A Systematic Literature Review on Predictive Cognitive Skills in Novice Programming

Jucelio S. Santos*, Wilkerson L. Andrade*, João Brunet*, Monilly Ramos Araujo Melo*
*Federal University of Campina Grande (UFCG), Campina Grande, Brazil
jucelio@copin.ufcg.edu.br, {wilkerson, joao.arthur}@computacao.ufcg.edu.br, monillyramos@gmail.com

***Abstract*—This Research Full Paper presents a Systematic Literature Review (SLR) investigating predictive programming skills and strategies to foster and measure such skills. Predictive skills are specific skills that precede a milestone or the development of other more structured skills, in this case, skills that precede programming skills. Introductory Programming Courses (CS1) feature students with a wide range of skill levels. This difference causes educators to get lost when formulating their practices to teach such a diverse group. There is no clear vision about what previous skills professors should foster before a student enters a CS1, much less which strategies to foster/measure such skills. Due to this limitation, we present/plan an RSL following the guidelines proposed by Kitchenham (2004) to achieve these goals. Our main results are a) Predictive programming skills are problem-solving, abstract thinking, mathematical reasoning, and cognitive flexibility; b) Different researchers use different approaches to teach programming skills based on different educational theories, teaching frameworks, or educational approaches; c) Several studies use the Classical Test Theory as a way to measure predictive programming skills. However, some universities have adopted other theories for this practice, such as Item Response Theory.**

I. Introduction

Programming Introduction Course (CS1) is a challenging course for novice students that conventionally covers problem-solving skills, basic programming concepts, syntax, and semantics of a programming language to formulate solutions [5]. The classroom teaching begins with declarative knowledge concepts followed by program writing skills. The most important activity for students while learning to program is how to create their programs. Therefore students write programs after being taught the syntax rules and a few examples [5]. However, educators need to emphasize precursor skills for writing code.

The learners learning the program need to think like a programmer, which involves making the necessary plans to solve problems, and they must learn to write code corresponding to those plans. To be a good programmer means to be creative and have an invention. Specific knowledge within programming can be learned based on each programmer's needs [56].

However, students enter the introductory classroom with a wide range of skill levels, particularly their background in programming and problem-solving skills [49]. In the first phase of learning programming, low-achieving students find it difficult and will not be inspired to do programming [56]. They often face difficulties during problem analysis, planning, and design solutions [51].

This difference in student skill levels causes educators to get lost in formulating their practices to teach such a diverse group. There is no consensus on how educational institutions should pre-proceed with these students, what previous skills teachers should foster before a student enters a CS1, and what strategies to foster/measure such skills. Previous research and recent Systematic Literature Review (SLR) cover articles that address students, teachers, curriculum, assessment, and trends in CS1 or focus on categorizing introductory programming challenges in Higher Education [38]. Therefore, there is a need for an SLR that provides a better understanding and categorization of predictive programming skills and disclosure strategies to foster/measure such skills. To achieve these goals, we planned and carried out an SLR following the guidelines proposed by Kitchenham et al. [28].

We have organized this article as follows. In Section II, we briefly discuss related work. In Section III, we present the methodology applied to the conduct of this article. In section IV, we present the results and their discussions in section V. Finally, in section VI, we present the conclusions and suggestions for future work.

II. Related Work

As far as we know, no SLR categorizes students' skills in learning to program. Existing studies address the challenges of teaching/assessment in CS1, alternative methods, formative feedback, or mention some programming skills without classification or cognitive hierarchy [38]. These studies are essential because they provide an overview of the fundamentals in CS1 and indicate the skills that professors should foster and assess, but they do not preclude which skills should be worked on in a CS1 at the predictive level.

Both works characterized the challenges in CS1. They highlighted some fundamental skills for a beginner student to learn to program and the difficulties they faced in this process. The main teaching challenge concerns the lack of adequate methods and tools for personalized teaching. Problem-solving remains one of the leading learning challenges, followed by motivation and engagement and difficulty learning programming languages' syntax [38].

This paper presents an SLR that categorizes predictive programming skills and highlights strategies that foster/measure those skills. We believe that the contributions of this work guide Higher Education institutions to prepare curricula/assessment tools to help those entering CS1 learn.

III. Research Methodology

We performed an SLR to gain knowledge about predictive skills related to programming and how to foster and measure them, following the guidelines proposed by Kitchenham [28]. First, we defined the review protocol that allowed us to identify the objectives, the research questions, the scope, the strategy search, and query selection studies, as presented below.

A. Objectives and scope

This SLR has the following objectives:

- **Objective 1:** identify the predictive skills involved in teaching programming to beginning students;
- **Objective 2:** identify studies which foster predictive programming skills;
- **Objective 3:** identify studies which measure predictive programming skills.

B. Research questions

To meet the proposed objectives, we divided the research question (RQ) of this SLR as follows:

- **Primary question (PQ):** what are predictive programming skills?
- **Secondary question (SQ1):** how to foster predictive programming skills?
- **Secondary question (SQ2):** how to measure predictive programming skills?

C. Search strategy

Our search strategy consisted of an online search in the 4 (four) main digital libraries with high relevance for Software Engineering, namely: IEEE Xplore [2], ACM Digital Library [1], Science Direct [3], and Scopus [4].

Search keywords are essential for the quality and results coverage, so they should be carefully defined to search in online digital libraries. The query has a raw string composed of 3 (three) terms, one standard term and the other a combination of identified keywords' synonyms. The first term (A) refers to Introduction to Programming and its synonyms, the second term (B) refers to Novice Programming or CS1, and the third term (C) refers to skill and its synonyms.

