


Introductory Programming in Higher Education: A Systematic Literature Review

Gabryella Rodrigues ✉ 

Research Centre on Education (CIEd), Institute of Education, University of Minho, Braga, Portugal

Ana Francisca Monteiro ✉ 

Research Centre on Education (CIEd), Institute of Education, University of Minho, Braga, Portugal

António Osório ✉ 

Research Centre on Education (CIEd), Institute of Education, University of Minho, Braga, Portugal

Abstract

A systematic literature review was performed on 33 papers obtained from the ACM, IEEE and Sciencedirect databases, in order to understand in depth, the introductory programming discipline (CS1) in higher education. Recently published works have been covered, providing an overview of the teaching-learning process of introductory programming and enabling to find out whether the research developed by universities worldwide is in line with the proposals made by ACM/IEEE-CS group for computer courses, regarding the transition to the competency-based model. The results show that the new techniques/technologies currently used in software development, as an example of agile methodology, has influenced the teaching-learning process of CS1 together with methods such as visual programming and e-learning. The analyzed papers discuss the importance of developing not only technical, but also social skills, corroborating that methodologies used in introductory programming courses need to focus on preparing students for an increasingly competitive market, associating new skills with technical aspects.

2012 ACM Subject Classification General and reference → Surveys and overviews

Keywords and phrases systematic literature review, CS1, introductory programming, teaching programming, learning programming

Digital Object Identifier 10.4230/OASICS.ICPEC.2022.4

Funding This work is supported by CIEd – Research Centre on Education, Institute of Education, University of Minho, and funded by FCT – Foundation for Science and Technology, scholarship n°2021.07850.BD.

1 Introduction

The Association for Computing Machinery (ACM) guides and recommends higher education institutions worldwide in the context of analyzing the characteristics of graduation students and in the construction of curricula in the field of computing, since 1960. The most recent document – published in partnership with the Computer Society of the Institute of Electrical and Electronics Engineers (IEEE-CS) – the Computing Curricula 2020: Paradigms for Global Computing Education [19], referred to as CC2020, records significant updates.

The main change from previous documents focuses on the transition from a traditional teaching model to a competency-based model. The first one is described through areas of knowledge, units of knowledge and learning outcomes. However, this paradigm has been shown to be inefficient through two new challenges: the new ways of acquiring knowledge and the gap between graduates' skills and the skills expected in professional activities, known as “the skill gap” [19].



© Gabryella Rodrigues, Ana Francisca Monteiro, and António Osório;
licensed under Creative Commons License CC-BY 4.0

Third International Computer Programming Education Conference (ICPEC 2022).

Editors: Alberto Simões and João Carlos Silva; Article No. 4; pp. 4:1–4:17

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Therefore, with the aim of promoting the success of learning and the effective reduction of the skill gap, the ACM/IEEE group made use of previous experiences and incorporated to CC2020 the concept of competence as a primary characteristic in the construction of the computer science curriculum. The curriculum guidelines for undergraduate degree programs in Information Technology [45] identified as IT2017 was the initial inspiration.

This research focuses on the introductory programming discipline, specifically in algorithms, which according to CC2013 [44] are fundamental to the development of any software system and present in all areas of computing. The study of algorithms provides an insight into the nature of the problem as well as possible solution techniques, independently of a programming language and programming paradigm, computer hardware or any other aspect of implementation.

A systematic literature review was conducted in June 2021 in order to systematically collect recent data available in scientific databases on: the course identity, expected skills, methodologies, programming languages and tools used in teaching and learning of introductory programming.

2 Previous Studies

Most of the recent systematic literature review on introductory programming focus on a specific teaching/learning method [32, 35, 53, 36] or on the assessment of tools, programming languages or programming paradigms [27, 24, 3]. In general, studies that address the teaching and learning programming on novice, do not cover results after 2018 [5, 37, 40] and centers only on dropout, failure rates, problems faced by students and technical skills expected before and at the end of the course.

Despite the large volume of work on introductory programming, there is a lack of studies that focus on competency instead of knowledge expectations [45]. To gain a different perspective in this review, we aiming in studies that also include a competence model.

3 Research Questions

Firstly, given the recommendation to parameterize the course's terminologies and classifications and the areas of knowledge proposed in CC2020, this research seeks to identify how the introductory programming discipline is identified and structured and in which computing degree program is targeted (Q1). In addition to pointing out, or not, a pattern in the discipline's identity, this research seeks to register what contents are covered, in order to recognize the presence of the algorithm subject. It also seeks to examine which undergraduate courses in computing place more emphasis on the development of algorithms, comparing these results with the panorama suggested in CC2020, in which it is quantified the importance that each area of knowledge offers to courses.

Secondly, driven by the transition to the CC2020 competency model, this research investigated whether universities are still based on the classical teaching model or are already concerned about the transition to the new model proposed and what skill set are classified as essential for computer science students taking the introductory programming course and whether these skills are explored within a real-world context. For that it is important to note which technical and social skills are involved in the process of creating and analyzing an algorithm (Q2).

After the analysis of the included texts, a list of methodologies, tools, languages and programming paradigms used in the teaching and learning of introductory programming course was compiled, in order to analyze the influence of the technology in this process and

observe how close are the tools used in universities of those required by job market, identified through the question: What are the methodologies, tools, languages and programming paradigms used in the teaching and learning process of introductory programming (Q3)?

The results of this work will define which terminologies will be used to reference this course in future work; the level of importance that the discipline has in each computer course will be used to define which curriculum recommendations documents (currently there are seven different reports regulating computer courses) will be used as a basis in the construction of a conceptual framework of a doctoral research project in the context of learning the introductory programming course in higher education. In addition to providing information to justify the choice of the courses that will be observed in the field work.

The data related to skills, technical skills, social skills and methodologies described in the collected documents will be used as a basis for the construction of a observation guide in a case study that will be developed in the future.

