Easy-Programming: towards a web collaborating algorithmic and programming aid for early apprentices

Ricardo Vardasca[1,2][0000-0003-4217-2882], Duarte Silva[1], Joao Fonseca[1], Marco Tereso[1][0000-0003-4644-5227], Fernando Bento[1,3][0000-0002-0233-4077] and Domingos Martinho[1][0000-0002-5887-4814]

[1] ISLA Santarem, Rua Teixeira Guedes 31, 2000-029 Santarem, Portugal
[2] INEGI, Universidade do Porto, Rua Dr. Roberto Frias 400, 4200-465 Porto, Portugal
[3] ISTAR, ISCTE-IUL, Av. Prof. Aníbal Bettencourt 9, 1600-189 Lisboa, Portugal
ricardo.vardasca@islasantarem.pt

Abstract. Science, Technology, Engineering and Mathematics (STEM) undergraduate students must develop problem solving skills through algorithmic and programming learning. There are several tools available to aid them in this process but none in a web collaborative environment that can be used for e-learning accommodating the three methods available for that development: code, flowchart, and pseudocode. It is aim of this research to outline the existing tools and their features, and to propose a new web based collaborative tool accommodating the main features found. An architecture, technological infrastructure, database structure, requirements definition, UML use case and class diagrams and a user interface were proposed. New STEM undergraduate students can develop a solid foundation in algorithmic thinking, problem-solving skills, and the ability to effectively communicate and collaborate with others, establishing the foundations for their success in specific fields.

Keywords: Algorithms, collaboration tool, flowcharts, programming learning.

1 Introduction

In today's rapidly evolving technological landscape, the importance of algorithmic and programming skills cannot be overstated, especially in the context of an academic degree. As the world becomes increasingly digitized, Science, Technology, Engineering and Mathematics (STEM) graduates are at the forefront of driving innovation and solving complex problems through the application of computational thinking. The ability to understand, design, and implement algorithms and programming logic is a foundational skillset that plays a vital role in a STEM academic and professional journey [1-4].

Early exposure to algorithmic and programming skills during an academic STEM degree provides students with a solid foundation for tackling real-world challenges. Examples of those skills include problem-solving and logical thinking, efficiency and optimization, interdisciplinary collaboration, innovation and creativity, and adaptability and futureproofing [1-4].

Learning algorithmic and programming skills cultivates a structured and analytical approach to problem-solving. STEM professionals encounter intricate problems that require breaking them down into smaller, manageable components. By understanding algorithms, students can devise systematic solutions, identify patterns, and apply logical thinking to efficiently solve complex problems [1-4].

Algorithms lie at the core of developing efficient and optimized solutions. Through learning early algorithmic skills, STEM students acquire the ability to evaluate the time and space complexities of different approaches. This knowledge empowers them to design algorithms that streamline processes, reduce resource consumption, and optimize system performance, ultimately contributing to enhanced solutions [5].

STEM projects often involve collaboration across various disciplines. Algorithmic and programming skills serve as a common language, enabling effective communication and collaboration among peers from different backgrounds. When peers possess a shared understanding of algorithms and programming concepts, it becomes easier to collaborate on designing and implementing solutions, leading to more integrated and comprehensive project outcomes [5].

Algorithmic and programming skills provide the foundation for innovation and creativity in engineering. Armed with the ability to develop custom algorithms, students can devise novel solutions to complex problems, explore unconventional approaches, and think outside the box. This creative thinking, combined with technical proficiency, allows students to propose groundbreaking solutions, drive technological advancements, and make a lasting impact on society [1-5].

The rapid advancement of technology requires students to be adaptable and future-proof their skillsets. Algorithmic and programming skills provide a strong foundation for learning new programming languages, frameworks, and technologies. By mastering the fundamentals early in their academic journey, STEM students develop a solid base from which they can easily adapt to emerging technologies and stay relevant in a rapidly evolving industry [5].

Learning algorithmic and programming skills at an early stage of an academic STEM degree is of paramount importance. These skills foster critical thinking, problem-solving abilities, and interdisciplinary collaboration. By equipping students with a strong foundation in algorithms and programming, educational institutions can empower future engineers to tackle complex challenges, drive innovation, and make meaningful contributions to the field of engineering. The subsequent sections will delve into specific approaches and methodologies to effectively integrate algorithmic and programming education into degrees curriculum [1-4].