We performed a pilot search on the IEEE Xplore to evaluate the query. After some trial and error with a databases range, we selected a combined search phrase that seemed to capture the interest area:

> ((“introduction to programming” OR “programming course” OR “programming language” OR “programming learning” OR “learning programming” OR “programming teaching” OR “teaching programming”) AND (“novice programming” OR CS1) AND (skill OR expertise OR ability OR proficiency OR experience OR art OR technique OR facility OR talent OR intelligence OR craft OR competence OR readiness OR accomplishment OR knack OR ingenuity OR finesse OR aptitude OR dexterity OR cleverness OR quickness OR adroitness OR handedness OR skilfulness))

D. Study selection strategy

We explicitly set the inclusion and exclusion criteria in reviewing the protocol for this SLR, namely:

- **Inclusion criteria (IC):** studies that define predictive programming skills in CS1.
- **Exclusion criteria (EC1):** incomplete studies, unavailable and/or duplicate studies (papers with the same or updated version, keeping only the most recent);
- **Exclusion criteria (EC2):** is a short paper or SLR;
- **Exclusion criteria (EC3):** is not written english;
- **Exclusion criteria (EC4):** published before last decade;
- **Exclusion criteria (EC5):** studies that do not define predictive programming skills to beginning students.

We organized our study selection process in four distinct phases, described below:

- **Phase 1:** as a preliminary selection, we performed the queries, applied EC4 and EC5 defined the study group that served for the second phase;
- **Phase 2:** based on the titles, abstracts, and words of the preliminary selection studies, we determined and kept which were studies relevant;
- **Phase 3:** based on the inclusion and exclusion criteria and full reading, we reviewed relevant studies from the previous phase;
- **Phase 4:** one specialist evaluated and validated the selected studies, with the studies' inclusion or exclusion possibility.

E. Data extraction

We prepared a form to extract and synthesize relevant study data to answer the SLR protocol research questions. Data extraction aims to summarize data from primary studies.

The SLR process defined in the previous section resulted in 5063 articles found in the four databases. After Phase 1, we selected 262 studies for Phase 2, in which we identified 56 studies [5] [6] [7] [8] [9] [10] [11] [13] [12] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [29] [30] [31] [32] [33] [34] [35] [36] [37] [39] [40] [41] [42] [43] [44] [45] [46] [47] [49] [50] [51] [52] [53] [54] [56] [57] [58] [59] [60] as relevant after a careful reading of the paper. In the works in which the principal researcher had difficulties accepting or not the paper in the inclusion criteria, we consulted university professors specialized in teaching programming to solve these doubts. We show the study's details in Table I by the library and Table II by phase.

Table I
Details of study search and selection by database

Digital Library	**Search result**
IEEE Xplore	421
ACM Digital Library	1,807
Science Direct	297
Scorpus	2,538
Total studies	**5,063**

Table II
SEARCH PHASE STUDY DETAILS

Phase	Descriptions	Included	Excluded
Phase 1	Search results	5,063	0
Phase 2	Title and abstract selection	262	4,801
Phase 3	Full reading selection	56	206
Phase 4	Selection validated by a specialist	56	0

IV. RESULTS

This section details the SLR result. We surveyed from February to April 2020. Next, we answer each research question defined in Section III.

A. What are predictive programming skills?

We have presented predictive programming skills in Table III.

Table III
PREDICTIVE PROGRAMMING SKILLS

Skill	Studies
Problem Solving	[5] [7] [8] [9] [10] [11] [12] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [29] [30] [31] [32] [34] [35] [36] [37] [39] [40] [41] [42] [43] [46] [47] [49] [50] [51] [53] [56] [57] [58] [59] [60]
Abstract Thinking	[11] [33] [44] [45] [46] [50] [54]
Mathematical Reasoning	[6] [13] [19] [52]
Cognitive Flexibility	[15]

1) Problem Solving: several studies point to evidence that the Problem Solving skill is a Predictive Programming skill [9] [15] [24] [26] [27] [29] [32] [34] [35] [36] [39] [46] [50] [58] [59]. These studies report the relationship between the skills perceived by students in Problem Solving and academic performance in Introductory Programming Assessment tasks formative and summative program.

In addition, some paper is aims to study and improve the Problem Solving skills of undergraduate students in Computer Science [10] [11] [16] [17] [19] [30]. Some studies developed a conceptual problem-based online learning framework for students' Problem Solving and Programming skills [5] [31] [40] [42] [43] [49]. Other studies investigated, through a survey, the self-efficacy of programming in the context of Problem Solving skills from the point of view of participants such as specialists, professionals, students or professors [60] [25] [37] [41] [57]. And, a lot of studies have focused on propose a solution, approach or instrument to help students improve Problem Solving skills [5] [7] [8] [12] [14] [18] [22] [21] [20] [23] [37] [40] [42] [43] [47] [51] [53] [56] [57].

2) Abstract Thinking: some studies create evidence that the Abstract Thinking skill is a predictive programming skill [44] [45] [46] [50] [54]. Abstract thinking positively affects understanding the most complex part of the program [45] [44]. Moreover, stimulating this skill is essential in Computer Science [45].

3) Mathematical Reasoning: Some studies prove that the Mathematical Reasoning skill is a Predictive Programming skill [19] [52]. The other preliminary studies explore the impact of Mathematical ability in programming learning in Higher Education [6] and describe a systematic process of introduction of Mathematical Reasoning skills in the CS1 curriculum, and evaluation of how well students have learned this skill [13].

