computers

Special Issue Reprint

Future Trends in Computer Programming Education

Edited by
Stelios Xinogalos

mdpi.com/journal/computers

MDPI

Future Trends in Computer Programming Education

Future Trends in Computer Programming Education

Guest Editor

Stelios Xinogalos

MDPI

Basel • Beijing • Wuhan • Barcelona • Belgrade • Novi Sad • Cluj • Manchester

Guest Editor
Stelios Xinogalos
Department of Applied
Informatics
University of Macedonia
Thessaloniki
Greece

Editorial Office
MDPI AG
Grosspeteranlage 5
4052 Basel, Switzerland

This is a reprint of the Special Issue, published open access by the journal *Computers* (ISSN 2073-431X), freely accessible at: https://www.mdpi.com/journal/computers/special_issues/41HX41V201.

For citation purposes, cite each article independently as indicated on the article page online and as indicated below:

Lastname, A.A.; Lastname, B.B. Article Title. *Journal Name* **Year**, *Volume Number*, Page Range.

ISBN 978-3-7258-9167-2 (Hbk)
ISBN 978-3-7258-9168-9 (PDF)
https://doi.org/10.3390/books978-3-7258-9168-9

© 2026 by the authors. Articles in this reprint are Open Access and distributed under the Creative Commons Attribution (CC BY) license. The reprint as a whole is distributed by MDPI under the terms and conditions of the Creative Commons Attribution-NonCommercial-NoDerivs (CC BY-NC-ND) license (https://creativecommons.org/licenses/by-nc-nd/4.0/).

Contents

About the Editor . . . . . **vii**

Stelios Xinogalos
Future Trends in Computer Programming Education
Reprinted from: *Computers* **2026**, *15*, 497, https://doi.org/10.3390/computers15080497 . . . . . **1**

Ling-Hsiu Chen and Ha Thi The Nguyen
Study Trends and Core Content Trends of Research on Enhancing Computational Thinking: An Incorporated Bibliometric and Content Analysis Based on the Scopus Database
Reprinted from: *Computers* **2024**, *13*, 91, https://doi.org/10.3390/computers13040091 . . . . . **9**

Dimitrios Yiatas and Athanassios Jimoyiannis
Enacting Computer Science Curriculum Reform: The Case of Model and Experimental Lower Secondary Schools in Greece
Reprinted from: *Computers* **2026**, *15*, 140, https://doi.org/10.3390/computers15030140 . . . . . **31**

María Jesús Manzanares, Diana Pérez Marín and Celeste Pizarro
A Preliminary Usability Study of a Novel Educational Training System to Teach ScratchJr. in School
Reprinted from: *Computers* **2026**, *15*, 17, https://doi.org/10.3390/computers15010017 . . . . . **51**

Oleg Sychev and Mikhail Denisov
Explain Trace: Misconceptions of Control-Flow Statements
Reprinted from: *Computers* **2023**, *12*, 192, https://doi.org/10.3390/computers12100192 . . . . . **80**

João P. J. Pires, Fernanda Brito Correia, Anabela Gomes, Ana Rosa Borges and Jorge Bernardino
Predicting Student Performance in Introductory Programming Courses
Reprinted from: *Computers* **2024**, *13*, 219, https://doi.org/10.3390/computers13090219 . . . . . **96**

Ming Lu and Zhongyi Hu
Digital Twin-Enhanced Programming Education: An Empirical Study on Learning Engagement and Skill Acquisition
Reprinted from: *Computers* **2025**, *14*, 322, https://doi.org/10.3390/computers14080322 . . . . . **115**

Jaroslav Porubän, Milan Nosáľ, Matúš Sulír and Sergej Chodarev
Teach Programming Using Task-Driven Case Studies: Pedagogical Approach, Guidelines, and Implementation
Reprinted from: *Computers* **2024**, *13*, 221, https://doi.org/10.3390/computers13090221 . . . . . **141**

Belkis Díaz-Lauzurica and David Moreno-Salinas
Gamified Project-Based Learning in Vocational Education and Training Computer Science Courses
Reprinted from: *Computers* **2026**, *15*, 82, https://doi.org/10.3390/computers15020082 . . . . . **170**

Juan Gabriel López Solórzano, Christian Jonathan Ángel Rueda and Osslan Osiris Vergara Villegas
Measuring Undergraduates' Motivation Levels When Learning to Program in Virtual Worlds
Reprinted from: *Computers* **2024**, *13*, 188, https://doi.org/10.3390/computers13080188 . . . . . **214**

Oleg Sychev
Searching Questions and Learning Problems in Large Problem Banks: Constructing Tests and Assignments on the Fly
Reprinted from: *Computers* **2024**, *13*, 144, https://doi.org/10.3390/computers13060144 . . . . . **236**

Bruno Pereira Cipriano, Bernardo Baltazar, Nuno Fachada, Athanasios Vourvopoulos and Pedro Alves
Bridging the Gap between Project-Oriented and Exercise-Oriented Automatic Assessment Tools
Reprinted from: *Computers* **2024**, *13*, 162, https://doi.org/10.3390/computers13070162 . . . . . **259**

About the Editor

Stelios Xinogalos

Stelios Xinogalos is a Professor at the Department of Applied Informatics, University of Macedonia, Greece. He is a member of the Software and Data Engineering Lab, and the Educational Technology Research Group of the Department of Applied Informatics, University of Macedonia. His research interests include Programming Environments and Techniques, Object-oriented Design and Programming, Didactics of Programming, Computer Science Education, Educational Technology, and Serious Games. He has published more than 150 research papers in international journals, conferences and books.

computers

MDPI

Editorial

Future Trends in Computer Programming Education

Stelios Xinogalos

Department of Applied Informatics, School of Information Sciences, University of Macedonia, GR-54634 Thessaloniki, Greece; stelios@uom.edu.gr

1. Introduction

Programming education has attracted the interest of instructors and researchers for several decades [1]. It is widely accepted that learning programming is difficult [2] and the classic approach to teaching programming [3], which is based on using a conventional programming language and environment for solving number and symbol processing problems, is quite problematic. Students must be supported, motivated and engaged in the learning process in order to comprehend the principles of programming and write good-quality code. Several types of educational programming environments and tools have been developed over the decades, such as programming microworlds and games, structure editors, program/software visualization tools, flowchart-based programming environments and so on [4,5]. Several teaching approaches have also been devised based on the aforementioned tools. Moreover, extended research has been carried out regarding the best strategy for introducing novices to programming, including the imperative-first, functions-first and objects-first strategies, as well as the most appropriate language for each strategy.

In recent years the educational community around the world has shown an even greater interest in teaching and learning programming as a means of promoting Computational Thinking (CT) [6], which is considered important for everyone nowadays, not just computer scientists. Primary school kids nowadays are educated to think computationally through specially designed programming environments, such as Scratch [7] or the widely known "Hour of Code" [8,9] games.

Another important dimension of the work around teaching and learning programming nowadays lies in the way that "digital natives" [10] grow up and learn. Game-based learning and educational/serious games are considered a promising approach for learning programming. Pair programming [11] and more recently distributed pair programming [12,13] is another contemporary approach that promotes collaboration among students, which is also considered important. Most recently, the advent of LLMs and generative AI tools has started transforming the computer programming education landscape [14].

The objective of the Special Issue (SI) on Future Trends in Computer Programming Education was to explore recent advances, as well as future trends in Computer Programming Education. This SI consists of eleven articles that were accepted in the SI after a rigorous peer-review process. Thirty-one researchers from China, Greece, Mexico, Portugal, Russia, Slovakia, Spain, Taiwan and Vietnam contributed their work in this SI.

The eleven articles included in the SI cover a wide range of issues related to computer programming education, as can be seen in the word cloud of the keywords included in them and presented in Figure 1. In the following paragraphs, the eleven articles are summarized, providing their goals, methodology, findings and implications for future research. The articles are organized in four thematic areas.

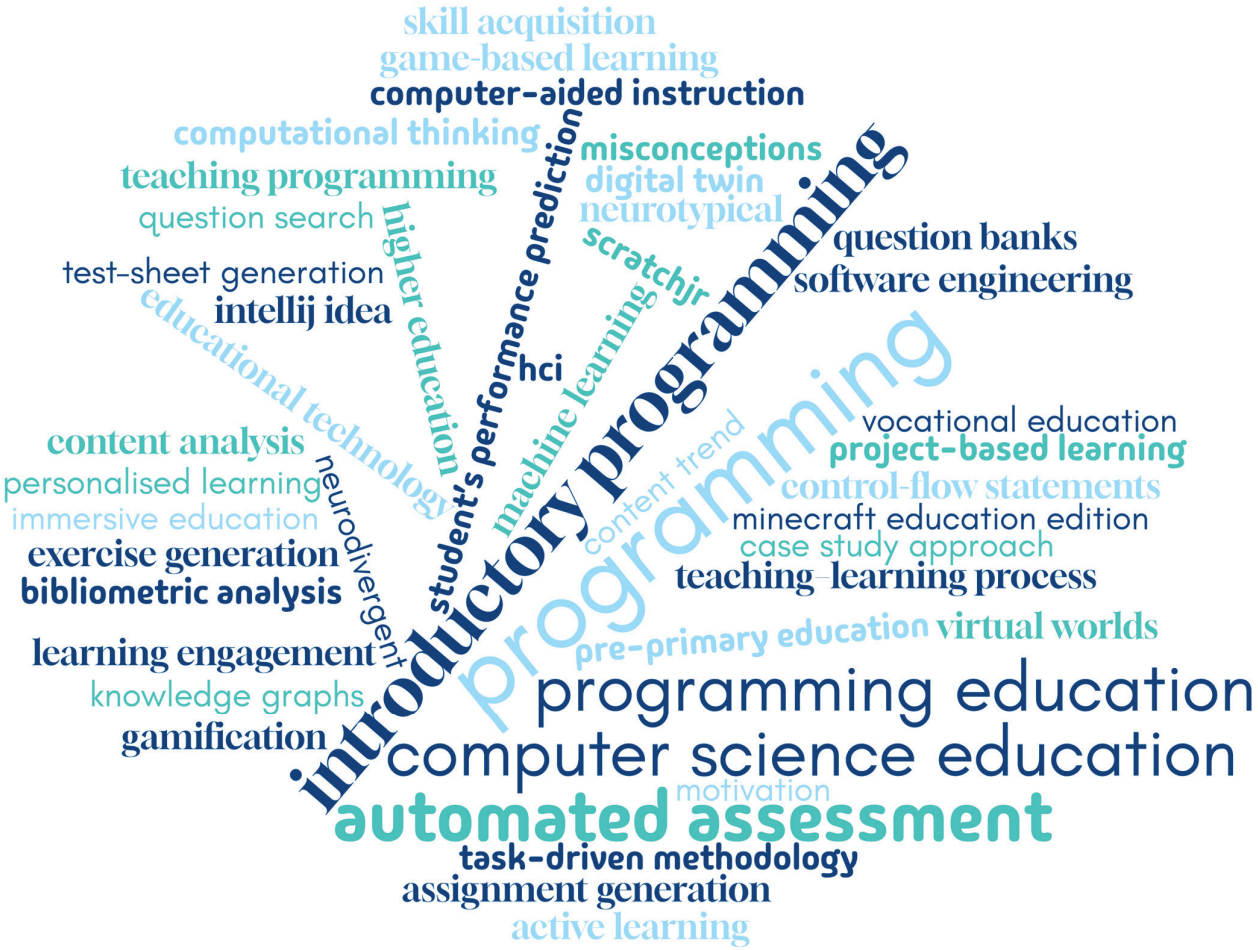


Figure 1. Word cloud of keywords included in the articles of the Special Issue.

2. Computational Thinking and the Computer Science Curriculum

The first article in this thematic area presents a bibliometric and content analysis of research on enhancing CT (contribution 1). This is followed by an article presenting a study on Computer Science teachers' views regarding the recent reform of the Computer Science curriculum in lower secondary education in Greece, which aims at cultivating CT, problem solving and digital skills, among others (contribution 2). The next article presents the design and evaluation of an agent-based training system aimed at helping both neurotypical students and students with special needs learn programming concepts with ScratchJr, an environment that can be used for introducing pre-primary students to CT (contribution 3).

The article "Study Trends and Core Content Trends of Research on Enhancing Computational Thinking: An Incorporated Bibliometric and Content Analysis Based on the Scopus Database" by Ling-Hsiu Chen and Ha Thi The Nguyen (contribution 1) provides a systematic review of research on CT over the period of 2008–2022. It employs a combined bibliometric and content analysis of 132 publications indexed in Scopus to identify publication patterns, influential contributors, and thematic developments in the field. The findings reveal a steady growth in CT-related research, with a notable increase in studies integrating game-based learning approaches. A peak in publication activity was recorded in 2021, while the analysis also highlights geographic trends, indicating that the United States leads in research output, while China has the highest citation impact. In terms of thematic content, the study identifies core research focusing on strategies, learner attitudes, and instructional methods aimed at enhancing CT skills, particularly through interactive and

https://doi.org/10.3390/computers15080497

technology-supported learning environments. Additionally, keyword and co-occurrence analyses demonstrate the centrality of educational applications in shaping the evolution of the field. The authors conclude that understanding these trends can inform future research directions, policymaking, and curriculum design, while the inclusion of other databases (like WOS and PubMEd) and alternative analytical tools (like VOSviewer and CiteSpace) in subsequent studies can improve comprehensiveness.

The article "Enacting Computer Science Curriculum Reform: The Case of Model and Experimental Lower Secondary Schools in Greece" by Dimitrios Yiatas and Athanassios Jimoyiannis (contribution 2) investigates teachers' views regarding the implementation of a new Computer Science curriculum in the Greek model and experimental lower secondary schools. The study is based on qualitative data (interviews), examining how educators interpret and apply the curriculum in real classroom settings. Findings show that teachers generally adopt the curriculum's student-centered philosophy, using it flexibly as a guide rather than a rigid framework. The curriculum promotes key competencies such as CT, collaboration, creativity, and problem solving, aligned with modern educational goals. Teachers report using innovative teaching methods, such as projects, programming activities, and digital tools, as well as alternative assessment practices. Overall, the study concludes that successful implementation is supported by teacher expertise, autonomy, and a supportive school environment, highlighting the importance of flexibility in curriculum design.

The article "A Preliminary Usability Study of a Novel Educational Training System to Teach ScratchJr. in School" by Maria Jesus Manzanares, Diana Perez Marin and Celeste Pizarro (contribution 3) presents the design and pilot evaluation of an interactive agent-based training system aimed at helping young students learn programming with ScratchJr. It focuses on early childhood education, exploring how an agent-based interactive system can support neurotypical students and students with special needs in learning basic coding concepts. The study conducts a usability evaluation to assess how easy, engaging, and effective the system is for classroom use. Findings indicate that the system is generally user-friendly and enhances students' engagement and understanding of fundamental programming ideas.

3. Misconceptions on Control-Flow Statements and Performance Prediction

The first study (contribution 4) in this thematic area investigates students' misconceptions on control-flow statements with the aim of designing targeted instructional interventions and intelligent tutoring systems aimed at improving fundamental programming skills. The next article (contribution 5) utilizes datasets, including academic records, analyzed by machine learning algorithms and identified metrics for predicting students' performance.

The article "Explain Trace: Misconceptions of Control-Flow Statements" by Oleg Sychev and Mikhail Denisov (contribution 4) investigates common misconceptions that novice programmers exhibit when reasoning about control-flow constructs. It adopts an empirical methodology based on a large set of 181 specially designed questions, each combining pseudocode with execution traces that may contain deliberate errors, prompting students to explain the correctness or incorrectness of specific steps. The study analyzes 10.799 student responses collected from 67 participants, enabling a systematic identification and classification of misconceptions related to sequences, loops, and conditional statements. Findings reveal 24 distinct misconceptions, of which some align with previously documented issues, while others represent novel or more fine-grained variants not extensively discussed in earlier literature. The authors highlight that misconceptions about control flow are often underexplored as a distinct category, despite their central role

https://doi.org/10.3390/computers15080497

in introductory programming comprehension. They emphasize the value of open-ended explanatory tasks for uncovering students' mental models, as opposed to relying solely on multiple-choice or correctness-based assessments. Overall, the article concludes that a more systematic understanding of these misconceptions can inform the design of targeted instructional interventions and intelligent tutoring systems aimed at improving fundamental programming skills.

The article "Predicting Student Performance in Introductory Programming Courses" by João P. J. Pires, Fernanda Brito Correia, Anabela Gomes, Ana Rosa Borges and Jorge Bernardino (contribution 5) reviews literature on the use of machine learning techniques to identify students at risk of poor performance in early programming education. It situates the problem within the broader challenge of high failure and dropout rates in introductory programming courses in Higher Education, emphasizing the need for early intervention mechanisms. Methodologically, the study evaluates the performance of various algorithms, using identified metrics and considering the applicability in Higher Education Introductory Programming courses. The results demonstrate that these models can achieve meaningful predictive accuracy, enabling instructors to detect at-risk students at an early stage. The findings highlight the potential of data-driven approaches to support targeted pedagogical interventions, thereby improving student success and retention. Overall, the study concludes that machine learning-based prediction constitutes a valuable tool in Computer Science Education, though its effectiveness depends on data quality and model selection.

4. Teaching Approaches and Tools for Enhancing Programming Education

The first article in this thematic area investigates students' engagement and skill acquisition in introductory programming courses utilizing digital-twin technology instead of traditional learning approaches (contribution 6). The next two articles propose two more alternatives to the traditional learning approach, namely task-driven case studies that combine the task-driven methodology with the case study approach (contribution 7) and gamified project-based learning focused on Vocational Education (contribution 8). The last paper in this thematic area refers to the use of Virtual Worlds in programming education for dealing with common difficulties that accompany the traditional learning approaches for programming (contribution 9).

The article "Digital Twin-Enhanced Programming Education: An Empirical Study on Learning Engagement and Skill Acquisition" by Ming Lu and Zhongyi Hu (contribution 6) explores the use of digital twin (DT) technology in introductory programming courses. It addresses limitations of traditional teaching, which often fails to connect theory with practice and results in low student engagement and weak problem-solving skills. Using a quasi-experimental study with 135 students, it compares a DT-based learning environment with conventional instruction. Results show that students using digital twins demonstrate higher engagement (behavioral, cognitive, emotional) and improved performance in coding tasks. This approach is especially beneficial for beginners, significantly enhancing algorithmic thinking, debugging, and practical programming skills. Overall, the study concludes that digital twin technology provides an immersive, interactive learning model that effectively bridges theory and practice in programming education.

The article "Teach Programming Using Task-Driven Case Studies: Pedagogical Approach, Guidelines, and Implementation" by Jaroslav Porubän, Milan Nosál', Matúš Sulir and Sergej Chodarev (contribution 7) proposes a novel method for teaching programming by combining task-driven learning with case study approaches. It is grounded in constructivism learning theory, emphasizing learning through solving real-world tasks rather than relying solely on theoretical instruction. The approach uses structured case studies

https://doi.org/10.3390/computers15080497

(e.g., a programming project like Minesweeper) to guide students step-by-step in applying concepts and developing solutions. The authors provide practical guidelines for designing tasks, organizing courses, and supporting students during implementation. Findings suggest that this method improves student engagement, understanding, and practical programming skills, as learners actively apply knowledge and learn from mistakes. Overall, the study concludes that task-driven case studies offer an effective, practice-oriented alternative to traditional programming instruction.