Algorithms can be learned in three different ways, with which they can be represented, they are pseudocode (a formal language between natural and a programming language), flowchart (graphical representation) or by code in an explicit programming language. The perception of each of these forms depends on everyone, being easier for some in one way and more difficult for others in another. Deep down, they are all complementary and allow a different view of a solution to a problem [6-7].

The aim of this research is to identify the existing collaborative tools to support algorithmic learning, identifying the innovative characteristics and based on them to

propose a tool that consolidates them for future implementation in an e-Learning environment.

2 Existing tools

There are several tools available to support algorithmic and programming language learning. These tools are designed to provide learners with a practical and interactive environment to acquire and practice algorithmic and programming skills. Examples of those tools include Integrated Development Environments (IDEs), Online Coding Platforms, Code Editors, Online Learning Platforms, Documentation and Reference Materials, Community Forums and Q&A Platforms and Educational Coding Games. These tools offer a range of resources and interactive experiences to support learners in their algorithmic and programming language journey. Depending on individual preferences and learning styles, learners can explore and utilize these tools to enhance their programming skills and gain practical experience.

At basic and secondary education students are starting to become familiarized with tools devoted to algorithmic thinking, examples of these tools are Scratch, Blocky and Flowgorithm [8-11]. Scratch is a visual programming language developed by the MIT Media Lab. It features a block-based interface where users can drag and snap together code blocks to create animations, games, and interactive stories. Scratch is widely used to introduce programming concepts to beginners, especially children, in a fun and intuitive way. Blockly is a web-based visual programming editor created by Google. It provides a similar block-based interface like Scratch but offers more flexibility and extensibility. Blockly supports multiple programming languages, allowing learners to transition from visual coding to text-based coding. It is often used as a foundation for other visual programming environments. Flowgorithm is a visual programming tool designed specifically for teaching algorithmic thinking and flowchart-based programming. It allows users to create flowcharts to represent algorithms and control structures. Flowgorithm supports multiple programming languages and can generate code from the flowcharts [8-11].

At table 1 a comparison between these tools is shown according to complexity, supported programming languages, allowing customization and available community and resources. Scratch is more oriented towards beginners and Flowgorithm to a more advanced users, being Blocky in between, the Scratch has its own programming language limiting the user development towards others, the other two support the most used. Flowgorithm, due to be based in flowcharts is very rigid and the two other options are customizable until a certain point. In terms of community, Scrach has a large one and plenty of resources, for the other tools the community is small and the resources scarce.

For more mature students there are other tools available such as Visual Alg, Portugol (in three versions: IDE, Studio and Online), Algorithmi, SICAS, COLLEGE, OOP-Admin and Progranimate.

VisuAlgo [12] is an online platform that provides visualizations for various algorithms and data structures. It covers a wide range of topics, including sorting, searching, graph algorithms, and dynamic programming. Users can choose an algorithm, input their data, and interactively step through the visualization to understand the algorithm's behavior.

Portugol [13-15] is a pseudocode-based programming language and tool that is widely used for teaching and learning programming concepts. It is specifically designed for beginners and serves as a steppingstone to learning actual programming languages. It comes in three versions: IDE (provides features such as code editing, compilation, and execution), Studio (local instance of IDE, can run offline) and Online (web-based platform that allows you to write, compile, and execute programs directly in a web browser).

Table 1. Table captions should be placed above the tables.

Tool	Complexity	Programming languages	Customization	Community & Resources
Scratch	Beginner-oriented	Limited (own)	Customizable	Large
Blocky	Simple and flexible	JavaScript, Python, PhP, Lua	Customizable	Small
Flowgorithm	More complex	C#, C++, Java, JavaScript, Lua, Perl, Python, Ruby, Swift, Visual Basic	Rigid	Small

Algorithmi [16] is an application aid for programming learners, which has as goal to help them to overcome their learning barriers, it can use pseudocode and code of several programming languages, allowing conversion between them.

Interactive system for algorithm development and simulation (SICAS) [17] is a basic procedural programming learning tool for concepts such as selection or repetition, its goal is to improve problem solving abilities using those concepts.