4) Cognitive Flexibility: recently, a study established the effect of cognitive flexibility in programming, indicating that this ability is predictive programming. The study showed that students who had stimulation in cognitive flexibility skills performed better in the activities schedule [15].

B. How to foster predictive programming skills?

In Table IV, we present the methods, approaches, and tools that foster predictive programming skills.

Table IV
METHODS, APPROACHES AND TOOLS THAT FOSTER PREDICTIVE PROGRAMMING SKILLS

Skill	Methods	Approaches	Tools
Problem Solving	Problem-Based Learning [5] [50] DECSAR [10] Bricolage Programming [47] ADRI Model [50] IDEAL Model [50]	Ne-Curse [16] [17] [18] Flowchart [21] [22] [23] [51] Sliding Window [40] Task Specific Program Design Teaching [55]	Kattis [7] Visual Problem-Solving Environment for Programming Learning [9] GameSalad [12] FMAS [20] [24] ADRI-based editor [34] [35] PROSOLVE [36] [37] Scratch [39] [43] [58] JavaGrinder [42] Karel the Robot [43] Turtle Graphic [43] Marine Biology Case Stud [43] Alice [43] Greenfoot [43] Roboot Tool [57]
Abstract Thinking	GOBSTONE [33] Game-themed programming [50]	Armoni's framework [54]	-
Mathematical Reasoning	-	-	-
Cognitive Flexibility	-	-	-

Based on Table IV, we can see limitations in studies that report approaches and tools that foster Mathematical Reasoning skills and Cognitive Flexibility. In Abstract Thinking, there are no tools that foster this skill.

C. How to measure predictive programming skills?

In Table V, we present the paper-and-pencil instruments that measure the predictive programming skills.

Table V
INSTRUMENTS THAT MEASURE PREDICTIVE PROGRAMMING SKILLS

Skill	Instruments
Problem Solving	Yuan Ze University [9] Bartin University [14] [15] Aladdin and his flying carpet [26] Programmer for International Student Assessment (PISA) [32] Brunel University [29] Bebras [53]
Abstract Thinking	Jeju National University [44] [45] Bebras [53] University of Turku [59]
Mathematical Reasoning	Limestone College [13] University of Coimbra [19] Universidade de São Paulo [52]
Cognitive Flexibility	Bartin University [15]

V. DISCUSSIONS

This section discusses the data presented in Section IV on the studies that characterize predictive programming skills, highlighting methods, approaches, tools, and instruments that foster/measure such skills.

A. Predictive programming skills

Predictive skills are specific skills that precede a milestone or develop other, more structured skills. In this case, the skills that precede programming skills are:

1) Problem Solving: is a valuable concept that describes the conscious effort in the controlled processing of information that identifies, discovers, or invents a solution to a problem [5]. Problem Solving requires much more than having a valid mental model of basic constructions. Transmitting generic problem-solving skills requires a more project-oriented constructivist model, where students design and build artifacts [57].

Problem Solving plays a significant role in people's technical capabilities in general, but even more so for students of applied science and engineering, such as Computer Science [10]. Several fields use problem-solving as an essential skill for professional development; therefore, it is helpful to have some basic familiarity with problem-solving, regardless of work or study [56].

Designing complex systems is like designing code; understanding how smaller parts of the code create a larger functional unit can help understand complex real-world systems from simpler parts (engines, buildings, power plants). Problem-solving skills are essential because merely understanding what questions to ask is as important as being able to seek answers [56].

Problem-solving skills are integral to understanding the programming domain's precise concepts for beginners in introductory programming courses. However, these skills are mainly addressed in the first lessons of such courses or included in only a few opening chapters of some relevant books [36]. Unfortunately, most universities' traditional curricula do not teach these skills to the level desired by employers [10]. As a result, introductory programming courses have a high failure, and dropout rates [36]. Further research in this field has revealed that the lack of ability to solve problems, considered one of the primary deficiencies faced by newcomers, is exacerbated by the language syntax that newcomers use [20].

Problem-Solving skills are measured and stimulated with exercises and training [10]. In this way, students with more effective problem-solving skills find programming easy and can master programming with little or no difficulty, regardless of the programming environment. On the contrary, students with poor problem-solving skills find programming challenging to understand and are often unable to master it [39].

Problem Solving produces an objective and quantifiable benefit to students' programming ability [29]. Also, having a valuable knowledge of problem-solving techniques is vital [56]. Because the ability to solve problems correlates significantly with performance in programming assignments [32].

2) Abstract Thinking: Programming skills involve not only problem solving but also a style of thinking. A thinking style is a preferred way of thinking, be it global (abstract) or local (concrete) thinking [33]. It is an essential strategy for computer program development, as the student needs to test and evaluate his computer programming solutions. To properly test and evaluate these solutions, students need good metacognitive skills. In other words, students need to be able to "think about think."

Abstract thinking influences the way people learn information, form judgments and regulate behavior. For example, abstract thinking can directly affect understanding a program that uses the iteration structure in iteration structures. If the student's abstract thinking level is higher, he will better understand the program with an iteration structure. When the teacher provides the student with an uncomplicated programming problem, the abstract thinking skill is not discriminatory in the student's evaluation. However, when a complex programming problem, the high level of abstract thinking capacity plays an essential role in solving the problem [44].

There is a moderating effect on the ability to program between the level of familiarity with programming and abstract thinking [44] [45]. When students have a high level of abstract thinking skills, their programming skills are improved much more than those with a low level of abstraction. When teachers increase students' abstract thinking, the psychological temporal distance also increases. Also, familiarity is a crucial factor in understanding the program; students learn more and practice more about the programming language in the future.

