A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education

Rodrigo Pessoa Medeiros, Geber Lisboa Ramalho, and Taciana Pontual Falcão

***Abstract*—*Contribution:* This paper adds to the results of previous systematic literature reviews by addressing a more contemporary context of introductory programming. It proposes a categorization of introductory programming challenges, and highlights key issues for a research roadmap on introductory programming learning and teaching in higher education.**

***Background:* Despite the advances in methods and tools for teaching and learning introductory programming, dropout and failure rates are still high. Published surveys and reviews either cover papers only up to 2007, or focus on methods and tools for teaching introductory programming.**

***Research Questions:* 1) What previous skills and background knowledge are key for a novice student to learn programming? 2) What difficulties do novice students encounter in learning how to program? 3) What challenges do teachers encounter in teaching introductory programming?**

***Methodology:* Following a formal protocol, automatic and manual searches were performed for work from 2010 to 2016. Of 100 papers selected for data extraction, 89 were retained after quality assessment.**

***Findings:* The most frequently cited skills necessary for learning programming were related to problem solving and mathematical ability. Problem solving was also cited as a learning challenge, followed by motivation and engagement, and difficulties in learning the syntax of programming languages. The main teaching challenges concern the lack of appropriate methods and tools, as well as scaling and personalized teaching.**

***Index Terms*—Achievement, faculty development, higher education, introductory programming, STEM, student experience, systematic review.**

I. Introduction

INTRODUCTORY programming courses are part of various undergraduate curricula, particularly in STEM degrees (Science, Technology, Engineering, and Mathematics). In this paper, "introductory programming" (commonly called CS1 in the United States) refers to a course for novice students that typically covers problem-solving skills, basic programming concepts, the syntax and semantics of a programming language, and the use of this programming language to formulate solutions. Failing an introductory programming course impacts various other courses taken [1].

Despite the advances in methods and tools for teaching and learning introductory programming [2], [3], and the gradual integration of programming fundamentals into high school curricula [2], dropout and failure rates are still high in introductory programming courses [4], [5]. There is no consensus on what the main challenges are [1], [3]–[5], nor a clear and complete categorization of them [6], [7]. Previous surveys and systematic literature reviews either cover papers up to 2007, [1], [8]–[10], or focus on methods and tools for teaching introductory programming [11]. There is thus a need for a systematic literature review that: (a) provides a better understanding and categorization of introductory programming problems; (b) covers recent research; and (c) includes both learning and teaching perspectives.

This paper is organized as follows: Section II presents some previous attempts to systematize the research results on introductory programming. Section III describes the systematic method adopted, and Section IV details its use for the work presented here. Section V presents and analyzes the answers to the research questions. Section VI further analyzes the answers, identifying possible key issues in learning and teaching introductory programming. Section VII makes some qualitative comparisons between this study and previous ones. Conclusions and future work are described in Section VIII.

II. Previous Studies

Most of the surveys and systematic reviews on introductory programming teaching and learning do not cover research results after 2008 [1], [8]–[10], and focus more on teaching methods and tools than on student problems. A more recent systematic review of introductory programming examined successful teaching practices [11]. Evaluating tools, methods, and practices for teaching introductory programming is essential, but given that dropout and failure rates are still high [4], [5], and that the literature shows no consensus on the main problems involved [1], [3], [4], there is a need for understanding

Manuscript received October 11, 2017; revised January 4, 2018 and July 3, 2018; accepted July 22, 2018. Date of publication August 27, 2018; date of current version May 3, 2019. *(Corresponding author: Rodrigo Pessoa Medeiros.)*

R. P. Medeiros is with the Centro de Informática, Universidade Federal de Pernambuco, Recife 50670-901, Brazil, and also with the Instituto Federal da Paraíba, Departamento de Design Gráfico, Cabedelo 58103-772, Brazil (e-mail: rpm3@cin.ufpe.br).

G. L. Ramalho is with the Centro de Informática, Universidade Federal de Pernambuco, Recife 50670-901, Brazil, and also with CESAR School, Recife 50030-220, Brazil (e-mail: glr@cin.ufpe.br).

T. P. Falcão is with the Departamento de Computação, Universidade Federal Rural de Pernambuco, Recife 52171-900, Brazil (e-mail: taciana.pontual@ufrpe.br).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TE.2018.2864133

and categorization. Indeed, a better understanding of challenges in introductory programming may help to develop better teaching tools, methods, and practices.

III. Method

The protocol adopted in this review follows the guidelines for systematic literature reviews presented by Kitchenham and Charters [12] and Petticrew and Roberts [13].

A. Research Questions

RQ1: What previous skills and background knowledge are key for a novice student to learn programming?

RQ2: What challenges do novice students encounter in learning how to program?

RQ3: What challenges do teachers encounter in teaching introductory programming?

The boundaries between learning (RQ1 and RQ2) and teaching (RQ3) are obviously fuzzy, especially in as complex a subject as programming. Nevertheless, this review keeps this distinction to provide a better understanding of challenges in introductory programming, already proposed in a programming context [6], [7].

B. Search Process

The search process started with a manual search in specific journals chosen for their relevance to the subject: the ACM Transactions on Computing Education (TOCE), the IEEE Transactions on Education, and Computer Science Education. A fourth source of data entry for the manual search was a systematic literature review written in 2014 and often cited in the literature of the area [11].

A manual search is frequently completed by an automatic search on scientific databases using a search string. Recurring key terms—such as "programming", "programming language", "programming teaching", "computer programming", "coding", "CS1", and "novice programmers" —were identified in the papers from the manual search. The search string for the automatic search was built from these terms and, after preliminary simulations, was defined as: ("learning programming" OR "teaching programming") AND ("novice programmers" OR "CS1").

The databases chosen for the automatic search were: ACM Digital Library, IEEE Xplore Digital Library, SpringerLink, ScienceDirect and ERIC (the Education Resources Information Center, sponsored by the Institute of Education Sciences). Before doing the automatic search, the search string was validated by being used in the IEEE and ACM databases. The papers retrieved were cross-checked with the papers selected manually (by applying inclusion and exclusion criteria - see below) from IEEE and ACM journals respectively.

C. Selection Criteria and Procedure

Papers included in the review had to be written in English, and published in conferences or journals, or as book chapters, between 2010 and 2016, on the theme of teaching and learning introductory programming in higher education.

Papers were excluded if they:

1) Did not address the research questions;
2) Were too short, such as workshop papers;
3) Were published in local conferences;
4) Were written by the same research group with the same data (in which case only the most recent was kept).

Because of the large number of papers retrieved in the search, a two-step procedure was performed. First, a preselection step applied the selection criteria on the basis of title, keywords and abstract. Next, the full text of the preselected papers was analyzed, and duplicates (criterion 4) were excluded.

Two researchers carried out the selection process, both of whom independently analyzed each paper. The reasons for inclusion or exclusion were carefully noted, and meetings were held to resolve any disagreements, with the help of a third researcher.

D. Snowballing

The results from both manual and automatic searches, after the application of the selection criteria, were "snowballed", that is, the bibliographical references of all the selected papers were considered as potential studies, and were then analyzed according to the inclusion and exclusion criteria.

E. Quality Assessment

The quality of the selected papers was assessed against criteria selected and adapted from Kitchenham & Charters' guidelines [12], focusing on the methods adopted and their scientific rigor.

For papers based on field studies (qualitative and quantitative approaches), the criteria were:

1) How well was the data collection carried out?
2) How well was the approach to, and formulation of, the analysis conveyed?
3) How well were the contexts and data sources retained and portrayed?
4) How clear and coherent were the links between data, interpretation, and conclusions?

For theoretical papers, the criteria were:

1) How well did the analysis address its original aims and objectives?
2) How has knowledge or understanding been extended by the research?
3) How well was diversity of perspective and context explored?

The assessment was performed independently by two researchers, and each paper was scored on the scale (for each criterion): 0 - very poorly; 1 - poorly; 2 - reasonably; 3 - well; 4 - very well. For scores that differed between the researchers by two or more points, a meeting was held with the third researcher to resolve the conflict. A quality threshold was adopted to decide whether to keep the papers in the analysis. The scores obtained for each question were averaged to give the final score for each paper.

F. Data Collection and Analysis

The following data were extracted from each study and recorded in the codebook (in spreadsheet format):

- Search mechanism (manual, automatic or snowballing) and source (journal name, database name and article title, respectively);
- Title, keywords, authors, publication venue, type (journal or proceedings), and year;
- Objective and participants;
- Scientific method (qualitative and/or quantitative approach, and specific methods adopted);
- Summary of the study;
- Answers to each research question;
- Quality assessment;

One researcher extracted the data, with the other checking the extraction. Any disagreements were discussed until resolved; if necessary, the third researcher was involved.

To analyze the answers to the research questions, a bottom-up inductive approach was adopted. Firstly, key terms were collected exactly as used in each paper. Different terms used to refer to similar concepts were identified, showing that authors choose words rather freely in discussing a topic, rather than following a standard terminology (perhaps drawn from prior experience or previous experience in programming). Such synonyms were consolidated under the most frequently-used terms. Then related concepts were grouped into categories under each research question, to clarify the presentation and discussion of the results, Section V. This grouping also drew on the literature where applicable (for example on theories on Computational Thinking [14], [15]), as discussed in Section V. Some terms were used by some authors with slightly different meanings; these are discussed in Section V. The first two researchers performed the inductive analysis together, then the third revised and validated it.

IV. Method Application

A. Search and Selection Results

The search for papers relevant to the review followed the process laid out in Fig. 1. The manual search provided 721 potentially relevant papers, from which 45 were pre-selected. The automatic search yielded 5,870 potentially relevant papers, from which 101 were pre-selected. This gave a total of 146 pre-selected papers. In the second phase of the selection process, 26 papers from the manual search and 62 papers from the automatic search were retained, giving 88 relevant papers in total. The snowballing process on these 88 selected papers provided another 102 candidates, of which 12 were selected. The final total at the end of the whole process was thus 100 papers. These are listed as Selected Papers in the Reference section of this paper [R01]–[R100].

