational thinking, in reality, the existing literature on the usability and effectiveness of learning tools for programming is truly scarce. Based on the studies analyzed, the authors of this article determined that there is a significant gap both when it comes to designing and developing tools that encourage computational thinking and are suited to different educational levels, and to studying the impact that these platforms have in terms of the cognitive load that students can tolerate.

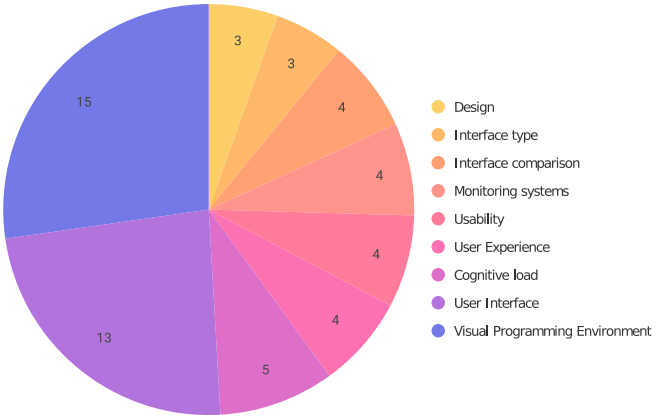


Fig. 5. Graph showing the categories into which the papers analyzed are classified.

This work shows that the type of interface used in platforms that promote computational thinking affects student learning, which makes it important to ascertain what types of interfaces are best suited to each educational stage. This makes it possible to determine how these tools will be presented to the user.

The authors' review of the literature yielded several types of interfaces: natural, tangible, graphic, and hybrid. All of them, depending on the degree of use, enhance computational thinking skills to some extent [14], [18], [25], [30], [34].

It is also important to consider the cognitive load imposed by these tools, since the user learning process could be frustrated, resulting in a negative experience and a counterproductive outcome [14], [15], [23], [33], [35].

As noted by Jiang *et al.*, since there is no design framework for developing this type of tool, it is difficult to establish a pattern that can be used to bridge or mitigate this gap [27]. Therefore, to provide a better user experience, it is necessary to follow certain design guidelines focusing on usability and multimodality [31], this being an iterative process that improves the final design of the product [24]. A proper evaluation of a software prototype is a fundamental part of the development process, as it provides useful information on usability and identifies potential problems so that the software can be improved later [36].

In terms of the design of graphical user interfaces, the developers of computational thinking platforms should use a user-centered approach in order to achieve tools that are intuitive, usable and appealing [13], [22], [31], [32]. As per the ISO 9241-210:2019 standard on the "Ergonomics of human–system interaction—Part 210: Human-centered design for interactive systems," six design principles are defined as follows.

1) The design is based upon an explicit understanding of users, tasks, and environments. It is important to know the target audience for which the tool is intended, the context, the needs and goals of the user,

as well as the tasks the user will carry out on the interface.
2) Users are involved throughout the design and development. This point suggests making the user a participant in the interface design process and learning their point of view, thus making it possible to design interfaces that are better suited to the user's needs.
3) The design is driven and refined by user-centered evaluation. The user's viewpoint is taken into account such that the user evaluates all the versions, from the start of the development phase until the final version.
4) The process is iterative. It is not a linear process; rather, the design is evaluated along the various phases.
5) The design addresses the whole user experience. Just as usability is an implicit part of this process, so must the user experience be.
6) The design team includes multidisciplinary skills and perspectives. This is a process that should involve every team member, not just designers or developers.

Various papers analyzed present a series of solutions that can address some of the problems exhibited by the current tools in terms of developing computational thinking.

These alternatives include the implementation of technology adapted to the different educational stages [22], [25], systems that measure the retention flow of students on the platform and that allow measuring the cognitive loads that are imposed on students [15], and assistive technologies that are activated when the user cannot find a solution to the problem [23].

In conclusion, and in response to the hypothesis proposed in this research, it may be deduced that user interfaces indeed affect the development of computational thinking skills. A more exhaustive research process would be beneficial in order to ascertain the impact that these technologies have on the development of intrinsic computational thinking skills. As analyzed in previous studies, there is a close relationship between solving a given problem and the learning interface used for this [13], [18], [21], [33], [37].

This work is the first step, and involved a study to systematically map the literature to see if there are any papers that conclude that user interfaces influence the development of computational thinking skills. As the results show, there are few studies in this field, meaning more work is required in this area of research to determine the extent to which each type of user interface is best suited to learning in the various stages of education, and to study the usability and user experience of these interfaces, the goal being to develop programming environments that are tailored to the user's cognitive skills.