3) Mathematical Reasoning: Programming skills are often associated with problem-solving skills, and certain types of math skills affect the analytical skills that contribute to the experience of learning to program on the computer. In addition to the syntax and semantics of a programming language, logical and mathematical thinking, numeracy and visualization skills, knowledge of algebra and calculus, and knowledge of conditional and recursive functions are essential for designing functions and procedures during the programming activity [6].

To write fully specified, verifiable code, computing students need the ability to reason mathematically about software components and their relationships, model them via mathematical

constructs, and understand and write formal specifications and assertions using the precise language of mathematical notation [13]. There is one correlation between mathematical skills and programming potential. The students consider the logic of programming and teaching methodology as the primary sources of difficulties [52]. Mathematical concepts related to induction, inference, and logical reasoning are among them.

4) Cognitive Flexibility: is one of the most preliminary individual skills that can guide the student during the regulation and evaluation of the solution and the exit in the problem-solving process. Cognitive flexibility allows the student to employ the most effective learning strategies related to the topic under study or identify the steps to solve a problem, find a solution, control the learning process and control the products or opportunities for self-regulation. A recent study shows that cognitive flexibility predicts academic performance in CS1 programming training. Professors should develop this skill in order to improve students' programming performance [15].

B. Fostering predictive programming skills

On the one hand, the number of employees employed in the IT industry grows continuously. On the other hand, there is a growing shortage of IT specialists, especially in software development. This problem can be solved by increasing students' interest in computer programs or decreasing students who leave the course early to learn some courses. While the first solution requires all stakeholders' involvement, influencing students (parents, teachers, schools, friends), in the second, students have already decided to study IT programs. Therefore, it is "easier" to invest time and resources in designing an appropriate educational concept, which could increase the likelihood of students successfully graduating and having the appropriate knowledge and skills required by the job market [50].

However, today's young people are not interested in listening to lectures or watching long-term educational videos. They prefer the immediate use or application of the knowledge and skills obtained. They require the option to learn anytime and anywhere, not just at the university. Also, they prefer to select the knowledge and skills whose usefulness they can imagine or prove in a short time.

Innovative educational models and structures that try to minimize obstacles and meet the requirements and expectations of a new generation of students must use well-known learning approaches and develop the skills and knowledge needed for available taxonomies. Programming teaching is a very complicated educational process. Educational theories and taxonomies are valuable tools for developing learning objectives and assessing student performance, but they do not apply directly to this area. Programming requires students to understand the relevant theory and apply it to solve real problems [50].

Researchers use different approaches to teach programming based on different educational theories, teaching structures, or educational approaches. Some studies have sought to simplify the complexity of acquiring knowledge and skills. On the other hand, others described and mapped very accurately.

1) Problem-solving: There are several models able to foster problem-solving in novices programming, to know: ADRI Model [34], DECSAR Model [10], and IDEAL Model [50].

ADRI Model is a well-known quality assurance model used extensively in the education and business sectors [34]. This model has the following steps: i) the first step (approach) consists of thinking and planning tasks; ii) the second step (deployment) provides a platform to perform or implement tasks; iii) the third step (result) refers to the output or findings as a consequence of the approach and implementation steps; iv) the fourth step (improvement) refers to the results step' conclusions.

The ADRI model has a positive impact on the results achieved by students during the course. The model engages students in their programming skills and provides a new style of presenting examples and exercises that discourage students from using programming shortcuts, significantly reducing dropouts and failures and positively affecting all students who pass CS1 [35].

DECSAR Model is a method to foster problem-solving skills [10]. The DECSAR model is a problem-solving strategy designed to model practical problem-solving. The method has the following steps: i) define the problem; ii) examine the situation; iii) consider the causes; iv) consider the solution; v) act and test, and vi) review the problem solution.

IDEAL Model [50] consists of the following steps: i) identify the problem; ii) define the problem precisely; iii) explore strategies to solve the problem (based on previous knowledge and experience); iv) execute the previously planned strategies; and, v) learn based on observations about the actions' effect and results.

Another strategy teachers in CS1 can use **Problem-Based Learning** (PBL), which follows constructivist principles and support the concept that learners actively build knowledge during the learning process. Likewise, the PBL method presents problems for students to discover what they know, what they do not know, and what they need to learn before solving the problem [5]. PBL is a method that can potentially help students perform better in CS1, in turn affecting their performance in projects [58];

Bricolage Programming is a creative feedback loop that encompasses the written algorithm, its interpretation, and the programmer's perception and reaction to its output or behavior. In a study [47], researchers used an experimental post-test design to explore Bricolage Programming's effects on problem-solving when playing an educational programming game. The study created two versions of the game, one that used a Scratch-like visual programming interface to encourage Bricolage Programming and one that used a more structured visual programming interface. The significant difference in Bricolage Programming between the groups supports the theory that Bricolage Programming occurs when using Scratch-like instructions, and scratch has additional freedom that encourages more movement and instructions' deletion.

However, confirming this basic assumption also highlights the debate's importance about Bricolage Programming's effects on problem-solving skills.

Different approaches foster problem-solving skills: Ne-Curse [18] [16], Flowchart [22] [23], Sliding Window [40], and Teaching Task-Specific Program Design [55].

Ne-Curse aims to make students manage and solve problems manually, such as through games, where they explore them with pleasure, without fear of making mistakes, and where the professor-student relationship and trust can be improved [18]. Ne-Curse allows the professor to understand how students analyze and understand aspects such as problem visualization and interpretation [16].