4 Method

The systematic literature review seeks to employ a research methodology with scientific rigor and great transparency. To ensure such rigor and transparency Elisa Nakagawa [42] affirm that all systematic review must contain a protocol of investigation, which describes the entire process in detail. This study follows the guidelines for systematic literature review presented by [29] and [42] and consists of the following stages: formulation of the research questions to be answered; strategies adopted in the search and selection process of the studies that will be included in the review; the procedure for data extraction and classification and finally the data synthesis strategies and analysis of results.

4.1 Search Strategy

To define the string used in this systematic review, the research questions were broken down into keywords and synonyms were searched for each term [29]. For the first research question (Q1), in which the objective is to reveal the identity of the discipline, we opted to use a manual search (test search) in computing databases with the terminology CS1.

The CS1 (Computer Science 1) acronym was originally created in 1978 by the ACM and refers to the first computing course that introduces the programming basics [18]. Although the contents have changed over the past four decades, the name and general principles have remained [23].

The result of this preliminary search showed us that the researches carried out on this field also make references to the keywords: “Introductory Programming Course” and “Introduction to Programming”.

Similarly to the second research question (Q2), the word “skill” was chosen to identify which skills are described as necessary to students who are attending and/or have attended the fundamentals of programming. The choice of not distinguishing social skills from technical skills in this process was based on the results obtained in this preliminary exercise, which showed that with the exclusive use of the word Skill researches involving both capabilities were returned.

For the third research question (Q3), the chosen words that proved to be timely in identifying the methodologies, tools, languages and programming paradigms used by teachers and students in the discipline were: method and tools. When the word “method” was associated with “CS1” it returned works that involve the use of new teaching methods in the teaching of introductory programming. Using the keyword “tools” combined with “CS1”,

we obtained papers that cited the use of tools and programming languages. For the last item of Q3: programming paradigms, it was not necessary to use a specific keyword, because programming paradigms can be revealed from the identification of the programming language used, as [11] explains.

Since this work seeks to research both teaching and learning processes of introductory programming, the identified keywords were associated to the terms: “teaching programming” and “learning programming” and the search strategy, described in Table 01, was built.

The use of a wide search string, provided its use in several databases [29] and its validation was made by an expert in the area, who reproduced the final protocol.

4.2 Database

After the process described above, an automatic search was performed in Scimedirect, IEEE Xplore and ACM Digital Library, using the previously constructed research strategy. The first two sources, according to the classification of Felizardo et al. [16] are identified as bibliographic databases, I.e only return studies published by their own publisher. And in order to mitigate possible limitations in the searches, we also used a hybrid database: ACM Digital Library, which indexes studies published by the publisher and studies from other sources [16].

In addition to the characteristics mentioned above, the ACM and IEEE associations, which respectively maintain the ACM Digital Library and IEEE Xplore database, are responsible for producing and publishing all the reports that guide computer courses. Also, they are classified as two of the main indexing services in the area of computing, electricity and electronics [9].

In order to collect all the evidence used to answer the research questions, we have made use of a third source: Scimedirect. A multidisciplinary database made available by Elsevier, which brought together papers published by researchers from various countries and a collection covering several thematic areas in the field of Science and Technology.

Another point to consider about the databases chosen was the fact that they index new studies regularly and with peer-reviewed. They have a search engine that allow an easy adaptation of the string, versatility in exporting the results and integration with a reference management software.

4.3 Selection Criteria

In order to include relevant documents and ensure that no important study was excluded, selection criteria were defined (inclusion and exclusion) in the search protocol before starting the automatic searching process in ScienceDirect, IEEE Xplore and ACM Digital Library databases.

4.3.1 Inclusion Criteria

The condition of having been published in the period from June 2017 to June 2021 (five-year period), was the inclusion criteria used to select documents in this systematic review, allowing the presentation of a current scenario of what has been investigated on “introductory programming”. In particular, it can provide insights into what new technologies are influencing the teaching and learning process in this discipline.

At the same time, the first document of the ACM/IEEE that embraced the concept of competencies as the main characteristic of a computing curriculum was the Information Technology (IT2017) report [45], published in 2017. This led to the adherence of CC2020 to the competence-based learning model.

Although the mentioned document was published only in December 2017, this review moved the observation window back to June 2017 with the intention to identify whether the literature already indicate this concern even before any publication by the ACM/IEEE.

We highlight that the selection criteria was not limited to a specific languages, the only criteria for including documents in this systematic review is limited to time.

4.3.2 Exclusion Criteria

And with the aim of eliminating texts considered irrelevant [29], a set of five exclusion criteria was adopted:

■ Duplicate Records

According to Kitchenham and Charters [29], it is important not to consider repeated evidence from the same study, to avoid create distortions in the conclusions. Thus, this criteria considers the most recent study, in cases of papers written by the same author or group of authors that address the same subject. However the study will only be deleted if the most recent one deals with all the contents of the older version. If the study contains an intersection, it will not be excluded.

■ Papers providing only the abstract or unfinished research

Detailed information on the methodological aspects, such as type of research, data collection method, target audience, instruments, programming languages used, as well as details of the results, may not be included in the abstract or in unfinished research. However, it is a type of information that needs to be collected to assist the data analysis phase. Therefore, works that do not provide access to the full text or are ongoing research will not be included in the analysis phase.

■ Papers that do not result from a scientific research

A systematic literature review aims to collect evidence from primary studies [42]. Thus, studies that are not the result of a research will not be considered. This case includes panel, journal column, tutorial, editorial and other systematic literature reviews.

■ Works carried out with students of different levels

The interest in programming is not exclusive to higher education courses and much less to courses in the exact sciences area. There are many researches carried out also with students of all levels and modalities of teaching. However the interest of this review is to focus only on how to learn and teach in introductory programming in undergraduate courses.

■ Works that do not address identity, basic learning conditions and the teaching methods of the discipline of “introductory programming”

The quality of a systematic review is linked to the papers chosen for the analysis. Works that do not answer at least one of the research questions of this review were not classified for the next phase.

After the definition of the inclusion and exclusion criteria, a research protocol, used in the selection process of the studies in this work was defined and is described in Table 01.