The article "Gamified Project-Based Learning in Vocational Education and Training Computer Science Courses" by Belkis Díaz-Lauzurica and David Moreno-Salinas (contribution 8) investigates how combining gamification with project-based learning (PBL) affects Computer Science Education. It uses a qualitative approach to analyze students' learning experiences and engagement within vocational education settings. Findings show that gamified PBL increases student motivation, participation, and collaboration, making learning more interactive and engaging. Students also develop stronger problem-solving, algorithmic thinking, and practical programming skills through hands-on, project-oriented activities. However, the study highlights challenges such as the need for careful instructional design and teacher support to effectively integrate gamification elements. Overall, the approach is considered beneficial for enhancing learning outcomes and aligning Computer Science Education with real-world, skill-based demands.

The article "Measuring Undergraduates' Motivation Levels When Learning to Program in Virtual Worlds" by Juan Gabriel López Solórzano, Christian Jonathan Ángel Rueda and Osslan Osiris Vergara Villegas (contribution 9) investigates the effectiveness of virtual-world environments as pedagogical tools for introductory programming education. It employs a quasi-experimental pre-test/post-test design, comparing undergraduate students' motivation in traditional classroom settings with those learning in immersive virtual environments. Findings indicate that students exposed to virtual worlds demonstrate significantly higher levels of motivation than those taught through conventional methods, suggesting that immersive and interactive contexts can enhance engagement in programming education. The study also reports gender-related differences, with female students exhibiting greater confidence levels in the virtual learning condition. Methodologically, the research highlights the importance of integrating information and communication technologies (ICTs) into teaching strategies, while emphasizing the need for structured instructional design when adopting virtual environments. Overall, the article concludes that virtual worlds represent a promising complementary approach to traditional instruction, though their effectiveness depends on careful pedagogical implementation and alignment with course objectives.

5. Automatic Assignment Creation and Assessment

This thematic area consists of two papers, with the first one referring to the automatic construction of tests and assignments from large problem banks and the second one on automatic assessment tools that can be used both for exercise and project assessment.

The article "Searching Questions and Learning Problems in Large Problem Banks: Constructing Tests and Assignments on the Fly" by Oleg Sychev (contribution 10) addresses the challenge of generating effective assessments from large-scale repositories of automatically created or curated problems. It highlights that modern educational systems increasingly rely on extensive question banks, where manual inspection and selection by instructors become impractical. To address this issue, the study proposes an algorithm that constructs individualized tests dynamically based on teacher-defined constraints, such as required concepts and difficulty levels. The algorithm ensures that generated assignments meet pedagogical objectives by controlling the distribution of target concepts

https://doi.org/10.3390/computers15080497

while adapting difficulty according to the available problem pool. Empirical evaluation on a large dataset demonstrates that the method reliably produces diverse and balanced exercises, reducing predictability and limiting opportunities for cheating through shared solutions. Additionally, the results show that task difficulty is primarily influenced by the characteristics of the problem bank rather than parameter tuning alone. Overall, the article concludes that algorithmic generation of assessments can support scalable, individualized learning in contexts such as massive open online courses, provided that problem banks are well-structured and sufficiently rich in content.

The article "Bridging the Gap between Project-Oriented and Exercise-Oriented Automatic Assessment Tools" by Bruno Pereira Cipriano, Bernardo Baltazar, Nuno Fachada, Athanasios Vourvopoulos and Pedro Alves (contribution 11) examines limitations in existing automatic assessment tools (AATs) for programming education, particularly their difficulty in supporting both large-scale projects and small exercises within a unified framework. It introduces an IntelliJ IDEA plugin that extends the Drop Project AAT, enabling more efficient handling of short, exercise-based assignments while preserving the benefits of project-oriented assessment. Methodologically, the study combines system design with an empirical evaluation based on a student survey conducted during the 2022–23 academic year. Results indicate that the plugin improves usability, streamlines submission workflows, and integrates instructions and feedback directly within the development environment, thereby enhancing learning efficiency. Student responses suggest a generally positive perception, particularly regarding time savings and ease of use, although the small number of survey respondents limits the generalizability of findings. Such IDE-integrated tools can bridge the pedagogical divide between project-based and exercise-based learning approaches in Computer Science Education. The study concludes that bridging the gap between project-oriented and exercise-oriented AAT offers a promising direction for future educational technologies, provided they are iteratively refined through user feedback and broader evaluation.

6. Conclusions

Research in the field of Computer Programming Education has been at the forefront for several decades. Students' difficulties and misconceptions with programming concepts have been thoroughly investigated but are still under investigation, with the aim of devising effective didactical interventions and tools, such as intelligent tutoring systems. Moreover, they are utilized in combination with other data and techniques for predicting performance at early stages, in order to support students and prevent them from dropping out. In addition, contemporary forms of Information and Communication Technologies such as digital twins and virtual worlds, as well as innovative teaching approaches or combinations of them, such as task-driven case studies and gamified project-based learning, have been proposed for supporting and motivating students in learning programming. Another field that has attracted researchers' interest is the creation of tools that automate the process of devising assignments and tests from large problem banks and the process of assessing students' assignments, tools that are considered important nowadays for supporting large audiences and massive open online courses. Finally, programming has been considered a valuable means of cultivating CT skills, which are considered so important for everyone that several educational systems around the world make efforts towards incorporating them into their Computer Science curriculum from early ages.

The articles published in this Special Issue have dealt with all the aforementioned challenges and have highlighted other challenges and future trends as well that require further research.

https://doi.org/10.3390/computers15080497

Funding: This research received no external funding.

Acknowledgments: We would like to thank all the authors and peer reviewers for their valuable contributions to the Special Issue on "Future Trends on Computer Programming Education".

Conflicts of Interest: The author declares no conflicts of interest.

List of Contributions:

1. Chen, L.-H.; Nguyen, H.T.T. Study Trends and Core Content Trends of Research on Enhancing Computational Thinking: An Incorporated Bibliometric and Content Analysis Based on the Scopus Database. *Computers* **2024**, *13*, 91. https://doi.org/10.3390/computers13040091.
2. Yiatas, D.; Jimoyiannis, A. Enacting Computer Science Curriculum Reform: The Case of Model and Experimental Lower Secondary Schools in Greece. *Computers* **2026**, *15*, 140. https://doi.org/10.3390/computers15030140.
3. Manzanares, M.J.; Pérez Marín, D.; Pizarro, C. A Preliminary Usability Study of a Novel Educational Training System to Teach ScratchJr. in School. *Computers* **2026**, *15*, 17. https://doi.org/10.3390/computers15010017.
4. Sychev, O.; Denisov, M. Explain Trace: Misconceptions of Control-Flow Statements. *Computers* **2023**, *12*, 192. https://doi.org/10.3390/computers12100192.
5. Pires, J.P.J.; Brito Correia, F.; Gomes, A.; Borges, A.R.; Bernardino, J. Predicting Student Performance in Introductory Programming Courses. *Computers* **2024**, *13*, 219. https://doi.org/10.3390/computers13090219.
6. Lu, M.; Hu, Z. Digital Twin-Enhanced Programming Education: An Empirical Study on Learning Engagement and Skill Acquisition. *Computers* **2025**, *14*, 322. https://doi.org/10.3390/computers14080322.
7. Porubän, J.; Nosál', M.; Sulír, M.; Chodarev, S. Teach Programming Using Task-Driven Case Studies: Pedagogical Approach, Guidelines, and Implementation. *Computers* **2024**, *13*, 221. https://doi.org/10.3390/computers13090221.
8. Díaz-Lauzurica, B.; Moreno-Salinas, D. Gamified Project-Based Learning in Vocational Education and Training Computer Science Courses. *Computers* **2026**, *15*, 82. https://doi.org/10.3390/computers15020082.
9. López Solórzano, J.G.; Ángel Rueda, C.J.; Vergara Villegas, O.O. Measuring Undergraduates' Motivation Levels When Learning to Program in Virtual Worlds. *Computers* **2024**, *13*, 188. https://doi.org/10.3390/computers13080188.
10. Sychev, O. Searching Questions and Learning Problems in Large Problem Banks: Constructing Tests and Assignments on the Fly. *Computers* **2024**, *13*, 144. https://doi.org/10.3390/computers13060144.
11. Cipriano, B.P.; Baltazar, B.; Fachada, N.; Vourvopoulos, A.; Alves, P. Bridging the Gap between Project-Oriented and Exercise-Oriented Automatic Assessment Tools. *Computers* **2024**, *13*, 162. https://doi.org/10.3390/computers13070162.

References

1. Robins, A.; Rountree, J.; Rountree, N. Learning and Teaching Programming: A Review and Discussion. *Comput. Sci. Educ.* **2003**, *13*, 137–172. [CrossRef]
2. Xinogalos, S. Designing and Deploying Programming Courses: Strategies, Tools, Difficulties and Pedagogy. *Educ. Inf. Technol.* **2016**, *21*, 559–588. [CrossRef]
3. Brusilovsky, P.; Calabrese, E.; Hvorecky, J.; Kouchnirenko, A.; Miller, P. Mini-Languages: A Way to Learn Programming Principles. *Educ. Inf. Technol.* **1997**, *2*, 65–83. [CrossRef]
4. Xinogalos, S.; Satratzemi, M.; Malliarakis, C. Microworlds, Games, Animations, Mobile Apps, Puzzle Editors and More: What Is Important for an Introductory Programming Environment? *Educ. Inf. Technol.* **2017**, *22*, 145–176. [CrossRef]
5. Xinogalos, S. Using Flowchart-Based Programming Environments for Simplifying Programming and Software Engineering Processes. In Proceedings of the 2013 IEEE Global Engineering Education Conference (EDUCON), Berlin, Germany, 13–15 March 2013; pp. 544–552. [CrossRef]
6. Wing, J.M. Computational Thinking. *Commun. ACM* **2006**, *49*, 33–35. [CrossRef]
7. Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Kafai, Y. Scratch: Programming for All. *Commun. ACM* **2009**, *52*, 60–67. [CrossRef]

8. Yauney, J.; Bartholomew, S.R.; Rich, P. A Systematic Review of "Hour of Code" Research. *Comput. Sci. Educ.* **2023**, *33*, 512–544. [CrossRef]
9. Du, J.; Wimmer, H.; Rada, R. Hour of Code: Can It Change Students' Attitudes toward Programming? *J. Inf. Technol. Educ. Innov. Pract.* **2016**, *15*, 53–70. [CrossRef] [PubMed]
10. Prensky, M. *Teaching Digital Natives: Partnering for Real Learning*; Corwin Press: Thousand Oaks, CA, USA, 2010.
11. Williams, L.A.; Kessler, R.R. All I Really Need to Know about Pair Programming I Learned in Kindergarten. *Commun. ACM* **2000**, *43*, 108–114. [CrossRef]
12. da Silva Estácio, B.J.; Prikladnicki, R. Distributed Pair Programming: A Systematic Literature Review. *Inf. Softw. Technol.* **2015**, *63*, 1–10. [CrossRef]
13. Satratzemi, M.; Xinogalos, S.; Tsompanoudi, D. Distributed Pair Programming in Higher Education: A Systematic Literature Review. *J. Educ. Comput. Res.* **2023**, *61*, 546–577. [CrossRef]
14. Raihan, N.; Siddiq, M.L.; Santos, J.C.; Zampieri, M. Large Language Models in Computer Science Education: A Systematic Literature Review. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS '25), Pittsburgh, PA, USA, 26 February–1 March 2025*; ACM: New York, NY, USA, 2025; pp. 938–944. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

https://doi.org/10.3390/computers15080497

computers

MDPI

Review

Study Trends and Core Content Trends of Research on Enhancing Computational Thinking: An Incorporated Bibliometric and Content Analysis Based on the Scopus Database

Ling-Hsiu Chen [1] and Ha Thi The Nguyen [1,2,*]

[1] Department of Information Management, Chaoyang University of Technology, Taichung 413310, Taiwan; ling@cyut.edu.tw
[2] Center for Research Development and Quality Assurance, Dong Thap Community College, 259 Thien Ho Duong Street, Hoa Thuan Ward, Cao Lanh City 81114, Vietnam
* Correspondence: theha@dtcc.edu.vn

Abstract: Over the last decade, research on evolving computational thinking (CT) has garnered heightened attention. Assessing the publication tendencies and nucleus contents of investigations on progressing CT to direct future research initiatives, develop policies, and integrate them into instructional materials is timely and exceedingly essential in education. Therefore, this research reviewed publications on progressing CT to identify research trends and core contents published in the Scopus database from 2008 to May 2022. For this reason, this study applied bibliometric and content analysis to 132 selected publications. After examining bibliometrics, the findings indicate a steady increase in publications related to game-based learning (GBL) and CT, reaching a peak in 2021, with the United States emerging as the most prolific contributor in terms of authors, institutions, and countries). The leading country in citations is primarily China. The document that received the most citations is Hsu's 2018 paper on "*Computers and Education*". Analysis of keywords and themes reveals core content tendencies, emphasizing teaching methods and attitudes aimed at improving CT via GBL. These results offer valuable insights for researchers and educators to inform their future work. However, future studies may benefit from including other databases such as Web of Science (WoS) and PubMed, employing alternative bibliometric software like VOSviewer or CiteSpace, as well as collecting data from June 2022.

Keywords: computational thinking; bibliometric analysis; game-based learning; content analysis; content trend

1. Introduction

CT has attracted the interest of educational scholars and practitioners in recent years as it has emerged as a vital ability [1] for the twenty-first century [2]. Others emphasize CT as a cognitive process, while some experts describe it as a [3] problem-solving strategy [4]. For computer science (CS) and all sciences, the acquisition of CT is fundamental [5]. Tsarava [6] mentioned that Wing also emphasized the importance of CT skills [7] for everyone and not only programmers and computer scientists [8]. Wing determined CT [9] as a method [10] for solving problems [11] that involve conceptualizing, creating abstractions [12], and designing systems [13]. It is similar to logical thinking [14] and necessitates understanding basic computing concepts [15].

According to a 2012 National Research Council (NRC) report, given the critical role that CT [16] plays in the twenty-first century, educators should help students understand CT [17] by incorporating its content into existing curricula [18] in schools across the globe [19]. The International Society of Education Technology urges teachers to teach and integrate these abilities into the curriculum [20] to equip students of all ages with problem-solving skills for

the real world [9]. In Wing's opinion, everyone can solve problems with CT skills with ideas from CS [2]. Utilizing CT skills, students can create solutions for complicated systems [21], and teachers can impart knowledge of the systems. For the management of information, solving problems, and understanding human behavior, CT skills must be developed [22]. The literature has given different definitions to CT, so it is arguable what universal abilities and skills comprise its development and how these should be incorporated into educational content [9]. In addition, significant research has been conducted to determine the CT-related skills, strategies, and tools that can support students' academic progress. Thus, the question of how to improve K–12 pupils' CT is receiving more attention [23].

Governments and non-governmental organizations (NGOs) have launched numerous programs and educational environments to assist the development of CT abilities as its popularity grows [24]. Notably, these have included a variety of kid- and teen-oriented computer-based, user-friendly, game-based platforms [24]. Because digital games are engaging and appealing to an extensive population of individuals, GBL has been suggested as one instructional framework for learning programming [25] that has been demonstrated to benefit students [9,26].

Due to technological advancements [27], digital GBL research [28] has recently garnered increased attention [29]. GBL research and application in education have exploded in the last ten years. In a learning environment known as GBL, knowledge and skills are incorporated into games so that students can learn while taking on challenges that require problem-solving [30] and competition [31]. A prior study built an application for virtual reality immersion with mini-games and an expedition to improve learner motivation [32], engagement, and immersive experience. Furthermore, students acquired knowledge to enhance their problem-solving abilities [33]. Some authors have asserted in recent years that teaching programming to children can help them learn and develop CT [34,35].

According to previous findings, students' CT skills can be improved by using programming approaches [36]. This method [2] has been positively linked to improving the student's cognitive abilities, self-management problem-solving, communication, collaboration, and critical thinking. In the context of a GBL setting, the study revealed that the most persistent learners demonstrated high levels of patience as they persistently sought the optimal solution across various themes [24]. Teachers can use computer gaming techniques to promote CT in elementary school children. These strategies not only help students develop their CT skills [37] and conceptual understanding [38] but also help students have less advanced prior learning attitudes and knowledge [39]. CT was not only a distinct discipline or field; it was also applicable to several other areas, particularly computer science and programming. Many nations have included courses about CT in K–12 curricula to develop students' programming, digital, and CT competencies. Previous results demonstrated that students' CT concerning conceptual skills and knowledge considerably improved after playing the adaptive computer game in education. Additionally, students who played the adaptive computer game had noticeably greater interest levels, happiness, acceptance, and flow state of technology in CT learning [38]. The advancement of computer curriculum and CT skills can be supported effectively by video game making in science material [40]. Previous research demonstrated [41] the effectiveness of learning CT via games [26] to foster students' motivation, teamwork, perseverance, abstraction, and creativity [16].

The above-mentioned evidence demonstrates that the GBL strategy and the students' CT skills are very significant because of their high effectiveness in educating students. The acquisition of CT skills [42] enhances students' problem-solving abilities [2], and GBL has been proposed as an effective instructional approach for enhancing CT skills [26,43]. Hence, it is crucial to discern trends via the analysis of literature aimed at enhancing CT skills in order to pinpoint the nation and paper that have made the most substantial contributions in identifying key themes, diverse keywords, and primary documents within each cluster, the interconnected network of author documents, and central content areas.

This study conducts content and bibliometric analysis with R programming [44] to analyze CT publications from 2008 to May 2022. The study adds to the body of literature

in the ensuing facets: (i) These identified trends and networks serve as valuable resources for researchers to search for the CT and GBL fields. The results serve as references, aiding researchers in understanding the current state of research, selecting research topics, determining suitable publication outlets, and providing guidance for future investigations. Collaboration with scholars from highly cited papers or countries is encouraged. (ii) Additionally, the findings are essential for administrators in formulating policies to support teachers and students, selecting research topics and trends for participation, and fostering international cooperation via research and exchange programs. Administrators can identify opportunities for development and training to assist instructors in integrating these trends into teaching practices. (iii) Furthermore, recognizing these trends assists instructors in creating dynamic and productive learning environments to prepare students for future challenges. Adaptation of teaching approaches allows for the seamless integration of these trends into the curriculum. The research also helps instructors design curricula that facilitate students' access to essential research and develop research skills. Instructors can identify related concepts and connections to integrate into lessons. In light of this, the main research objectives (RO) are listed as follows:

1. RO1. Publication trends of CT and GBL as factors to influence student education;
2. RO2. The most cited countries and the most globally cited documents;
3. RO3. The co-occurrence network and co-citation network.