Flowchart is a diagrammatic representation type of an algorithm, a step-by-step approach to solving a task [22]. A flowchart is an approach that converts text into a flowchart to make newbies focus on the solution rather than the programming syntax, engaging them in meaningful planning activities and solution design before attempting to implement [23].

The beginning programmer can easily represent his solution using flowcharts. Likewise, many challenging tasks can be shown directly in flowchart form to help newbies gain greater understanding [22]. Flowcharts are a wise approach to problem-solving and can help novice programmers keep track of where they are and guide what they need to do next, similar to how a roadmap helps navigate. High-level flowcharts aid in recognizing decomposition sub-tasks and lighten the cognitive load as each sub-task is handled independently [51].

Sliding Window is a common term among professionals to simplify the algorithm's description and help beginners solve algorithmic problems. This metaphorical term allows it to be abstract and communicate ideas and, at the same time, is easy to implement with elementary programming tools [40].

Teaching Task-Specific Program Design helps in fostering problem-solving. The traditional computer science program offers programming design and analysis content in higher education courses such as systems analysis and software development. CS1 students generally have limited exposure to the program design [55]. The task-specific design' structure consists of three steps: i) the professor presents classes that contain linguistic resources; ii) they provide algorithmic solutions and critical thinking skills to solve the task problem, and iii) after each task problem is solved correctly, the professors require students to combine the error-free segments and debug the entire program before submitting the final solution.

Thus, different strategies integrate writing and narrative elements into problem-solving courses for primary and non-core computer systems. The professor can adopt a module to develop narrative and writing skills to program computer courses or develop an interdisciplinary course in Creative Writing and Computational Thinking for non-graduates. These purposeful interdisciplinary approaches to problem-solving will promote the learning' transfer, providing students with the skills they need to succeed in college and beyond [31].

Finally, there are several tools that foster problem-solving skills: Katis [7], Visual Problem Solving Environment for Programming Learning [9], FMAS [20] [24], ADRI-based editor [34], PROSOLVE [37], JavaGrinder [42], Roboot [57], and GameSalad [12].

Katis is an online service to automate students' assessment and classification in the CS1. Kattis provides personalized feedback based on student performance using statistical modeling techniques. Kattis provides thousands of solutions to choose from, developed by professionals and educators worldwide. Kattis has demonstrated good usability due to her activities being motivated by the level of student satisfaction with the system, the degree of confirmation of the student's expectations, and the system's perceived usefulness of the system [7].

Visual Problem Solving Environment for Programming Learning aims to support programming learning. The students exhibited different patterns of computational practice in the environment. Computational practice patterns were correlated with computational design and performance [9].

FMAS aims to improve students' problem-solving skills and introduce them to basic programming algorithms before the surface structure, using an automatic text-to-flowchart approach. The conversion of text into a flowchart, as a visualization-based approach, is employed in FMAS to involve students in developing flowcharts for subsequent programming stages [20] [24].

ADRI-based editor is based on the ADRI model, discouraging the programming shortcut. Also, he focuses on paying equal attention to programming knowledge and problem-solving strategies. The results show an improvement in obtaining the students' learning results and positively impacting the dropout rate [34]. The ADRI-based editor is a simple editor based on the Java language, and Java is a platform-independent language, so it is compatible with different operating systems.

PROSOLVE is an educational game based on the pseudocode technique and available on the web. PROSOLVE aims to improve the problem-solving skills of novice programmers. It covers all teaching topics of the Introductory Programming course, such as algorithm and pseudocode, essential C++ elements, input/output, selection control structure, repetition control structure, functions, and matrices. The game is an excellent alternative to the traditional pen and paper learning approach to attract students' interest in programming [37]. The results show that students and professors appreciated the application, and its use favors cognitive gains and student involvement. Also, they promote the students' affective involvement in the course. Using the application improves the beginners' understanding of programming, logic resources, and problem-solving skills. The students' grades showed that the students' performance improved and the wear rates reduced after introducing the course application.

JavaGrinder is a task-specific Web 2.0 environment where students can work individually or as a team on minor problems focusing on sound software engineering practices and concept mastery. The environment presents concepts in real-world contexts that defend Computer Science as an exciting multidisciplinary field, not as an abstract world of mysteri-

ous codes and syntax. JavaGrinder facilitates problem-solving skills, exposing the most important aspects of a problem, providing guided practice and immediate feedback [42].

Roboot is an instrument that presents a flexible and incremental visual constructivist model that allows different paths for each student. Formative and summative assessments based on assignment tasks suggest that this approach can significantly improve learning outcomes and student satisfaction, even when students have varying cognitive skills [57].

GameSalad is a game development tool designed to empower everyone to create games for various platforms without regard to their proficiency in a specific programming language. The tool provides easy "drag and drop" features that allow individuals to create the games. research indicates that a better approach to introducing programming concepts to students is to use such "drag and drop" tools that eliminate the complexity of a programming language's syntax and resulting compilation errors, thus decreasing initial frustration with the course. Immediate visual feedback from such tools engaged students in computing effectively [12].

Some tools support programming by dragging and dropping control blocks to provide a simple interface for novice students and eliminate the complexity of syntax errors. However, these tools can be perceived as disconnected from "real" programming languages [43], namely: **Karel the Robot**, **Turtle Graphics**, **Marine Biology Case Study**, **Alice**, **Scratch** and **Green-foot**.

2) Abstract Thinking: GOBSTONE [33], Game-themed programming [50] and Armoni's framework [54] are approaches that foster Abstract Thinking skills.