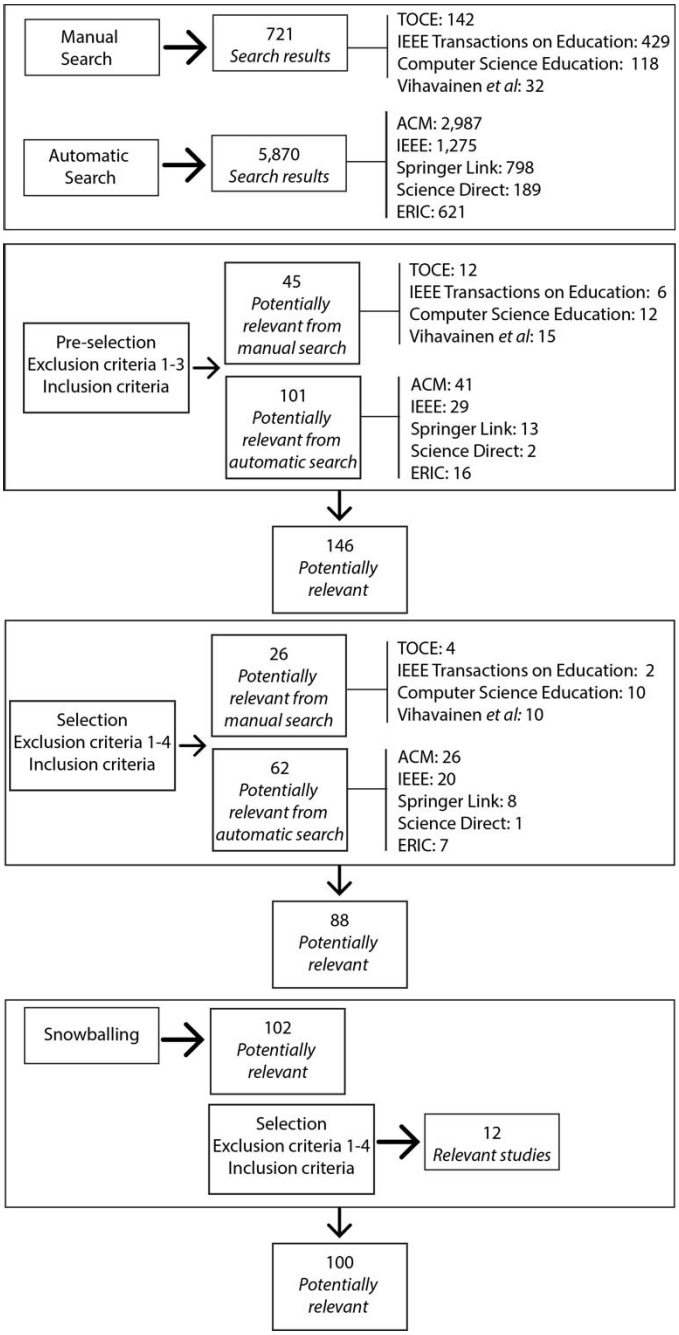

Fig. 1. Search process, indicating numbers of papers at each step.

B. Quality Assessment Results

To assess the quality of the selected studies (by answering a series of questions—see Section V), the scientific methods they adopted to support their arguments were collected, Table I. This information per se can be of interest to the computer science education community.

Generally, there was good agreement between raters: only six of the 100 papers had scores for at least one of the criteria that differed by two or more points. In the histogram of the quality assessment scores, Fig. 2, the x-axis labels the upper border of each histobar range.

Most studies received a good score, which is to be expected since they were published in peer-reviewed scientific journals and conferences. A few studies, however, obtained significantly low scores, and were not considered of acceptable quality to be included in the analysis. The threshold score adopted was 1.81, which corresponds to one of the apparent discontinuities in the histogram. Consequently, 11 papers (of

TABLE I
OVERVIEW OF METHODS

Method	Selected References
Pedagogical intervention evaluated through analysis of student performance and/or feedback	R04, R08, R09, R11, R12, R17, R18, R22, R23, R24, R25, R26, R27, R29, R30, R35, R38, R40, R41, R42, R43, R44, R45, R48, R51, R53, R55, R56, R57, R58, R59, R60, R61, R62, R63, R64, R66, R74, R75, R78, R79, R81, R82, R83, R85, R89, R92, R94, R96, R98, R99, R100
Survey, questionnaires, interviews and /or focus groups	R03, R06, R07, R10, R11, R12, R15, R19, R20, R21, R28, R31, R32, R33, R34, R35, R37, R41, R42, R44, R45, R46, R47, R50, R52, R53, R54, R55, R57, R62, R65, R61, R69, R70, R71, R76, R80, R81, R83, R87, R93, R95, R97
Observation of classroom activities	R03, R04, R09, R19, R28, R44, R50, R51, R83
Analysis of students' learning styles and strategies	R13, R33, R45, R51, R67, R70, R74, R85, R88
Analysis of student performance with regard to type of assessment, pedagogical approach or contextual factors	R64, R66, R67, R70, R73, R75, R77
Analysis of student errors	R36, R65, R84
Comparison between teachers' predictions/perceptions and students' actual performance	R01, R16
Theoretical reflection based on teaching experience	R02, R49, R68, R72, R86
Literature review	R05, R14, R39, R68, R71, R72, R74, R75, R80, R81, R86, R90, R91

the 100) were excluded. The results and discussion below only consider the remaining 89 papers.

C. Protocol Limitations

This review shares the most common limitations of the systematic method: search coverage and possible biases introduced during study selection, data extraction, and analysis. These limitations were addressed following the general recommendations for systematic reviews—using a combined manual and automatic search complemented by a snowballing process, and having two or more researchers selecting studies, assessing quality and extracting data [12], [13].

The specific limitation of the study lies in the fact that the stated research questions require answers that are not binary or objective. This review required a categorization of terms used by different primary studies to characterize introductory programming challenges. However, as is typical of education-related issues, the categories have fuzzy boundaries, so a given lexical term is used by different authors with the same meaning.

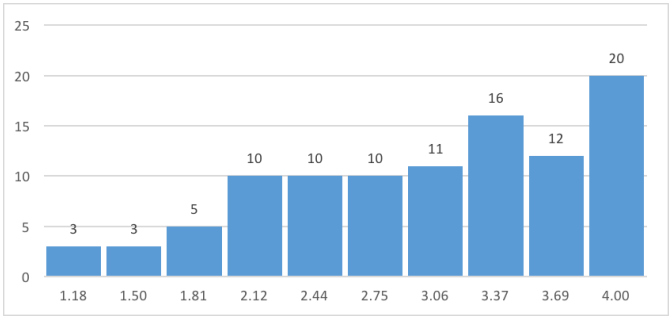

Fig. 2. Histogram of the quality assessment scores, showing the apparent discontinuities in the threshold score adopted. X-axis labels give the upper border of each histobar range, and the number of papers in that range.

TABLE II
STUDENT SKILLS

Category	Skill	References
Programming -Related Skills	problem solving	R04, R09, R10, R15, R19, R27, R39, R41, R51, R67, R73, R74, R75, R77, R80, R87
	mathematical ability	R04, R07, R10, R28 R38, R44, R45, R49, R69, R73, R75, R77, R80, R85, R86
	previous knowledge in programming	R06, R15, R23, R29, R48, R67, R69, R73, R75, R77, R86
	abstraction	R07, R12, R19, R28, R81, R86
General Educational Skills	basic knowledge in English	R01, R23, R38, R75, R80
	critical thinking and discussion skills	R10, R12, R14, R28
	creativity	R12, R28, R87, R97
	time management	R04, R81

V. RESULTS

A. Skill and Background Knowledge (RQ1)

The skills extracted from the papers were grouped into two categories: Programming-Related and General Educational, Table II.

1) Programming-Related Skills: Problem solving, which can be defined as understanding the context of a problem, identifying key information, and making a plan to solve it [16], was among the most cited skills for answering RQ1 (16 publications). It is considered a prerequisite for learning how to program: "*There are researchers that emphasize that the computer programming requires the use of problem-solving techniques. The lack of mastery of these techniques makes it difficult to learn programming.*" [R39, p. 02]; and even the main goal of introductory programming courses: "*[the goal is] to familiarize the students to the art of problem*

solving and to teach them how to express solutions in a simple way" [R15, p. 68]; and "*the long-term goals for the introductory programming course are that the students should eventually develop problem-solving ability and design competence*" [R41, p. 130]. It is also important that students can solve problems efficiently when working cooperatively with peers, teachers, and mentors [R39].

Mathematical ability is cited as a necessary skill closely related to problem-solving ability. Fifteen studies argue that deficient mathematical ability is one reason for the difficulty encountered by novice learners of programming: "*When talking with programming teachers, most of them claim that students don't know how to program because they don't know how to solve problems and do not have enough mathematical background*" [R07, p. 113]; "*Moreover many students did not show basic mathematical skills that are expected when entering higher education (...). Students had difficulties in calculus; Students did not have enough basic mathematical concepts concerning number theory; Students had difficulties to transform a textual problem into a mathematical formula that solves it; (...) Students had weak abstraction levels; Students had lack of logical reasoning*" [R45, pp. 01–02]. However, the correlation between mathematics and programming is not necessarily a cause-effect one: "*some teachers believe that mathematical ability is important for programming. However, they believe that the positive correlation between them is due to more basic cognitive functions that would be common to both domains. Thus, when developing a "mathematical logic", the student acquires structures or cognitive abilities that facilitate or promote learning to program*" [R28, p. 04].

Eleven publications discuss the impact of students' previous knowledge in programming; there is no consensus as to its consequences. For instance, [R15] and [R29] consider previous knowledge in programming as a predictor of success for novice learners in higher education: "*Although a few of the students have already learnt some programming language before getting enrolled in Computer Science degree program at university level, and it is believed that prior programming has a positive impact on Computer Science studies (...)*" [R15, p. 64]; "*Students who studied programming in the past tend to do well compared to those with no programming experience.*" [R29, p. 296].