This protocol summarizes information on research questions, keywords and boolean operators used in the search process, databases, inclusion and exclusion criteria, methodological validation criteria and analysis of results.

■ **Table 1** Research Protocol.

Research Questions
Q1 – How the introductory programming discipline is identified and structured and in which computer courses are the researches carried out??
Q2 – What are the technical and social skills involved in the process of creating and analyzing an algorithm?
Q3 – What are the methodologies, tools, languages and programming paradigms used in the process of teaching and learning introductory programming?
Search Strategy
All: (“introductory programming course” OR “introduction to programming” OR “cs1”) AND All: (“teaching programming” OR “learning programming”) AND Abstract: (“skill” OR “knowledge” OR “method” OR “tools”) AND (publication date: 01/01/2016 TO 12/31/2021)
Database
ACM Digital Library, IEEE Xplore e ScienceDirect
Inclusion Criteria
1. Papers published between June 2017 and June 2021
Exclusion Criteria
1. Duplicate Records 2. Papers providing only the abstract or unfinished research 3. Papers that do not result from a scientific research 4. Works carried out with students of different levels 5. Works that do not address identity, basic learning conditions and the teaching methods of the discipline of “introductory programming”
Methodological validation criteria
Replication of the search process by another researcher

4.3.3 Selecting Studies

The first step was to conduct an automatic search in the databases, using the string and inclusion criteria defined in the search protocol. In total, 136 papers have been provided of which 90 from the ACM Digital Library, 21 from IEEE Xplore and finally 25 from Science Direct.

The identification and organization of all studies was done. The metadata were exported to a reference management software, which enabled the proper organization and automatic detection of duplicate articles. Eleven documents were excluded for duplicity, resulting 125 selected to the next stage of the process. After reading the respective titles and abstracts, the exclusion criteria 2, 3 and 4 have been applied.

In the end, 70 documents were selected for the full reading phase. Then, the last exclusion criterion has been applied, generating a set of 33 eligible studies (24,6% of the total) to data extraction and analysis step.

4.3.4 Data Classification

In addition to documenting the search and selection strategy used, the authors Kitchenham and Charters [29] indicate the need to also document the strategy used to extract the data contained in the selected studies. Pointing to the importance of constructing a data extraction form, which contains fields that record all documents uniformly – built in parallel with the search protocol – providing the foundation for appraising, analysing, summarising and interpreting a body of evidence.

The first data was collected by the reference management software. Items such as: title, authors, institution, country of publication, type of document, year of publication, abstract, keywords and database were inserted in the data extraction form.

Information such as research questions, data collection technique, type of study and results obtained were also identified in the studies to assist in the final analysis. All information extracted from the primary studies were managed by a spreadsheet program.

In order to standardize the extracted data, a data dictionary [29] was created to restrict the possibilities of categorizing certain form items, establishing a set of permissible values in each field, in order to facilitate the classification process.

5 Results

5.1 Identity of the Introductory Programming discipline (Q1)

The first research question launched by this study aims to determine how researchers, who investigate this topic, refer to this discipline. Thus, analyzing the collected data it was possible to realize that 22 papers refers to the name “introductory program course”. However, similar nomenclatures have also been identified as “introduction to programming” and “introductory computer programming”. In spite of being similar terminologies, all studies use the acronym CS1 to quote the discipline. That term, as previously identified, refers to Computer Science I, the first discipline created for the computer science course, in which the fundamentals of programming are addressed [30]. However, it is still used in the curriculum of all courses in the computing field, aiming to facilitate the transfer of students between educational institutions and also as a keyword for research developed in this field [23].

As can be seen in Table 02, other denominations were also identified and a new proposal is reported in [21]: “Computational Thinking Course”, being considered a lighter version of CS1 designed to engage and stimulate future CS1 students. And training skills such as: problem solving, logical thinking, abstraction, decomposition and pattern recognition, recommendations already foreseen in the ACM/IEEE-CS 2013 Report [44], that arise mainly to address students outside of computer or engineering courses.

This proposal can also be found in specific courses in the computing field, but is classified as an optional discipline and often offered before the beginning of the regular academic period. Identified by the terminology CS0. It is responsible for presenting computer science to students without previous skill, promoting mainly the development of problem solving and mathematical skills, in order to reduce retention rates and withdrawals in the future [14].

Of the papers that identified the audience involved in the study, 73% engaged students of computer courses, namely: computer science, computer engineering, information systems, information technology and software engineering. Also 9% of the research included students from other STEM courses like: electrical, electronics, civil, industrial and telecommunications engineering or bachelor’s degree in mathematics and 18% developed activities with graduates from other courses, that is, outside the computing field, which are also identified in the literature as non-Computer Science (non-CS) [11].

Offer introductory programming courses for audiences with different backgrounds, expectations and with a thematic focus became popular [44]. The great interest in programming in non-CS courses is related to initiatives that seek to make computing accessible to all [12], enabling students to interact consciously with new technologies, allowing the acquisition of the ability to read, write, analyze and modify program codes.

■ **Table 2** Introductory Programming Identity.

Term	References
Algorithms and Problem Solving	[22]
Algorithms and Programming I	[52]
Computational Thinking Course	[21]
Introductory Computer Programming	[51]
Introductory Programming Course	[63, 33, 25, 47, 1, 2, 55, 58, 50], [13, 7, 38, 46, 12, 28, 4, 59, 56], [57, 6, 41, 48]
Introduction to Programming	[49, 20, 34, 17]
Programming 1	[15]
Programming Course	[39]

With regard to the approaches, most of the research focused on the use of a certain programming language. Leaving aside the emphasis on coding, 33% ([33, 39, 47, 2, 58, 34, 21, 52, 22, 46, 41]) provided a broader introduction to the concepts of programming, using algorithmic as an alternative approach.

This last perspective is analogous to the “Algorithms-first” model, proposed in CC2001 [18], which emphasizes the importance of students working with a variety of data and control structures, without having to deal with the specificities that programming languages inevitably introduce.