GOBSTONE method promotes Abstract Thinking. The method focuses on representing information at the code level (abstraction procedures) and the discourse universe, allowing information representation. The method showed a positive impact by improving student approval rates [33].

Game-themed programming is a slightly different approach where abstract programming concepts are taught by exploring small game applications. The main aim is not to teach game programming but help students to understand programming concepts through simple game assignments [50].

Armoni's framework is a generic, rationalized, structured, and even simple strategy for teaching CS abstraction to beginners, providing detailed guidance to achieve this goal. The results point to solid evidence that supports the new teaching method in several aspects. Students in the experimental group were likelier to work at the algorithmic level than limiting the process of solving problems given to the lowest programming and execution levels. They also tended to use black boxes in their solutions, thus demonstrating a high level of abstraction. They also realized the course's objective at a higher level of abstraction and with more in-depth insights, tended more to explain their solutions, demonstrated a better understanding of CS ideas, such as startup, and exhibited a higher level of skills, and CS' knowledge [54].

3) Mathematical Reasoning and Cognitive Flexibility: No studies addressed methods, processes, and tools to foster Mathematical Reasoning and Cognitive Flexibility skills. Based on this limitation, new research must arise to increase the discussions that such skills are predictive programming, supporting scientifically proven tools to foster them. Despite the limitations, some studies used paper-and-pencil instruments, elaborating items that measure all the skills mentioned above, some with a scientific comparison of their impact on CS1, which we will report below.

C. Measure predictive programming skills

Several approaches and instruments are in their initial conception, in the items bank' elaboration, in paper-and-pencil, to be used in the measurement of predictive programming skills: Problem Solving [9], [14], [15], [26], [32], [29], [53]; Abstract Thinking [44], [45], [53], [59]; Mathematical Reasoning [13], [19], [52]; Cognitive Flexibility [15].

Several studies use the Classical Test Theory [9] [13] [14] [15] [19] [32] [29] [44] [45] [48] [52] [53] [59] as a way to measure cognitive programming skills. However, some universities have been adopting other theories for this practice, such as Item Response Theory [26].

Some methods support educators in identifying difficulties in program predictive skills for students in institutions. A probable solution to this problem is the psycho tests (generally on paper-pencil) based on the Classical Test Theory (CTT), which uses as a reference the total number of correct answers of an instrument to evaluate the individual's performance in such a way that, the makes it simple and straightforward in its application [13].

The CTT has limitations in its assessment area, including the individual sample who responds to the instrument. This limitation is justified because the instrument's validation occurs through its application to another group of individuals with mandatory characteristics similar to the first application. Another limitation presented by the CTT is the response' individualization [44] [53]. Each individual submitted may or may not respond consistently to a given item. However, in theory in question, the variance of measurement errors of individuals must be the same, not taking into account the individual particularity mentioned above [14].

Considering the limitations in CTT, the Item Response Theory (IRT) emerged as a compliment that aims to overcome the first' limitations. In order to apply standardized educational assessments on a large scale, many countries use the IRT to construct these, aiming to assess skills in multiple-choice tests [26]. For example, in Brazil, the IRT is used to prepare questions for the Exame Nacional do Ensino Médio (ENEM).

The IRT considers the item as the basic unit of analysis. Different people or the same person can measure their abilities and compare them at different times. The IRT measures the items' parameters independently of the sample [26]. In them, the identical items' bank is applied to all individuals, enabling greater precision, speed, and updating ease, and instruments based on the IRT are minor, subject to errors in their results' dissemination.

VI. Conclusions and Futures Works

This paper aimed to present an SLR to categorize the predictive programming skills and strategies to foster/measure such skills. For that, we defined the SLR protocol and presented this review's search and results.

As a result, we found that, in recent years, several studies show that: (PQ) Predictive programming skills are problem-solving, abstract thinking, mathematical reasoning, and cognitive flexibility; (SQ1) Different researchers use different approaches to teach programming skills based on different educational theories, teaching frameworks, or educational approaches; (SQ2) Several studies use the Classical Test Theory as a way to measure predictive programming skills. However, some universities have adopted other theories for this practice, such as Item Response Theory.

This SLR's limitations are that, although the researched libraries index the most influential magazines and conferences in this area, we cannot guarantee that the SLR will cover them completely. Therefore, for future work, we intend to extend this SLR to the annals of the leading conferences in the field of Informatics and periodicals.

References

[1] Acm digital library. https://dl.acm.org/. Accessed: 2020-02-10.

[2] Ieee xplore library. https://ieeexplore.ieee.org/. Accessed: 2020-02-10.

[3] Sciencedirect library. https://www.sciencedirect.com/. Accessed: 2020-02-10.

[4] Scopus library. https://www.scopus.com/. Accessed: 2020-02-10.

[5] I. Alshaye, Z. Tasir, and N. F. Jumaat. The conceptual framework of online problem-based learning towards problem-solving ability and programming skills. In *Proceedings of the Conference on e-Learning, e-Management e-Services (IC3e)*. IEEE, Pulau Pinang, Malaysia, 2019.

[6] B. Attallah, Z. Ilagure, and Y. K. Chang. The impact of competencies in mathematics and beyond on learning computer programming in higher education. In *Proceedings of the Information Technology Trends (ITT)*. IEEE, Dubai, United Arab Emirates, 2018.

[7] R. B. Basnet, T. Doleck, D. J. Lemay, and P. Bazelais. Exploring computer science students' continuance intentions to use kattis. *Education and Information Technologies*, 23(3), 2018.