However, the associated benefits are not straightforward: "*We normally think of the ability to code as the main benefit of having prior programming experience, but student interviews suggest that the impact of this experience is more complex: it affects students' expectations, work habits, attitude and confidence, and perceptions of self and peers*" [R06, p. 244].

Finally, abstraction is cited in six studies. Wing [17] defines abstraction as "*the most important and high-level thought process in computational thinking (...) used in defining patterns, generalizing from specific instances, and parameterization (...) to capture essential properties common to a set of objects*". Indeed, "*a programmer needs to apply abstraction when analyzing computational problems, as well as to instantiate abstract programming concepts and techniques to solve particular computational problems*" [R12, p. 624]; "*It is expected that students will use it [abstraction] to pass from the concrete world to a more semantic and symbolic reality, formulating or representing a problem in a more abstract manner, less attached to the details and singular concrete elements. When a student faces difficulties while attempting to do this abstraction, he/she is not capable of moving from reality to a more generic language such as the notation used in programming*" [R28, p. 04].

2) General Educational Skills: Critical thinking and discussion skills were cited in four publications, associated with the students' level of maturity at the university. They affect the interpretation and analysis of problems, and figure in the low incidence of classroom debate.

Four studies comment on the importance of creativity in programming, for example, "*it is important to stress that these students also believe that mathematical dexterity is important as well as creativity, even though they state that programming is basically reasoning*" [R28, p. 04].

Five publications add that limited knowledge in English is a barrier, as it is used in the syntax of all consolidated programming languages. Reference [R23] argues that, in the course they analyzed, fluent English speaking students were more likely than non-fluent ones to be successful.

Finally, time management was also cited as an essential skill by [R04] and [R81], in terms of planning use of time and the stages in the execution of a project.

B. Difficulties Encountered by Students (RQ2)

The first three categories in Table III are related to the three stages of computational thinking (problem formulation, solution expression and solution execution and evaluation) presented by Repenning, Basawapatna and Escherle [14, pp. 268–269]. The remaining category is behavior, related to social, emotional and self-management aspects.

1) Problem Formulation: As expected, problem-solving reappeared as a challenge faced by students, with 23 studies identifying this as a difficulty. These echo the arguments discussed with respect to RQ1 in Section V-A above: "*As may be expected, the learners face more difficulties to establish relations between different problem instances. Conceivably, a student who has gained a little familiarity with this problem-solving approach should provide logically correct answers*" [R33, p. 164]; "*Many of these [students] end up dropping the course due to not being able to solve problems and therefore feeling inadequate.*" [R26, p. 93].

Another problem cited, also discussed under RQ1, concerns the abstract nature of programming, with similar arguments: "*(...) students in learning programming need to imagine and comprehend many abstract terms that do not have equivalents in real life: how does a variable, a data type, or a memory address relate to a real-life object? These concepts are difficult to grasp. Consequently, many students struggle to comprehend even the most basic of programming concepts*" [R49, p. 16]; *"Students face diverse problems when they are learning programming (...) mainly because programming is dynamic and abstract"* [R54, p. 414].

Algorithmic reasoning, which appeared in three studies, is understood here as *"a pool of abilities that are connected to*

TABLE III
STUDENT DIFFICULTIES

Category	Difficulty	References
problem formulation	problem solving	R10, R14, R15, R16, R22, R26, R33, R37, R39, R41, R44, R45, R48, R49, R70, R73, R74, R81, R87, R89, R92, R94, R99
	abstract nature of programming	R10, R14, R33, R49, R54, R99
	algorithmic and logical reasoning	R28, R41, R99
solution expression	syntax of programming language	R02, R09, R15, R18, R21, R27, R33, R36, R39, R42, R44, R48, R49, R74, R82, R83, R84, R87, R90, R92
	control structures	R16, R21, R25, R27, R30, R43, R44, R48, R65, R74, R86, R90
	data structures	R21, R30, R34, R61, R90
	structure of the code	R21, R25, R34, R65
	others	R03, R21, R25, R34, R53
solution execution and evaluation	debugging	R02, R14, R18, R21, R27, R48, R55, R84, R87, R89, R90, R91, R96
	tracing the execution	R14, R16, R19, R27, R42, R53, R61, R68, R84
behavior	motivation and engagement	R03, R04, R10, R11, R12, R13, R17, R20, R23, R24, R26, R30, R44, R45, R48, R55, R56, R70, R71, R72, R73, R74, R75, R77, R79, R80, R84, R87, R88, R89, R90, R94, R96, R97, R98, R100
	time management	R01, R13, R44, R46, R50, R55, R56, R69, R80, R81, R88, R93, R94, R96, R100
	study skills	R10, R13, R46, R50, R56, R69, R75, R80, R93, R94, R97
	confidence	R10, R56, R69, R75, R85, R92, R93, R98, R100
	perception of programming as a complex discipline	R03, R12, R21, R65, R81, R93

constructing and understanding algorithms: to analyze given problems; to specify a problem precisely; to find basic actions that are adequate to the given problem; to construct a correct algorithm to a given problem using the basic actions" [18, p. 160]. According to teachers, "*(...) it is this shift in perception that allows them [the students] to see the whole based on the parts that will allow the student to elaborate the algorithm. In words of one of the teachers, when thinking this way, the student 'no longer thinks as a person but starts thinking as a computer, taking into account the machine's limited comprehension."'* [R28, pp. 03–04].

2) Solution Expression: Once the problem is formulated, students can move on to expressing a solution through programming structures. The first problem within the solution expression scope, with 20 occurrences, concerns the syntax of programming languages, as reported by [R48, p. 02]: *"The overhead of learning the syntax and semantics of a language at the same time, and difficulties in combining new and previous knowledge and developing their general problem-solving skills"*. According to [R49, p. 02], "*even students, who have adequate problem-solving skills and manage to phrase a solution to a programming problem in terms of a pseudo code, find it difficult to turn the pseudo code into a syntactically correct computer program.*" In this context, it is important to consider syntax error as part of the learning process and help students learn to recover from them [R36], [R48]: "*Syntax errors are an important area of programming pedagogy research. Experienced programmers rarely make syntax errors, and when they do, they have clear strategies to correct them very quickly. However, syntax errors are significant for novice programmers; correcting them is a time-consuming process and often leads to random debugging behavior, also influenced by the fact that students do not understand compiler messages"* [R48, p. 16].

Control structures also appeared as a challenge (13 occurrences). Among them, conditional statements [R25], [R43], loops [R14], [R25], [R27], [R43], [R48], [R65], [R86], [R90], recursion [R18], [R27], [R90], and sequence, selection and repetition [R21], [R30] were the most cited. Selecting the appropriate structures for solving a problem (if, if/else, with, for, while) was also cited [R21] as a difficulty.

Several papers also mention difficulties related to the use of data structures. References [R21] and [R34] explore this topic the most deeply, but only on the basis of previous literature, and not on their findings. Data structures most often cited as potentially difficult were arrays [R21], [R30], [R61], [R90].

For code structure, some papers mention the use of functions/classes [R21], [R65], the relationship between classes and objects [R25], and using language libraries [R34]. Finally, other aspects of solution expression cited were: pointers [R03], [R21], [R34], [R53], [R65], references [R34], [R65], parameters [R25], [R34], [R65], variable scoping [R53] and error handling [R34].

3) Solution Execution and Evaluation: Students should test and analyze their code to identify and correct problems. In this context, debugging (13 publications) and code tracing (nine publications) were the most cited challenges.

Debugging is described as a complex activity, requiring several aspects to be mastered: *"Debugging, one of the essential skills for successful programmers, is difficult for novices as it requires the application of many new skills simultaneously. Students must understand the problem domain, know rudimentary programming concepts and understand at least one programming language well enough to read and write instructions, comprehend the logic of the intended program, and be able to track down and fix bugs"* [R55, p. 390]. It demands

much practice and should be explicitly supported by teachers: *"debugging should be explicitly taught. Assuming students will simply "pick up" debugging skills as a by-product of learning to program may lead them to develop some of the ineffective strategies observed in this study"* [R55, p. 395]. Reference [R02] states that students rarely test their code, perhaps because they do not know how to formulate proper test cases, or because they lack the discipline to test their code systematically.

Finally, students should also be given the opportunity for developing code tracing skills, as cited by nine papers, since this helps them understand programming more holistically. This activity may be related to debugging, but can also be used to understand code passages. The authors argue that: *"code tracing activities exposed students' not viable models and, more importantly, provided a much clearer starting point with which to discuss correct models"* [R53, p. 562].

4) Behavior: This category encompasses social and emotional aspects that can have an impact on student learning, such as motivation, engagement, and confidence (including the perception of programming as being difficult), as well as study habits and time management.

The most discussed issue in this category (36 occurrences), is the relationship between students' motivation and engagement, and positive learning outcomes [R14]: *"For learners, engagement correlates with improvements in specific desirable outcomes (...), such as general abilities and critical thinking, cognitive development, improved grades and persistence"* [R14, p. 02]. The perception of programming as a complex discipline, coupled with the previously discussed learning challenges, contributes to students' demotivation, particularly in the first year of higher education [R10]: *"(...) an unmotivated student will hardly succeed. This is aggravated by the fact that programming courses usually gain a reputation that passes from student to student of being difficult that so many of them start already feeling defeated"* [R12, p. 02]. *"The motivational dimension has a great impact on the individual's cognitive development and is a determinant factor for success in the learning process (. . .). It is motivation that fosters in the student the disposition to want to progress and reach the goals that were set, maintaining an adequate level of volition to overcome the demands that are being dealt with"* [R13, pp. 1–2].