The research studies that used this approach were developed in the courses of: computer science [33, 2, 34] and [46], computer engineering [52, 41] and [58], information systems [58] and software engineering [58]. This information corroborates with the data published in CC2020 about the degree of importance that algorithms and data structure have in each course.

5.2 Technical and Social Skills (Q2)

Considering the 33 articles analyzed, 29 identified one or more previous skills needed or expected at the end of an introductory programming course. They are identified in table 03 and were gathered in two groups: technical skills (related to programming) and general skills.

Problem solving was the most cited skill, which it is identified by the computer science – CS2013 ([44]) and computer engineering – CE2016 ([43]) reports and defined by [34] as being the ability to understand a given context, identify key information and build a plan to solve it. To plan a solution, the programmer needs to divide the problem into smaller parts, analyze input and output data, and formulate the necessary steps for resolution [62]. The ability to divide a large and complex problem into parts that are manageable to solve, test, and maintain is also known as decomposition. Another skill identified in three papers as a prerequisite for learning programming. [25] states that learn to decompose a computational problem facilitates the software development process, but [13] emphasize that it is not a determining factor of success.

Besides decomposition, another skill, also related to problem solving, is algorithmic thinking. It is defined by [34] as a set of skills connected to building and analyzing algorithms; problem analysis; detailed specification/description of the problem; definition of the necessary actions and construction of an algorithm to solve the problem.

■ **Table 3** Technical and General Skills.

Skill	References	Category
Abstraction	[25, 20, 34, 7, 22, 46]	Technical Skill
Algorithmic Thinking	[25, 47, 20, 15]	Technical Skill
Decomposition	[33, 25, 58, 20, 34, 52, 57]	Technical Skill
Debugging	[51, 2, 55, 58, 7, 59]	Technical Skill
Mathematical Skill	[46, 28]	Technical Skill
Problem Solving	[51, 33, 25, 2, 58, 20, 34, 50, 52, 56, 57]	Technical Skill
Critical Thinking	[51, 20, 21, 52, 59]	General Skill
Communication, creativity, persistence, voluntary participation, perseverance and trust	[51, 28, 41, 10]	General Skill
Spatial Visualization	[17]	General Skill
Team Work and collaboration	[52, 22, 28, 59]	General Skill

In six publications the importance of abstraction was discussed. [61] claims that it is the process used in setting patterns, generalized from specific instances and parameterization in order to capture essential properties that are common to a set of objects. The author also describes that an algorithm is an example of a process abstraction, that starts by receiving input values (input), running a sequence of steps (algorithm), and producing a result (output) to satisfy a given objective (problem).

These skills (abstraction, decomposition, algorithmic thinking and problem resolution) are the key concepts of computational thinking, proposed by [60] and defined as a set of skills that allows us to recognize aspects of computing in everyday life. Beyond computational thinking, another feature developed throughout the discipline and evidenced by [51, 2, 55, 58, 7] and [59] is the debugging. Defined as the process of finding and reducing errors in a code [51], it is identified as the most neglected topic in teaching introductory programming. [59] have identified that software developers tend to debug codes using a scientific method that requires cycles of generations and hypothesis testing. This process, according to the authors, promotes a deeper learning and provides an insight into how critical the individual is and the strategies used. However, the curriculum tends not to focus on teaching strategies that promote critical thinking [59]. Authors such as [20] and [52] claim that this ability is usually noticeable only in experienced programmers, as it is related to the individual's level of maturity.

Mathematical knowledge is cited as necessary and closely related to problem solving, decomposition, and abstraction skills. The lack of this ability can influence programming learning [46] and [28]. However, [34] rejects the hypothesis of correlation between grades obtained by high school students in mathematics and learning programming. Nevertheless, mathematical skills can be used to identify cognitive problems in students as incapable of connecting algorithmic thinking to other mathematical concepts [13]. Associated with mathematical skills, geometry and spatial visualization is also present in the collected data, which influences the understanding and the ability to mentally manipulate a two/three-dimensional figure in space [54]. [17] exposes an example in which a 3D object is placed on a table, and the student needs to imagine how the object will be rotated, without having any physical/real interaction or change of perspective.

And finally, teamwork, communication, creativity, persistence, voluntary participation, perseverance and trust are social skills identified in eight different papers. The authors [51] and [52] emphasized several aspects related to the attitude of students and [22] sustain that knowledge is built from the participation and collaboration between peers. CC2020

corroborates with this statement, and clarifies that all computer courses emphasize the professional knowledge required of each professional, including problem solving, critical thinking, communication, and teamwork [19].

In summary, two areas of computational thinking were outlined in the collected data: Computational practices (algorithm, decomposition, abstraction, etc.) and computer perspectives, i.e. the understanding that students have of themselves and their interaction with others and with technology [8]. This idea broadens the original definition of computational thinking proposed by [60], changing the individualistic conception of programming to a vision that includes focusing on social dimensions, called computational participation [26]. This new conception explore computational practices and perspectives that together enable insight into sociological and cultural dimensions, with an emphasis on learning to code so that learners are able to meaningfully participate as critical thinkers, as well as producers, consumers, and distributors of technology [26].

5.3 Methodologies, Tools, Languages and Programming Paradigms(Q3)

Many teaching strategies have been used and reported. However, a trend towards the use of traditional approach was identified: where the teacher reviews the content, explain the terms and concepts followed by paper-based programming exercises ([10, 50] and [52]). The practical activities are developed later in a laboratory with the support of software/tools ([63, 51, 39, 47, 55, 20, 7, 52, 15, 4, 59, 56, 57, 6] and [48]) or hardware ([10] and [50]). Facing the constraints of using the traditional teaching method, or in order to increase students' motivation ([21, 39, 47, 1, 56]), reduce failure rates in CS1 ([34, 56, 25, 20]), attend classes with large numbers of students ([55, 50] and [41]), active methods have been incorporated as innovative teaching strategy and are cited in Table 4.

■ **Table 4** Methodologies.