COLLEGE [18], also known as Real Time Collaborative Programming system that allows remotely distributed programmers to work concurrently and collaboratively in the same programming task.

OOP-Anim [19] is a support tool that allows students animate and simulate small Java programs, it aims to support student learning, through execution simulation and errors detection.

Progranimate [20, 21] is a simplified development environment that uses dynamic structured flowchart program construction, can generate code in several languages and provides an animated execution.

A comparison between the outlined algorithmic learning tools according to features such as web orientation, having flowchart support, having pseudocode support, having a code editor, having a depurator, providing a program structure, having a variable

inspector, accommodating a chat, providing a set o examples, providing automatic evaluation and programming languages support is given at table 2.

From that comparison, one two tools provide web environment, the most provided feature is the incorporation of a code editor and a depurator, followed by a variable inspector, being scarcer the support to pseudocode and flowchart, provision of examples, and rarer a chat and automatic evaluation. In terms of programming languages, the most supported is Java.

From this evaluation there is no web solution providing pseudocode, flowchart and any language code support with a code editor and depurator available to aid early algorithmic and programming learners.

Table 2. Comparison of algorithmic tools for more mature users.

Tool	a)	b)	c)	d)	e)	f)	g)	h)	i)	j)	k)
Visual Alg	X		X	X	X		X		X		Pascal
Portugol IDE		X	X	X	X		X				
Portugol Studio				X	X	X	X		X		
Portugol Online	X		X	X					X		
Algorithmi		X	X	X	X	X	X		X		C, Java, Python
SICAS		X		X	X		X		X		
COLLEGE				X	X	X	X	X			
OOP-Anim				X	X		X				Java
Progranimate				X	X	X	X		X	X	Java, Visual Basic

a) Web oriented, b) Flowchart support, c) Pseudocode support, d) Code editor, e) Depurator, f) Program structure, g) Variable inspector, h) Chat, i) Provision of examples, j) Automatic evaluation and k) Programming languages

3 Proposed tool modelling

Based on the verified opportunity of lack of a collaborative web tool to support the learning of algorithms and programming language, a proposal will be elaborated based on a definition of architecture, technological infrastructure, database Entity-Relationship diagram, requirements analysis, UML modulation of use cases and classes and finally, it is proposed a user interface.

The proposed architecture for the Easy Programming collaborative tool is Model-View-Controller (MVC), which is a software architecture pattern commonly used in the development of web applications. It separates the application logic into three interconnected components: the Model, the View, and the Controller. The Model represents the application's data and business logic. The View is responsible for the presentation layer of the application. The Controller acts as the intermediary between the

Model and the View. The MVC enforces a clear separation between the different aspects of the application, allowing code reutilization, easy maintenance, testability, and scalability. Due to its benefits and is supported by many popular frameworks and technologies [22].

In terms of technological infrastructure (see Fig. 1), the Easy Programming application is expected to run at the client in a web browser support by a JavaScript framework (e.g. angular.js or react.js), which provides interfaces, components, directives and services to the application, that connects with a remote server through a REST API and on the server side there is a database repository supporting the application server.

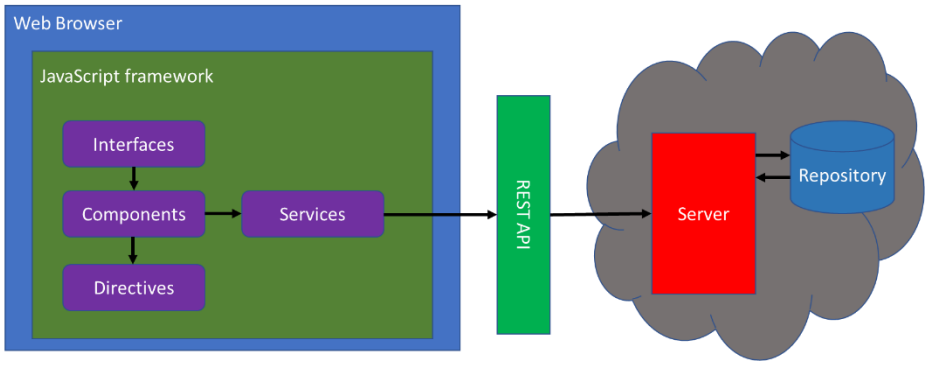

Fig. 1. The technological infrastructure for the proposed Easy Programming application.