## VI. Conclusion

Recent years have seen a great wave of research on computational thinking and its effect in the field of education. However, there are no mapping studies that consider the implications of the impact caused by the various tools that have been developed to promote computational thinking skills. This article presents a systematic review that relies on PRISMA as the analytical methodology. Twenty-two documents out of a total of 1069 were identified and filtered for further study. The result is an analysis of the state of the art involving the intersection between the areas of Computer Science, Human–Computer Interaction, and Education, the goal of which is to identify the scope of the research, trends and any existing gaps.

The results showed that despite the relative dearth of literature on the relationship between user interfaces and the development of computational thinking skills, it is safe to say that user interfaces do affect the development of these skills. There is also the need to conduct research that further analyzes the effect caused by the technologies employed in the development of these interfaces. This is thus a worthy area of endeavor for future research.

Over the course of this study, a series of more specific questions emerged as follows.

1) How do the interfaces of visual programming languages affect the development of computational thinking skills?
2) How does the design of a user interface influence the development of computational thinking skills?
3) How does the user experience involving visual programming tools affect the development of computational thinking?
4) What are the differences between the interfaces of visual programming languages?
5) What improvements can be applied to visual programming tools or platforms to make their designs more inclusive and accessible?
6) Is it possible to improve the design of a user interface by applying other thought models?

The answer to these questions will provide a starting point for future research.

## References

[1] C. Frauenberger and P. Purgathofer, "Ways of thinking in informatics," *Commun. ACM*, vol. 62, no. 7, pp. 58–64, Jun. 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3329674

[2] J. M. Wing, "Computational thinking," *Commun. ACM*, vol. 49, no. 3, p. 33, Mar. 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1118178.1118215

[3] A. V. Aho, "Computation and computational thinking," *Comput. J.*, vol. 55, no. 7, pp. 832–835, Jul. 2012. [Online]. Available: https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/bxs074

[4] M. Tedre and P. J. Denning, "The long quest for computational thinking," in *Proc. 16th Koli Calling Int. Conf. Comput. Educ. Res. Koli Calling '16*, Koli, Finland, 2016, pp. 120–129. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2999541.2999542

[5] I. Harel and S. Papert, "Software design as a learning environment," *Interact. Learn. Environ.*, vol. 1, no. 1, pp. 1–32, 1990. [Online]. Available: https://doi.org/10.1080/1049482900010102

[6] M. Zapata-Ros, "Pensamiento computacional: Una nueva alfabetización digital," *Revista de Educación a Distancia*, vol. 2015, no. 46, Oct. 2015. [Online]. Available: https://revistas.um.es/red/article/view/240321

[7] M. Ritter and C. Winterbottom, *UX for the Web: Build Websites for User Experience and Usability*. Birmingham, U.K.: Packt Publ., Sep. 2017.

[8] D. Stone, C. Jarrett, M. Woodroffe, and S. Minocha, *User Interface Design and Evaluation*, 1st ed. Amsterdam, Netherlands: Elsevier, 2005. [Online]. Available: https://www.elsevier.com/books/user-interface-design-and-evaluation/stone/978–0-12–088436-0

[9] NSAI Stand, Dublin, Ireland, ISO 9241–11:2018(en). "Ergonomics of Human-System Interaction–Part 11: Usability: Definitions and Concepts." 2018. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en

[10] F. J. M. Lopez, *Instalación y Actualización De Sistemas Operativos –UF0852*. Madrid, Spain: Editorial Paraninfo, 2017.

[11] S. Yadav and P. Chakraborty, "Human-computer interaction as an important aspect of software: A tutorial," in *Proc. IEEE Int. Conf. Comput. Power Commun. Technol. (GUCON)*, 2020, pp. 40–44, doi: 10.1109/GUCON48875.2020.9231155.

[12] M. Rauterberg, "How to measure and to quantify usability attributes of man-machine interfaces," in *Proc. 5th IEEE Int. Workshop Robot Human Commun. RO-MAN*, 1996, pp. 262–267.