Students' confidence is another issue (nine occurrences). According to [R10, p. 02], *"some students may have motivation but the self-confidence may be blocked, as such these types of students need the teacher to work closer with them to develop self-esteem"*. Similarly, [R98, p. 14] argues that *"Student programmers who lack confidence are less able to make independent progress with coding exercises. They frequently become "stuck", and will wait for assistance from the instructor, rather than try an alternative approach on their own*". Confidence in programming is expected to change rapidly as the student gains experience [R98].

Fifteen publications mention students' poor time management as another challenge: *"most students indicated that they regularly spent far less time studying for the unit than the recommended eight hours [per week]"* [R46, p. 125]. Students themselves acknowledge this difficulty: *"Another constantly recurring theme in the free comments section was time management issues. Many students reported that they have too many concurrent courses, all with high workload. Without careful time management and prioritizing, there is not enough time for deep-level learning in every study topic"* [R44, p. 04].

Another issue is study skills (eleven occurrences) such as organization and minimal work habits [R10], or the comprehension of students' own learning styles [R46], [R75], [R80]: "*The acquisition of well-functioning learning strategies and skills seems, in this specific context, to be affected by group work strategies, extrinsic motivation, and some issues related to study habits. It is not always easy to establish a direction of causality, but one important aspect in learning programming seems to be the work that students are required to do on their own time, outside the instructed learning sessions. For one reason or another, in too many cases students are not able to find effective ways of working (...)"* [R80, p. 306].

TABLE IV
FACULTY CHALLENGES

Challenges	References
methods and tools for teaching programming	R13, R18, R19, R24, R28, R35, R41, R46, R47, R49, R53, R57, R58, R59, R60, R61, R63, R64, R70, R72, R73, R74, R76, R78, R79, R81, R82, R88, R89, R93, R98, R99, R100
scale problems	R03, R09, R10, R32, R44, R50, R57, R68, R72, R93, R96
keeping students' motivation, engagement and persistence	R03, R10, R12, R14, R20, R24, R27, R51, R78, R79, R80, R81, R86, R87, R89, R93, R96
teacher-student communication and feedback	R10, R14, R18, R24, R27, R29, R41, R76, R82, R83, R85, R87, R90, R91, R96, R97
programming language	R05, R15, R19, R47, R48
curriculum and instructional sequences	R03, R51, R75
addressing students' inadequate mathematical background	R28, R90, R91

C. Faculty Challenges (RQ3)

Challenges faced by faculty when teaching introductory programming are listed in Table IV. Obviously, learning challenges (RQ1 and RQ2) would naturally appear in RQ3 as teaching ones. To avoid redundancy, only teaching challenges corresponding to the most-cited learning difficulties, Table III, were included in the discussion here. Other teaching challenges that have not yet been discussed are also treated here.

1) Revisited Challenges: Maintaining student motivation, engagement and persistence is fundamental (17 occurrences), but teachers struggle to find strategies to attract students' interest [R10]. Reference [R24, p. 499] argues that the

teacher should "*be a motivator, not just a provider*", and that "*Not many teachers realize that it is also their responsibility to motivate the students and arouse the students' interests*" [R24, p. 499]. Reference [R51] highlights the role of the instructors in helping students to persist: *"Instructors should thoroughly ground novices in foundational skills and simple problem solving before presenting problem-solving plans, so that novices have the background needed to understand and appreciate those plans. Doing so should increase the persistence of less-advanced novices in introductory courses"* [R51, p. 311]. A third concept closely related to this discussion is keeping student's engagement: *"(...) improving the student engagement can be different from case to case and must be elaborated into practical approach and strategies"* [R24, p. 502]; *"Some of the factors that can result in un-engagement among students are: the current style of education doesn't appeal to everyone; some educators are not sensitive to students' responses to educational methods in class; students overwhelmed by all they have to do - 'quality of effort"'* [R14, p. 03].

Students' inadequate mathematical background is also revisited; teachers should create specific activities and/or change the pedagogical approach to deal with it: *"(...) we propose the introduction of activities that foster the development of mental models that are fit to the cognitive skills needed in programming, thus helping the students succeed in the course. While problem based learning tries, by definition, to promote problem solving abilities, we have verified that it is not enough when dealing with the students that present greater difficulties"* [R28, p. 06].

2) Methods and Tools for Teaching Programming: The main challenge in RQ3, cited by 33 publications, is related to teaching methods and tools. This cross-cutting concern underlines various challenges discussed in RQ3. Various methods are suggested: learning by doing and learning by examples; problem-based learning; pre-recorded classes; active learning exercises and demonstrations; live coding, canned examples and trace-driven teaching approach; extreme apprenticeship; games, game-themed assignments and gamification; team based-learning; media computing; collaborative learning; collaborative tutoring session; mentor support; peer instruction and pair programming. However, the results of the application of these methods are not yet conclusive [11]. Reference [R46] reinforces the necessity of rethinking the current teaching methods: *"This apparent mismatch in teacher expectations and student behavior might give universities the impetus to rethink the type of educational experiences we provide for our students"* [R46, p. 128].

Some studies report applications of specific teaching methods in introductory programming in classes, but these do not always succeed, for reasons which are often unclear. One case of partial failure is reported in [R53, pp. 562–563]: *"Like many institutions, our introductory classes suffered from low retention rates and poor student performance. These problems occurred despite our making a number of pedagogical changes including adding active learning exercises and demonstrating how code features worked using live coding and canned examples. (...) The classes already included in-class code tracing and coding activities, labs, small programming homework exercises, larger programming projects, and other opportunities for students to develop valid mental models. We observed that some of these activities seemed to hurt as much as they helped."*. On the other hand, [R53] reports an experiment using a trace-driven teaching approach that decreased the dropout and grade failures by 25.49% and 8.51% respectively. Reference [R47] comments on the need to prioritize and adjust teaching methods to improve problem-solving, one of the most cited skills necessary for learning programming (RQ2). Reference [R49] comments on the lack of appropriate tools for teaching the object-oriented paradigm. Reference [R19] comments that the choices of tools and the initial programming language are important and challenging for teachers.

3) Scale Problems: The scale topic is related to number and diversity of students in classes, and it involves various aspects: heterogeneity of students [R09], [R10], [R31], [R57], [R68], [R72], staff resource limitations and classes' size [R03], [R09], [R32], [R44], [R72], [R93], [R96] and personalized teaching [R09], [R10], [R50].

Teachers struggle to develop problem-solving skills in classes with students with heterogeneous levels of knowledge, commitment and different learning styles [R10]. Furthermore, it is extremely hard to address cognitive needs and individual difficulties in large groups [R10]. Staff is also essential since tutors and mentors can help increase the number of activities and assessments [R03], [R32], [R44]. Using extrinsic motivators to engage students also requires planning and extra effort [R03].

The heterogeneity of students, cited by six publications, relates to knowledge, motivation, commitment and learning rhythm: *"It is possible to explain concepts and examples, but it is very hard to promote the development of problem solving skills and address the variety of cognitive needs, learning styles, difficulties and motivations present in a group of students that is often very heterogeneous"* [R10, p. 01]; *"Therefore, it is difficult for the teacher to follow an approach suitable for every individual student. In an attempt to reach all students, teachers often design lectures and activities to the 'average student', who may not even exist in the class. To improve this situation personalized support and guidance are necessary, so that individual needs and difficulties can be addressed"* [R09, p. 01].

Five publications comment on staff resource limitations and how this affects the teaching and learning process: *"The general problems covered contextual issues such as staff resource limitations, university control systems (...). Changing a course and increasing the number of student assignments required extra resources, but a course revision should, in the long run, aim at optimizing the resource needs and uses"* [R03, p. 23]. In general, teachers say that having more people involved in the teaching process, such as tutors and mentors, would help to make teaching more fluid, with better classroom dynamics and fewer differences in learning rhythms [R03], [R32], [R44].

4) Teacher-Student Communication and Feedback: Another challenge is student-teacher communication and the feedback process (16 occurrences): *"A good pedagogical theory for*

teaching programming should focus on students' learning, and effective communication between teacher and student. This could be achieved by clearly stating goals and keep the students motivated" [R14, p. 104]. The way of evaluating and providing feedback to students can be a determining factor in their demotivation [R09]–[R12], [R14], [R29], particularly for women [R71]. However, giving quality feedback is not simple: *"Give feedback, not only judgment. Assessment can be classified into two categories: formative and summative. In formative assessment we can get the feedback of the current teaching and learning condition and use the feedback to improve them. On the other hand, the summative intention is solely to form a judgment"* [R24, p. 499]. Besides its inherent complexity, the feedback process is also affected by lack of time, quantity of students or format of courses.

More generally speaking, there are two reasons that teacher-student communication seems to fail: too many students in one class [R10]; and students' resistance to discussing their errors in exercises due to lack of rapport [R24]. There is a clear relationship between the communication and feedback challenges, and the category of scale problems.

5) Choice of Programming Language: The most specific introductory programming challenge is choosing the first programming language to be taught to students. Several programming languages can be used for introductory programming courses, as discussed by [R21, p. 02]: "*The available programming languages are numerous and selecting the one that will be used is a multi-criteria decision"*. This choice has a direct impact on the development of novices' programming skills [R48] and can be key in facilitating the teaching process [R05] and shaping programming style and coding technique [R15]. Reference [R47] discusses the choice between commercial and academic languages: *"The teacher is exposed to risks when teaching programming courses for novices in which the emphasis is on the programming language. One of the risks is that the novices focus their attention on syntax issues and not on the computational semantic power of the language, which at the end of the day is what makes it possible to build solutions using computer programs. This approach prevents novices from understanding that the main role of a programming language is to serve as a means to express computational solutions proposed in the training exercises. Moreover, choosing a standard de facto language in the industry offers the advantage of training the student to develop skills that the market is looking for. However, it can also generate a bias of the concept of the student regarding the futility of learning other languages and, besides, it can reduce the environment in which students develop the ability to learn to learn"* [R47, p. 02].