The database in the repository (see Fig. 2) will have at least three tables: user, exercise, and solution. The relationship between user and exercise is Many-to-Many being option from the user side, and the relationship between exercise and solution is One-to-Many.

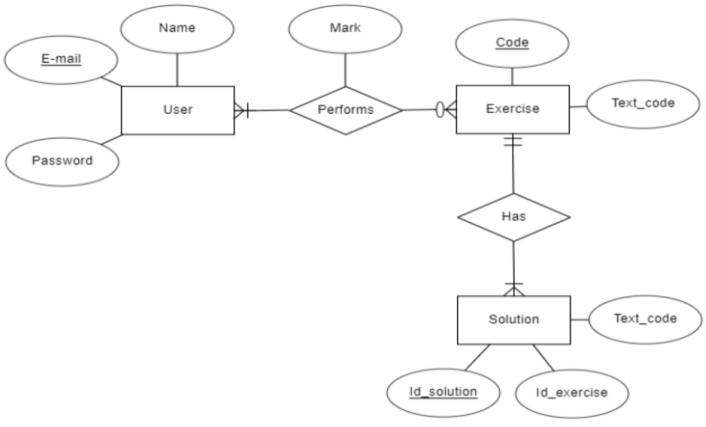

Fig. 2. The Entity-Relationship diagram for the repository of the proposed Easy Programming application.

The identified requirements for the proposed Easy Programming tool are:

Functional Requirements:

- User Registration and Authentication, allow users to create accounts, log in, and manage their profiles.
- Pseudocode Editor, provide an interactive editor that allows users to write, edit, and save pseudocode.
- Flowchart Editor, implement a visual editor that enables users to create, modify, and save flowcharts.
- Code Editor, include a code editor with syntax highlighting and code completion for writing and editing programming code.
- Code Execution, integrate a code execution engine that can run code written in various programming languages and provide immediate feedback on the output or errors.
- Algorithm Visualization, incorporate features to visually represent algorithms and their execution steps.
- Problem-Solving Exercises, offer a collection of algorithmic problems or exercises for users to practice and apply their algorithmic skills.
- Progress Tracking, track and display user progress, including completed exercises, achievements, and performance metrics.
- Social Features, enable users to share their pseudocode, flowcharts, and code with others, provide feedback, and engage in discussions.
- Collaborative Learning, support collaborative learning by allowing users to work together on solving problems or reviewing algorithms.
- Learning Resources, provide additional learning resources such as tutorials, reference materials, and examples related to algorithms and problem-solving.

Non-Functional Requirements:

- Usability, to ensure that the system is intuitive, easy to navigate, and provides a user-friendly interface.
- Performance, optimize the system to handle concurrent users, execute code efficiently, and provide real-time feedback.
- Scalability, design the system to handle increasing user loads and accommodate future growth.
- Reliability, ensure the system is stable, available, and resilient to failures or disruptions.
- Security, implement proper security measures to protect user data, prevent unauthorized access, and handle potential vulnerabilities.
- Compatibility, ensure the system works well across different web browsers, devices, and operating systems.
- Accessibility, design the system to be accessible to users with disabilities, adhering to relevant accessibility guidelines.

- Maintainability, develop the system using modular and maintainable code, allowing for easy updates, bug fixes, and future enhancements.
- Data Privacy, adhere to privacy regulations and safeguard user data by implementing appropriate data protection measures.
- Integration, allow for integration with other systems or platforms, such as learning management systems or external APIs for additional functionality.

A simple UML use case diagram (see Fig. 3) would have a Lecturer that needs to login, create a new exercise, a new solution to that exercise or a set of solutions, check students' grades and be able to chat with students to support them, at the end of his interaction he should be able to logout. A student would be able to login, solve an exercise, check grade, change to flowchart or code, or change again to pseudocode, create a new exercise, or logout.