[13] S. Fincher and I. Utting, "Machines for Thinking," *ACM Trans. Comput. Educ.*, vol. 10, no. 4, pp. 1–7, Nov. 2010. [Online]. Available: https://dl.acm.org/doi/10.1145/1868358.1868360

[14] S. L. Oviatt, "Interfaces for thinkers: Computer input capabilities that support inferential reasoning," in *Proc. 15th ACM Int. Conf. Multimodal Interact. ICMI*, Sydney, NSW, Australia, 2013, pp. 221–228. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2522848.2522849

[15] A. R. Basawapatna and A. Repenning, "Employing retention of flow to improve Online tutorials," in *Proc. ACM SIGCSE Tech. Symp. Comput. Sci. Educ.*, Seattle, WA, USA, Mar. 2017, pp. 63–68. [Online]. Available: https://dl.acm.org/doi/10.1145/3017680.3017799

[16] D. Moher, A. Liberati, J. Tetzlaff, D. G. Altman, and The PRISMA Group, "Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement," *PLoS Med.*, vol. 6, no. 7, Jul. 2009, Art. no. e1000097. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/19621072

[17] K. Roy, "App inventor for android: Report from a summer camp," in *Proc. 43rd ACM Tech. Symp. Comput. Sci. Educ.*, New York, NY, USA, 2012, pp. 283–288. [Online]. Available: https://doi.org/10.1145/2157136.2157222

[18] A. Strawhacker, A. Sullivan, and M. U. Bers, "TUI, GUI, HUI: Is a bimodal interface truly worth the sum of its parts?" in *Proc. 12th Int. Conf. Interact. Design Children - IDC*, New York, NY, USA, 2013, pp. 309–312. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2485760.2485825

[19] K. Perdikuri, "Students' experiences from the use of MIT app inventor in classroom," in *Proc. 18th Panhellenic Conf. Informat.*, New York, NY, USA, 2014, p. 1–6. [Online]. Available: https://doi.org/10.1145/2645791.2645835

[20] A. Han, J. Kim, and K. Wohn, "Entry: visual programming to enhance children's computational thinking," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput. Symp. Wearable Comput. (UbiComp)*, Osaka, Japan, 2015, pp. 73–76. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2800835.2800871

[21] B. Xie, I. Shabir, and H. Abelson, "Measuring the usability and capability of app inventor to create mobile applications," in *Proc. 3rd Int. Workshop Program. Mobile Touch (PROMOTO)*, Pittsburgh, PA, USA, 2015, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2824823.2824824

[22] C. Hill, H. A. Dwyer, T. Martinez, D. Harlow, and D. Franklin, "Floors and flexibility: Designing a programming environment for 4th-6th grade classrooms," in *Proc. 46th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, Kansas City, MO, USA, 2015, pp. 546–551. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2676723.2677275

[23] C.-K. Chang, "Using computational thinking patterns to scaffold program design in introductory programming course," in *Proc. 5th IIAI Int. Congr. Adv. Appl. Informat. (IIAI-AAI)*, Kumamoto, Japan, Jul. 2016, pp. 397–400. [Online]. Available: http://ieeexplore.ieee.org/document/7557641/

[24] C. Hu, "Can students design software? The answer is more complex than you think," in *Proc. Assoc. Comput. Mach.*, Feb. 2016, pp. 199–204.

[25] A. Pugnali, A. Sullivan, and M. U. Bers, "The Impact of User Interface on Young Children's Computational Thinking," *J. Inf. Tech. Educ. Innov. Pract.*, vol. 16, no. 1, pp. 171–193, 2017. [Online]. Available: https://www.informingscience.org/Publications/3768

[26] G. Cheng, "Exploring factors influencing the acceptance of visual programming environment among boys and girls in primary schools," *Comput. Human Behav.*, vol. 92, pp. 361–372, Mar. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0747563218305788

[27] X. Jiang, C. Harteveld, X. Huang, and A. Y. H. Fung, "The computational puzzle design framework: a design guide for games teaching computational thinking," in *Proc. 14th Int. Conf. Found. Digit. Games*, San Luis Obispo, CA, USA, Aug. 2019, pp. 1–11. [Online]. Available: https://dl.acm.org/doi/10.1145/3337722.3337768

[28] T. Turchi, D. Fogli, and A. Malizia, "Fostering computational thinking through collaborative game-based learning," *Multimedia Tools Appl.*, vol. 78, no. 10, pp. 13649–13673, May 2019. [Online]. Available: http://link.springer.com/10.1007/s11042–019-7229–9

[29] P. Vanvorce and H. M. Jamil, "Computational thinking with the Web crowd using codemapper," in *Proc. 34th ACM/SIGAPP Symp. Appl. Comput.*, New York, NY, USA, 2019, pp. 2532–2534. [Online]. Available: https://doi.org/10.1145/3297280.3298913

[30] M. L. Barron-Estrada, R. Zatarain-Cabada, and B. A. Cardenas-Sainz, "A natural user interface implementation for an interactive learning environment," in *Proc. IEEE 20th Int. Conf. Adv. Learn. Technol. (ICALT)*, 2020, pp. 341–343.