6) Curriculum and Instructional Sequences: [R03] highlights the idea that the introductory programming course should build more explicitly upon skills acquired in previous courses. *"At the curriculum level, departments should develop explicit course sequences that build upon the skills acquired in other courses to increase the student intrinsic motivation to complete the courses as planned. The departments should also track the performance of their courses to be able to act on problems early, since rehabilitating a course is a lengthy process. (...) In general, a competent instructor may be able to deliver a successful course without all the proposed systemic frameworks"* [R03, p. 23].

There also seems little research on instructional sequences for teaching programming for novices in higher education, as pointed out by [R51]: *"(...) no study has directly compared the effectiveness and efficiency of the instructional sequences used in presenting programming material to novices. Is it better to begin with a strategic overview, to start with syntax details and work upward, or to work through entire programming concepts one at a time? Knowing the effect of sequence on instructional effectiveness can help instructional designers avoid sequences that make learning unnecessarily difficult in an already difficult domain"* [R51, p. 292].

VI. General Discussion

Some issues for inclusion in the research roadmap on teaching and learning introductory programming have emerged from this review:

A. Better Understanding and Characterization of Problem-Solving in Programming

Problem solving is a crucial concept in introductory programming. Some authors in this review [R09], [R15], [R41], [R48], [R49] consider that the primary aim of an introductory programming course is to develop problem-solving skills through algorithmic thinking and basic concepts of programming, rather than to teach the syntactic particularities of a specific programming language. Positive student perceptions have been reported from approaches aligned with this perspective [20]. Computer Science Curricula 2013 [21] recommends that introductory programming courses should avoid conveying the idea to students that computer science is mainly about learning the specifics of a programming language, and instead emphasize general concepts in computing within the context of learning how to program.

Although problem solving appears in several articles, its definitions are generic, lacking, or inconsistent across authors. The reasons for students' limitations in problem solving are not detailed: is it because they do not understand the statement of the problem, do not know the strategy to solve it, or do not know a step-by-step plan to implement the strategy? How is problem-solving related to algorithmic thinking, computational thinking, or abstraction? Is problem-solving a monolithic issue, or does it encompass various sub-problems? And if so, how are they related? Some current definitions of computational thinking [15] may help. Deeper investigations should be carried out to understand what problem-solving really means in programming.

B. Improving Background Knowledge

Students' inadequate background knowledge (mainly of mathematics and English) is cited by introductory programming teachers as a challenge, but how can they "solve" this problem within the timeframe of an introductory programming course? Efforts to teach programming to children and

teenagers at school [2] are expected to improve this situation for the future generations of programming students, but this is not certain. Methods and metrics should be established to evaluate the efficacy of this approach. In a broader perspective, it could be desirable to foster a productive and systematic exchange of information between school and university teachers, with the involvement of policymakers who define curricula at all levels. This would help bridge gaps in students' competencies between school and university. Such interaction should be encouraged.

C. Better Specific Tools and Methods for Problem Formulation and Solution Expression

Some teachers believe that learning to properly formulate a problem and express its solution in a specific programming language, in the timeframe of an introductory programming course, is a great cognitive load for students. The idea of separating problem formulation strategies and solution expression into two different courses is gaining ground, but this assumes that problem framing and expression is one of the roots of the learning problems in introductory programming. More empirical evidence is needed.

More methods and tools are also needed if these two phases are to be developed separately. Nowadays, block-based programming environments for beginners help developing computational thinking, and are not tied to a professional programming language as taught in university courses. However, the vast majority of these are designed for children (e.g., Scratch, Alice, and others from code.org). Although they have been shown to help [2], [R61], most are too childish to be transferable to the university level. Building similar tools higher education should be considered.

D. Improving Motivation

Motivation appears is a significant concern for teachers, with high impact on the other learning problems. Approaches are needed to help teachers maintain student motivation.

Many papers discuss motivation, but only one examines the reasons for students' demotivation [R07]. A more in-depth understanding of this phenomenon seems necessary. Several promising teaching methods (such as gamification, problem-based learning, or unplugged computing) are available, but these are still not commonly adopted in higher education. The barriers to adoption could be teaching culture, insufficient knowledge of the methods, the work required to put them into practice, or other unknowns.

E. More Empirical Basis and Standards

The last point concerns the basis upon which the papers analyzed deliver their conclusions. Several authors make claims mainly based on observations made by teachers in their daily classroom work. Although the importance of these observations cannot be denied, empirical data from more formal experimental projects would be welcome. When empirical results exist, they are not comparable because the experimental protocol, the number and profile of participants, and even the goals are too heterogeneous.

Therefore the creation of experimental standards for tasks, participant profiles and number, expected results, and evaluation criteria should be encouraged. Benchmarks for evaluating methods and tools for introductory programming would be very welcome.

VII. Comparing Results With Previous Studies

Studies covering papers published before 2010 [1], [8]–[10] are surveys rather than systematic literature reviews. They do not follow a rigid protocol in presenting results, so these cannot be quantitatively compared with the results of this review. Nevertheless it is possible to qualitatively compare some issues from the 1990s or 2000s. Remarkably, the solution expression problems (language syntax, control structures, and data structures) highlighted in the earlier studies are still relevant, but are now of similar importance to problem formulation.

Several topics listed as novices' difficulties by Robins et al. in their review from 2003 [9] were also identified in the present review, including: problem solving (strategies and skills); solution expression (abstraction, use of data structures like variables and arrays, use of control structures like loops, conditional statements, and recursions); and solution execution (tracing/tracking code, dedicating time to planning and testing; and debugging). This 2003 review did not discuss issues of choosing a programming language and learning syntax, or the importance of students' mathematical background, algorithmic and logical reasoning—all of which were identified in the present review.

Topics involving methods and tools for teaching and learning introductory programming were a significant concern in 2003, and, as discussed in the present review, still are. Robins *et al.* [9] discussed the importance of fostering learning of core principles of programming, focusing on student learning rather than instructor teaching, linked to methods such as learning-by-doing, problem solving-based methods, peer and collaborative work, and lab-based learning (all of which should be combined with appropriate assessment and feedback given through effective student-teacher communication). The authors also discussed tools for teaching programming; curriculum and instructional sequences, but did not discuss scale problems.

From the 2014 review [11, p. 25] focused on evaluating methods for teaching introductory programming: *"What may be missing however, are the reports on interventions that did not yield an improvement. Thus, educators that have tried an intervention but received poor results should also be encouraged and supported in reporting the results to create a more stable picture of the field"*.

Robins *et al.* [9] discuss the importance of engagement and indicate motivation and confidence as potential factors for better understanding effective and ineffective novice learners. This confirms the finding of the present review of the growing interest in student engagement and motivation in recent years. Further topics related to the behavior of novice programming learners discussed here, such as time management and study skills, were not mentioned in [9].

VIII. Conclusion

A systematic literature review was performed on 100 papers to move towards a better understanding of introductory programming problems. With respect to previous systematic reviews, this study has as main contributions: (i) covering more recent papers; (ii) proposing novel research questions, including an original discussion on previous skills and background knowledge; (iii) stimulating discussion of a categorization of introductory programming challenges; (iv) suggesting potentially key issues for a research roadmap on introductory programming learning and teaching.

For RQ1, on students' background knowledge and skills, the most frequently cited issues were problem-solving abilities and mathematical knowledge. For RQ2, on challenges faced by students in introductory programming courses, the major issues cited were related to motivation and engagement, problem-solving, and the syntax of programming languages. For RQ3, about the challenges faced by faculty, the need for appropriate methods and tools for teaching programming at the introductory level were the most cited issue.

This review has also shown the need to: (i) clarify the concept of problem-solving in programming; (ii) foster the dialogue between the communities of primary and higher education; (iii) create specific tools for the problem formulation stage; (iv) understand why promising teaching methods are not frequently adopted in higher education; (v) stimulate the creation of experimental standards.

In future work, the authors aim to pursue some of the issues raised in Section VI, deepening the investigation based on a better understanding of problem-solving in programming, following recent results on the categorization of computational thinking [14], [15]. Another stream of future research, already under way, is an analysis of barriers to successful methods in basic education on computational thinking being used in higher education. Finally, the authors intend to create a new systematic review devoted to the investigation of empirical methods, tools, and type of data employed in the identification of problems in learning and teaching introductory programming, as well as in the evaluation of solutions.

Acknowledgment

The authors would like to thank Dr. G. Cabral and Dr. P. Tedesco from Universidade Federal de Pernambuco and C.E.S.A.R (Centro de Estudos e Sistemas Avançados do Recife) for the support and assistance in the preparation of this paper.

References

[1] E. Lahtinen, K. Ala-Mutka, and H. M. Järvinen, "A study of the difficulties of novice programmers," *ACM SIGCSE Bull.*, vol. 37, no. 3, pp. 14–18, 2005.

[2] M. Resnick *et al.*, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009.

[3] S. Mohorovičić and V. Strčić, "An overview of computer programming teaching methods," in *Proc. Central Eur. Conf. Inf.*, 2011, pp. 47–52.

[4] C. Watson and F. W. B. Li, "Failure rates in introductory programming revisited," in *Proc. Conf. Innov. Technol. Comput. Sci. Educ.*, Uppsala, Sweden, 2014, pp. 39–44.

[5] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," *ACM SIGCSE Bull.*, vol. 39, no. 2, pp. 32–36, 2007.

[6] D. Teague, "Pedagogy of introductory computer programming: A people-first approach," M.S. thesis, Dept. Inf. Syst., Queensland Univ. Technol., Brisbane, QLD, Australia, 2011.

[7] P. Kinnunen, "Challenges of teaching and studying programming at a University of Technology—Viewpoints of students, teachers and the University," Ph.D. dissertation, Dept. Comput. Sci. Eng., Helsinki Univ. Technol., Espoo, Finland, 2009.

[8] T. Jenkins, "On the difficulty of learning to program," in *Proc. 3rd HEA Conf. ICS Learn. Teach. Support Netw.*, Loughborough, U.K., 2002, pp. 1–8.