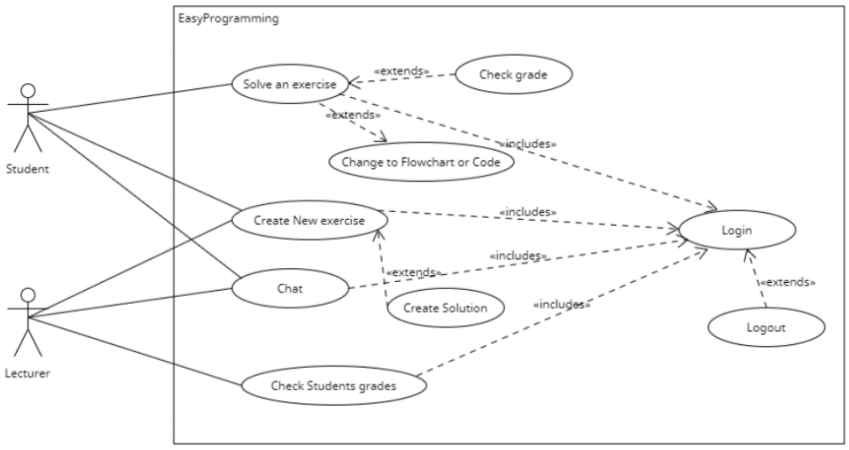

Fig. 3. A simple UML use case diagram for the proposed Easy Programming application.

To allow the development of the proposed easy programming tool in an object-oriented approach, a simple UML class diagram is proposed (Fig. 4), it has five classes: user, exercise, exercise-student (to store the mark), exercise_code, exercise_flowchart and exercise_solution. A user can perform several exercises, an exercise may have a grade to a student, an exercise must have a correspondent code and flowchart and may have one or more solutions.

For aiding the user interface designer, the proposed main interface for the Easy Programming tool is presented in the Fig. 5, when the user logs in it will display its name and the interface is split in three sections with more option over the username button, such as run, check grade, chat, add solution, add new exercise, view student grades, change programming language and logout. The three areas on the main screen are: on the left the pseudocode, at the middle the flowchart and at the right the code of the specified programming language.

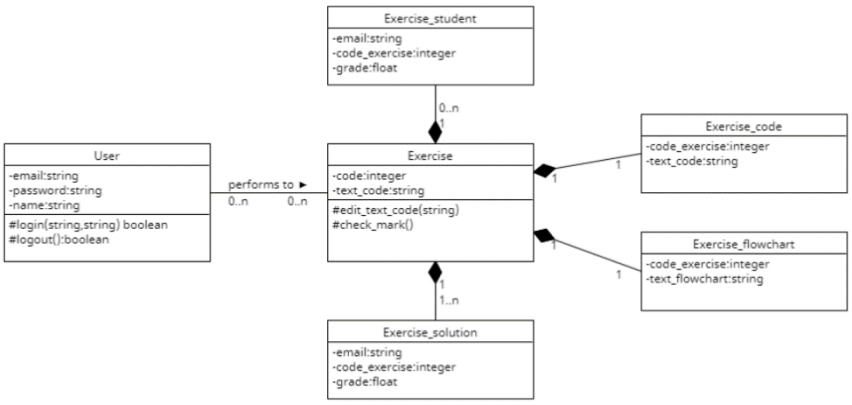

Fig. 4. A simple UML class diagram for the proposed Easy Programming application.

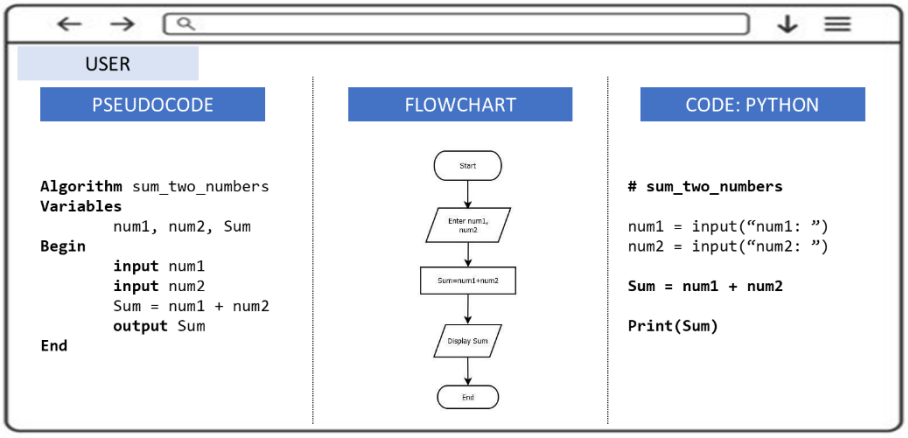

Fig. 5. Proposed user interface for the proposed Easy Programming application.