Teaching Strategy	References
Active Learning/Peer learning	[21, 22]
Blended Learning	[38]
E-learning	[25, 1, 12, 48]
Flipped Classroom	[25, 2]
Gamification	[39, 38]
Peer Programming	[49, 21, 22]
Problem Based Learning	[52]
Project Based Learning	[10, 22, 46]
Storytelling/Storyboard	[58, 38]
Tradicional Classes	[63, 51, 47, 55, 20, 50, 7, 52, 15, 4, 59, 56, 57, 6, 48]

This scenario, in which traditional classes are enriched with laboratory practice are foreseen in the reports published in 2013 (computer science), 2014 (software engineering) and 2016 (computer engineering). The suggestion for change is clear and incisive from the CC2020, which suggests that exploring new methods of learning can augment the learning of knowledge and allow students to interact with each other to develop new skill sets as well as to develop both communication and teamwork skills by studying with others [19].

About programming languages and paradigms, 57,5% of the authors used a programming language in the discipline under investigation. The summary of these data is identified in Table 5 and from this information it was possible to correlate which programming paradigms the courses chose to use.

A programming paradigm, according to [31] is a way to classify programming languages according to their functionalities, it will determine how the program will be structured and executed. Most of the research studies opted for the use of the object-oriented paradigm, followed by five papers that addressed the structured paradigm and two multi-paradigm. The courses that used one of the programming languages, also associated the use of an Integrated Development Environment (IDE), a software for building applications that combines common developer tools into a single graphical user interface (GUI). As an example were mentioned: Spyder, Visual Studio, ArduinoStudio and Eclipse.

■ **Table 5** Programming Languages used in CS1.

Programming Languages	Paradigms	References
C	Structured	[63, 1, 7, 52, 46]
C++	Object-oriented	[39, 47, 10]
Java	Object-oriented	[25, 49, 13, 12]
Phyton	Multi-paradigm	[20, 21]
C#	Object-oriented	[15]

Research such as [20] and [10] also integrated the use of an Arduino, an electronic prototyping platform which allows the development and control of interactive systems of low cost. Another particularity was the adoption of a code review tools used by [25, 49, 1, 55, 7, 59] e [48] in order to facilitate the work done by the teachers in correcting coding exercises, consequence of a large number of students per class.

Despite the facilities offered by these tools, [7] and [6] proved the importance of a qualitative feedback, so that students can identify not only where the syntax error is, but also what cause the error. Others ([33, 58, 22, 15]) have made the option to use block-based programming environments, such as: Alice, Inventior app, Blockly games, code.org, gameblox, Pencil code, microsoft makecode and Scratch. These platforms allow more accessible programming environments than programming languages, as they use graphical interfaces that enable programs by dragging and dropping blocks.

Studies ([33, 34, 13, 28]) in which tools or code evaluation techniques have been developed, did not mention methods of teaching or learning introductory programming. In contrast, [21, 22] and [38] cited the use of one or more strategies in their research.

Although 54% of studies referenced the use of innovative methods, but most described conducting uniquely a single experiment. Only [10] narrates a four-year experience with students of computer engineering, software engineering and information systems courses.

6 Conclusion

A systematic review was performed with the analysis of 33 recently published studies obtained from ACM, IEEE and ScienceDirect databases in order to understand in depth, the introductory programming discipline in higher education.

For the first research question, the data revealed that most studies mentioned the name “Introductory Program Course”. In addition to this, the acronym CS1 was also widely used as a synonym for “introductory program course”. Thus, in future work, the authors will opt to use the term and the acronym to indicate the first course in which students have the first contact with the fundamentals of programming.

Throughout the history of computing, the structure of the CS1 discipline has been the subject of intense debate. Many strategies have been proposed and numerous discussions have been raised around. As explained, the analyzed papers present several approaches.

Some have focused on the core concepts of software development associated with a particular programming language, others put aside the emphasis on programming and provide a broader introduction to the concepts, with an emphasis on algorithmic. And the courses that were involved in the development of these reflections were: Computer Science, Computer Engineering, Software Engineering and Information System. This corroborates with the CC2020 report about the level of importance that this discipline has in each course.

For Q2, there was a concern in stimulating technical skills linked to programming. On the other hand, skills such as critical thinking, communication, creativity, persistence, voluntary participation, perseverance, trust and teamwork emerged in twelve studies. 69% of the papers that identified the use of some active method in teaching-learning CS1, identified one or more social skills developed in programming practice. These data lead us to believe that knowledge and skills, be they technical or social are clearly identified, transferred and achieved through the practices described. However, research has not yet identified this triad as part of a competency-based model.

Finally, in response to Q3, it was noted that instead of a particular programming language or paradigm being favored, a list of programming languages and paradigms are successfully used in the courses. As this systematic review collected data that refers only to CS1, it was not possible to observe whether the analyzed courses address more than one paradigm in their programs. But, restricting the students' experience to just one paradigm can make the transition to the next more complicated [64]. Shifting the focus from the programming languages and paradigms to the tools, it was observed that the courses used different platforms, that can bring the classroom learning closer to professional contexts, as is the case of IDE.

The same applies for teaching and learning strategies. Methods like pair programming, a practice derived from the agile software (EXtreme programming - XP) and other methods like problem-based and design-based learning, are techniques that provide the development of the triad: knowledge, technical and social skills, in a practical context.

In conclusion, this systematic literature review in the area of introductory programming education over the past 5 years, have explored a variety of themes in the computing field, making at least the following contributions: identifying a standard identity for introductory programming discipline; identifying the aspects of CS1 have been focus of publication; summarizing the strategies, tools and technology used in teaching and learning programming; and highlighting the evidence of the use of a competency-based model in learning-teaching process.

This work is part of a PhD thesis and as future work the authors intend to further explore these issues and to focus on how the dimensions of computational participation may influence the learning processes of introductory programming, promoting the skills revealed in this review.