[31] C. Karakasis and S. Xinogalos, "Blocklyscript: Design and pilot evaluation of an RPG platform game for cultivating computational thinking skills to young students," *Informat. Educ.*, vol. 19, no. 4, pp. 641–668, Dec. 2020.

[32] V. Nandan, A. Spittlemeister, and F. Brubacher, "Pixasso: A development stage-based learning application for children," in *Proc. 7th ACM Conf. Learn. Scale*, New York, NY, USA, 2020, pp. 361–364. [Online]. Available: https://doi.org/10.1145/3386527.3406747

[33] S.-Y. Lin, S.-Y. Chien, C.-L. Hsiao, C.-H. Hsia, and K.-M. Chao, "Enhancing computational thinking capability of preschool children by game-based smart toys," *Electron. Commer. Res. Appl.*, vol. 44, Nov./Dec. 2020. Art. no. 101011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1567422320300880

[34] A. Almjally, K. Howland, and J. Good, "Comparing TUIs and GUIs for primary school programming," in *Proc. 51st ACM Tech. Symp. Comput. Sci. Educ.*, New York, NY, USA, 2020, pp. 521–527. [Online]. Available: https://doi.org/10.1145/3328778.3366851

[35] S. Ahmed, M. A. A. Walid, and M. Islam, "EEG-based cognitive load assessment in matlab GUI and impact on learning system," in *Proc. 2nd Int. Conf. Adv. Inf. Commun. Technol. (ICAICT)*, 2020, pp. 484–487.

[36] A. Zitek, M. Poppe, M. Stelzhammer, S. Muhar, and B. Bredeweg, "Evaluating the effects of a new qualitative simulation software (Dynalearn) on learning behavior, factual and causal understanding," in *Artificial Intelligence in Education* (Lecture Notes in Computer Science), G. Biswas, S. Bull, J. Kay, and A. Mitrovic, Eds. Berlin, Germany: Springer, 2011, pp. 594–596, doi: 10.1007/978-3-642-21869-9_112

[37] A. C. Calderon and T. Crick, "Using interface design to develop computational thinking skills," in *Proc. Workshop Primary Sec. Comput. Educ. (WiPSCE)*, New York, NY, USA, 2015, pp. 127–129. [Online]. Available: https://doi.org/10.1145/2818314.2818333

**Sara Rijo-García** received the bachelor's degree in computer science and the master's degree in teacher training for secondary education, baccalaureate, vocational training, and foreign language teaching from the Universidad de La Laguna, San Cristóbal de La Laguna, Spain, in 2014 and 2015, respectively, where she is currently pursuing the Ph.D. degree in industrial, computer and environmental engineering.

She has worked with the Technology Transfer Office, Universidad de La Laguna for three years.

**Eduardo Segredo** (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from the Universidad de La Laguna, San Cristóbal de La Laguna, Spain, in 2006, 2008, and 2014, respectively.

He is currently a Lecturer with the Departamento de Ingeniería Informática y de Sistemas, Universidad de La Laguna. He has authored or coauthored over 50 technical papers and book chapters, including more than 15 journal papers. His publications currently report over 400 citations in Google Scholar with an H-index of 13. His current research interests include single/multi/many-objective optimization, evolutionary algorithms, metaheuristics, machine learning, and computational thinking.

Dr. Segredo currently serves for the editorial board of multiple international conferences.

**Coromoto León** received the M.S. degree in mathematics and the Ph.D. degree in computer science from the Universidad de La Laguna, San Cristóbal de La Laguna, Spain, in 1990 and 1996, respectively.

She currently teaches programming languages and paradigms in graduate and master's programs, with the Departamento de Ingeniería Informática y de Sistemas, Universidad de La Laguna. She has led dozens of doctoral courses and multiple seminars and has directed several research projects and contracts for innovation and industry transfer. She has published more than 30 journal papers and refereed conferences. Her research interests include programming languages, algorithmic techniques, optimization, parallel programming, and computational thinking.