[8] C. Cabo and R. D. Lansiquot. Synergies between writing stories and writing programs in problem-solving courses. In *Proceedings of the Frontiers in Education Conference (FIE)*. IEEE, Madrid, Spain, 2014.

[9] P. Y. Chao. Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers Education*, 95, 2016.

[10] N. Chaudhry and G. Rasool. A case study on improving problem solving skills of undergraduate computer science students. *World Applied Sciences Journal*, 20(1), 2012.

[11] J. Chetty and G. Barlow-Jones. Novice students and computer programming: Toward constructivist pedagogy. *Mediterranean Journal of Social Sciences*, 5(14), 2014.

[12] S. Dekhane, X. Xu, and M. Y. Tsoi. Mobile app development to increase student engagement and problem solving skills. *Journal of Information Systems Education*, 24(4), 2019.

[13] S. V. Drachova, J. O. Hallstrom, J. E. Hollingsworth, J. Krone, R. Pak, and M. Sitaraman. Teaching mathematical reasoning principles for software correctness and its assessment. *ACM Transactions on Computing Education (TOCE)*, 15(3), 2015.

[14] H. Y. Durak. The effects of using different tools in programming teaching of secondary school students on engagement, computational thinking and reflective thinking skills for problem solving. *Technology, Knowledge and Learning*, 2018.

[15] H. Y. Durak. Modeling different variables in learning basic concepts of programming in flipped classrooms. *Journal of Educational Computing Research*, 58(1), 2020.

[16] J. Figueiredo and F. J. García-Peñalvo. Improving computational thinking using follow and give instructions. In *Proceedings of the Technological Ecosystems for Enhancing Multiculturality (TEEM)*. ACM Cádiz, Spain, 2017.

[17] J. Figueiredo and F. J. García-Peñalvo. Teaching and learning strategies of programming for university courses. In *Proceedings of the Technological Ecosystems for Enhancing Multiculturality (TEEM)*. ACM León, Spain, 2019.

[18] J. Figueiredo, G. Natália, and F. J. García-Peñalvo. Ne-course for learning programming. In *Proceedings of the Technological Ecosystems for Enhancing Multiculturality (TEEM)*. ACM Salamanca, Spain, 2016.

[19] A. Gomes, A. J. Mendes, M. J. Marcelino, W. Ke, S. K. Im, and C. T. Lam. Student's characteristics and programming learning—a macanese perspective. In *Proceedings of the International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE, Hong Kong, China, 2017.

[20] D. Hooshyar, R. B. Ahmad, M. Fathi, M. Yousefi, and M Hooshyar. Flowchart-based bayesian intelligent tutoring system for computer programming. In *Proceedings of the International Conference on Smart Sensors and Application (ICSSA)*. IEEE, Kuala Lumpur, Malaysia, 2015.

[21] D. Hooshyar, R. B. Ahmad, and M. H. N. M. Nasir. A framework for automatic text-to-flowchart conversion: A novel teaching aid for novice programmers. In *Proceedings of the International Conference on Computer, Control, Informatics and Its Applications (IC3INA)*. IEEE, Bandung, Indonesia, 2014.

[22] D. Hooshyar, R. B. Ahmad, M. H. N. M. Nasir, and W. C. Mun. Flowchart-based approach to aid novice programmers: A novel framework. In *Proceedings of the International Conference on Computer and Information Sciences (ICCOINS)*. IEEE, Kuala Lumpur, Malaysia, 2014.

[23] D. Hooshyar, R. B. Ahmad, M. H. N. M. Nasir, S. Shamshirband, and S. J. Horng. Flowchart-based programming environments for improving comprehension and problem-solving skill of novice programmers: A survey. *International Journal of Advanced Intelligence Paradigms*, 7(1), 2015.

[24] D. Hooshyar, R. B. Ahmad, R. G. Raj, M. H. N. M. Nasir, M. Yousef, S. J. Horng, and J. Rugelj. A flowchart-based multi-agent system for assisting novice programmers with problem solving activities. *Malaysian Journal Of Computer Science*, 28(2), 2015.

[25] Y. C. Huei. Benefits and introduction to python programming for freshmore students using inexpensive robots. In *Proceedings of the International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE, Wellington, New Zealand, 2014.

[26] F. Jakoš and D. Verber. Learning basic programming skills with educational games: A case of primary schools in slovenia. *Journal of Educational Computing Research*, 55(5), 2017.

[27] G. B. Jones and D. V. Westhuizen. Pre-entry attributes are thought to influence the performance of students in computer programming. In *Proceedings of the Southern African Computer Lecturers' Association*. Springer, Cham, 2017.

[28] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

[29] T. Koulouri, S. Lauria, and R. D Macredie. Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education (TOCE)*, 14(4), 2014.

[30] A. N. Kumar. A mid-career review of teaching computer science i. In *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Denver, Colorado, United States, 2013.

[31] R. D. Lansiquot and C. Cabo. Strategies to integrate writing in problem-solving courses: Promoting learning transfer in an interdisciplinary context. In *Proceedings of the American Society for Engineering Education (ASEE)*. American Society for Engineering Education, Seattle, WA, United States, 2015.

[32] 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 *Proceedings of the Special Interest Group on Computer Science Education (SIGCSE)*. ACM Memphis, TN, United States, 2016.

[33] P. E. M. López, D. Ciolek, G. Arévalo, and D. Pari. The gobstones method for teaching computer programming. In *Proceedings of the Latin American Computer Conference (CLEI)*. IEEE, 2017.