[9] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Comput. Sci. Educ.*, vol. 13, no. 2, pp. 137–172, 2003.

[10] A. Pears *et al.*, "A survey of literature on the teaching of introductory programming," in *Proc. ITiCSE-WGR*, Dundee, U.K., 2007, pp. 204–223, doi: 10.1145/1345443.1345441.

[11] A. Vihavainen, J. Airaksinen, and C. Watson, "A systematic review of approaches for teaching introductory programming and their influence on success," in *Proc. ICER*, Glasgow, U.K., 2014, pp. 19–26.

[12] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering, version 2.3," School Comput. Sci. Math., Keele Univ., Keele, U.K., Rep. EBSE-2007-01, 2007.

[13] M. Petticrew and H. Roberts, *Systematic Reviews in the Social Sciences: A Practical Guide*. Hoboken, NJ, USA: Blackwell, 2008, p. 352, doi: 10.1002/9780470754887.

[14] A. Repenning, A. Basawapatna, and N. Escherle, "Computational thinking tools," in *Proc. IEEE Symp. Vis. Lang. Human-Centric Comput. (VL/HCC)*, Cambridge, U.K., 2016, pp. 218–222.

[15] M. Berry, *QuickStart Computing: A CPD Toolkit for Primary Teachers*, BCS, Swindon, U.K., 2015. [Online]. Available: http://primary.quickstartcomputing.org/resources/pdf/qs_handbook.pdf

[16] S. Lehoczky and R. Rusczyk, *The Art of Problem Solving, Volume 1: The Basics*, 7th ed. Alpine, CA, USA: AoPS, 2006, p. 288.

[17] J. M. Wing, *Computational Thinking—What and Why?* Link Mag., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2011, pp. 20–23. [Online]. Available: http://link.cs.cmu.edu/files/11-399_The_Link_Newsletter-3.pdf

[18] G. Futschek, "Algorithmic thinking: The key for understanding computer science," in *Informatics Education—The Bridge Between Using and Understanding Computers. ISSEP 2006* (LNCS 4226), R. T. Mittermeir, Ed. Berlin, Germany: Springer-Verlag, 2006, pp. 159–168, doi: 10.1007/11915355_15.

[19] V. Trowler, *Student Engagement Literature Review*, Higher Educ. Acad., York, U.K., 2010. [Online]. Available: https://www.heacademy.ac.uk/system/files/studentengagementliteraturereview_1.pdf

[20] S. Turner and G. Hill, "Robots in problem-solving and programming," in *Proc. 8th Annu. Conf. Subject Centre Inf. Comput. Sci.*, Southampton, U.K., 2007, pp. 82–85, doi: 10.13140/2.1.2425.6800.

[21] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. New York, NY, USA: ACM, 2013, doi: 10.1145/2534860.

Selected Papers

[R01] I. Utting *et al.*, "A fresh look at novice programmers' performance and their teachers' expectations," in *Proc. ITiCSE-WGR*, Canterbury, U.K., 2013, pp. 15–32.

[R02] A. N. Kumar, "A mid-career review of teaching computer science I," in *Proc. SIGCSE 44th ACM Tech. Symp. Comput. Sci. Educ.*, Denver, CO, USA, 2013, pp. 531–536.

[R03] U. Nikula, O. Gotel, and J. Kasurinen, "A motivation guided holistic rehabilitation of the first programming course," *ACM Trans. Comput. Educ.*, vol. 11, no. 4, 2011, Art. no. 24, doi: 10.1145/2048931.2048935.

[R04] S. Haatainen, A.-J. Lakanen, V. Isomöttönen, and V. Lappalainen, "A practice for providing additional support in CS1," in *Proc. Learn. Teach. Comput. Eng. (LaTiCE)*, 2013, pp. 178–183.

[R05] J. Sorva, V. Karavirta, and L. Malmi, "A review of generic program visualization systems for introductory programming education," *ACM Trans. Comput. Educ.*, vol. 13, no. 4, Nov. 2013, Art. no. 15, doi: 10.1145/2490822.

[R06] A. Tafliovich, J. Campbell, and A. Petersen, "A student perspective on prior experience in CS1," in *Proc. SIGCSE 44th ACM Tech. Symp. Comput. Sci. Educ.*, Denver, CO, USA, 2013, pp. 239–244.

[R07] A. J. Gomes and A. J. Mendes, "A study on student performance in first year CS courses," in *Proc. ITiCSE*, Ankara, Turkey, 2010, pp. 113–117.

[R08] C. Gavan and M. Anderson, "Engaging undergraduate programming students: Experiences using lego mindstorms NXT," in *Proc. 13th Annu. Conf. Inf. Technol. Educ. (SIGITE)*, Calgary, AB, Canada, 2012, pp. 139–144.

[R09] A. Santos, A. Gomes, and A. Mendes, "A taxonomy of exercises to support individual learning paths in initial programming learning," in *Proc. Front. Educ. Conf. (FIE)*, Oklahoma City, OK, USA, 2013, pp. 87–93, doi: 10.1109/FIE.2013.6684794.

[R10] A. Gomes and A. Mendes, "A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations," in *Proc. Front. Educ. Conf. (FIE)*, Madrid, Spain, 2014, pp. 1–8, doi: 10.1109/FIE.2014.7044086.

[R11] S. Alhazbi, "ARCS-based tactics to improve students' motivation in computer programming course," in *Proc. Comput. Sci. Educ. (ICCSE)*, Cambridge, U.K., 2015, pp. 317–321, doi: 10.1109/ICCSE.2015.7250263.

[R12] M. A. Brito and F. de Sá-Soares, "Assessment frequency in introductory computer programming disciplines," *ACM Comput. Human Behav.*, vol. 30, pp. 623–628, Jan. 2014, doi: 10.1016/j.chb.2013.07.044.

[R13] A. P. Ambrosio, S. Martins, L. Almeida, A. Franco, and F. Georges, "Assessment of self-regulated attitudes and behaviors of introductory programming students," in *Proc. Front. Educ. Conf. (FIE)*, Seattle, WA, USA, 2012, pp. 1–6, doi: 10.1109/FIE.2012.6462314.

[R14] N. Eltegani and L. Butgereit, "Attributes of students engagement in fundamental programming learning," in *Proc. Comput. Control Netw. Electron. Embedded Syst. Eng. (ICCNEEE)*, Khartoum, Sudan, 2015, pp. 101–106, doi: 10.1109/ICCNEEE.2015.7381438.

[R15] M. Ateeq, H. Habib, A. Umer, and M. U. Rehman, "C++ or Python? Which one to begin with: A learner's perspective," in *Proc. Teach. Learn. Comput. Eng. (LaTiCE)*, Kuching, Malaysia, 2014, pp. 64–69, doi: 10.1109/LaTiCE.2014.20.

[R16] Simon *et al.*, "Can computing academics assess the difficulty of programming examination questions?" in *Proc. Koli Calling*, 2012, pp. 160–163, doi: 10.1145/2401796.2401822.

[R17] D. Horton, M. Craig, J. Campbell, P. Gries, and D. Zingaro, "Comparing outcomes in inverted and traditional CS1," in *Proc. ITiCSE*, Uppsala, Sweden, 2014, pp. 261–266, doi: 10.1145/2591708.2591752.

[R18] T. Wang, X. Su, P. Ma, Y. Wang, and K. Wang, "Ability-training-oriented automated assessment in introductory programming course," *Comput. Educ.*, vol. 56, no. 1, pp. 220–226, 2011.

[R19] J. Holvikivi, "Conditions for successful learning of programming skills," in *Key Competencies in the Knowledge Society* (IFIP Advances in Information and Communication Technology (AICT)), vol. 324, N. Reynolds and M. Turcsányi-Szabó, Eds. Berlin, Germany: Springer, 2010, pp. 155–164, doi: 10.1007/978-3-642-15378-5_15.

[R20] P. Kinnunen and B. Simon, "CS majors' self-efficacy perceptions in CS1: Results in light of social cognitive theory," in *Proc. ICER*, Providence, RI, USA, 2011, pp. 19–26, doi: 10.1145/2016911.2016917.

[R21] S. Xinogalos, "Designing and deploying programming courses: Strategies, tools, difficulties and pedagogy," *Educ. Inf. Technol.*, vol. 21, no. 3, pp. 559–588, Jul. 2014, doi: 10.1007/s10639-014-9341-9.

[R22] O. Seppälä, P. Ihantola, E. Isohanni, J. Sorva, and A. Vihavainen, "Do we know how difficult the rainfall problem is?" in *Proc. Koli Calling*, 2015, pp. 87–96, doi: 10.1145/2828959.2828963.

[R23] D. Horton and M. Craig, "Drop, fail, pass, continue: Persistence in CS1 and beyond in traditional and inverted delivery," in *Proc. SIGCSE*, Kansas City, MO, USA, 2015, pp. 235–240, doi: 10.1145/2676723.2677273.

[R24] B. Hartanto, "Enhancing the student engagement in an introductory programming: A holistic approach in improving the student grade in the informatics Department of the University of Surabaya," in *Intelligence in the Era of Big Data. ICSIIT 2015* (Communications in Computer and Information Science), vol. 516, R. Intan, C. H. Chi, H. Palit, and L. Santoso, Eds. Berlin, Germany: Springer, 2015, pp. 493–504, doi: 10.1007/978-3-662-46742-8_45.

[R25] T. Sirkiä and J. Sorva, "Exploring programming misconceptions: An analysis of student mistakes in visual program simulation exercises," in *Proc. Koli Calling*, 2012, pp. 19–28, doi: 10.1145/2401796.2401799.

[R26] A. Vihavainen, M. Paksula, and M. Luukkaine, "Extreme apprenticeship method in teaching programming for beginners," in *Proc. SIGCSE*, Dallas, TX, USA, 2011, pp. 93–98, doi: 10.1145/1953163.1953196.