In addition, big and time-changing data presents a challenge in detailed, specific, and accurate analysis to recognize trends in the acquisition of CT knowledge. It is very crucial to have a suitable method to perform this analysis. Bibliometric analysis is an appropriate choice in this case because some researchers claim the advantages of this analysis. As an illustration, Aria and Cuccurrulo [45] asserted that bibliometric analysis can bring about a transparent, systematic, and reproducible evaluation procedure. In contrast to other methods, bibliometrics offers more impartial and trustworthy evaluations. Bibliometrics is helpful in the context of the enormous amount of conceptual advancements, new information, and data. Unlike manually coding text-based data sources and customary content-based analysis, bibliometric analysis has distinctive aspects. The efficiency of implementing manual technology is typically constrained by the ongoing growth of literary data with the spread of "big data", while bibliometric analysis is particularly ideal for dealing with a large-scale dataset. Furthermore, since bibliometric analysis is predominantly quantitative-based [46], its findings are typically generated more objectively and reliably than those obtained via content analysis research methods, which often rely on prioritizing conceptual categories [47]. This analysis [48] provides advantages over experimentally based research approaches in terms of the quantity of data acquired and a more objective examination, allowing for the discovery of novel research patterns and topics. Also, this analysis was chosen because of its capacity for storing comprehensive data and its potential for making it freely accessible to academics.

Moreover, researchers used the bibliometric method to analyze trends and development in education, almost on GBL and rarely focusing on CT. For example, in a previous study, GBL related to distant learning was analyzed using bibliometric analytic techniques [49]. Hwang and Chen [31] conducted a comprehensive review and bibliometric analysis of the published papers to emphasize the applications of GBL and its tendencies.

The previous authors [50] utilized bibliometrics, text mining, and social network analysis to provide meaningful, current data to illustrate the evolution of the issue and the state of research at this time. In the bibliometric analysis of trends, Chen et al. 2022 [29] analyzed papers on GBL in mathematics education and science published from 1991 to 2020. The previous authors [51] investigated bibliometric data from papers about determining research trends from 2013 to 2016. The bibliometric analysis [48] of publications on STEM education games published in conference proceedings and journals between 2010 and 2020 is presented. In terms of CT, the previous study employed the bibliometric analysis approach [52] to examine changes in study trends related to CT over the past twelve years.

Additionally, raising CT skills on GBL approaches has a noteworthy impact on the existing educational system. In recent years, bibliometric analysis has been used in research on GBL or CT skills. However, using this analysis of documents on boosting CT via GBL is a gap in research. This study applies bibliometric and content analysis in the literature to find publication and content trends of CT improvement on GBL in education. The research utilizes data archived in the Scopus database, ensuring unlimited access for exploration. Section 2 of the study details the specific research procedure and the bibliometric methodology employed. Following this, Section 3 delves into the analysis of results, followed by a discussion and conclusion in Section 4. The final section of the research concludes by addressing the limitations of the employed methodology and recommendations for future work.

2. Materials and Methods

For data collection, this study utilized Elsevier's Scopus database, which integrates meticulously curated citations and abstract databases containing extensive academic literature across various disciplines [53]. Scopus boasts a substantial collection of peer-reviewed publications, including conference proceedings, journals, and books [54]. The bibliometric analysis provides a static representation of the field at a given point in time; the field growth over time can be tracked by dividing the timeframe into multiple periods [45,50]. The main emphasis of this study's methodology was the bibliometric analysis, which was reviewed in [55]. Bibliometric analysis can offer a quantitative evaluation [48] of a product of science in a particular field of research over a specific period. The investigation covered factors such as trends in publications and citations, prolific writers, journals, geography, collaboration, and the development of research themes [31,47,56]. Via the application of bibliometric analysis, this research can be evaluated using quantitative and qualitative methods, with parameter and theme analysis being the respective focuses of bibliometric studies [56]. The research procedure comprises two steps: data collection and data analysis, as illustrated in Figure 1. This figure demonstrates the specific activities undertaken in each step.

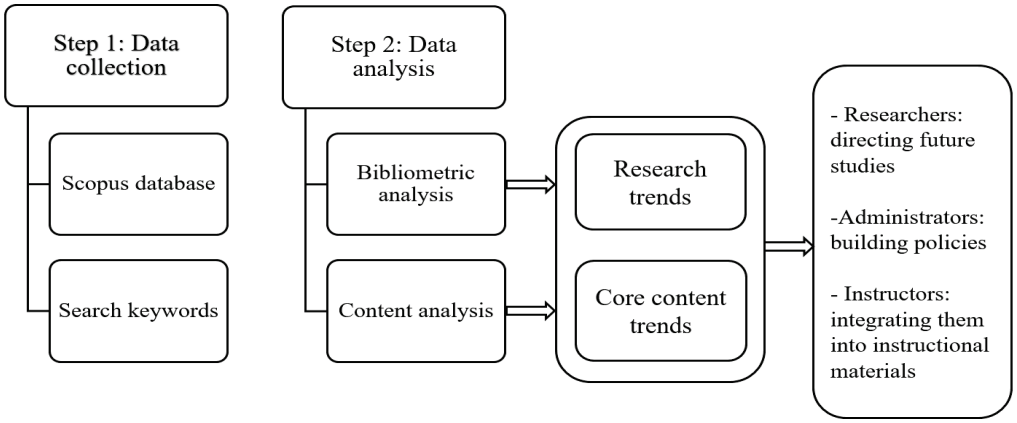


Figure 1. Steps of bibliometric and content analysis.

2.1. Step 1: Data Collection

The bibliometric analysis relies on data sourced from Scopus, which is renowned for its comprehensive coverage of scientific outputs relevant to this investigation. To conduct the necessary data collection, the search terms "game-based learning" and "computational thinking" were utilized in the Scopus interface, employing binary operators such as "AND" and restricting the search timeframe up to May 2022 with the language set to "English". Protocols were applied to ensure consistency and accuracy in database querying. The publications included in this study encompass book chapters, journal articles, and conference proceedings. According to Chand Bhatt [56], book chapters provide a theoretical

foundation for the fields of education in CT and GBL, while conference papers introduce new ideas for education research. The initial search yielded 4029 publications discussing CT and 5927 papers discussing GBL in education. After combining the two keywords using the "AND" operator, 132 publications were obtained, as shown in Table 1. The BibTeX method was employed to download the full text of each paper for further analysis.

Table 1. The keyword selection technique of the study.

Search Keywords	Result	Combination #1 AND #2
"computational thinking"	4029	132
"game-based learning"	5927	

2.2. Step 2: Data Analysis

2.2.1. Bibliometric Analysis

The bib file was downloaded to be used for data analysis by applying Biblioshiny from R-Studio with version 4.1.1 on 10 August 2021. Before uploading the data to Biblioshiny, the data were merged into a single bib file for using R-Studio, a development environment integrated with the R language [57]. Following this, the shiny application Biblioshiny, which provides a web interface for the bibliometric R-package [55], was utilized for data analysis. In the Biblioshiny interface, the Scopus file containing raw data was imported for analysis. This process resulted in the acquisition of the dataset, sources, authors, documents, and conceptual structure required for conducting descriptive and network analyses.

2.2.2. Content Analysis

The second technique employed in this study involves content analysis. The main aim of content evaluation is to categorize scientific findings conceptually and categorically to uncover underlying dynamics [58]. Previous research has suggested that employing content analysis helps in uncovering significant trends and crucial insights that may otherwise remain undiscovered [59]. Consequently, following the bibliometric method, all selected publications are analyzed to identify the trends of core contents.

3. Results

3.1. Bibliometric Analysis

3.1.1. Analysis of Trends

Data information is obtained after employing bibliometric analysis. Table 2 provides the distribution of the vital details of data collection. Documents with the keywords "computational thinking" and "game-based learning" were identified from 2008 to 2022, including 78 sources, 132 papers as book chapters, articles, conference papers, and 4320 references. A variety of document types were identified, with conference papers totaling the highest number at 84, followed by articles at 42. Conference reviews constituted the lowest count with 4, and book chapters were the least common with only 1. The document contents encompassed a total of 534 keywords, supplemented by 337 author's keywords. Among the document authors, a total of 327 were identified, with 14 authors of documents single-authored and 313 authors of documents multi-authored.

Publication trends are one of the most significant indicators in a field or sector [52]. A total of 132 papers published from 2008 to 31 May 2022 were identified. Figure 2 illustrates the development of scientific production trends using the keywords "CT" and "GBL" over the years. The publication trend exhibits slight oscillations within three documents from 2008 to 2014. However, there was a significant increase in documents from one paper in 2014 to five in 2015 and from 7 in 2016 to 13 in 2017. Subsequently, the number of published documents stabilized at around four each year from 2017 to 2021, with a notable nine papers in 2022. It is important to note that data for 2022 are collected from the first five months of the year. Based on these findings, publications focusing on CT and GBL

have garnered interest from researchers between 2008 and 2022. The publication trend has shown a consistent increase, reaching its peak in 2021.

Table 2. Principal data-related information.

Description	Results
Timespan	2008:2022
Sources (Books, Journals, etc.)	78
Publications	132
Average years from publication	3.26
The mean quantity of citations found in every document	7.386
Annual average of citations for each document	1.476
References	4320
Article	43
Chapter of a book	1
Conference document	84
Review of the conference	4
Plus keywords	534
Author's Keywords	337
Authors	327
Authors of single-authored documents	14
Authors of multi-authored documents	313
Documents for each author	0.404
Writers for each document	2.48
Co-authors for each document	3.52
Index of collaboration	2.75

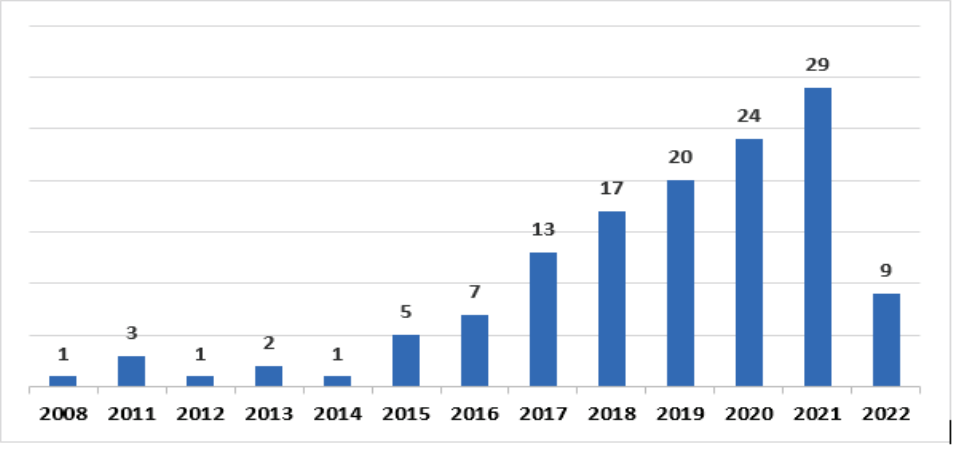


Figure 2. Annual scientific production.

Studies based on CT and GBL show relationships between authors, keywords, and countries. After applying bibliometrics to the top 20 authors, 19 authors, 7 countries, and 13 keywords were identified in a relationship in the research passage. Figure 3 illustrates a plot depicting the relationships among countries, authors, and keywords. Over the 14-year period from 2008 to 2022, three countries emerged as prominent among the seven countries, with many authors using keywords related to CT and GBL in their publications. For instance, the USA leads with nine authors, while Israel and Spain follow as the second countries, each with five authors.

During this period, the authors used a total of 13 keywords in their published documents. "CT" and "GBL" were the most commonly used keywords employed by 19 and 18 authors, respectively, in their publications. Additionally, "middle school" was utilized by 11 authors, while "educational games" and "collaboration" were used by six authors

each. These findings suggest that the top 19 authors, 13 keywords, and seven countries are interrelated in the context of CT and GBL during the specified period.

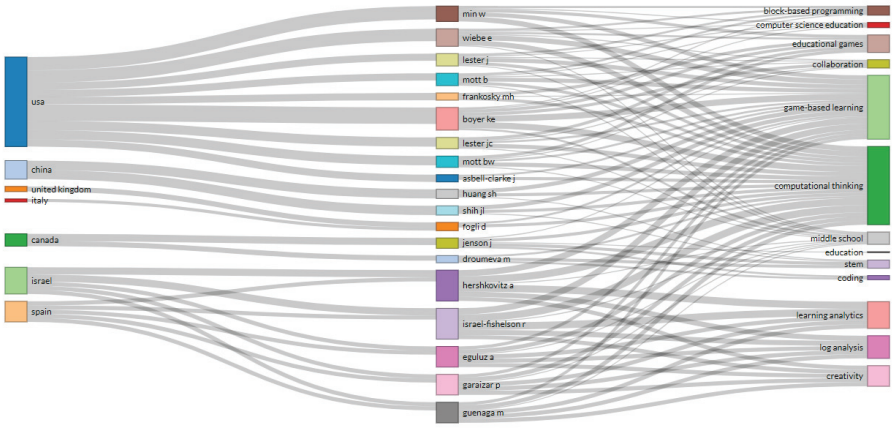
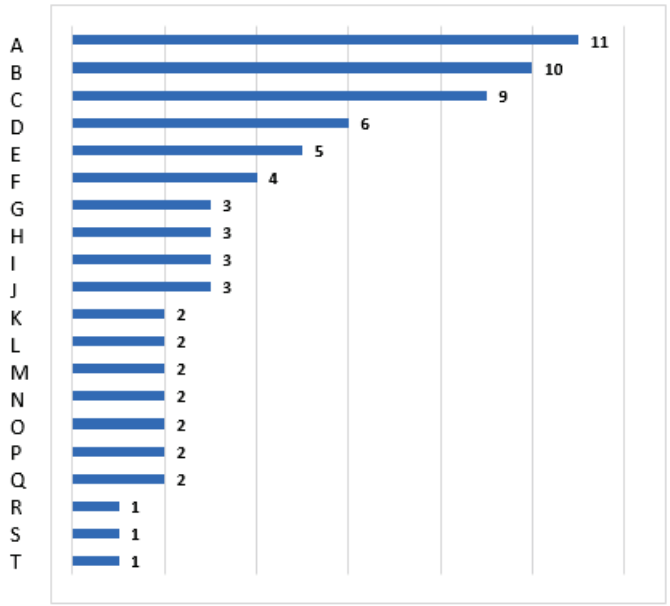


Figure 3. Three fields plot.

Based on the data obtained in the study, the top 20 publications comprising 74 papers related to CT and GBL for educational purposes were identified. Figure 4 illustrates the sources with the highest value during the specified period. Among these sources, a group of journal publications consistently increased the number of published papers. Among them, "*Lecture Notes in Computer Science*" emerged as the leading source with the highest number of published papers, totaling 11. Following closely is "*Conference Proceedings of the European Conference on GBL*" with 10 papers, and the "*ACM International Conference Proceeding Series*" with 9 papers. The group with the lowest number of published papers gradually increased, with "*Advances in Intelligent Systems and Computing*" starting with four papers and progressively increasing to five and six papers for the "*Journal of Educational Computing Research*" and the "*International Conference on CT Knowledge Proceedings*", respectively. Additionally, during this period, four journal sources had three publications each, seven sources had two publications each, and many sources had only one publication each. These findings indicate that six sources significantly influence the CT and GBL topic. "*Lecture Notes in Computer Science*" stands out as the most influential leading source during the specified period.

After identifying the top 20 contributing authors on CT and GBL topics from the selected data, a total of 90 papers were identified from the beginning of the research to May 2022. Figure 5 illustrates the authors who had the most impact during this period. The top three authors emerged as the most prominent, with publications related to CT and GBL. For instance, Boyer K. stood as the most active author with eight published documents. Following closely by authors Hershkovitz A. and Israel-Fishelson R., both have seven publications each. Min W and Wiebe E each have six published documents during this period.

Among the top 20 institutes identified from the selected data, the total number of published papers amounted to 88. Figure 6 illustrates the institutes with the most relevance. The universities ranked in the top four of documents related to CT and GBL published from 2008 to 2022. The most prominent university is North Carolina State University, with 11 published documents. The University of Tübingen follows in second place, albeit with the number of publications decreased, totaling eight during this period. Next in descending order are the University of Florida and Tel Aviv University, each with seven published documents, followed by the University of Salamanca with six.



A: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS)
B: PROCEEDINGS OF THE EUROPEAN CONFERENCE ON GAMES-BASED LEARNING
C: ACM INTERNATIONAL CONFERENCE PROCEEDING SERIES
D: PROCEEDINGS OF INTERNATIONAL CONFERENCE ON COMPUTATIONAL THINKING EDUCATION
E: JOURNAL OF EDUCATIONAL COMPUTING RESEARCH
F: ADVANCES IN INTELLIGENT SYSTEMS AND COMPUTING
G: CEUR WORKSHOP PROCEEDINGS
H: COMMUNICATIONS IN COMPUTER AND INFORMATION SCIENCE
I: EDUCATION AND INFORMATION TECHNOLOGIES
J: INFORMATICS IN EDUCATION
K: COMPUTERS IN HUMAN BEHAVIOR
L: ELECTRONIC JOURNAL OF E-LEARNING
M: IEEE GLOBAL ENGINEERING EDUCATION CONFERENCE EDUCON
N: INTERNATIONAL JOURNAL OF INTERACTIVE MOBILE TECHNOLOGIES
O: PROCEEDINGS - FRONTIERS IN EDUCATION CONFERENCE FIE
P: PROCEEDINGS OF THE 11TH EUROPEAN CONFERENCE ON GAMES BASED LEARNING ECGBL 2017
Q: PROCEEDINGS OF THE 14TH INTERNATIONAL CONFERENCE ON GAME BASED LEARNING ECGBL 2020
R: 2015 RESEARCH ON EQUITY AND SUSTAINED PARTICIPATION IN ENGINEERING COMPUTING AND TECHNOLOGY RESPECT 2015
S: 2017 IEEE 5TH INTERNATIONAL CONFERENCE ON SERIOUS GAMES AND APPLICATIONS FOR HEALTH SEGAH 2017
T: 2019 11TH INTERNATIONAL CONFERENCE ON VIRTUAL WORLDS AND GAMES FOR SERIOUS APPLICATIONS VS-GAMES 2019 - PROCEEDINGS

Figure 4. Most relevant sources.

Based on the obtained data, the top 20 countries with contributions to 210 published papers on CT and GBL have ranked. The publications of these countries are presented in Table 3. Active countries in this context refer to those that have contributed significantly to the literature body on CT and GBL by publishing ten or more papers during the specified period. These countries are considered to have made substantial contributions to research in the field, indicating a high level of engagement and productivity in CT and GBL-related research endeavors. Leading the pack with 55 publications is the United States, making it the most active nation. China follows in second place, albeit with a significant gap compared to the USA, with 23 publications. The third-ranking country, Greece, has several publications totaling 18.

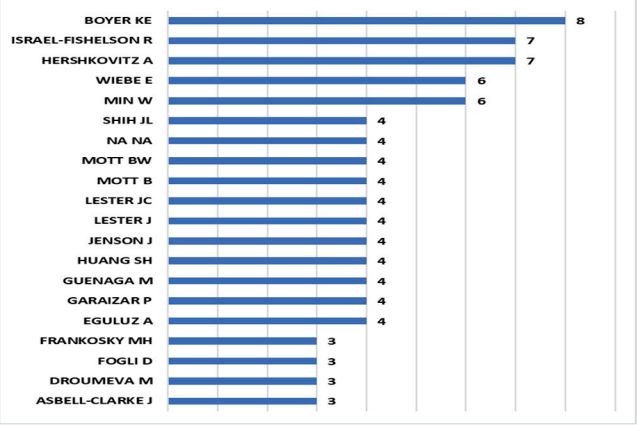


Figure 5. Most relevant authors.