[34] S. I. Malik. Enhancing practice and achievement in introductory programming using an adri editor. In *Proceedings of the IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE, Bangkok, Thailand, 2016.

[35] S. I. Malik and J. Coldwell-Neilson. Impact of a new teaching and learning approach in an introductory programming course. *Journal of Educational Computing Research*, 55(6), 2017.

[36] S. I. Malik, R. Mathew, R. Al-Nuaimi, A. Al-Sideiri, and J. Coldwell-Neilson. Learning problem solving skills: Comparison of e-learning and m-learning in an introductory programming course. *Education and Information Technologies*, 24(5), 2019.

[37] R. Mathew, S. I. Malik, and R. M. Tawafak. Teaching problem solving skills using an educational game in a computer programming course. *Informatics in Education*, 18(2), 2019.

[38] 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, 2018.

[39] M. Mladenović, D. Krpan, and S. Mladenović. Learning programming from scratch. In *Proceedings of the International Conference on New Horizons in Education*, 2017.

[40] O. Muller, A. Butman, and M. Butman. Opening a (sliding) window to advanced topics. In *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Bologna, Italy, 2017.

[41] P. Niemelä, T. Partanen, M. Harsu, L. Leppänen, and P. Ihantola. Computational thinking as an emergent learning trajectory of mathematics. In *Proceedings of the Koli Calling International Conference on Computing Education Research*. ACM Koli, Finland, 2017.

[42] J. D. Palmer, J. Flieger, and E. Hillenbrand. Javagrinder: A web-based platform for teaching early computing skills. In *Proceedings of the American Society for Engineering Education (ASEE)*. American Society for Engineering Education, Seattle, WA, United States, 2011.

[43] Y. Papadopoulos and S. Tegos. Using microworlds to introduce programming to novices. In *Proceedings of the Panhellenic Conference on Informatics*. IEEE, 2012.

[44] C. J. Park and J. S. Hyun. Effects of abstract thinking and familiarity with programming languages on computer programming ability in high schools. In *Proceedings of the International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE, Wellington, New Zealand, 2014.

[45] C. J. Park, J. S. Hyun, and J. Heuilan. Effects of gender and abstract thinking factors on adolescents' computer program learning. In *Proceedings of the Frontiers in Education Conference (FIE)*. IEEE, El Paso, TX, United States, 2015.

[46] M. Philip, V. G. Renumol, and R. Gopeekrishnan. A pragmatic approach to develop computational thinking skills in novices in computing education. In *Proceedings of the IEEE International Conference in MOOC, Innovation and Technology in Education (MITE)*. IEEE, 2013.

[47] S. Rose. Bricolage programming and problem solving ability in young children: An exploratory study. In *Proceedings of the European Conference on Games Based Learning*. University of the West of Scotland, Paisley, Scotland, 2016.

[48] K. Sanders, M. Ahmadzadeh, T. Clear, S. H. Edwards, M. Goldweber, C. Johnson, R. Lister, R. McCartney, E. Patitsas, and J. Spacco. The canterbury questionbank: Building a repository of multiple-choice cs1 and cs2 questions. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Canterbury, United Kingdom, 2013.

[49] P. Sands. Addressing cognitive load in the computer science classroom. *ACM Inroads*, 10(1), 2019.

[50] J. Skalka and M. Drlík. Educational model for improving programming skills based on conceptual microlearning framework. In *Proceedings of the International Conference on Interactive Collaborative Learning (ICL)*. Springer, Cham, 2018.

[51] R. Smetsers-Weeda and S. Smetsers. Problem solving and algorithmic development with flowcharts. In *Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE)*. ACM Nijmegen, Netherlands, 2017.

[52] L. M. Souza, B. M. Ferreira, I. M. Félix, L. Oliveira Brandão, A. A. F. Brandão, and P. A. Pereira. Mathematics and programming: Marriage or divorce? In *Proceedings of the World Conference on Engineering Education (EDUNINE)*. IEEE, Lima, Peru, 2019.

[53] T. H. Spangsberg, S. Fincher, and S. Dziallas. Non-traditional novices' perceptions of learning to program: A framework of developing mental models. In *Proceedings of the Frontiers in Education Conference (FIE)*. IEEE, San Jose, CA, United States, 2018.

[54] D. Statter and M. Armoni. Teaching abstraction in computer science to 7th grade students. *ACM Transactions on Computing Education (TOCE)*, 20(1), 2020.

[55] X. Suo. Toward more effective strategies in teaching programming for novice students. In *Proceedings of the International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE, Hong Kong, China, 2012.

[56] S. M. Taheri, M. Sasaki, and H. T. Ngetha. Evaluating the effectiveness of problem solving techniques and tools in programming. In *Proceedings of the Science and Information Conference (SAI)*. IEEE, Logon, United Kingdom, 2015.

[57] C. Thevathayan and M. Hamilton. Supporting diverse novice programming cohorts through flexible and incremental visual constructivist pathways. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Vilnius, Lithuania, 2015.

[58] D. Topalli and N. E. Cagiltay. Improving programming skills in engineering education through problem-based game projects with scratch. *Computers Education*, 120, 2018.

[59] A. K. Veerasamy, D. D'Souza, R. Lindén, and M. J. Laakso. Relationship between perceived problem-solving skills and academic performance of novice learners in introductory programming courses. *Journal of Computer Assisted Learning*, 35(2), 2019.

[60] A. N. Çoklar and A. Akçay. Evaluating programming self-efficacy in the context of inquiry skills and problem-solving skills: A perspective from teacher education. *World Journal on Educational Technology: Current Issues*, 10(3), 2018.