[R27] L. Ma, J. Ferguson, M. Roper, and M. Wood, "Investigating and improving the models of programming concepts held by novice programmers," *Comput. Sci. Educ.*, vol. 21, no. 1, pp. 57–80, 2011, doi: 10.1080/08993408.2011.554722.

[R28] A. P. Ambrósio, F. M. Costa, L. Almeida, A. Franco, and J. Macedo, "Identifying cognitive abilities to improve CS1 outcome," in *Proc. Front. Educ. Conf. (FIE)*, Rapid City, SD, USA, 2011, pp. F3G-1–F3G-7, doi: 10.1109/FIE.2011.6142824.

[R29] L. Porter and D. Zingaro, "Importance of early performance in CS1: Two conflicting assessment stories," in *Proc. SIGCSE*, Atlanta, GA, USA, 2014, pp. 295–300, doi: 10.1145/2538862.2538912.

[R30] H. Aris, "Improving students performance in introductory programming subject: A case study," in *Proc. Comput. Sci. Educ. (ICCSE)*, Cambridge, U.K., 2015, pp. 657–662, doi: 10.1109/ICCSE.2015.7250328.

[R31] M. Konecki, N. Kadoić, and R. Piltaver, "Intelligent assistant for helping students to learn programming," in *Proc. Inf. Commun. Technol. Electron. Microelectron. (MIPRO)*, Opatija, Croatia, 2015, pp. 924–928, doi: 10.1109/MIPRO.2015.7160406.

[R32] M. Hertz and S. M. Ford, "Investigating factors of student learning in introductory courses," in *Proc. SIGCSE*, Denver, CO, USA, 2013, pp. 195–200, doi: 10.1145/2445196.2445254.

[R33] C. Mirolo, "Learning (through) recursion: A multidimensional analysis of the competences achieved by CS1 students," in *Proc. ITiCSE*, Ankara, Turkey, 2010, pp. 160–164, doi: 10.1145/1822090.1822136.

[R34] M. Piteira and C. Costa, "Learning computer programming: Study of difficulties in learning programming," in *Proc. ISDOC*, Lisbon, Portugal, 2013, pp. 75–80, doi: 10.1145/2503859.2503871.

[R35] C. Watson, F. W. B. Li, and R. W. H. Lau, "Learning programming languages through corrective feedback and concept visualisation," in *Advances in Web-Based Learning—ICWL 2011* (LNCS 7048), H. Leung, E. Popescu, Y. Cao, R. W. H. Lau, and W. Nejdl, Eds. Heidelberg, Germany: Springer, 2011, pp. 11–20, doi: 10.1007/978-3-642-25813-8_2.

[R36] D. McCall and M. Kölling, "Meaningful categorisation of novice programmer errors," in *Proc. Front. Educ. Conf. (FIE)*, Madrid, Spain, 2014, pp. 1–8, doi: 10.1109/FIE.2014.7044420.

[R37] R. Matthews, H. S. Hin, and K. A. Choo, "Novice programming students' perception of learning object," in *Proc. Informat. Creative Multimedia (ICICM)*, Kuala Lumpur, Malaysia, 2013, pp. 292–297, doi: 10.1109/ICICM.2013.67.

[R38] M. Yousoof and M. Sapiyan, "Optimizing instruction for learning computer programming—A novel approach," in *Intelligence in the Era of Big Data. ICSIIT 2015* (Communications in Computer and Information Science), vol. 516. R. Intan, C. H. Chi, H. Palit, and L. Santoso, Eds. Berlin, Germany: Springer, 2015, pp. 128–139, doi: 10.1007/978-3-662-46742-8_12.

[R39] O. D. L. Tavares, C. S. de Menezes, and R. A. de Nevado, "Pedagogical architectures to support the process of teaching and learning of computer programming," in *Proc. Front. Educ. Conf. (FIE)*, Seattle, WA, USA, 2012, pp. 1–6, doi: 10.1109/FIE.2012.6462427.

[R40] A. Shargabi, S. A. Aljunid, M. Annamalai, S. M. Shuhidan, and A. M. Zin, "Program comprehension levels of abstraction for novices," in *Proc. Comput. Commun. Control Technol. (I4CT)*, Kuching, Malaysia, 2015, pp. 211–215, doi: 10.1109/I4CT.2015.7219568.

[R41] N. Thota, "Programming course design: Phenomenographic approach to learning and teaching," in *Proc. Teach. Learn. Comput. Eng. (LaTiCE)*, Kuching, Malaysia, 2014, pp. 125–132, doi: 10.1109/LaTiCE.2014.30.

[R42] D. Teague and R. Lister, "Programming: Reading, writing and reversing," in *Proc. ITiCSE*, Uppsala, Sweden, 2014, pp. 285–290, doi: 10.1145/2591708.2591712.

[R43] M. Yamamoto, T. Sekiya, and K. Yamaguchi, "Relationship between programming concepts underlying programming skills," in *Proc. Inf. Technol. Based Higher Educ. Train. (ITHET)*, Izmir, Turkey, 2011, pp. 1–7, doi: 10.1109/ITHET.2011.6018678.

[R44] M. Apiola, N. Moisseinen, and M. Tedre, "Results from an action research approach for designing CS1 learning environments in Tanzania," in *Proc. Front. Educ. Conf. (FIE)*, Seattle, WA, USA, 2012, pp. 1–6, doi: 10.1109/FIE.2012.6462321.

[R45] A. Gomes and A. J. Mendes, "Studies and proposals about initial programming learning," in *Proc. Front. Educ. Conf. (FIE)*, Washington, DC, USA, 2010, pp. 1–6, doi: 10.1109/FIE.2010.5673426.

[R46] J. Sheard, A. Carbone, D. Chinn, and M.-J. Laakso, "Study habits of CS 1 students: What do they say they do?" in *Proc. Learn. Teach. Comput. Eng. (LaTiCE)*, 2013, pp. 122–129, doi: 10.1109/LaTiCE.2013.46.

[R47] S. M. M. Rubiano, O. López-Cruz, and E. G. Soto, "Teaching computer programming: Practices, difficulties and opportunities," in *Proc. Front. Educ. Conf. (FIE)*, El Paso, TX, USA, 2015, pp. 1–9, doi: 10.1109/FIE.2015.7344184.

[R48] T. Koulouri, S. Lauria, and R. D. Macredie, "Teaching introductory programming: A quantitative evaluation of different approaches," *ACM Trans. Comput. Educ.*, vol. 14, no. 4, 2014, Art. no. 26, doi: 10.1145/2662412.

[R49] L. M. M. Giraffa, M. C. Moraes, and L. Uden, "Teaching object-oriented programming in first-year undergraduate courses supported by virtual classrooms," in *Proc. 2nd Int. Workshop Learn. Technol. Educ. Cloud*, 2013, pp. 15–26, doi: 10.1007/978-94-007-7308-0_2.

[R50] V. Isomöttönen and V. Tirronen, "Teaching programming by emphasizing self-direction: How did students react to the active role required of them?" *ACM Trans. Comput. Educ.*, vol. 13, no. 2, Jun. 2013, Art. no. 6, doi: 10.1145/2483710.2483711.

[R51] D. A. Kranch, "Teaching the novice programmer: A study of instructional sequences and perception," *Educ. Inf. Technol.*, vol. 17, no. 3, pp. 291–313, 2011, doi: 10.1007/s10639-011-9158-8.

[R52] Á. Matthíasdóttir and H. J. Geirsson, "The novice problem in computer science," in *Proc. CompSysTech*, Vienna, Austria, 2011, pp. 570–576, doi: 10.1145/2023607.2023702.

[R53] M. Hertz and M. Jump, "Trace-based teaching in early programming courses," in *Proc. SIGCSE*, Denver, CO, USA, 2013, pp. 561–566, doi: 10.1145/2445196.2445364.

[R54] C. J. Costa, M. Aparicio, and C. Cordeiro, "Web-based graphic environment to support programming in the beginning learning process," in *Entertainment Computing—ICEC 2012* (LNCS 7522), M. Herrlich, R. Malaka, and M. Masuch, Eds. Heidelberg, Germany: Springer, 2012, pp. 413–416, doi: 10.1007/978-3-642-33542-6_41.

[R55] S. Fitzgerald *et al.*, "Debugging from the student perspective," *IEEE Trans.*, vol. 53, no. 3, pp. 390–396, Aug. 2010, doi: 10.1109/TE.2009.2025266.

[R56] M. Haungs, C. Clark, J. Clements, and D. Janzen, "Improving first-year success and retention through interest-based CS0 courses," in *Proc. 43rd ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, Raleigh, NC, USA, 2012, pp. 589–594.

[R57] J. Kurhila and A. Vihavainen, "Management, structures and tools to scale up personal advising in large programming courses," in *Proc. Conf. Inf. Technol. Educ. (SIGITE)*, New York, NY, USA, 2011, pp. 3–8.

[R58] P. Lasserre and C. Szostak, "Effects of team-based learning on a CS1 course," in *Proc. 6th Annu. Joint Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, Darmstadt, Germany, 2011, pp. 133–137.

[R59] L. Porter and B. Simon, "Retaining nearly one-third more majors with a trio of instructional best practices in CS1," in *Proc. SIGCSE*, Denver, CO, USA, 2013, pp. 165–170.

[R60] C. F. Reilly and N. De La Mora, "The impact of real-world topic labs on student performance in CS1," in *Proc. Front. Educ. Conf.*, Seattle, WA, USA, 2012, pp. 1–6, doi: 10.1109/FIE.2012.6462329.

[R61] M. Rizvi and T. Humphries, "A scratch-based CS0 course for at-risk computer science majors," in *Proc. Front. Educ. Conf.*, Seattle, WA, USA, 2012, pp. 1–5, doi: 10.1109/FIE.2012.6462491.

[R62] G. Rößling and M. Mühlhäuser, "An unusual CS 1 with high standards and confirming results," in *Proc. 15th Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, Ankara, Turkey, 2010, pp. 169–173.