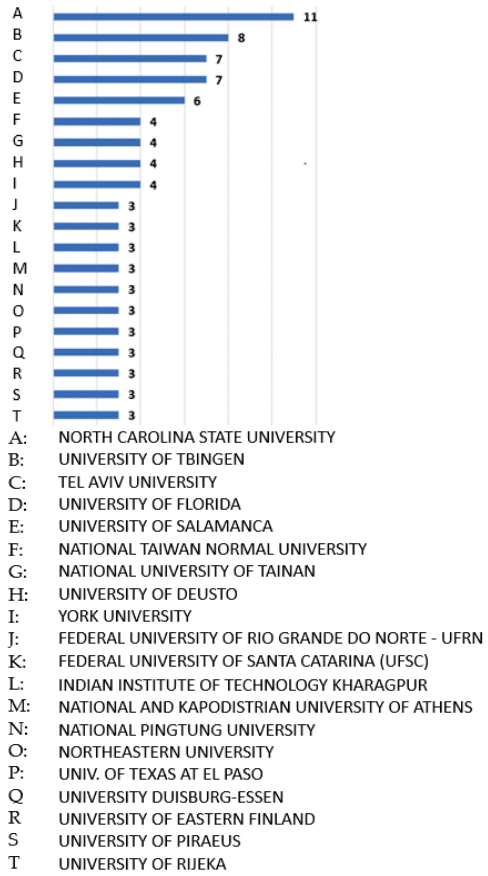


Figure 6. Most relevant institutes.

Table 3. The number of country article publications.

Rank	Country	Publication
1	USA	55
2	China	23
3	Greece	18
4	Germany	16
5	UK	14
6	Brazil	12
7	Spain	11
8	Canada	10
9	Malaysia	10

Table 3. *Cont.*

Rank	Country	Publication
10	Israel	7
11	Croatia	5
12	Finland	5
13	Sweden	5
14	Denmark	4
15	India	3
16	Italy	3
17	Thailand	3
18	Austria	2
19	Poland	2
20	Portugal	2

3.1.2. Citation Analysis of Countries and Documents

Citation Analysis of Countries

The countries with the most citations represent valuable contributions to the education community related to CT and GBL. A ranking of 20 nations based on citation volume is provided in Table 4. The most influential countries in CT and GBL research are from Asia and Europe, particularly Taiwan and Spain. Publication trends from Europe are also widespread. For example, China and Spain lead in citations with 206 and 164, respectively. The United Kingdom (Europe) and the USA (North America) follow closely, with 118 and 80 citations, respectively, ranking third and fourth. However, other European countries such as Greece, Estonia, Germany, Denmark, Poland, and Switzerland are also noteworthy. The number of quality papers on CT and GBL from various countries across continents has advanced, indicating intense global competition in investigations related to CT and GB.

Table 4. Most cited countries.

Rank	Country	Total Citations
1	China	206
2	Spain	164
3	UK	118
4	USA	80
5	Greece	40
6	Israel	40
7	Canada	16
8	Estonia	11
9	South Korea	5
10	Malaysia	3
11	Germany	2
12	Denmark	1
13	Poland	1
14	South Africa	1
15	Switzerland	1

Citation Analysis of Documents

The most cited articles demonstrate high-quality content with research hotspots on CT and GBL, which are attractive to researchers in related areas. The total number of citations of a document represents the value of its contribution to the community. From the given data, twenty published papers have been sorted by citation count using the R language and the Biblioshiny bibliometric program. The documents that have received the most citations worldwide are presented in Table 5. The top three journal articles are cited the most during the specified period.

Table 5. Most globally cited documents.

Paper	Total Citations
Hsu Tc, 2018, Comput Educ	186
Garca-Pealvo Fj, 2018, Comput Hum Behav	119
Kazimoglu C, 2012, Procedia Comput Sci	101
Jenson J, 2016, Electron J E-Learning	36
Kazimoglu C, 2011, International Journal of Game-Based Learning	30
Tsarava K, 2017, Proc Eur Conf Games Based Learn, Ecgbl	28
Kanaki K, 2018, Educ Inf Technol	27
Akram B, 2018, Proc Int Conf Educ Data Min, Edm	23
Wu Ml, 2011, Lect Notes Comput Sci	22
Hershkovitz A, 2019, Interact Learn Environ	21
Garca-Pealvo Fj, 2018, CT In The Stem Disciplines: Foundations And Research Highlights	21
Min W, 2015, Lect Notes Comput Sci	19
Francisco Jgp, 2016, Acm Int Conf Proc Ser	18
Min W, 2020, Ieee Trans Learn Technol	17
Turchi T, 2019, Multimedia Tools Appl	17
Israel-Fishelson R, 2020, J Educ Comput Res	14
Min W, 2017, Lect Notes Comput Sci	13
Hooshyar D, 2021, J Educ Comput Res	11
Altanis I, 2018, Educ Sci	11
Lin Sy, 2020, Elect Commer Res Appl	10

Table 5 shows evidence that the paper authored by Hsu TC in 2018, published in the journal "*Computers & Education*", has received the highest number of citations, totaling 186. Following closely, the paper by García-Peñalvo FJ, published in 2018 in "*Computers in Human Behavior*", includes 119 citations. Ranking third is the paper by Kazimoglu C., published in 2012 in "*Procedia Computer Science*".

3.1.3. Network Analysis

Co-Occurrence Network

In this section, emphasis is placed on analyzing co-occurrence networks using the Biblioshiny bibliometric software with the R language. In terms of selecting keywords, authors can establish connections between different research topics and identify closely related ones. Moreover, the frequency of a keyword can indicate the primary subjects covered in articles within the field [52]. In this paper, the bibliometric method is applied to organize themes into clusters related to CT and GBL. Consequently, the primary themes have emerged from the beginning of the research to the present. Figure 7 shows the co-occurrence network with four clusters containing various nodes based on CT and GBL keywords. Betweenness represents the correlation of a keyword for CT and GBL domains in papers. A higher betweenness value suggests a stronger correlation of the keyword within the documents of a cluster. The number of nodes reflects the diversity of themes in the CT and GBL research field.

From Figure 7, the red network represents cluster 1, comprising 29 nodes forming the primary network. The node "CT" exhibits the highest betweenness at 473.4, followed by "GBL" with a betweenness of 176.8. "Students" comes next with a betweenness of 127.8. Among the top five nodes, "education computing" has the lowest betweenness at 27.3, whereas "computer games" has a higher betweenness at 54.3. This evidence highlights the correlation between keywords such as CT [60], GBL [61], students [62], computer games [63], and others.

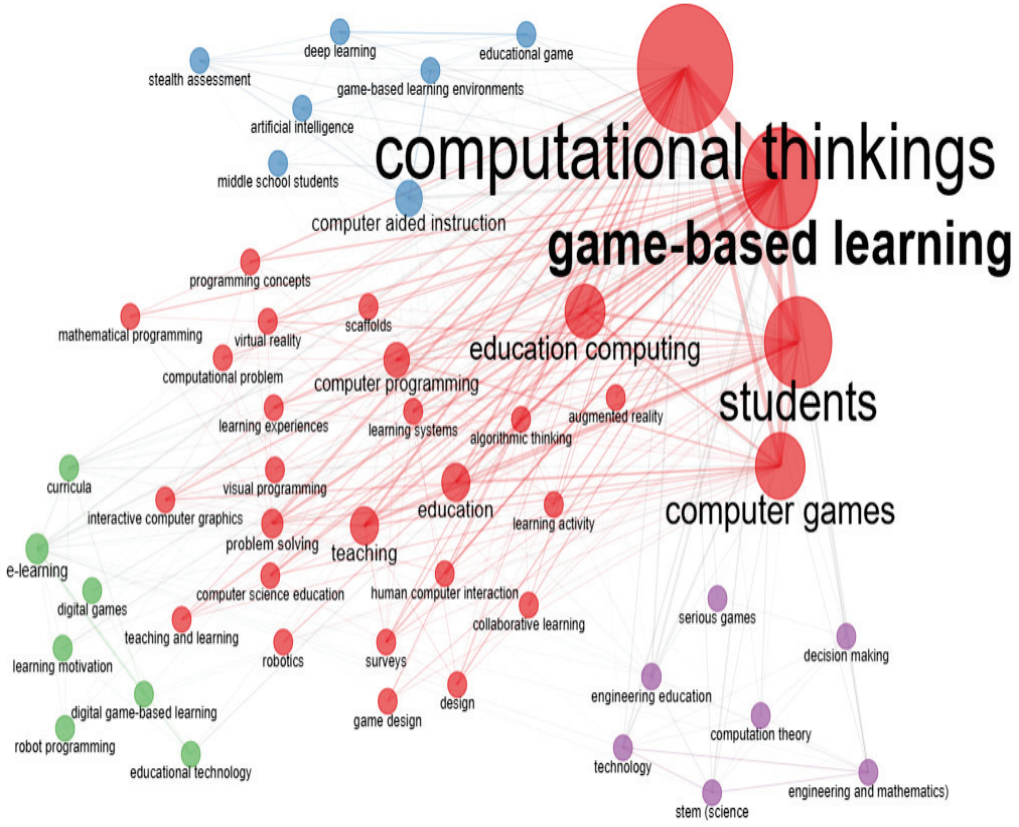


Figure 7. Co-occurrence network.

Cluster two appears as the blue network with seven nodes. The top three nodes with the highest betweenness include "computer-aided instruction" at 5.96, "GBL environments" at 0.51, and "artificial intelligence" at 0.25. This demonstrates the relationship among keywords like computer-aided instruction [24], GBL environments [64], artificial intelligence, educational games [65], and middle school students [66].

Cluster 3 forms a green network with seven nodes. The node "e-learning" has the highest betweenness at 9.5, while nodes such as "curricula" and "educational technology" have lower betweenness at 1.9 and 0.5, respectively. This cluster suggests themes related to e-learning, including curricula, digital games, learning motivation, digital GBL [67], and others.

Cluster 4 emerges a purple network with seven nodes, with three nodes exhibiting the highest betweenness. "Engineering education" leads with a betweenness of 0.9, followed by "technology" at 0.5 and "decision-making" at 0.4. This cluster identifies domains related to engineering education, such as technology [68], decision-making [69], computation theory, serious games [70,71], and others.

Co-Citation Network

Figure 8 illustrates the co-citation network comprising three distinct networks identified after applying the Biblioshiny bibliometric software with the R language. The co-citation network reveals three clusters and 38 documents centered around CT and GBL. In the co-citation network, betweenness is described as the interaction of documents' authors on themes based on CT and GBL and indicates the most cited author's documents. A higher betweenness signifies increased interaction among authors' documents. The nodes in the network represent the number of authors' documents interacting in the research field based on CT and GBL.

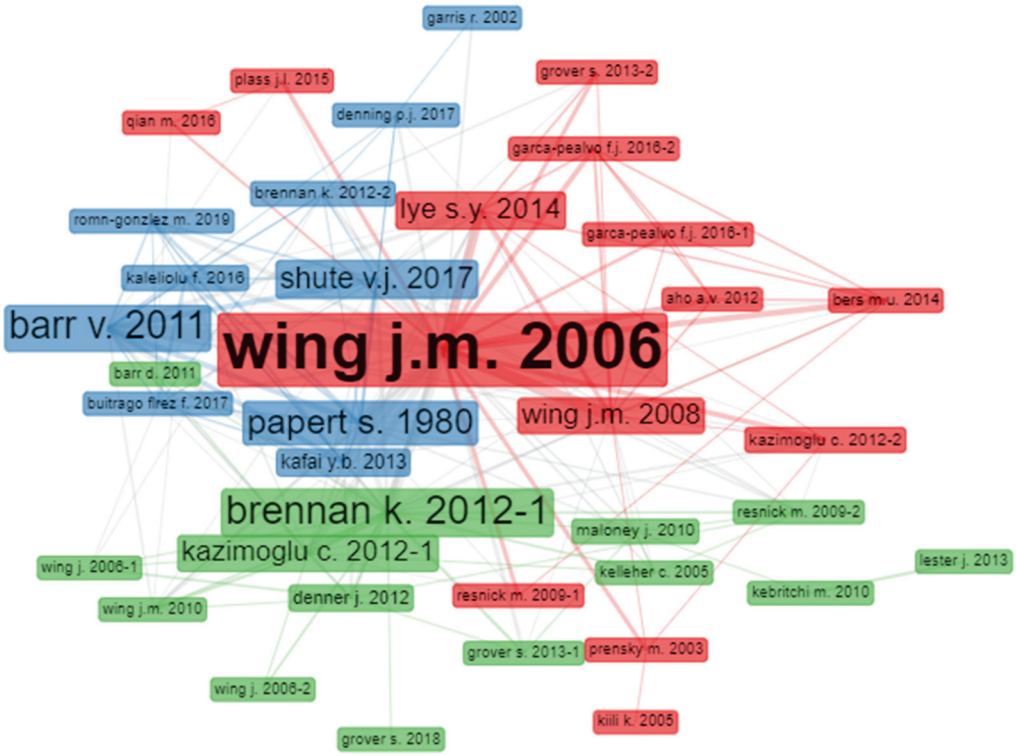


Figure 8. Co-citation network.

The first cluster forms the primary network with the red nodes encompassing 14 documents. The most significant article in this cluster, with a betweenness of 321.9, is based on CT and was explained by Wing [72] who initially introduced the concept as skills and attitudes for learning and application. This article has garnered over 4000 citations. Another article by the same author emphasizes that CT will impact students as they face new academic challenges [73]. Additionally, a paper by Prensky [74] identifies that digital GBL helps children stay motivated in their learning.

The second noticeable cluster is a green network with 14 nodes. The primary documents in this cluster are based on the [75] development of CT via digital games [76] to improve students' performance [77] and attitudes. The document with the highest betweenness in this cluster, at 128.6, revolves around the development of CT via Scratch programming [78]. This paper, authored by Brennan and Resnick, has amassed over 4000 citations. Additionally, Maloney et al. [79] illustrated an independent study facilitated by experimentation, peer collaboration, and creations [80] promoted by the Scratch Programming Language. Furthermore, Kebritchi et al. [81] demonstrated that digital games contribute to motivation [82] and students performance.

An impressive third cluster is blue nodes with ten documents. The co-citation study draws attention to the primary theme of CT in K–12 and teaching programming. In this cluster, Barr and Stephenson [83] identified that the delivery of CT for K–12 is the most significant article, and it necessitates systemic change, teacher involvement, and valuable significant resource creation. More than 4000 citations are included in the paper. It provides a comprehensive explanation of CT, covering aspects such as definition, intervention, curricular evaluation, and learning models [84]. Moreover, it demonstrates how CT skills can be developed via various means, including games, interactive stories [85], and simulations [78].

3.2. Emerging Content Trends

Trends in emerging content are discerned by analyzing documents from the co-occurrence and the co-citation network. These networks, as shown in Figures 7 and 8, reveal the interconnections among key concepts and prominent authors in the CT and GBL

fields. By examining documents, researchers can identify research hotspots and academic interest trends.

The analysis of content within publications based on the co-occurrence network highlights several primary trends across different clusters. In cluster 1, research primarily focuses on CT and GBL, encompassing areas such as teaching and learning tools, teaching and learning activities, teaching approach, GBL assessment framework, and attitudes like engagement, motivation, learning interest, confidence, encouragement [86–99]. Cluster 2 delves into CT, GBL, computer science, and assessment frameworks, particularly emphasizing strategies for teaching CT [99,100]. Cluster 3 researches CT, GBL, digital tools, and teaching strategies, focusing on attitudes such as motivation and engagement [15,101–103]. Cluster 4 explores factors related to CT development, game design, teaching method, attitudes (especially engagement, encouragement, and confidence), GBL, and learning activities [93,104–107]. Overall, the analysis underscores trends in teaching methods, activities in teaching and learning, tools for learning and teaching, frameworks for assessment, and attitudes such as engagement, motivation, encouragement, and confidence across the clusters.

The analysis of main content trends derived from content analysis on studies based on the co-citation network reveals distinct themes across various clusters. In cluster 1, a prevalent trend is the digital game utilization to facilitate learning and teaching CT, which serves to motivate student learning [108]. Cluster 2 predominantly focuses on promoting CT via digital games to enhance attitudes and improve students' learning outcomes [34,109]. Notably, computer games are found to positively impact students' achievement and motivation.

Meanwhile, cluster 3 highlights themes such as CT integration in K-12 education, instructional methods for CT, programming instruction, and the development of CT skills via learning games, interactives, and simulations [78,110–113]. These tendencies in co-citation networks underscore the importance of teaching approaches aimed at enhancing students' achievement and attitudes via GBL [114].

3.3. Sustainable Education in the Future

By integrating games into teaching and fostering students' CT skills, this study aims to provide a valuable resource for researchers, educational policymakers, and instructors, ensuring long-term sustainability.

In terms of teaching students, it is hoped that the findings will positively influence student education with more attention from teachers, education leaders, and researchers on trends of developing students' CT skills via GBL. Additionally, it anticipates that educators will consistently employ creativity and dedication in each teaching session to accomplish educational objectives, including enhancing student engagement [109], motivation [43], attitudes [115], and academic performance [116]. This underscores the importance of integrating educational games into the curriculum to foster the development of CT skills within the context of the technology age [90]. Games help students enhance mental computation skills [117] by acquiring CT skills [118]. The Trachtenberg system, recognized for its rapid mental computation capabilities [119], consists of a set of easily memorized operations facilitating quick arithmetic calculations. These operations entail straightforward algorithms aimed at helping students enhance their CT skills [120]. Therefore, it expects that there are appropriate policies to promote the development of the application of GBL to develop students' CT skills and help students achieve academic performance. This study will be a helpful source to researchers as a springboard for planning upcoming studies. Given the significance of GBL's effectiveness in fostering CT [34], it is anticipated that this teaching strategy will continue to gain traction across various subjects and academic levels, including mathematics, language arts, and science [16] in the foreseeable future [121].

In this research, the most prevalent themes included CT and GBL, along with computer-aided instruction, e-learning, and engineering education. Given the global impact of COVID-19, these themes are particularly pertinent, as they address challenges faced in

the shift to online learning environments. When students engage in GBL to acquire CT skills [26,122] online [123], various challenges arise. However, the integration of CT skill development via GBL offers numerous benefits to learners, including achieving learning objectives, fostering motivation [124], and enhancing problem-solving abilities. Ultimately, this approach cultivates a generation equipped with the skills and knowledge essential for the modern era.

4. Discussion and Conclusions

The bibliometric analysis delves into trends, co-occurrence, and co-citation network patterns sourced from the Scopus database, emphasizing parameter and theme analysis. Out of a pool of 132 publications selected based on two keywords, the study utilizes R-studio in conjunction with Biblioshiny software to facilitate bibliometric analysis. Via this approach, the research objectives are identified, leading to the emergence of valuable insights that can benefit researchers, administrators, and instructors alike.