4 Discussion

There are several advantages of having a collaborative web system for an integrated algorithmic learning tool that incorporates pseudocode, flowcharts, and code, such as: enhanced learning experience, visual representation, language agnostic, seamless transition from design to implementation, immediate feedback and error analysis, collaborative learning and discussion, progress tracking and personalization, accessible and flexible learning, and a centralized learning resource.

The integration of pseudocode, flowcharts, and code provides a comprehensive learning experience. Students can understand algorithms at different levels of abstrac-

tion, from high-level problem-solving using pseudocode and flowcharts to low-level implementation using code. This holistic approach helps in developing a deep understanding of algorithmic concepts.

Pseudocode and flowcharts offer visual representations of algorithms, making them easier to understand and analyze. Visual aids help students grasp complex algorithms and their execution flow, enhancing their algorithmic thinking and problem-solving skills.

A web system that supports pseudocode, flowcharts, and code is language-agnostic. This allows students to focus on algorithmic principles rather than being tied to a specific programming language. It prepares them to adapt to different programming languages and environments, which is particularly valuable in the rapidly evolving field of computer science.

The integration of pseudocode, flowcharts, and code facilitates a smooth transition from algorithm design to implementation. Students can start by designing algorithms using pseudocode and flowcharts, and then directly translate them into code. This helps in bridging the gap between problem-solving and coding skills.

With code execution integrated into the system, students can receive immediate feedback on their code's correctness and output. This allows for instant error detection and analysis, helping students identify and understand common mistakes, debug their code, and improve their problem-solving skills.

The web system can foster collaborative learning by allowing students to share their pseudocode, flowcharts, and code with others. This encourages collaboration, discussions, and the exchange of ideas among learners. Peer feedback and engagement enhance the learning process and provide different perspectives on problem-solving approaches.

The system can track students' progress, performance, and achievements. This enables personalized learning experiences, as the system can recommend specific exercises or resources based on individual strengths, weaknesses, and learning goals. Progress tracking also allows students to monitor their growth and development over time.

A web-based system provides accessibility and flexibility, allowing students to access the learning tool from anywhere with an internet connection. This enables self-paced learning, facilitates remote or distance education, and accommodates diverse learning styles and schedules.

The Easy Programming proposed tool serves as a centralized platform for all algorithmic learning resources, including pseudocode examples, flowchart templates, code samples, and interactive exercises. This organized and comprehensive resource hub supports efficient learning and reduces the need for students to search for scattered materials.

Overall, a web system for integrated algorithmic learning offers a multifaceted and interactive learning environment that promotes effective algorithmic understanding, problem-solving skills, collaboration, and personalized learning experiences.

5 Conclusion

This research has identified the existing collaborative tools to support algorithmic learning, and their innovative characteristics, with such information a proposal of a collaborative e-Learning environment tool for algorithmic and programming skills learning was outlined with architecture, technological infrastructure, database structure, requirements definition, UML use case and class diagrams and a user interface.

An integrated algorithmic learning tool that incorporates pseudocode, flowcharts, and code can be highly beneficial for new STEM (Science, Technology, Engineering, and Mathematics) undergraduate students for several reasons, such as: conceptual understanding, transitional support, algorithmic thinking, error identification and debugging, collaboration and communication.

The next phase in this research project will be to develop, test and integrate the outlined features into a fully functional pilot to be used with first year students and evolve from there to a study support tool.

By integrating pseudocode, flowcharts, and code in the learning process, new STEM undergraduate students can develop a solid foundation in algorithmic thinking, problem-solving skills, and the ability to effectively communicate and collaborate with others. These skills are crucial for success in STEM disciplines and provide a strong basis for further learning and specialization in their respective fields.