[R63] S. C. Shaffer and M. B. Rosson, "Increasing student success by modifying course delivery based on student submission data," *ACM Inroads*, vol. 4, no. 4, pp. 81–86, 2013, doi: 10.1145/2537753.2537778.

[R64] B. Simon, P. Kinnunen, L. Poter, and D. Zazkis, "Experience report: CS1 for majors with media computation," in *Proc. 15th Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, Ankara, Turkey, 2010, pp. 214–218.

[R65] R. Cacceffo, S. Wolfman, K. S. Booth, and R. Azevedo, "Developing a computer science concept inventory for introductory programming," in *Proc. 47th ACM Tech. Symp. Comput. Sci. Educ.*, Memphis, TN, USA, 2016, pp. 364–369.

[R66] J. Figueiredo, N. Gomes, and F. J. García-Peñalvo, "Ne-course for learning programming," in *Proc. 4th Int. Conf. Technol. Ecosyst. Enhancing Multiculturality*, Salamanca, Spain, 2016, pp. 549–553.

[R67] A. Lishinski, A. Yadav, R. Enbody, and J. Good, "The influence of problem solving abilities on students' performance on different assessment tasks in CS1," in *Proc. 47th ACM Tech. Symp. Comput. Sci. Educ.*, Memphis, TN, USA, 2016, pp. 329–334.

[R68] A. Luxton-Reilly, "Learning to program is easy," in *Proc. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, Arequipa, Peru, 2016, pp. 284–289.

[R69] A. Petersen, M. Craig, J. Campbell, and A. Tafliovich, "Revisiting why students drop CS1," in *Proc. 16th Koli Calling Int. Conf. Comput. Educ. Res.*, 2016, pp. 71–80.

[R70] D. F. Shell, L.-K. Soh, A. E. Flanigan, and M. S. Peteranetz, "Students' initial course motivation and their achievement and retention in college CS1 courses," in *Proc. 47th ACM Tech. Symp. Comput. Sci. Educ.*, Memphis, TN, USA, 2016, pp. 639–644.

[R71] A. Lishinski, A. Yadav, J. Good, and R. Enbody, "Learning to program: Gender differences and interactive effects of students' motivation, goals, and self-efficacy on performance," in *Proc. ACM Conf. Int. Comput. Educ. Res.*, Melbourne, VIC, Australia, 2016, pp. 211–220.

[R72] C. Ott, A. Robins, and K. Shephard, "Translating principles of effective feedback for students into the CS1 context," *ACM Trans. Comput. Educ.*, vol. 16, no. 1, Jan. 2016, Art. no. 1, doi: 10.1145/2737596.

[R73] G. Silva-Maceda, P. D. Arjona-Villicaña, and F. E. Castillo-Barrera, "More time or better tools? A large-scale retrospective comparison of pedagogical approaches to teach programming," *IEEE Trans.*, vol. 59, no. 4, pp. 274–281, Nov. 2016, doi: 10.1109/TE.2016.2535207.

[R74] M. P. Uysal, "Improving first computer programming experiences: The case of adapting a Web-supported and well-structured problem-solving method to a traditional course," *Contemporary Educ. Technol.*, vol. 5, no. 3, pp. 198–217, 2014.

[R75] R. A. Alturki, "Measuring and improving student performance in an introductory programming course," *Informat. Educ.*, vol. 15, no. 2, pp. 183–204, 2014, doi: 10.15388/infedu.2016.10.

[R76] K. Hartness and L.-J. Shannon, "Peer mentors and their impact for beginning programmers," *Inf. Syst. Educ. J.*, vol. 9, no. 6, pp. 21–29, 2011.

[R77] A. Hoskey and P. S. M. Maurino, "Beyond introductory programming: Success factors for advanced programming," *Inf. Syst. Educ. J.*, vol. 9, no. 5, pp. 61–70, 2011.

[R78] B. Hosack, B. Lim, and W. P. Vogt, "Increasing student performance through the use of Web services in introductory programming classrooms: Results from a series of quasi-experiments," *J. Inf. Syst. Educ.*, vol. 24, no. 4, pp. 373–383, 2012.

[R79] K. Sung *et al.*, "Game-themed programming assignment modules: A pathway for gradual integration of gaming context into existing introductory programming courses," *IEEE Trans. Edu.*, vol. 54, no. 3, pp. 416–427, Aug. 2011, doi: 10.1109/TE.2010.2064315.

[R80] M. Apiola and M. Tedre, "New perspectives on the pedagogy of programming in a developing country context," *Comput. Sci. Educ.*, vol. 22, no. 3, pp. 285–313, 2012.

[R81] T. B. Bati, H. Gelderblom, and J. van Biljonc, "A blended learning approach for teaching computer programming: Design for large classes in Sub-Saharan Africa," *Comput. Sci. Educ.*, vol. 24, no. 1, pp. 71–99, 2014.

[R82] B. A. Becker *et al.*, "Effective compiler error message enhancement for novice programming students," *Comput. Sci. Educ.*, vol. 26, nos. 2–3, pp. 148–175, 2016.

[R83] J. Bennedsen and M. E. Caspersen, "Persistence of elementary programming skills," *Comput. Sci. Educ.*, vol. 22, no. 2, pp. 81–107, 2012.

[R84] M. C. Hughes, M. C. Jadud, and M. M. T. Rodrigo, "String formatting considered harmful for novice programmers," *Comput. Sci. Educ.*, vol. 20, no. 3, pp. 201–228, 2010.

[R85] C. Ott, A. Robins, P. Haden, and K. Shephard, "Illustrating performance indicators and course characteristics to support students' self-regulated learning in CS1," *Comput. Sci. Educ.*, vol. 25, no. 2, pp. 174–198, 2015.

[R86] A. Robins, "Learning edge momentum: A new account of outcomes in CS1," *Comput. Sci. Educ.*, vol. 20, no. 1, pp. 37–71, 2010.

[R87] S. Shuhidan, M. Hamilton, and D. D'Souza, "Instructor perspectives of multiple-choice questions in summative assessment for novice programmers," *Comput. Sci. Educ.*, vol. 20, no. 3, pp. 229–259, 2010.

[R88] S. Willman *et al.*, "On study habits on an introductory course on programming," *Comput. Sci. Educ.*, vol. 25, no. 3, pp. 276–291, 2015, doi: 10.1080/08993408.2015.1073829.

[R89] L. Beck and A. Chizhik, "Cooperative learning instructional methods for CS1: Design, implementation, and evaluation," *ACM Trans. Comput. Educ.*, vol. 13, no. 3, pp. 1–21, 2013.
[R90] A. Berglund and R. Lister, "Introductory programming and the didactic triangle," in *Proc. 12th Aust. Comput. Educ. Conf. (ACE)*, Brisbane, QLD, Australia, 2010, pp. 35–44.
[R91] R. Cardell-Oliver, "How can software metrics help novice programmers?" in *Proc. 13th Aust. Comput. Educ. Conf. (ACE)*, Perth, WA, Australia, 2011, pp. 55–62.
[R92] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx, "Understanding the syntax barrier for novices," in *Proc. 16th Annu. Joint Conf. Innov. Technol. Comput. Sci. Educ.*, Darmstadt, Germany, 2011, pp. 208–212.
[R93] N. Hawi, "Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course," *Comput. Educ.*, vol. 54, no. 4, pp. 1127–1136, 2010.
[R94] L. J. Höök and A. Eckerdal, "On the bimodality in an introductory programming course: An analysis of student performance factors," in *Proc. Int. Conf. Learn. Teach. Comput. Eng.*, Taipei, Taiwan, 2015, pp. 79–86.
[R95] R. Ibrahim, R. C. M. Yusoff, H. M. Omar, and A. Jaafar, "Students perceptions of using educational games to learn introductory programming," *Comput. Inf. Sci.*, vol. 4, no. 1, pp. 205–216, 2011.
[R96] V. Isomöttönen and V. Lappalaine, "CS1 with games and an emphasis on TDD and unit testing: Pilling a trend upon a trend," *ACM Inroads*, vol. 3, no. 3, pp. 62–68, 2012.
[R97] D. F. Shell *et al.*, "Improving learning of computational thinking using computational creativity exercises in a college CSI computer science course for engineers," in *Proc. Front. Educ. Conf. (FIE)*, Madrid, Spain, 2014, pp. 1–7, doi: 10.1109/FIE.2014.7044489.
[R98] K. Wood, D. Parsons, J. Gasson, and P. Haden, "It's never too early: Pair programming in CS1," in *Proc. 15th Aust. Comput. Educ. Conf. (ACE)*, Adelaide, SA, Australia, 2013, pp. 13–21.
[R99] A. Yadin, "Reducing the dropout rate in an introductory programming course," *ACM Inroads*, vol. 2, no. 4, pp. 71–76, 2011.
[R100] B. Simon, J. Parris, and J. Spacco, "How we teach impacts student learning: Peer instruction vs. lecture in CS0," in *Proc. 44th ACM Tech. Symp. Comput. Sci. Educ.*, Denver, CO, USA, 2013, pp. 41–46.

Rodrigo Pessoa Medeiros received the B.S. degree in Internet systems from Faculdade Marista, Brazil, in 2005 and the M.S. degree in digital arts and technology from Universidade do Minho, Portugal, in 2009. He is currently pursuing the Ph.D. degree in computer science with the Universidade Federal de Pernambuco, Brazil.

Geber Lisboa Ramalho received the Ph.D. degree in computer science from the University of Paris VI in 1997. He is an Electronic Engineer and an Associate Professor with the Centro de Informática, Universidade Federal de Pernambuco, Brazil, where he is currently the Head of the Systems Engineer Department and the Chair of the Board of CESAR, a Brazilian innovation institute.

Taciana Pontual Falcão received the bachelor's degree in computer science and the Ph.D. degree from the University of London in 2014. She is an Assistant Professor with the Departamento de Computação, Universidade Federal Rural de Pernambuco, Brazil.