RO1 focuses on investigating the publication trends influencing student education, particularly in the areas of CT and GBL. Analysis of data spanning from 2008 to May 2022 reveals a noteworthy increase in publications related to these areas, particularly evident in 2021. This upward trend indicates a growing interest [125] in leveraging GBL [29] to enhance CT skills and positively impact student learning. Contributions from regions such as North America (USA), Asia (Israel), and Europe (Spain) have been substantial and are expected to continue expanding. Notably, publications from sources and countries like "*Lecture Notes in Computer Science*" by Springer and the United States, including the most published authors and most published institutes have emerged as vital references for future research. For example, Boyer is the most contributing author with eight papers. The book "*Lecture Notes in Computer Science*" and the author's document, Boyer, are suggested as references for researchers for future studies. Searching for CT and GBL areas to support educators in teaching and learning, this book and this author are highlighted. Institutions like North Carolina State University in the USA have made significant contributions, offering valuable insights for educators and administrators alike. The results hope contributions from the highest publications on CT and GBL, such as developing instructors' professional capacity to improve students' performance, motivation, and engagement. Moreover, it expects to inspire other schools' instructors in their teaching and research to enhance learning communities.

By integrating these trends into teaching strategies and curricula, educators can support instructors via professional development and training and enhance student achievement, engagement, and motivation, fostering a conducive learning environment. This analysis serves as a foundational understanding of the current research landscape in CT and GBL, guiding future educational initiatives and research endeavors.

RO2 focuses on investigating the most cited countries and the most globally cited documents. The evaluation of citation results highlights China as the most cited country, followed by Spain during the study period. This underscores the significant contributions of China and Spain in the field of CT and GBL, attracting attention from researchers worldwide and prompting policymakers to support educational goals such as assisting instructors and brightening up students' achievement. Additionally, the most globally cited document, authored by Hsu in 2018 from Taiwan, focuses on acquiring and instructing CT, particularly via GBL. Hsu et al. focused on issue-based education, learning via projects, GBL in CT activities, collaborative learning, mathematical computing, programming skills training, and the CT performance and viewpoints of the students. This document stands out for its comprehensive exploration of CT development via GBL strategies and teaching tools [126], making it a valuable resource for instructors, policymakers, and researchers alike. Regarding understanding global research trends, researchers can identify impactful topics and potential collaborators, while administrators can facilitate international collaboration and curriculum development.

Instructors can leverage this data to enhance student access to significant research and emphasize the importance of research skills in their teaching. Policymakers can make appropriate policies to accommodate instructors and learners and for researchers who direct their future research.

RO3 focuses on investigating the co-occurrence network and co-citation network analysis of CT and GBL keywords. The co-occurrence analysis conducted in this study revealed the interaction among the most popular keywords, unveiling research hotspots and academic interests [31]. Clusters, representing groups of interconnected keywords, emerged from this analysis, each encapsulating distinct themes within the realm of CT and GBL. Notably, clusters 1, 2, 3, and 4 showcased prominent keywords such as CT and GBL, computer-aided instruction, e-learning, and engineering education, respectively. Cluster 1 exhibited the most diverse range of keywords, totaling 29 interactions, highlighting the breadth of topics explored in the realm of CT and GBL. Research within each cluster focused on distinct themes, including [127] the development of CT skills via GBL [128,129] teaching approaches [130] in cluster 1, assessment frameworks for GBL [131] in cluster 2, various aspects of CT and GBL, along with attitudes like engagement [132] and motivation [103] in cluster 3, and CT-related topics to be interesting in cluster 4. The content analysis of these clusters identified prevalent themes, including GBL evaluation frameworks, teaching strategies, and attitudes like confidence, encouragement, motivation, and [133] engagement. These research trends provide valuable insights for guiding future investigations and curriculum development in educational support. Administrators can leverage these findings to allocate resources effectively, while instructors can integrate relevant concepts into their teaching methodologies to enhance student learning experiences. Researchers can recognize the scope and interconnectedness of different concepts in the field to direct the development of research projects and identify current and emerging trends on CT and GBL to choose pertinent research topics in the future. Ultimately, these instructional approaches aim to equip students with the knowledge and skills necessary for success in today's digital age.

In the co-citation network analysis, clusters represent distinct groupings of scholarly documents and authors to be closely linked based on the frequency of citations between them. Three major clusters were identified in this study, each highlighting specific themes within the realm of CT and GBL research. Cluster 1 focuses on CT skills and their application in computer games [134], emphasizing the role of gaming in fostering learning and skill development. Notably, Wing's work stands as a seminal document in this cluster, highlighting key aspects of CT implementation. Cluster 2 delves into the advancement of CT via digital games, particularly emphasizing its impact on student achievement [40] and attitudes [135]. Brennan and Resnick's research [78] on CT evolution via Scratch programming emerges as a central document within this cluster. Cluster 3 centers around CT in K-12 education [136] and programming instruction. Barr and Stephenson's work serves as a cornerstone document in this cluster, advocating for the integration of CT into K-12 curricula, underscoring the importance of systemic reforms and teacher involvement. These clusters reveal the primary documents within each thematic area and highlight the interconnectedness of influential authors within the CT and GBL domains. By analyzing the content tendencies within these clusters, valuable insights into instructional strategies that enhance student performance and attitude via GBL are uncovered. This information can aid researchers and policymakers in orienting their research efforts and supporting student learning effectively. Additionally, these findings serve as a valuable resource for instructors and researchers seeking relevant materials on CT, GBL, and associated authors, guiding curriculum design and planning for the future.

5. Recommendation for Future Work

The research, conducted via a comprehensive review of the literature on "CT" and "GBL", provides valuable insights and guidance for future researchers. However, it is subject to certain limitations. For instance, it relies solely on research contributions listed in

the Scopus index, potentially overlooking relevant studies from databases like WoS and PubMed. Additionally, while the research employs Biblioshiny bibliometric software with Rstudio programming for analysis, alternative software options such as VOSviewer and CiteSpace could yield different perspectives. Future studies may benefit from exploring these alternative tools to enrich the analysis and broaden the scope of findings. Moreover, this research employed data collection of CT publications from 2008 to May 2022 to analyze the contents and bibliometric with R language. Future research should select data from June 2022.

Author Contributions: Conceptualization, L.-H.C. and H.T.T.N.; methodology, L.-H.C. and H.T.T.N.; software, L.-H.C. and H.T.T.N.; validation, L.-H.C. and H.T.T.N.; formal analysis, L.-H.C. and H.T.T.N.; investigation, L.-H.C. and H.T.T.N.; resources, L.-H.C. and H.T.T.N.; data curation, L.-H.C. and H.T.T.N.; writing—original draft preparation, L.-H.C. and H.T.T.N.; writing—review and editing, L.-H.C. and H.T.T.N.; supervision, L.-H.C. and H.T.T.N.; project administration, L.-H.C. and H.T.T.N.; funding acquisition, L.-H.C. and H.T.T.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data conducted in this research are available upon request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Zaibon, S.B.; Yunus, E. Perceptions of computational thinking in game based learning for improving student problem solving skills. *Int. J. Adv. Trends Comput. Sci. Eng.* **2019**, *8*, 181–184. [CrossRef]
2. Ezeamuzie, N.O.; Leung, J.S.C. Computational Thinking Through an Empirical Lens: A Systematic Review of Literature. *J. Educ. Comput. Res.* **2022**, *60*, 481–511. [CrossRef]
3. Israel-Fishelson, R.; Hershkovitz, A. Micro-persistence and difficulty in a game-based learning environment for computational thinking acquisition. *J. Comput. Assist. Learn.* **2021**, *37*, 839–850. [CrossRef]
4. Fanfarelli, J.R. Assessing Computational Thinking Pedagogy in Serious Games Through Questionnaires, Think-aloud Testing, and Automated Data Logging. In Proceedings of the 2021 IEEE/ACIS 20th International Fall Conference on Computer and Information Science (ICIS Fall), Xi'an, China, 13–15 October 2021; pp. 149–152. [CrossRef]
5. Voon, X.P.; Wong, S.L.; Wong, L.H.; Khambari, M.N.M.; Syed-Abdullah, S.I.S. Developing Computational Thinking Competencies through Constructivist Argumentation Learning: A Problem-Solving Perspective. *Int. J. Inf. Educ. Technol.* **2022**, *12*, 529–539. [CrossRef]
6. Tsarava, K.; Moeller, K.; Román-González, M.; Golle, J.; Leifheit, L.; Butz, M.V.; Ninaus, M. A cognitive definition of computational thinking in primary education. *Comput. Educ.* **2021**, *179*, 104425. [CrossRef]
7. Theodoropoulos, A.; Poulopoulos, V.; Lepouras, G. Towards a Framework for Adaptive Gameplay in Serious Games that Teach Programming: Association Between Computational Thinking and Cognitive Style. In *Advances in Intelligent Systems and Computing*; Springer International Publishing: Cham, Switzerland, 2021; Volume 1328, pp. 530–541, ISBN 9783030681975.
8. Yunus, E.; Zaibon, S.B. Connecting Computational Thinking (CT) Concept with the Game-Based Learning (GBL) Elements. *Int. J. Interact. Mob. Technol.* **2021**, *15*, 50–67. [CrossRef]
9. Kazimoglu, C.; Kiernan, M.; Bacon, L.; MacKinnon, L. Learning programming at the computational thinking level via digital game-play. *Procedia Comput. Sci.* **2012**, *9*, 522–531. [CrossRef]
10. Israel-Fishelson, R.; Hershkovitz, A.; Eguíluz, A.; Garaizar, P.; Guenaga, M. The Associations Between Computational Thinking and Creativity: The Role of Personal Characteristics. *J. Educ. Comput. Res.* **2021**, *58*, 1415–1447. [CrossRef]
11. Israel-Fishelson, R.; Hershkovitz, A.; Eguíluz, A.; Garaizar, P.; Guenaga, M. A Log-Based Analysis of the Associations Between Creativity and Computational Thinking. *J. Educ. Comput. Res.* **2021**, *59*, 926–959. [CrossRef]
12. Grizioti, M.; Kynigos, C. Children as players, modders, and creators of simulation games: A design for making sense of complex real-world problems. In Proceedings of the IDC '21: 20th Annual ACM Interaction Design and Children Conference, Athens, Greece, 24–30 June 2021; ACM: New York, NY, USA, 2021; pp. 363–374.
13. Hershkovitz, A.; Sitman, R.; Israel-Fishelson, R.; Eguíluz, A.; Garaizar, P.; Guenaga, M. Creativity in the acquisition of computational thinking. *Interact. Learn. Environ.* **2019**, *27*, 628–644. [CrossRef]
14. Rose, S.P.; Habgood, M.P.J.; Jay, T. Designing a Programming Game to Improve Children's Procedural Abstraction Skills in Scratch. *J. Educ. Comput. Res.* **2020**, *58*, 1372–1411. [CrossRef]
15. Garneli, V.; Chorianopoulos, K. Programming video games and simulations in science education: Exploring computational thinking through code analysis. *Interact. Learn. Environ.* **2018**, *26*, 386–401. [CrossRef]

16. Tran, Y. Computational Thinking Equity in Elementary Classrooms: What Third-Grade Students Know and Can Do. *J. Educ. Comput. Res.* **2019**, *57*, 3–31. [CrossRef]
17. Steinmaurer, A.; Pirker, J.; Gütl, C. Scool—Game-based learning in computer science class A case study in secondary education. *Int. J. Eng. Pedagog.* **2019**, *9*, 35–50. [CrossRef]
18. Fessakis, G.; Prantsoudi, S. Computer science teachers' perceptions, beliefs and attitudes on computational thinking in Greece. *Inform. Educ.* **2019**, *18*, 227–258. [CrossRef]
19. Kanaki, K.; Kalogiannakis, M. Introducing fundamental object-oriented programming concepts in preschool education within the context of physical science courses. *Educ. Inf. Technol.* **2018**, *23*, 2673–2698. [CrossRef]
20. Jenson, J.; Droumeva, M. Revisiting the media generation: Youth media use and computational literacy instruction. *E-Learning Digit. Media* **2017**, *14*, 212–225. [CrossRef]
21. Gao, X.; Hew, K.F. Toward a 5E-Based Flipped Classroom Model for Teaching Computational Thinking in Elementary School: Effects on Student Computational Thinking and Problem-Solving Performance. *J. Educ. Comput. Res.* **2022**, *60*, 512–543. [CrossRef]
22. Laura-Ochoa, L.; Bedregal-Alpaca, N. Incorporation of Computational Thinking Practices to Enhance Learning in a Programming Course. *Int. J. Adv. Comput. Sci. Appl.* **2022**, *13*, 194–200. [CrossRef]
23. Zhao, L.; Liu, X.; Wang, C.; Su, Y.S. Effect of different mind mapping approaches on primary school students' computational thinking skills during visual programming learning. *Comput. Educ.* **2021**, *181*, 104445. [CrossRef]
24. Israel-Fishelson, R.; Hershkovitz, A. Persistence in a Game-Based Learning Environment: The Case of Elementary School Students Learning Computational Thinking. *J. Educ. Comput. Res.* **2020**, *58*, 891–918. [CrossRef]
25. Demirkiran, M.C.; Hocanin, F.T. An investigation on primary school students' dispositions towards programming with game-based learning. *Educ. Inf. Technol.* **2021**, *26*, 3871–3892. [CrossRef]
26. Turchi, T.; Fogli, D.; Malizia, A. Fostering computational thinking through collaborative game-based learning. *Multimed. Tools Appl.* **2019**, *78*, 13649–13673. [CrossRef]
27. Portugal, E.; Cruz, F.; Ayala, A.; Fernandes, B. Analysis of Explainable Goal-Driven Reinforcement Learning in a Continuous Simulated Environment. *Algorithms* **2022**, *15*, 91. [CrossRef]
28. Lu, Y.; Li, W.; Li, W. Official International Mahjong: A New Playground for AI Research. *Algorithms* **2023**, *16*, 235. [CrossRef]
29. Chen, P.Y.; Hwang, G.J.; Yeh, S.Y.; Chen, Y.T.; Chen, T.W.; Chien, C.H. Three decades of game-based learning in science and mathematics education: An integrated bibliometric analysis and systematic review. *J. Comput. Educ.* **2022**, *9*, 455–476. [CrossRef]
30. Almeda, M.V.; Asbell-Clarke, J. Scaffolding executive function in game-based learning to improve productive persistence and computational thinking in neurodiverse learners. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer International Publishing: Cham, Switzerland, 2021; Volume 12790, pp. 155–172, ISBN 9783030774134.
31. Hwang, G.J.; Chen, P.Y. Interweaving gaming and educational technologies: Clustering and forecasting the trends of game-based learning research by bibliometric and visual analysis. *Entertain. Comput.* **2022**, *40*, 100459. [CrossRef]
32. Israel, M.; Liu, T.; Moon, J.; Ke, F.; Dahlstrom-Hakki, I. Methodological Considerations for Understanding Students' Problem Solving Processes and Affective Trajectories During Game-Based Learning: A Data Fusion Approach. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer International Publishing: Cham, Switzerland, 2021; Volume 12790, pp. 201–215, ISBN 9783030774134.
33. Joseph Agbo, F.; Oyelere, S.S.; Suhonen, J.; Tukiainen, M. IThinkSmart: Immersive Virtual Reality Mini Games to Facilitate Students' Computational Thinking Skills. In Proceedings of the Koli Calling '21: 21st Koli Calling International Conference on Computing Education Research, Joensuu, Finland, 18–21 November 2021.
34. Pérez-Marín, D.; Hijón-Neira, R.; Bacelo, A.; Pizarro, C. Can computational thinking be improved by using a methodology based on metaphors and scratch to teach computer programming to children? *Comput. Human Behav.* **2020**, *105*, 105849. [CrossRef]
35. Zhang, L.C.; Nouri, J. A systematic review of learning computational thinking through Scratch in K-9. *Comput. Educ.* **2019**, *141*, 103607. [CrossRef]
36. Sun, L.; Hu, L.; Zhou, D. Single or Combined? A Study on Programming to Promote Junior High School Students' Computational Thinking Skills. *J. Educ. Comput. Res.* **2022**, *60*, 283–321. [CrossRef]
37. Grizioti, M.; Kynigos, C. Code the mime: A 3D programmable charades game for computational thinking in MaLT2. *Br. J. Educ. Technol.* **2021**, *52*, 1004–1023. [CrossRef]
38. Hooshyar, D.; Pedaste, M.; Yang, Y.; Malva, L.; Hwang, G.J.; Wang, M.; Lim, H.; Delev, D. From Gaming to Computational Thinking: An Adaptive Educational Computer Game-Based Learning Approach. *J. Educ. Comput. Res.* **2021**, *59*, 383–409. [CrossRef]
39. Hooshyar, D. Effects of technology-enhanced learning approaches on learners with different prior learning attitudes and knowledge in computational thinking. *Comput. Appl. Eng. Educ.* **2021**, *30*, 64–76. [CrossRef]
40. Garneli, V.; Chorianopoulos, K. The effects of video game making within science content on student computational thinking skills and performance. *Interact. Technol. Smart Educ.* **2019**, *16*, 301–318. [CrossRef]
41. Teixeira, S.; Barbosa, D.; Araújo, C.; Henriques, P.R. Improving Game-Based Learning Experience through Game Appropriation. *OpenAccess Ser. Inform.* **2020**, *81*, 27:1–27:10. [CrossRef]