7 Limitation of Our Systematic Review

This work shares the most limitations of systematic review method: the bias in selecting articles and in data extraction due to our choices of eligibility criteria. Furthermore, other limitation lies to the fact that the research was not supplemented with a complementary process or made use of a quality evaluation to selecting papers. These limitations were addressed developing a strong protocol that answer the search problem and using a combined manual and automatic search. For ensuring the quality we selected papers based on the characteristics of the studies, described in item "Data Classification" and used different types of databases.

References

- 1 Ella Albrecht, Fabian Gumz, and Jens Grabowski. (R08) Experiences in Introducing Blended Learning in an Introductory Programming Course. In *Proceedings of the 3rd European Conference of Software Engineering Education*, ECSEE'18, pages 93–101, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Seon/ Bavaria, Germany. doi:10.1145/3209087.3209101.
- 2 Saleh Alhazbi and Osama Halabi. (R09) Flipping Introductory Programming Class: Potentials, Challenges, and Research Gaps. In *Proceedings of the 10th International Conference on Education Technology and Computers*, ICETC '18, pages 27–32, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Tokyo, Japan. doi:10.1145/3290511.3290552.
- 3 Mike Barkmin and Torsten Brinda. Analysis of programming assessments — building an open repository for measuring competencies. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, Koli Calling '20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3428029.3428039.
- 4 Brett A. Becker, Catherine Mooney, Amruth N. Kumar, and Sean Russell. (R26) A Simple, Language-Independent Approach to Identifying Potentially At-Risk Introductory Programming Students. In *Australasian Computing Education Conference*, ACE '21, pages 168–175, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual, SA, Australia. doi:10.1145/3441636.3442318.
- 5 Brett A. Becker and Keith Quille. 50 years of cs1 at sigcse: A review of the evolution of introductory programming education research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 338–344, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3287324.3287432.
- 6 Jeremiah Blanchard, Christina Gardner-McCune, and Lisa Anthony. (R30) Dual Modality Instruction & Programming Environments: Student Usage & Perceptions. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, pages 481–487, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual Event, USA. doi:10.1145/3408877.3432434.
- 7 Yoram Bosse, David Redmiles, and Marco A. Gerosa. (R18) Pedagogical Content for Professors of Introductory Programming Courses. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 429–435. Association for Computing Machinery, New York, NY, USA, 2019. doi:10.1145/3304221.3319776.
- 8 Karen Brennan and Mitchel Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association*, Vancouver, Canada, volume 1, page 25, 2012.
- 9 Pearl Brereton, Barbara Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571–583, 2007. doi:10.1016/j.jss.2006.07.009.
- 10 David W. Brown, Sheikh K. Ghafoor, and Stephen Canfield. (R13) Instruction of Introductory Programming Course Using Multiple Contexts. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018, pages 147–152, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Larnaca, Cyprus. doi:10.1145/3197091.3197105.
- 11 Parmit Chilana, Celena Alcock, Shruti Dembla, Anson Ho, Ada Hurst, Brett Armstrong, and Philip Guo. Perceptions of non-CS majors in intro programming: The rise of the conversational programmer. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 251–259, 2015. doi:10.1109/VLHCC.2015.7357224.
- 12 Rodrigo Silva Duran, Jan-Mikael Rybicki, Arto Hellas, and Sanna Suoranta. (R24) Towards a Common Instrument for Measuring Prior Programming Knowledge. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 443–449. Association for Computing Machinery, New York, NY, USA, 2019. doi:10.1145/3304221.3319755.

- 13 Nikita Dümmel, Bernhard Westfechtel, and Matthias Ehmann. (R17) MuLE: A Multiparadigm Language for Education. The Object-Oriented Part of the Language. In *Proceedings of the 4th European Conference on Software Engineering Education, ECSEE '20*, pages 32–41, New York, NY, USA, 2020. Association for Computing Machinery. event-place: Seon/Bavaria, Germany. doi:10.1145/3396802.3396806.
- 14 Cenk Erdil and Darcy Ronan. Implementing CS0 with Computer Science Principles Curriculum. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education, SIGCSE '19*, page 1272, New York, NY, USA, 2019. Association for Computing Machinery. event-place: Minneapolis, MN, USA. doi:10.1145/3287324.3293791.
- 15 Osman Erol and Adile Aşkın Kurt. (R22) The effects of teaching programming with scratch on pre-service information technology teachers' motivation and achievement. *Computers in Human Behavior*, 77:11–18, 2017. doi:10.1016/j.chb.2017.08.017.
- 16 Katia Romero Felizardo, Emilia Mendes, Marcos Kalinowski, Érica Ferreira Souza, and Nandamudi L. Vijaykumar. Using forward snowballing to update systematic reviews in software engineering. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '16*, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2961111.2962630.
- 17 José Figueiredo and Francisco García-Peñalvo. (R32) Teaching and Learning Tools for Introductory Programming in University Courses. In *2021 International Symposium on Computers in Education (SIIE)*, pages 1–6, 2021. doi:10.1109/SIIE53363.2021.9583623.
- 18 The Joint Task Force for Computing Curricula 2001. Computing curricula 2001. *Journal on Educational Resources in Computing (JERIC)*, 1(3^a), 2001. Publisher: ACM New York, NY, USA.
- 19 The Joint Task Force for Computing Curricula 2020. *Computing Curricula 2020: Paradigms for Global Computing Education*. Association for Computing Machinery, New York, NY, USA, 2020.
- 20 G. Cooper, R. Walker, E. Hill, and N. Waksanski. (R12) Incorporating IoT and Data Analytics in an Introductory Programming Course. In *2020 15th International Conference on Computer Science & Education (ICCSE)*, pages 169–175, August 2020. Journal Abbreviation: 2020 15th International Conference on Computer Science & Education (ICCSE). doi:10.1109/ICCSE49874.2020.9201863.
- 21 Luke Gusukuma, Austin Cory Bart, Dennis Kafura, and Jeremy Ernst. (R16) Misconception-Driven Feedback: Results from an Experimental Study. In *Proceedings of the 2018 ACM Conference on International Computing Education Research, ICER '18*, pages 160–168, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Espoo, Finland. doi:10.1145/3230977.3231002.
- 22 H. Amer and S. Harous. (R20) Smart-Learning Course Transformation for an Introductory Programming Course. In *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*, pages 463–465, July 2017. Journal Abbreviation: 2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT). doi:10.1109/ICALT.2017.91.
- 23 Matthew Hertz. What Do "CS1" and "CS2" Mean? Investigating Differences in the Early Courses. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education, SIGCSE '10*, pages 199–203, New York, NY, USA, 2010. Association for Computing Machinery. event-place: Milwaukee, Wisconsin, USA. doi:10.1145/1734263.1734335.
- 24 Theresia Devi Indriasari, Andrew Luxton-Reilly, and Paul Denny. Improving student peer code review using gamification. In *Australasian Computing Education Conference, ACE '21*, pages 80–87, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3441636.3442308.
- 25 J. Skalka, M. Drlik, and J. Obonya. (R05) Automated Assessment in Learning and Teaching Programming Languages using Virtual Learning Environment. In *2019 IEEE Global Engineering Education Conference (EDUCON)*, pages 689–697, April 2019. Journal Abbreviation: 2019 IEEE Global Engineering Education Conference (EDUCON). doi:10.1109/EDUCON.2019.8725127.