References

1. Prinsloo, G. J., Helleveeg, C.: The integration of algorithmic thinking in undergraduate information systems courses. International Journal of Educational Technology in Higher Education, 15(1), 1-24 (2018).
2. Guerra, S. R., Reis, C. R.: Algorithmic problem-solving in undergraduate computer science education: A systematic literature review. Computer Science Education, 28(4), 284-316 (2018).
3. Sanders, K.: Algorithmic thinking and computational problem solving. Communications of the ACM, 59(8), 33-34 (2016).
4. Körner, T.: Algorithms for undergraduate students. Cambridge University Press (2016).
5. Mehta, D., Gupta, S.: Algorithmic thinking and problem-solving skills: An integral part of computer science education. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education, pp. 1076-1076 (2018).
6. Andrzejewska, M., Stolińska, A.: Do Structured Flowcharts Outperform Pseudocode? Evidence From Eye Movements. IEEE Access, 10, 132965-132975 (2022).
7. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: Introduction to algorithms. MIT press (2022).
8. Fagerlund, J., Häkkinen, P., Vesisenaho, M., Viiri, J.: Computational thinking in programming with Scratch in primary schools: A systematic review. Computer Applications in Engineering Education, 29(1), 12-28 (2021).
9. Weintrop, D., Shepherd, D. C., Francis, P., Franklin, D.: Blockly goes to work: Block-based programming for industrial robots. In 2017 IEEE Blocks and Beyond Workshop (B&B), pp. 29-36 (2017).

10. Gajewski, R. R., Smyrnova-Trybulska, E.: Algorithms, Programming, Flowcharts and Flowgorithm. E-Learning and Smart Learning Environment for the Preparation of New Generation Specialists, 393-408 (2018).
11. Cook, D. D.: Flowgorithm: Principles for teaching introductory programming using flowcharts. In Proc. American Society of Engineering Education Pacific Southwest Conf.(ASEE/PSW), pp. 158-167 (2015).
12. Borba, F. H., Marchi, M. I., Rehfeldt, M. J. H.: Utilização do Software VisuAlg no Ensino da Lógica de Programação. Revista de Ensino, Educação e Ciências Humanas, 22(3), 295-304 (2021).
13. Manso, A., Oliveira, L., Marques, C.: Portugol IDE–Uma ferramenta para o ensino de programação. In PAEE'2009—Project Approaches in Engineering Education—Guimaraes (2009).
14. Noschang, L. F., Pelz, F., de Jesus, E., Raabe, A.: Portugol studio: Uma ide para iniciantes em programaçao. In Anais do XXII SBC Workshop sobre Educação em Computação, pp. 1-10 (2014).
15. Silva, D. G. S., Silva, D. G. S., Soussa, M. R. B.: Portugol WebStudio: IDE Online de Desenvolvimento em Portugol como instrumento de ensino-aprendizagem. Ensino e Tecnologia em Revista, 6(1), 16-30 (2022).
16. Manso, A., Marques, C. G., Santos, P., Lopes, L., Guedes, R.: Algorithmi IDE-Integrated learning environment for the teaching and learning of algorithmics. In 2019 IEEE International Symposium on Computers in Education (SIIE), pp. 1-6 (2019).
17. Gomes, A., Mendes, A. J.: SICAS: Interactive system for algorithm development and simulation. Computers and Education: Towards an Interconnected Society, 159-166 (2001).
18. Bravo, C., Redondo, M. A., Ortega, M.: Aprendizaje en grupo de la programación mediante técnicas de colaboración distribuida en tiempo real. In Proceedings of V Congreso Interacción Persona Ordenador, Lleida, Spain, pp. 351-357 (2004).
19. Santos, A., Gomes, A., Mendes, A. J.: Integrating new technologies and existing tools to promote programming learning. Algorithms, 3(2), 183-196 (2010).
20. Scott, A., Watkins, M., McPhee, D.: E-learning for novice programmers; a dynamic visualisation and problem solving tool. In 2008 3rd IEEE International Conference on Information and Communication Technologies: From Theory to Applications, pp. 1-6 (2008).
21. Scott, A., Watkins, M., McPhee, D.: A step back from coding–an online environment and pedagogy for novice programmers. In Proceedings of the 11th Java in the Internet Curriculum Conference, pp. 35-41. The Higher Education Academy, London Metropolitan University (2007).
22. Dey, T.: A comparative analysis on modeling and implementing with MVC architecture. International Journal of Computer Applications, 1, 44-49 (2011).