42. Valente, A.; Marchetti, E. Stickandclick—sticking and composing simple games as a learning activity. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer International Publishing: Cham, Switzerland, 2020; Volume 12206, pp. 333–352, ISBN 9783030505059.
43. Kazimoglu, C. Enhancing confidence in using computational thinking skills via playing a serious game: A case study to increase motivation in learning computer programming. *IEEE Access* **2020**, *8*, 221831–221851. [CrossRef]
44. Ionescu, Ș.; Delcea, C.; Chiriță, N.; Nica, I. Exploring the Use of Artificial Intelligence in Agent-Based Modeling Applications: A Bibliometric Study. *Algorithms* **2024**, *17*, 21. [CrossRef]
45. Aria, M.; Cuccurullo, C. bibliometrix: An R-tool for comprehensive science mapping analysis. *J. Informetr.* **2017**, *11*, 959–975. [CrossRef]
46. Dhanasekar, Y.; Anandh, K.S.; Szóstak, M. Development of the Diversity Concept for the Construction Sector: A Bibliometric Analysis. *Sustainability* **2023**, *15*, 15424. [CrossRef]
47. Chen, X.; Zou, D.; Xie, H. Fifty years of British Journal of Educational Technology: A topic modeling based bibliometric perspective. *Br. J. Educ. Technol.* **2020**, *51*, 692–708. [CrossRef]
48. Novia, N.; Permanasari, A.; Riandi, R. Research on educational games in STEM area 2010-2020: A bibliometric analysis of literature. *J. Phys. Conf. Ser.* **2021**, *1806*, 012209. [CrossRef]
49. Xu, J.; Duffy, B.M.; Duffy, V.G. Data Mining in Systematic Reviews: A Bibliometric Analysis of Game-Based Learning and Distance Learning. In *Lecture Notes in Computer Science*; Springer International Publishing: Cham, Switzerland, 2021; Volume 12777, pp. 343–354, ISBN 9783030778163.
50. Martí-Parreño, J.; Méndez-Ibáñez, E.; Alonso-Arroyo, A. The use of gamification in education: A bibliometric and text mining analysis. *J. Comput. Assist. Learn.* **2016**, *32*, 663–676. [CrossRef]
51. Arici, F.; Yildirim, P.; Caliklar, Ş.; Yilmaz, R.M. Research trends in the use of augmented reality in science education: Content and bibliometric mapping analysis. *Comput. Educ.* **2019**, *142*, 103647. [CrossRef]
52. Tekdal, M. Trends and development in research on computational thinking. *Educ. Inf. Technol.* **2021**, *26*, 6499–6529. [CrossRef]
53. Lefosse, D.; Timmeren, A.V.; Ratti, C. Biophilia Upscaling: A Systematic Literature Review Based on a Three-Metric Approach. *Sustainability* **2023**, *15*, 15702. [CrossRef]
54. Burnham, J.F. Scopus database: A review. *Biomed. Digit. Libr.* **2006**, *3*, 1. [CrossRef]
55. Agbo, F.J.; Sanusi, I.T.; Oyelere, S.S.; Suhonen, J. Application of virtual reality in computer science education: A systemic review based on bibliometric and content analysis methods. *Educ. Sci.* **2021**, *11*, 142. [CrossRef]
56. Chand Bhatt, P.; Kumar, V.; Lu, T.C.; Daim, T. Technology convergence assessment: Case of blockchain within the IR 4.0 platform. *Technol. Soc.* **2021**, *67*, 101709. [CrossRef]
57. Guerreschi, A.; Díaz López, F.J. A Bibliometric Analysis on Cooperatives in Circular Economy and Eco-Innovation Studies. *Sustainability* **2023**, *15*, 15595. [CrossRef]
58. Tang, K.Y.; Chou, T.L.; Tsai, C.C. A Content Analysis of Computational Thinking Research: An International Publication Trends and Research Typology. *Asia-Pac. Educ. Res.* **2020**, *29*, 9–19. [CrossRef]
59. Lin, T.J.; Lin, T.C.; Potvin, P.; Tsai, C.C. Research trends in science education from 2013 to 2017: A systematic content analysis of publications in selected journals. *Int. J. Sci. Educ.* **2019**, *41*, 367–387. [CrossRef]
60. Arroyo, I.; Closser, A.H.; Castro, F.; Smith, H.; Ottmar, E.; Micciolo, M. The WearableLearning Platform: A Computational Thinking Tool Supporting Game Design and Active Play. *Technol. Knowl. Learn.* **2023**, *28*, 1815–1824. [CrossRef]
61. Sukirman, S.; Ibharim, L.F.M.; Said, C.S.; Murtiyasa, B. A Strategy of Learning Computational Thinking through Game Based in Virtual Reality: Systematic Review and Conceptual Framework. *Inform. Educ.* **2022**, *21*, 179–200. [CrossRef]
62. Zhang, S.; Wong, G.K.W.; Chan, P.C.F. Playing coding games to learn computational thinking: What motivates students to use this tool at home? *Educ. Inf. Technol.* **2023**, *28*, 193–216. [CrossRef]
63. Hooshyar, D.; Malva, L.; Yang, Y.; Pedaste, M.; Wang, M.; Lim, H. An adaptive educational computer game: Effects on students' knowledge and learning attitude in computational thinking. *Comput. Hum. Behav.* **2020**, *114*, 106575. [CrossRef]
64. Min, W.; Frankosky, M.H.; Mott, B.W.; Wiebe, E.N.; Boyer, K.E.; Lester, J.C. Inducing stealth assessors from game interaction data. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer International Publishing: Cham, Switzerland, 2017; Volume 10331, pp. 212–223, ISBN 9783319614243.
65. Giannakoulas, A.; Terzopoulos, G.; Xinogalos, S.; Satratzemi, M. A Proposal for an Educational Game Platform for Teaching Programming to Primary School Students. In *Technology and Innovation in Learning, Teaching and Education*; Springer International Publishing: Cham, Switzerland, 2021; Volume 1–4, pp. 463–475, ISBN 9781668437117.
66. Werner, L.; Denner, J.; Campe, S. Using Computer Game Programming to Teach Computational Thinking Skills. In *Learning, Education and Games*; ETC Press: Pittsburgh, PA, USA, 2014; pp. 37–53, ISBN 978-1-312-54285-3.
67. Panskyi, T.; Rowińska, Z. A Holistic Digital Game-Based Learning Approach to Out-of-School Primary Programming Education. *Inform. Educ.* **2021**, *20*, 255–276. [CrossRef]
68. Antonio, M.S.P. Computational thinking beyond STEM: An introduction to "moral machines" and programming decision making in Ethics classroom. In Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality, Salamanca, Spain, 2–4 November 2016; ACM International Conference Proceeding Series; ACM: New York, NY, USA, 2016; pp. 37–44.

69. Pardo, A.M.S. Computational Thinking between Philosophy and STEM—Programming Decision Making Applied to the Behavior of "Moral Machines" in Ethical Values Classroom. *IEEE Rev. Iberoam. Tecnol. Aprendiz.* **2018**, *13*, 20–29. [CrossRef]
70. Ayman, R.; Sharaf, N.; Ahmed, G.; Abdennadher, S. MiniColon; Teaching kids computational thinking using an interactive serious game. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer International Publishing: Cham, Switzerland, 2018; Volume 11243, pp. 79–90, ISBN 9783030027612.
71. Akkaya, A.; Akpinar, Y. Experiential serious-game design for development of knowledge of object-oriented programming and computational thinking skills. *Comput. Sci. Educ.* **2022**, *32*, 476–501. [CrossRef]
72. Wing, J.M. Computational thinking. *Commun. ACM* **2006**, *49*, 33–35. [CrossRef]
73. Wing, J.M. Computational thinking and thinking about computing. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* **2008**, *366*, 3717–3725. [CrossRef]
74. Prensky, M. Digital game-based learning. *Comput. Entertain.* **2003**, *1*, 21. [CrossRef]
75. Troiano, G.M.; Schouten, D.; Cassidy, M.; Tucker-Raymond, E.; Puttick, G.; Harteveld, C. All Good Things Come in Threes: Assessing Student-Designed Games via Triadic Game Design. In Proceedings of the 15th International Conference on the Foundations of Digital Games, Bugibba, Malta, 15–18 September 2020; ACM International Conference Proceeding Series; ACM: New York, NY, USA, 2020.
76. Prompolmaueng, W.; Wetmaha, A.; Jamsri, P. A Game Development to Promote Computational Thinking. In Proceedings of the 2021 13th International Conference on Information Technology and Electrical Engineering (ICITEE), Chiang Mai, Thailand, 14–15 October 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 116–121.
77. Lin, S.Y.; Chien, S.Y.; Hsiao, C.L.; Hsia, C.H.; Chao, K.M. Enhancing Computational Thinking Capability of Preschool Children by Game-based Smart Toys. *Electron. Commer. Res. Appl.* **2020**, *44*, 101011. [CrossRef]
78. Brennan, K.; Resnick, M. 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, BC, Canada, 13–17 April 2012; Volume 1, p. 25.
79. Maloney, J.; Resnick, M.; Rusk, N.; Silverman, B.; Eastmond, E. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* **2010**, *10*, 1–15. [CrossRef]
80. Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J.; Silverman, B.; et al. Scratch: Programming for All. *Commun. ACM* **2009**, *52*, 60–67. [CrossRef]
81. Kebritchi, M.; Hirumi, A.; Bai, H. The effects of modern mathematics computer games on mathematics achievement and class motivation. *Comput. Educ.* **2010**, *55*, 427–443. [CrossRef]
82. Fernandes, K.T.; Da Silva Aranha, E.H.; Lucena, M.J.N.R.; De Souza Fernandes, G.L. Developing Computational Thinking and Reading and Writing Skills through an Approach for Creating Games. In Proceedings of the Frontiers in Education Conference, FIE 2020, Uppsala, Sweden, 21–24 October 2020.
83. Barr, V.; Stephenson, C. Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads* **2011**, *2*, 48–54. [CrossRef]
84. Shute, V.J.; Sun, C.; Asbell-Clarke, J. Demystifying computational thinking. *Educ. Res. Rev.* **2017**, *22*, 142–158. [CrossRef]
85. Miller, E.S. Designing Interactive Storytelling Games to Teach Computational Thinking. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer International Publishing: Cham, Switzerland, 2021; Volume 12789, pp. 342–351, ISBN 9783030772765.
86. Higashi, R.; Harpstead, E.; Solyst, J.; Kemper, J.; Odili Uchidiuno, J.; Hammer, J. The Design of Co-Robotic Games for Computer Science Education. In Proceedings of the Extended Abstracts of the 2021 Annual Symposium on Computer-Human Interaction in Play, Virtual Event, Austria, 18–21 October 2021; ACM: New York, NY, USA, 2021; pp. 111–116.
87. Hsu, T.C.; Chang, S.C.; Hung, Y.T. How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Comput. Educ.* **2018**, *126*, 296–310. [CrossRef]
88. Sharma, K.; Papavlasopoulou, S.; Giannakos, M. Coding games and robots to enhance computational thinking: How collaboration and engagement moderate children's attitudes? *Int. J. Child-Comput. Interact.* **2019**, *21*, 65–76. [CrossRef]
89. Threekunprapa, A.; Yasri, P. Unplugged coding using flowblocks for promoting computational thinking and programming among secondary school students. *Int. J. Instr.* **2020**, *13*, 207–222. [CrossRef]
90. Panskyi, T.; Rowinska, Z.; Biedron, S. Out-of-school assistance in the teaching of visual creative programming in the game-based environment—Case study: Poland. *Think. Ski. Creat.* **2019**, *34*, 100593. [CrossRef]
91. Komis, V.; Karachristos, C.; Mourta, D.; Sgoura, K.; Misirli, A.; Jaillet, A. Smart toys in early childhood and primary education: A systematic review of technological and educational affordances. *Appl. Sci.* **2021**, *11*, 8653. [CrossRef]
92. Agbo, F.J.; Oyelere, S.S.; Suhonen, J.; Laine, T.H. Co-design of mini games for learning computational thinking in an online environment. *Educ. Inf. Technol.* **2021**, *26*, 5815–5849. [CrossRef] [PubMed]
93. Kazimoglu, C.; Kiernan, M.; Bacon, L.; Mackinnon, L. Understanding computational thinking before programming: Developing guidelines for the design of games to learn introductory programming through game-play. *Int. J. Game-Based Learn.* **2011**, *1*, 30–52. [CrossRef]
94. Nutz, M.; Leifheit, L. Computer science unplugged: Developing and evaluating a "Traveling Salesperson Problem" board game. In Proceedings of the European Conference on Games-Based Learning, Brighton, UK, 23–24 September 2021; pp. 561–568.

95. Cano, S.; Naranjo, J.S.; Henao, C.; Rusu, C.; Albiol-pérez, S. Serious game as support for the development of computational thinking for children with hearing impairment. *Appl. Sci.* **2021**, *11*, 115. [CrossRef]
96. Dlab, M.H.; Hoic-Bozic, N.; Mezak, J.; Zunic, M. Supporting Croatian primary school teachers in designing game based learning activities: A case study. In Proceedings of the 14th International Conference on Game Based Learning, ECGBL 2020, Brighton, UK, 24–25 September 2020; pp. 125–133.
97. Romero-Mendez, E.A.; Santana-Mancilla, P.C.; Garcia-Ruiz, M.; Montesinos-López, O.A.; Anido-Rifón, L.E. The Use of Deep Learning to Improve Player Engagement in a Video Game through a Dynamic Difficulty Adjustment Based on Skills Classification. *Appl. Sci.* **2023**, *13*, 8249. [CrossRef]
98. Anuar, N.H.; Mohamad, F.S.; Minoi, J.L. Contextualising computational thinking: A case study in remote rural sarawak borneo. *Int. J. Learn. Teach. Educ. Res.* **2020**, *19*, 98–116. [CrossRef]
99. Merkouris, A.; Garneli, V.; Chorianopoulos, K. Programming human-robot interactions for teaching robotics within a collaborative learning open space: Robots playing capture the flag game: Programming human-robot interactions within a collaborative learning open space. In Proceedings of the CHI Greece 2021: 1st International Conference of the ACM Greek SIGCHI Chapter, Online, Greece, 25–27 November 2021; ACM International Conference Proceeding Series; ACM: New York, NY, USA, 2021.
100. Marcelino, M.J.; Pessoa, T.; Vieira, C.; Salvador, T.; Mendes, A.J. Learning Computational Thinking and scratch at distance. *Comput. Hum. Behav.* **2018**, *80*, 470–477. [CrossRef]
101. Francisco, J.G.P.; Juan, C.B. Computational thinking in pre-university education. In Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality, Salamanca, Spain, 2–4 November 2016; ACM International Conference Proceeding Series; ACM: New York, NY, USA, 2016; pp. 13–17.
102. Bauer, A.; Butler, E.; Popovic, Z. Dragon architect: Open design problems for guided learning in a creative computational thinking sandbox game. In Proceedings of the 12th International Conference on the Foundations of Digital Games, Hyannis, MA, USA, 14–17 August 2017; ACM International Conference Proceeding Series; ACM: New York, NY, USA, 2017. Part F1301.
103. Sáez-López, J.M.; Román-González, M.; Vázquez-Cano, E. Visual programming languages integrated across the curriculum in elementary school: A two year case study using "scratch" in five schools. *Comput. Educ.* **2016**, *97*, 129–141. [CrossRef]
104. Troiano, G.M.; Chen, Q.; Alba, Á.V.; Robles, G.; Smith, G.; Cassidy, M.; Tucker-Raymond, E.; Puttick, G.; Harteveld, C. Exploring How Game Genre in Student-Designed Games Influences Computational Thinking Development. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, 25–30 April 2020; ACM: New York, NY, USA, 2020; pp. 1–17. [CrossRef]
105. Miller, J.A.; Cooper, S. Case studies in game-based complex learning. *Multimodal Technol. Interact.* **2021**, *5*, 72. [CrossRef]
106. Psycharis, S.; Theodorou, P. Teaching Programming Skills to Blind and Visually Impaired Learners. In *Lecture Notes in Networks and Systems*; Springer International Publishing: Cham, Switzerland, 2022; Volume 349, pp. 270–279, ISBN 978-3-030-90676-4.
107. Camacho-Sánchez, R.; Rillo-Albert, A.; Lavega-Burgués, P. Gamified Digital Game-Based Learning as a Pedagogical Strategy: Student Academic Performance and Motivation. *Appl. Sci.* **2022**, *12*, 11214. [CrossRef]
108. Benakli, N.; Kostadinov, B.; Satyanarayana, A.; Singh, S. Introducing computational thinking through hands-on projects using R with applications to calculus, probability and data analysis. *Int. J. Math. Educ. Sci. Technol.* **2017**, *48*, 393–427. [CrossRef]
109. Luo, F.; Antonenko, P.D.; Davis, E.C. Exploring the evolution of two girls' conceptions and practices in computational thinking in science. *Comput. Educ.* **2020**, *146*, 103759. [CrossRef]
110. Buitrago Flórez, F.; Casallas, R.; Hernández, M.; Reyes, A.; Restrepo, S.; Danies, G. Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming. *Rev. Educ. Res.* **2017**, *87*, 834–860. [CrossRef]
111. Sharma, V.; Bhagat, K.K.; Huang, H.H.; Chen, N.S. The design and evaluation of an AR-based serious game to teach programming. *Comput. Graph.* **2022**, *103*, 1–18. [CrossRef]
112. Pellas, N.; Vosinakis, S. How can a simulation game support the development of computational problem-solving strategies? In Proceedings of the IEEE Global Engineering Education Conference, EDUCON, Athens, Greece, 25–28 April 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1129–1136.
113. Wang, D.; Luo, L.; Luo, J.; Lin, S.; Ren, G. Developing Computational Thinking: Design-Based Learning and Interdisciplinary Activity Design. *Appl. Sci.* **2022**, *12*, 11033. [CrossRef]
114. Jenson, J.; Droumeva, M. Exploring media literacy and computational thinking: A game maker curriculum study. *Electron. J. e-Learn.* **2016**, *14*, 111–121.
115. Nouri, J.; Zhang, L.; Mannila, L.; Norén, E. Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Educ. Inq.* **2020**, *11*, 1–17. [CrossRef]
116. Wong, Y.S.; Hayati, M.Y.M.; Tan, W.H.; Yap, L.C. A Game-Based Learning Assessment Framework for Learning Ubiquitous Computational Thinking. In *Advances in Intelligent Systems and Computing*; Springer International Publishing: Cham, Switzerland, 2020; Volume 1135, pp. 607–615.
117. DEBRENTI, E.; LÁSZLÓ, B. Developing Elementary School Students' Mental Computation Skills through Didactic Games. *Acta Didact. Napoc.* **2020**, *13*, 80–92. [CrossRef]
118. Pourdavood, B.R.; McCarthy, K.; McCafferty, T. The impact of mental computation on children's mathematical communication, problem solving, reasoning, and algebraic thinking. *Athens J. Educ.* **2020**, *7*, 241–254. [CrossRef]
119. Parajuli, K.K. Three separate methods for squaring: A connective prospective on Lilavati, Vedic and Trachtenberg. *Int. J. Stat. Appl. Math.* **2021**, *6*, 43–47. [CrossRef]

120. Ziatdinov, R.; Musa, S. Rapid mental computation system as a tool for algorithmic thinking of elementary school students development. *Eur. Res.* **2012**, *25*, 1105–1110.
121. Buffum, P.S.; Frankosky, M.; Boyer, K.E.; Wiebe, E.; Mott, B.; Lester, J. Leveraging collaboration to improve gender equity in a game-based learning environment for middle school computer science. In Proceedings of the 2015 Research on Equity and Sustained Participation in Engineering, Computing, and Technology, RESPECT, Charlotte, NC, USA, 13–14 August 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 1–8.
122. García-Peñalvo, F.J.; Mendes, A.J. Exploring the computational thinking effects in pre-university education. *Comput. Hum. Behav.* **2018**, *80*, 407–411. [CrossRef]
123. Schmidt, M.; Beck, D. Computational Thinking and Social Skills in Virtuoso: An Immersive, Digital Game-Based Learning Environment for Youth with Autism Spectrum Disorder. In *Communications in Computer and Information Science*; Springer International Publishing: Cham, Switzerland, 2016; pp. 113–121.
124. Altanis, I.; Retalis, S.; Petropoulou, O. Systematic design and rapid development of motion-based touchless games for enhancing students' thinking skills. *Educ. Sci.* **2018**, *8*, 18. [CrossRef]
125. Diago, P.D.; González-Calero, J.A.; Yáñez, D.F. Exploring the development of mental rotation and computational skills in elementary students through educational robotics. *Int. J. Child-Comput. Interact.* **2022**, *32*, 100388. [CrossRef]
126. Ausiku, M.; Matthee, M. Preparing Primary School Teachers for Teaching Computational Thinking: A Systematic Review. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer International Publishing: Cham, Switzerland, 2021; Volume 12511, pp. 202–213, ISBN 9783030669058.
127. Asbell-Clarke, J.; Rowe, E.; Almeda, V.; Edwards, T.; Bardar, E.; Gasca, S.; Baker, R.S.; Scruggs, R. The development of students' computational thinking practices in elementary- and middle-school classes using the learning game, Zoombinis. *Comput. Hum. Behav.* **2021**, *115*, 106587. [CrossRef]
128. Liu, H.; Wu, Z.; Lu, Y.; Zhu, L. Exploring the Balance between Computational Thinking and Learning Motivation in Elementary Programming Education: An Empirical Study with Game-Based Learning. *IEEE Trans. Games* **2023**, *15*, 95–107. [CrossRef]
129. Asbell-Clarke, J.; Robillard, T.; Edwards, T.; Bardar, E.; Weintrop, D.; Grover, S.; Israel, M. Including Neurodiversity in Foundational and Applied Computational Thinking (INFACT). In Proceedings of the SIGCSE 2022: Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2, Providence, RI, USA, 3–5 March 2022; ACM: New York, NY, USA, 2022; p. 1076. [CrossRef]
130. Alam, A. A Digital Game based Learning Approach for Effective Curriculum Transaction for Teaching-Learning of Artificial Intelligence and Machine Learning. In Proceedings of the International Conference on Sustainable Computing and Data Communication Systems, ICSCDS 2022—Proceedings, Erode, India, 7–9 April 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 69–74.
131. Min, W.; Frankosky, M.; Mott, B.W.; Rowe, J.P.; Wiebe, E.; Boyer, K.E.; Lester, J.C. DeepStealth: Leveraging deep learning models for stealth assessment in game-based learning environments. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer International Publishing: Cham, Switzerland, 2015; Volume 9112, pp. 277–286.
132. Weintrop, D.; Holbert, N.; Horn, M.S.; Wilensky, U. Computational thinking in constructionist video games. *Int. J. Game-Based Learn.* **2016**, *6*, 1–17. [CrossRef]
133. Yoon, C.S.; Khambari, M.N.M. Design, Development, and Evaluation of the Robobug Board Game: An Unplugged Approach to Computational Thinking. *Int. J. Interact. Mob. Technol.* **2022**, *16*, 41–60. [CrossRef]
134. Atmatzidou, S.; Demetriadis, S. Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Rob. Auton. Syst.* **2016**, *75*, 661–670. [CrossRef]
135. Repenning, A.; Webb, D.C.; Koh, K.H.; Nickerson, H.; Miller, S.B.; Brand, C.; Horses, I.H.M.; Basawapatna, A.; Gluck, F.; Grover, R.; et al. Scalable Game Design: A Strategy to Bring Systemic Computer Science Education to Schools through Game Design and Simulation Creation. *ACM Trans. Comput. Educ.* **2015**, *15*, 1–31. [CrossRef]
136. Lye, S.Y.; Koh, J.H.L. Review on teaching and learning of computational thinking through programming: What is next for K-12? *Comput. Hum. Behav.* **2014**, *41*, 51–61. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

computers

MDPI

Article

Enacting Computer Science Curriculum Reform: The Case of Model and Experimental Lower Secondary Schools in Greece