- 26 Yasmin B. Kafai and Quinn Burke. Computational Participation: Teaching Kids to Create and Connect Through Code. In Peter J. Rich and Charles B. Hodges, editors, *Emerging Research, Practice, and Policy on Computational Thinking*, pages 393–405. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-52691-1_24.
- 27 Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. A systematic literature review of automated feedback generation for programming exercises. *ACM Trans. Comput. Educ.*, 19(1), September 2018. doi:10.1145/3231711.
- 28 Natalie Kiesler. (R25) Towards a Competence Model for the Novice Programmer Using Bloom’s Revised Taxonomy - An Empirical Approach. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE ’20, pages 459–465, New York, NY, USA, 2020. Association for Computing Machinery. event-place: Trondheim, Norway. doi:10.1145/3341525.3387419.
- 29 Barbara Kitchenham and Stuart Charters. *Guidelines for performing systematic literature reviews in software engineering*. Citeseer, 2007.
- 30 Elliot Koffman, Philip Miller, and Caroline Wardle. Recommended Curriculum for CS1, 1984. *Commun. ACM*, 27(10):998–1001, October 1984. Place: New York, NY, USA Publisher: Association for Computing Machinery. doi:10.1145/358274.358279.
- 31 Sirojiddin Komolov, Nursultan Askarbekuly, and Manuel Mazzara. An Empirical Study of Multi-Threading Paradigms Reactive Programming vs Continuation-Passing Style. In *2020 the 3rd International Conference on Computing and Big Data*, ICCBD ’20, pages 37–41, New York, NY, USA, 2020. Association for Computing Machinery. event-place: Taichung, Taiwan. doi:10.1145/3418688.3418695.
- 32 Patrick Korber and Renate Motschnig. The effects of pair-programming in introductory programming courses with visual and text-based languages. In *2021 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, 2021. doi:10.1109/FIE49875.2021.9637186.
- 33 L. Carlos Begosso, L. Ricardo Begosso, and N. Aragao Christ. (R03) An analysis of block-based programming environments for CS1. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, October 2020. Journal Abbreviation: 2020 IEEE Frontiers in Education Conference (FIE). doi:10.1109/FIE44824.2020.9273982.
- 34 L. M. de Souza, B. M. Ferreira, I. M. Félix, L. de Oliveira Brandão, A. A. F. Brandão, and P. A. Pereira. (R14) Mathematics and programming: marriage or divorce? In *2019 IEEE World Conference on Engineering Education (EDUNINE)*, pages 1–5, March 2019. Journal Abbreviation: 2019 IEEE World Conference on Engineering Education (EDUNINE). doi:10.1109/EDUNINE.2019.8875849.
- 35 Philip I.S. Lei and António José Mendes. A systematic literature review on knowledge tracing in learning programming. In *2021 IEEE Frontiers in Education Conference (FIE)*, pages 1–7, 2021. doi:10.1109/FIE49875.2021.9637323.
- 36 Madeleine Lorås, Guttorm Sindre, Hallvard Trætteberg, and Trond Aalberg. Study behavior in computing education – a systematic literature review. *ACM Trans. Comput. Educ.*, 22(1), October 2021. doi:10.1145/3469129.
- 37 Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. Introductory programming: A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018 Companion, pages 55–106, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3293881.3295779.
- 38 M. F. Ercan and D. Sale. (R21) Teaching programming: An evidence based and reflective approach. In *2020 IEEE REGION 10 CONFERENCE (TENCON)*, pages 997–1001, November 2020. Journal Abbreviation: 2020 IEEE REGION 10 CONFERENCE (TENCON). doi:10.1109/TENCON50793.2020.9293812.