Dimitrios Yiatas and Athanassios Jimoyiannis *

Department of Social and Education Policy, University of the Peloponnese, 20100 Korinthos, Greece
* Correspondence: ajimoyia@uop.gr

Abstract

This paper presents the findings of a study on computer science teachers' views regarding the reform of the computer science curriculum and its implementation in Greek model and experimental lower secondary schools (called Gymnasiums). Research data were collected through interviews with 19 computer science teachers who implemented the new curriculum in 12 model or experimental junior high schools. The results showed that the teachers acknowledged the important role of the computer science curriculum, which is perceived as a tool guiding their instruction. They have also understood and adopted, to a large extent, many elements of the instructional framework proposed by the new curriculum: (a) student-centred teaching approaches; (b) learning activities that promote students' active participation, inquiry, and collaborative learning; (c) focus on the expected students' learning outcomes related to developing a range of computational competences such as digital skills, computational thinking, problem-solving skills, creativity, and collaboration; and (d) students' assessment is constructively aligned with the anticipated learning outcomes, including the learning activities implemented as well as their digital-computational creations.

Keywords: computer science instruction; computer science curriculum; curriculum enactment; teachers' views; learning design

1. Introduction

Over the past decade, integrating computer science (CS) into the K-12 curriculum, as a distinct subject even in the early grades of primary school, has been an emerging policy among many countries around the globe. A common element across the recommended CS curriculum standards is that they are not limited to students' development of digital skills but, instead, focus on cultivating computational thinking skills as well as connecting computational competencies with 21st-century skills [1–4]. In this context, two major educational challenges emerged concerning: (a) the content knowledge of computer science that should be taught in primary and secondary education and the CS competencies students are expected to develop [5–7]; and (b) the competencies that computer science educators need to possess to effectively implement a modern computer science curriculum [8–10].

Undoubtedly, the implementation of a new curriculum constitutes a significant educational reform, shaped by multiple factors related to educators' work. Teachers are required to depart from conventional practices and roles, embracing instead new objectives, strategies, and student-centred learning processes. Consequently, studying the actual enactment of a new CS curriculum often reveals substantial discrepancies when compared to the curriculum's intended goals and the underlying design philosophy [11]. Investigating

teachers' perspectives as well as their response to the educational changes demanded by new curricula has been the focus of extensive research in recent years with regard to various curricular subjects and school levels [12–16]. Such notable studies have focused on primary education programmes [17], mathematics [11,18–20], science [21,22], and history [23,24], among other subjects.

Computer science is a relatively new subject in most school curricula internationally. Consequently, research findings on teachers' views regarding CS curricula, as well as their instructional choices in enacting the CS curriculum, remain limited [25–27]. The present study aspires to contribute to the broader discourse regarding educational reforms in primary and secondary education by exploring the perspectives of Greek teachers who have systematically implemented the CS curriculum in model and experimental lower secondary education schools (K7–K9) over consecutive years. The analysis of the interview data revealed that the participating teachers acknowledged the significant role of the new CS curriculum in shaping their instructional choices. Furthermore, teachers appear to accurately identify the core elements of the curricular changes, as suggested by the CS curriculum, many of which were actively integrated into their enactment practices.

2. Literature Review

The discourse regarding K-12 curricula encompasses multiple dimensions and is not confined to subject-specific content but extends to both pedagogical considerations and educational policy factors as well. Unlike a syllabus, the curriculum is not limited to the content and organization of the subject matter to be taught. Although diverse approaches are observed in practice, the core characteristics of a curriculum generally establish an educational framework that includes: (a) the learning objectives pursued through the teaching of the subject; (b) the design of educational learning experiences with the aim to achieving these objectives; (c) the organization and support of students' educational experiences (i.e., the underlying pedagogical philosophy); and (d) suggested evaluation methods for the attainment of the anticipated learning objectives.

Modern curricula are oriented toward: (a) a comprehensive description of the learning outcomes that students are expected to achieve, which correspond to specific competencies (knowledge, skills, attitudes, and values), and (b) new forms of educational designs that promote active, exploratory, and collaborative learning while leveraging students' prior knowledge and experiences. Consequently, the design principles and the philosophy of a new curriculum need to be comprehensively understood, adopted, and applied by the teachers themselves so that the reforms entailed by a new curriculum will be effectively implemented.

Hewitt [28] identified two phases that educators pass through to implement a new curriculum: (a) adoption, where the curriculum is accepted by the educators and implemented according to how they perceive it (either precisely as designed or with adjustments), and (b) adaptation, which refers to modifications educators themselves make during curriculum implementation, diverging from the designers' original intentions. Educational research has highlighted numerous aspects and factors related to the gap between the official curriculum (as defined by the education policy documents) and what is ultimately enacted in the classroom. Remillard & Heck [19] noted that the enacted curriculum has been found to significantly differ when compared to the official one.

Curriculum implementation is a dynamic process shaped by multiple factors that structure the broader framework of teaching and learning within a subject. It requires shifts and transformations in teachers' perceptions regarding the role of the subject matter in student development, as well as in their approach to teaching and learning [29]. According to Pajares [30], educators, guided by their beliefs about teaching and learning, often modify

https://doi.org/10.3390/computers15030140

the curriculum, leading to adaptations, reinterpretations, or even misinterpretations during implementation. Bergqvist & Bergqvist [11] argue that adopting a new curriculum is neither automatic nor guaranteed to align with the intentions of the designers. Teachers frequently interpret the official curriculum in diverse ways, selectively utilizing elements they understand best or can easily adapt to traditional or familiar teaching practices. This gap often arises because entrenched teaching habits, reinforced by classroom experience, hinder educators from altering long-established practices. Therefore, the enacted curriculum is essentially reflecting a negotiation between policy mandates and the pragmatic realities of the classroom.

Research in other disciplines, concerning teachers' perceptions of curriculum reforms and their experiences of implementing a new curriculum in practice, provided some insight into how CS teachers come to apply the expected changes in the classroom as well as the challenges they face [12,18,22,31]. Compared to other STEM fields (i.e., science and mathematics), computer science has instructional differences due to the fast evolution of digital technologies as well as the computational methods used to solve real-life problems. Therefore, CS teachers may face new forms of difficulties in their instruction. Among other issues, many CS teachers are not well prepared for teaching specific topics (e.g., by using new types of programming languages), they lack appropriate pedagogical knowledge, they have limited lesson planning and classroom management skills, they lack sufficient time to design students' learning activities in the computer lab, etc.

An early study in New Zealand high schools by Thompson et al. [32] confirmed that CS teachers faced several barriers and difficulties in implementing the CS curriculum. They reported a lack of teaching support resources (e.g., lesson plans, assessment exemplars), a lack of quality material to familiarize students with new topics, low confidence or experience with programming tools, and insufficient time or opportunities for professional development and preparation for the new curriculum standards.

Similarly, an investigation in the USA [27] explored CS teachers' perspectives on the demands of teaching computer science. The findings suggested that they faced several challenges, including isolation, a lack of adequate background in computer science, and limited professional development resources. Another survey, conducted in the UK with 339 primary and secondary school teachers of computing [9], identified two main categories of teachers' challenges. The first were intrinsic challenges, related to teachers' subject knowledge and their pedagogical approaches. The second were extrinsic challenges, including lack of time and support, technical problems, limited resources, and issues related to students' assessment. In addition, the researchers highlighted five key pedagogical strategies commonly used in practice: unplugged activities, contextualized tasks, collaborative learning, computational thinking, and scaffolded programming tasks.

An international study by Falkner et al. [33] explored K-12 CS curriculum implementation across Australia, England, Ireland, Italy, Malta, Scotland, and the USA. The key idea was to compare the CS curriculum requirements (i.e., the intended curriculum), through national reports, with the enacted curriculum as identified by the teachers participating in a survey. The findings highlighted the similarities and differences between the intended and the enacted CS curriculum within and across countries, specifically exploring the differences in terms of programming languages used and CS topics implemented.

Samarasekara, Ott & Robins [34] conducted an online survey in New Zealand with 91 CS teachers who had been involved with the new CS standards. Their analysis identified five main categories of barriers and related challenges for both teachers and students that hinder successful curriculum implementation: (a) CS standards, in terms of interpretation and implementation in their instruction; (b) a lack of resources like exemplars, learning material, support material, and teaching guides; (c) teacher support, in terms of time and

https://doi.org/10.3390/computers15030140

opportunities for professional development; (d) relevant knowledge and understanding of the requirements of the assessment; and (e) student-centred factors. The key suggestions that emerged were the inclusion of more practical work instead of theory and writing, and more time for teachers to practice the subject.

More recently, Zhu & Wang [35] explored CS instruction in K-12 schools by using semi-structured interviews with five CS teachers and eight CS instructors in higher education. The findings identified three types of effective teaching strategies: (a) connection to real-world problems, (b) project-based assessment, and (c) teamwork. The main teachers' challenges included gaps in CS knowledge, pedagogical knowledge, professional development needs, and opportunities as well. Using a mixed-method of analyzing quantitative and qualitative data, Sadik & Ottenbreit-Leftwich [26] explored secondary CS teachers' challenges and needs in the USA. Curriculum resources, materials to assess students' learning, and embedding the principles of computational thinking into the curriculum were identified as major needs for secondary teachers. Teachers also reported that they need to enhance their pedagogical knowledge to support student-centred strategies, like problem-based learning, pair programming, scaffolding, and team-management strategies.

The analysis above clearly indicates that most studies were mainly focused on CS teachers' challenges and difficulties, as well as on their instructional strategies applied in practice. Therefore, research findings concerning teachers' views about the CS curriculum and its enactment in schools remain notably limited.

3. Context and Research Questions

3.1. The New CS Curriculum for Lower Secondary Education

In 2011, the Greek Ministry of Education announced a major curriculum revision for lower secondary education (K7–K9). With regard to computer science and information and communication technologies (ICT), this revision aimed to establish a new competency framework, redetermining the knowledge, skills, attitudes, and values that were deemed to be essential components of the students' compulsory education as well as their lifelong development. Furthermore, the revised CS curriculum was aligned with the key principles of 21st-century education [36]. Thus, the new curriculum was structured upon four dimensions, with all of them requiring significant shifts in teaching and learning:

- CS as a scientific field and a technological tool (core concepts).
- CS-ICT as a problem-solving tool (computational thinking, computer programming, collaborative inquiry, and innovation).
- ICT as a cognitive and learning tool (exploration, inquiry, creativity, communication, and cross-disciplinary learning).
- ICT as a social phenomenon (digital citizenship, cultural awareness, ethics, and values).

3.2. Research Context—Model and Experimental Schools in Greece

The implementation of a new curriculum often requires a radical transformation in teachers' perceptions regarding subject matter knowledge, pedagogy, and learning processes [29,37]. A critical factor influencing educators' responsiveness is the time required to consolidate the actual integration of a new curriculum within schools. As Kagan emphasized [38], to achieve visible outcomes of a new curriculum, at least three years of school implementation are required.

For this reason, we decided to conduct the present study four years after the introduction of the new CS curriculum in the schools. The research period lasted 14 months, spanning late 2016 to early 2018. This research timeframe was expected to provide sufficient time for the necessary organizational, instructional, and pedagogical adjustments in the CS classrooms to have taken place in the schools. Therefore, we hypothesized that the

https://doi.org/10.3390/computers15030140

participating teachers have developed a more stable perception of the new CS curriculum and its enactment, since they were free from any barrier or challenge that they would inevitably have faced during the first years of implementation.

The Greek educational system incorporates a specialized type of schools, named model and experimental schools, designed to function as catalysts for systemic reform and academic excellence. Experimental schools operate as laboratories for the pilot application of new curricula, innovative pedagogies and teaching methodologies, instructional materials, etc. They also operate in close synergy with university educational departments to facilitate longitudinal educational research. Model schools, on the other hand, are designed to foster academic excellence and leadership. These schools target high-aptitude students through meritocratic selection, providing an intensive educational environment that promotes cognitive depth and scholastic distinction.

These schools are both characterized by decentralized governance, rigorous faculty evaluation, implementation of individualized educational modules and extracurricular activities, and a clear strategy for European orientation and internationalization (e.g., engagement in eTwinning programs, Erasmus+ mobility, etc.). By bridging the gap between university-led research and secondary education and aligning with international educational standards to cultivate high-level outcomes, they serve as the primary vehicle for educational reforms and the pilot for the standard public schools.

Therefore, the selection of model and experimental lower secondary schools was due to their nature of educational orientation, pedagogical autonomy, and the advanced academic credentials of the faculty that were expected to create a friendly and flexible educational environment for the enactment of the new CS curriculum.

3.3. Research Aim and Questions

The aim of the present study was to investigate the perceptions of CS teachers serving at model and experimental lower secondary schools regarding their understanding and adoption of the philosophy and the educational changes in the new CS curriculum, as well as their instructional choices and practices in the enactment of the CS curriculum. The research questions addressed were as follows:

- What are the key aspects of the new CS curriculum as they are perceived by teachers in model and experimental lower secondary schools?
- How have the participating teachers responded in terms of adoption and enactment of the new CS curriculum?
- How do the participating teachers evaluate the outcomes of implementing the new CS curriculum?

4. Research Method

4.1. Participants

Purposeful sampling is considered the most appropriate method to gather information-rich feedback from individuals [39]. Therefore, the selection criterion in this study was CS teachers serving in model or experimental lower secondary schools (Gymnasiums). Twelve schools from different districts of Greece were selected. Eventually, nineteen CS teachers (twelve female and seven male) participated in the study on a voluntary basis. Prior to the interviews, participants were briefed on the study's objectives and explicitly informed of their right to withdraw at any stage. To ensure confidentiality, all personal identifiers were anonymized using pseudonyms, and data was stored in compliance with privacy standards regarding both teachers and schools.

Table 1 presents demographic information concerning the participating teachers. With regard to their qualifications, eleven teachers have a bachelor's degree in computer science,

https://doi.org/10.3390/computers15030140

four in physics, and four in mathematics. In addition, twelve teachers hold a master's degree and five a doctoral degree as well.

Table 1. Demographic characteristics of participating teachers.

Teacher	Gender	Age	Studies	Teaching Experience (Years)	Curriculum Implementation (Years)	School Type
E1	A	58	PhD	29	6	Experimental
E2	A	38	Masters	13	4	Experimental
E3	Γ	57	Masters	30	4	Model
E4	Γ	51	Masters	24	5	Experimental
E5	A	41	PhD	14	3	Model
E6	A	50	Masters	20	6	Experimental
E7	Γ	42	PhD	17	3	Experimental
E8	Γ	53	PhD	25	6	Model
E9	Γ	40	Masters	18	5	Experimental
E10	A	47	PhD	17	4	Experimental
E11	Γ	46	PhD	22	4	Experimental
E12	Γ	55	PhD	26	6	Model
E13	Γ	51	Bachelor	25	6	Experimental
E14	Γ	48	PhD	24	5	Experimental
E15	Γ	43	Masters	17	2	Experimental
E16	A	40	PhD	15	4	Model
E17	Γ	54	Bachelor	27	6	Experimental
E18	Γ	41	Masters	15	6	Model
E19	A	49	Masters	12	3	Experimental

The majority of participants had long teaching experience, ranging from twelve to thirty years (M = 20.5 years). Seven teachers implemented the new curriculum for the full six-year period, while the rest did so for a time ranging between three and five years. Consequently, the selected participants were expected to provide important information on how the new CS Curriculum was perceived and implemented in these schools, the competencies achieved by the students, and the challenges teachers faced in translating the curriculum into effective instructional designs aimed at developing students' computational skills.

4.2. Data Collection and Analysis

To investigate the educators' perspectives, semi-structured interviews were selected as the primary method. The particular approach was deemed as the most appropriate for two reasons [39]: (a) the understudied nature of the topic both internationally and within the national context, and consequently, the scarce existing knowledge, called for more in-depth feedback from the interviewees, which semi-structured interviews can provide, and (b) the limited number of participating schools and educators allowed for the use of the method.

A specific interview protocol was developed and used as a tool for collecting qualitative data. The items were aligned with the key factors of curriculum enactment that appear in the literature, and the research questions of the present study as well. The mean time of the individual interviews was 45–50 min.

Thematic analysis of the data followed a constant inductive method [39], including preliminary exploratory analysis of the dataset to identify initial patterns, open coding,

https://doi.org/10.3390/computers15030140

category development through grouping coded data, refinement and prioritization of core themes and subthemes, and descriptive synthesis of findings. Before starting the coding phase and to achieve inter-code reliability, the researchers agreed on a common codebook including code name, code description, and inclusion/exclusion criteria. Following, they independently analyzed a sample of data coming from three teachers' interviews. The level of inter-rater agreement was very satisfactory (85%). The two researchers then discussed all discrepancies and agreed to redefine the corresponding codes.

After the initial exploratory review, the entire body of interview data was imported into NVivo for systematic analysis. Recurring patterns were coded and categorized, followed by iterative theme identification. Cross-referencing material from all interviews enabled the synthesis of overarching themes and subthemes, ultimately leading to the descriptive and interpretive presentation of the results. Interpretation was conducted within the theoretical and research framework, determining teachers' perceptions of curriculum enactment in school settings.