- 39 B. Marín, J. Frez, J. Cruz-Lemus, and M. Genero. (R04) An Empirical Investigation on the Benefits of Gamification in Programming Courses. *ACM Trans. Comput. Educ.*, 19(1), November 2018. Place: New York, NY, USA Publisher: Association for Computing Machinery. doi:10.1145/3231709.
- 40 Rodrigo Pessoa Medeiros, Geber Lisboa Ramalho, and Taciana Pontual Falcão. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, 62(2):77–90, 2019. doi:10.1109/TE.2018.2864133.
- 41 Nathan Mills, Allen Wang, and Nasser Giacaman. (R31) Visual Analogy for Understanding Polymorphism Types. In *Australasian Computing Education Conference, ACE '21*, pages 48–57, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual, SA, Australia. doi:10.1145/3441636.3442304.
- 42 Elisa Yumi Nakagawa, Kátia Romero Felizardo Scannavino, Sandra Camargo Pinto Ferraz Fabbri, and Fabiano Cutigi Ferrari. *Revisão sistemática da literatura em engenharia de software: teoria e prática*. Elsevier Brasil, 2017.
- 43 Task Group on Computer Engineering Curricula. Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering. Technical report, Association for Computing Machinery, New York, NY, USA, 2016.
- 44 Task Group on Computer Science Curricula. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery, New York, NY, USA, 2013.
- 45 Task Group on Information Technology Curricula. *Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology*. Association for Computing Machinery, New York, NY, USA, 2017.
- 46 P. E. Martínez López, D. Ciolek, G. Arévalo, and D. Pari. (R23) The GOBSTONES method for teaching computer programming. In *2017 XLIII Latin American Computer Conference (CLEI)*, pages 1–9, September 2017. Journal Abbreviation: 2017 XLIII Latin American Computer Conference (CLEI). doi:10.1109/CLEI.2017.8226428.
- 47 Fredi E. Palominos, Seomara K. Palominos, Claudia A. Durán, Felisa M. Córdova, and Hernán Díaz. (R06) Challenges in the use of a support tool with automated review in student learning of programming courses. *Procedia Computer Science*, 139:424–431, 2018. doi:10.1016/j.procs.2018.10.260.
- 48 Filipe Dwan Pereira, Samuel C. Fonseca, Elaine H. T. Oliveira, Alexandra I. Cristea, Henrik Bellhäuser, Luiz Rodrigues, David B. F. Oliveira, Seiji Isotani, and Leandro S. G. Carvalho. (R33) Explaining Individual and Collective Programming Students' Behavior by Interpreting a Black-Box Predictive Model. *IEEE Access*, 9:117097–117119, 2021. doi:10.1109/ACCESS.2021.3105956.
- 49 Reinhold Plösch and Cornelia Neumüller. (R07) Does Static Analysis Help Software Engineering Students? In *Proceedings of the 2020 9th International Conference on Educational and Information Technology, ICEIT 2020*, pages 247–253, New York, NY, USA, 2020. Association for Computing Machinery. event-place: Oxford, United Kingdom. doi:10.1145/3383923.3383957.
- 50 James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. (R15) Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research, ICER '18*, pages 41–50, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Espoo, Finland. doi:10.1145/3230977.3230981.
- 51 Vijayalakshmi Ramasamy, Hakam W. Alomari, James D. Kiper, and Geoffrey Potvin. (R02) A Minimally Disruptive Approach of Integrating Testing into Computer Programming Courses. In *Proceedings of the 2nd International Workshop on Software Engineering Education for Millennials, SEEM '18*, pages 1–7, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Gothenburg, Sweden. doi:10.1145/3194779.3194790.

- 52 S. M. Souza and R. A. Bittencourt. (R19) Report of a CS1 Course for Computer Engineering Majors Based on PBL. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 837–846, April 2020. Journal Abbreviation: 2020 IEEE Global Engineering Education Conference (EDUCON). doi:10.1109/EDUCON45650.2020.9125121.
- 53 Leonardo Silva, António José Mendes, and Anabela Gomes. Computer-supported collaborative learning in programming education: A systematic literature review. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 1086–1095, 2020. doi:10.1109/EDUCON45650.2020.9125237.
- 54 Sheryl Sorby. A Course in Spatial Visualization and its Impact on the Retention of Female Engineering Students. *Journal of Women and Minorities in Science and Engineering*, 7:50, January 2001. doi:10.1615/JWomenMinorScienEng.v7.i2.50.
- 55 Kristin Stephens-Martinez and Armando Fox. (R10) Giving Hints is Complicated: Understanding the Challenges of an Automated Hint System Based on Frequent Wrong Answers. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2018*, pages 45–50, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Larnaca, Cyprus. doi:10.1145/3197091.3197102.
- 56 Shelsey Sullivan, Hillary Swanson, and John Edwards. (R28) Student Attitudes Toward Syntax Exercises in CS1. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, SIGCSE '21*, pages 782–788, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual Event, USA. doi:10.1145/3408877.3432399.
- 57 Lasang Jimba Tamang, Zeyad Alshaikh, Nisrine Ait Khayi, Priti Oli, and Vasile Rus. (R29) A Comparative Study of Free Self-Explanations and Socratic Tutoring Explanations for Source Code Comprehension. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, SIGCSE '21*, pages 219–225, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual Event, USA. doi:10.1145/3408877.3432423.
- 58 Damla Topalli and Nergiz Ercil Cagiltay. (R11) Improving programming skills in engineering education through problem-based game projects with Scratch. *Computers & Education*, 120:64–74, 2018. doi:10.1016/j.compedu.2018.01.011.
- 59 Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. (R27) Novice Reflections on Debugging. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, SIGCSE '21*, pages 73–79, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual Event, USA. doi:10.1145/3408877.3432374.
- 60 Jeannette Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006. Publisher: ACM New York, NY, USA.
- 61 Jeannette Wing. Computational Thinking. *J. Comput. Sci. Coll.*, 24(6):6–7, June 2009. Place: Evansville, IN, USA Publisher: Consortium for Computing Sciences in Colleges.
- 62 Lan Wu, Yang Liu, Axi Wang, YuanLi Gong, and ShengQuan Yu. An analysis of Interaction of Cognitive and Social Aspects during Collaborative Problem Solving. In *2021 International Conference on Advanced Learning Technologies (ICALT)*, pages 105–107, 2021. doi:10.1109/ICALT52272.2021.00039.
- 63 Jooyong Yi, Umair Z. Ahmed, Amey Karkare, Shin Hwei Tan, and Abhik Roychoudhury. (R01) A Feasibility Study of Using Automated Program Repair for Introductory Programming Assignments. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017*, pages 740–751, New York, NY, USA, 2017. Association for Computing Machinery. event-place: Paderborn, Germany. doi:10.1145/3106237.3106262.
- 64 Daeng Zuhud. Some Prospective Approaches for the Shift of Programming Paradigms. In *Proceedings of the 2013 International Conference on Information Systems and Design of Communication, ISDOC '13*, pages 87–93, New York, NY, USA, 2013. Association for Computing Machinery. event-place: Lisboa, Portugal. doi:10.1145/2503859.2503873.