5. Results

Thematic analysis of the interview data revealed a range of factors related to participants' understanding and adoption of the curriculum and its application. These factors were organized into two thematic dimensions reflecting CS teachers' beliefs and practices: (a) Understanding of the CS curriculum, and (b) enactment of the CS curriculum. Six themes were identified, including various subcategories (subthemes), as presented in Table 2. The themes in Table 2 and the subthemes therein represent the CS curriculum elements that the participating teachers regarded as important, and, in their view, affect the acceptance and the implementation of the new curriculum in school practice.

Table 2. Themes and elements determining the enactment of CS curriculum.

Themes	Subthemes
Perceptions of the CS curriculum	Positive opinions/views about the curriculum The curriculum as a tool guiding teachers' instructional work The curriculum as a framework for instructional design
Key pedagogical features	Open learning philosophy Student-centred instruction Shift to learning outcomes Teacher's autonomy and creativity Enacting e-portfolio as a learning and assessment tool
Content of the CS curriculum	Suitable organization of CS thematic units Digital and computing skills Computational thinking, algorithmic thinking Constructive alignment (between learning outcomes, learning activities, and students' assessment)
Pedagogical strategies	Instruction based on learning activities (student-centred) Open and constructivist learning strategies Inquiry-based learning, collaborative learning Formative assessment as an integral component of students' learning
Educational resources	Programming tools Unplugged learning activities, educational robotics Open Educational Resources (OER) Learning Management System (eClass) Students' worksheets
Perceived outcomes	Instruction shifted to learning outcomes Achievement of the expected learning outcomes Collaboration and creativity skills Metacognition and learner autonomy skills

https://doi.org/10.3390/computers15030140

5.1. Understanding the CS Curriculum

5.1.1. Perceptions of the CS Curriculum

The results showed that the teachers in the sample presented a positive view of the new CS curriculum and recognized its critical features that bring about changes in their teaching. All participants focused on this theme, which was mentioned many times in the interviews with regard to its pedagogical features, the open instructional framework provided, the educational resources, and the assessment of students' learning outcomes. For example, we have the following:

> *E13: "I am very positive… Personally, I tried to assign integrative tasks to my students…"*

Other teachers reported that many colleagues are enthusiastic about the new CS curriculum and they have embraced it in their instruction:

> *E14: "…I know many computer science teachers, at other experimental schools, who clearly welcomed the new curriculum with great enthusiasm."*

> *E16: "…In general, computer science teachers like the new curriculum. I think they have adopted it."*

Conversely, other teachers, who were positive about the new CS curriculum, noted the difficulties inherent in any curricular change. For example, we have the following:

> *E1: "In general, there are reactions to every change in the schools. This is the challenge and depends on the individual teachers' perceptions."*

In addition, teacher E6 highlighted another factor of difficulty, i.e., the instructional freedom that the new CS curriculum provides to the educators. Teachers had to redesign their instruction by considering new elements in their planning and creating new types of learning activities:

> *E6: "We faced difficulties, especially during the first year of implementing the new curriculum; we were a bit lost. The freedom we suddenly had was overwhelmingly large at first, as we had to create our own teaching scenarios."*

Most participants reported that they have changed their ideas about the role of the curriculum, which they consider not as a bureaucratic document but as a tool for organizing their instruction. Others emphasized teachers' guidance and freedom, which are recognized as important features of the new CS curriculum. Examples of these perceptions appear in the following excerpts:

> *E1: "The curriculum is a tool … I came to understand and appreciate that it is a document of real value."*

> *E4: "The curriculum is a very important tool; while it guides teachers it gives freedom as well to adapt instruction into various educational contexts."*

> *E13: "It is not just a bureaucratic document; I consider it is extremely important…"*

Moreover, teachers perceived the new curriculum as providing a structured framework for long-term CS course design. It enables them to design their instruction and implement their own learning activities, instructional scenarios, and project-based tasks:

> *E8: "In the first year of implementing the new CS curriculum, I used many of the suggestions provided; afterwards, I started to create my own instructional scenarios."*

> *E7: "I place great emphasis on the curriculum. It's important because it shows the directions your teaching may take…"*

https://doi.org/10.3390/computers15030140

5.1.2. Key Pedagogical Features

Almost all teachers recognized the shift in the curriculum from instructional objectives to students' learning outcomes. They also agree with this change, since it places the students themselves and their expected achievements at the centre of the instructional design. This shift appears to be embraced by most educators, resulting in a reduction in traditional lecture-based teaching:

E1: "The shift from educational objectives to learning outcomes is appropriate."

E2: "Focusing on learning outcomes is absolutely right. . . The student, not the teacher, must be at the centre; this has changed the way we teach CS. . ."

Similarly, other teachers highlighted the open philosophy of the new CS curriculum and its innovative components, which enhance teachers' flexibility, autonomy, and creativity, while granting freedom to tailor their choices to the needs of their instruction and their classroom:

E12: "I consider openness and learner-centred orientation as the most innovative feature of the CS curriculum."

E11: "I find no weak points; the CS curriculum is structured clearly. The spiral approach of instruction is innovative and offers to both, teachers and students, freedom, autonomy, and creativity."

E10: "The CS curriculum gives flexibility for more creative and engaging lessons. . ."

Most teachers regard the adoption of new forms of assessing students' learning outcomes as both significant and innovative. Students' personal e-portfolio is considered by the participants as the most significant change, employed both as part of everyday teaching and as a tool for formative assessment of students' learning outcomes:

E2: "Using students' e-portfolio is very effective. . . Student's e-portfolio reveals her/his profile and achievements over the course of the year."

E6: "Using the e-portfolio is heading in the right direction. . . we must move away from a grade-collecting approach to authentic/constructive assessment to help students identify their weaknesses and encourage them to improve their artefacts."

Other teachers, like E14, emphasized the curriculum's recommendation that student assessment be considered an integral component of the educational process within the CS classroom:

E14: "Since students work in groups, assessment is based on the group deliverable. This allows me to evaluate both the manner and the extent of individual contribution to the collaborative task."

5.1.3. Computer Science Content

An open curriculum also concerns the CS content, since it is essential for facilitating teachers to overcome the inherent inertia of traditional frameworks and integrate new tools and emerging technologies in their instruction. Furthermore, openness of the CS curriculum transitions the pedagogical focus from transient digital proficiencies to the development of computational thinking, problem-solving, and cognitive flexibility skills through the utilization of dynamic and contextually relevant tools. The majority of participants think that the thematic units effectively address the requirements and the objectives of a contemporary CS course, and that CS content is organized in a clear, comprehensive, and open manner:

E7: "The CS curriculum covers the fundamental thematic units. . . The curriculum is dynamic and open with respect to current advances in computer science."

https://doi.org/10.3390/computers15030140

E5: "The open features of the CS curriculum allow me to include new topics in my instruction. For example, in Grade 9, I incorporated programming activities with Python. This is not clearly suggested by the CS curriculum; however, it is in line to the general scope of the CS curriculum and appeared as an engaging topic/learning activity for the students…"

The participating teachers identified, as one of the most significant features of the new CS curriculum, the emphasis placed on students' development of digital and computational competencies (knowledge, skills, and attitudes), which they consider essential for the students to actively participate in 21st century society:

E10: "Students are native citizens of the digital world… It's crucial to cultivate their attitudes toward handling information and information overload."

E9: "Digital literacy is essential; students also need to master the basic principles of computer programming."

Most teachers stressed the importance of fostering students' algorithmic and computational thinking:

E8: "Students need to understand the pervasiveness and impact of computer science in the modern world… I consider the development of algorithmic thinking/problem-solving to be vitally important."

E5: "There should be more time allocated for problem-solving… I believe computer programming should receive greater emphasis."

Consistency between the intended learning outcomes, the ways of students' learning engagement, and the assessment forms of the learning outcomes was recognized by most teachers as an important feature of the new CS curriculum:

E2: "The intended learning outcomes of the CS Curriculum are clearly aligned to students' work and assessment. The curriculum recommends open-ended computing projects rather than closed-form tests and quizzes."

The preceding comment can be interpreted through the lens of constructive alignment, a framework for curriculum and instructional design proposed by John Biggs [40]. Rooted in a constructivist view of learning, constructive alignment underscores the systematic coherence among intended learning outcomes, students' cognitive learning activities, and assessment tasks. Importantly, the majority of teachers in the sample identified this principle as a defining feature of the CS curriculum and applied it in their instructional decisions. The following excerpt from Teacher E4's interview explicitly connects alignment with the quality of CS teaching:

E4: "I feel that integrating assessment into my instructional practice made me a better teacher…"

5.2. Enactment of the CS Curriculum

The existing literature suggests that the enactment of a new curriculum brings about major reforms in education practice related to the learning objectives, the instructional strategies applied, and students' learning activities, as well as the teachers' role and work [11,12,14,15]. Thematic analysis of teachers' interview data revealed three main categories of factors regarding the implementation of the new CS curriculum in the schools, namely pedagogical strategies, educational resources, and learning outcomes.

https://doi.org/10.3390/computers15030140

5.2.1. Pedagogical Strategies

The majority of CS teachers in the model and experimental secondary schools under investigation reported implementation of learning activities and authentic problem-solving that foster students' engagement, autonomous learning, and self-regulation:

E6: "My teaching philosophy is to assign a program or a task and let the students to undertake initiatives and learn on their own…"

E8: "I ask my students to work both independently and autonomously…"

E12: "I use learning activities to change the way students learn; they place them in a process of learning how to learn."

In addition, most teachers recognized the pedagogical shifts advocated by the new CS curriculum and reported adoption of inquiry-based and constructivist approaches in their instructional practices. The following excerpts illustrate the implementation of student-centred learning strategies, including the use of cognitive-conflict situations, scaffolding techniques, and collaborative learning activities:

E5: "The philosophy of the new curriculum allows me to work in the computer lab, primarily by using inquiry-based learning activities."

E3: "It is important to use cognitive conflict situations, i.e., to help students realize that, by restricting to their existing knowledge, they cannot complete a learning task (e.g., developing a program) nor interpret it correctly."

E2: "I use scaffolding methods providing guidance and support to the students; then I am gradually fading… I use collaborative learning scenarios as well."

Almost all teachers mentioned that the curriculum's openness enables them to adapt their teaching to students' prior knowledge, skills, and interests by using open pedagogical strategies (e.g., inquiry-based and collaborative learning), and implement project-based tasks and educational robotics activities:

E6: "The open orientation of the curriculum offers many degrees of freedom and helped me to tailor my instruction to students' interests…"

E18: "I took advantage of the curriculum's open orientation … I employed robotics platforms (Lego and Arduino)."

In their attempts at curriculum implementation, most participants reported that they adopted the spiral model [41] to achieve students' reinforcement, particularly in the computer programming unit. Teachers E5 and E7 put emphasis on their adoption of the spiral model of instruction:

E5: "With regards to programming, the spiral model is very helpful when moving to a new programming language, since you can approach the same concepts from a new perspective…"

E7: "I have adopted the spiral approach of instruction. It is particularly effective, as it allows instruction to begin with a core CS concept and subsequently return through the gradual addition of new elements, thereby enriching CS content and supporting progressive learning…"

In addition, many teachers appear to implement scaffolded programming projects of increasing complexity to foster student engagement and autonomous learning, thus moving beyond the rigid instructional frameworks to which they had previously been accustomed. The following excerpts are characteristic of this:

E4: "Initially, I assign students guided programming activities; after completing several tasks, I encourage them to independently explore ways to extend their solutions."

https://doi.org/10.3390/computers15030140

E18: "I used to assign programming problems that can be developed incrementally by introducing new concepts. For example, a task might initially require only a conditional structure and subsequently be extended to incorporate loops, counters, accumulators, and, finally, lists."

On the other hand, teacher E18 expressed caution, arguing that such projects may have limited effectiveness in enhancing students' programming skills:

E18: "A project primarily reflects the final outcome of teamwork, even though not all students may have fully mastered the skills targeted by the specific project."

5.2.2. Educational Resources

Most teachers reported instructional practices that encompass collaborative learning activities and computational projects of increasing complexity. In their efforts to enact the new CS curriculum, they used various educational resources, like programming tools (i.e., Scratch, Kodu), CS Unplugged activities [42], and digital learning objects, as well as open-source software:

E4: "This school year I asked my students to create a Pac-Man style game using Scratch or Kodu…Thus, they harnessed their previous gaming experiences and had opportunities to develop abstraction and creative thinking skills."

E13: "I primarily use materials from CS Unplugged to help students conceptualize an algorithm. I try to simulate some of the basic algorithms."

Some teachers, like E1, had the opportunity to incorporate educational robotics in their instruction:

E1: "I had the opportunity to integrate educational robotics using three new kits we have at school. I organized six two-hour sessions in groups, and all third-grade students (K-9) had the chance to engage with educational robotics."

In addition, teacher E9 reported the utilization of relevant CS learning objects from Photodentro, the Greek National Learning Object Repository [43]:

E9: "There are important resources in Photodentro that can be used and integrated into CS teaching."

Other digital tools, reported by the teachers in the lower secondary schools of the present investigation, were Web 2.0 applications and the Learning Management System (i.e., Open eClass), approved by the Ministry of Education. For example, the following responses were recorded:

E6: "I systematically use Web 2.0 tools and computational projects."

E4: "I use the Open eClass (LMS) where I can upload educational and support materials. Students are asked to submit their artifacts to the LMS responding to their in-class assignments."

Other teachers, like E1, highlighted the design of suitable worksheets, which they use to guide students' learning activities, programming tasks, or individual/team assignments in the CS classroom:

E1: "Student worksheets are the primary means in my instruction."

5.2.3. Perceived Outcomes

Many teachers reported that their instruction was designed around the intended learning outcomes set by the new CS curriculum. They highlighted that the new curriculum offers enhanced opportunities for student-centred learning approaches, thus elevating the

https://doi.org/10.3390/computers15030140

quality of their instruction as well as students' achievements in terms of the intended digital skills and computing competencies. The following excerpts are indicative of many similar teachers' perceptions:

> *E2: "The new CS curriculum greatly supports students' development of critical skills, like problem-solving, metacognition, and self-regulation."*

> *E15: "I think that the new CS curriculum gives more freedom to the teachers to choose suitable approaches in order to achieve the intended learning outcomes; I find this very positive."*

> *E5: "(The new CS curriculum) represents a different philosophy and, I believe, it will contribute to improving educational quality, and to supporting and strengthening students' learning…"*

Almost all teachers believe that the new CS curriculum enhanced students' computational knowledge and skills, as well as their attitudes, values, and behaviour concerning digital culture and citizenship:

> *E9: "… I observed that students who started CS in Grade 7, under the framework proposed by the new curriculum, have already acquired many key components of computational thinking."*

> *E17: "Students achieved the expected learning outcomes to a large extent. It is notable how confidently the 9th grade students responded when they assigned a task using digital tools (e.g., creating a wiki, a timeline, a comic, etc.); they needed no further explanation or support to proceed."*

Moreover, many teachers feel that the new CS curriculum fosters students' development in terms of collaboration and creativity skills:

> *E4: "Students are working collaboratively …. having a common goal, to complete the assigned digital project. This was greatly reinforced by the curriculum, and I noticed significant improvement regarding their teamwork skills…"*

> *E1: "When using Web 2.0 tools, emphasis is set on their creative features… Collaboration is at the centre of students' work whether the Web 2.0 tool intrinsically supports it or not. The creative part of students' activity, i.e., creativity, is what I am interested in as a teacher".*

Other teachers also consider the implementation of the new CS curriculum in lower secondary schools to be successful, since students were able to develop a wide range of 21st century skills, as illustrated in the following excerpts:

> *E18: "Ultimately, students were able to mostly work autonomously. My main instructional goal is to teach them how to learn. So, they will continue evolving on their own…."*

Regarding the assessment of students' learning outcomes, the majority of teachers reported employing students' personal e-portfolio as a means of aligning achievements with the curriculum's intended learning outcomes. On the other side, a small group considered the combination of e-portfolio with written examinations as the best choice:

> *E10: "My teaching was adapted to the constructivist model. Emphasis was placed on assessing students' achievement of the anticipated learning outcomes."*

> *E17: "…Students' learning outcomes are assessed through their e-portfolio of digital artefacts. In the end, my students came very close to the intended learning outcomes set by the curriculum. I am very satisfied of their achievements!"*

https://doi.org/10.3390/computers15030140

6. Discussion and Conclusions

The central aim of the present study was to examine the relationship between teachers' understanding of the new CS curriculum, their instructional choices, and the challenges they encounter in enacting the curriculum in school practice. Analysis of the interview data indicated that CS teachers serving in model and experimental lower secondary schools largely identified and understood the principles and the underlying philosophy of the new curriculum. The findings of this investigation do not confirm previous studies reporting that CS teachers faced several barriers and difficulties in implementing the CS curriculum [27,32–34]. It was evident that the teachers in the sample not only incorporated the new CS curriculum into their instructional planning but also demonstrated a strong understanding of its core principles. Overall, they expressed positive perceptions and attitudes toward the CS curriculum, and they appeared to effectively implement it in practice.

Figure 1 depicts the core dimensions and the related factors revealed in the present analysis that represent the teachers' perceptions of enacting the new CS curriculum in the schools explored. These factors were organized along three core dimensions that determine the curriculum implementation in model and experimental lower secondary schools: (a) pedagogical strategies applied, (b) educational resources used, and (c) perceived outcomes.

Specifically, participants perceived the curriculum both as a guide for teachers' work and as a framework for instructional design. Another key feature they highlighted was the shift from a content-driven approach toward clearly articulated learning outcomes to be achieved by students. Moreover, teachers recognized the emphasis that the CS curriculum places on the development of students' digital competencies and computational thinking, as well as on cultivating essential life skills such as problem-solving, collaboration, and creativity.

With regard to the pedagogical strategies used in the enactment of the CS curriculum, student-centred learning approaches and constructivist, inquiry-based, and collaborative learning were reported by the teachers as routine instructional practices in model and experimental lower secondary schools, which is in line with the curriculum guidelines. The openness of the CS curriculum was perceived by the teachers as affording pedagogical flexibility and autonomy, thereby creating favourable conditions for effective implementation in classroom practice. This curriculum feature was acknowledged by participants as an enabling factor for incorporating new subject themes as well as emerging computing tools, in response to the continuous evolution of digital technologies. Notably, most teachers reported the adoption of formative- and self-assessment as an integral component of students' learning trajectories.

Consistent with findings from similar studies [26,27,35], the implementation of the CS curriculum in model and experimental lower secondary schools was characterized by the adoption of student-centred strategies (e.g., unplugged activities, contextualized tasks, collaborative learning, project-based assignments, etc.) and scaffolded programming tasks that connect programming to real-world problem solving and promote students' computational thinking. The participating teachers were positive about the integration of computing with other subject areas through transdisciplinary scenarios and projects. They further reported the effectiveness of appropriately designed learning activities, programming projects, and digital creation tasks in enabling students to achieve the intended learning outcomes specified in the new CS curriculum. In addition, aligning students' learning activities with alternative assessment tools resulted in limiting or avoiding traditional written examinations.

https://doi.org/10.3390/computers15030140

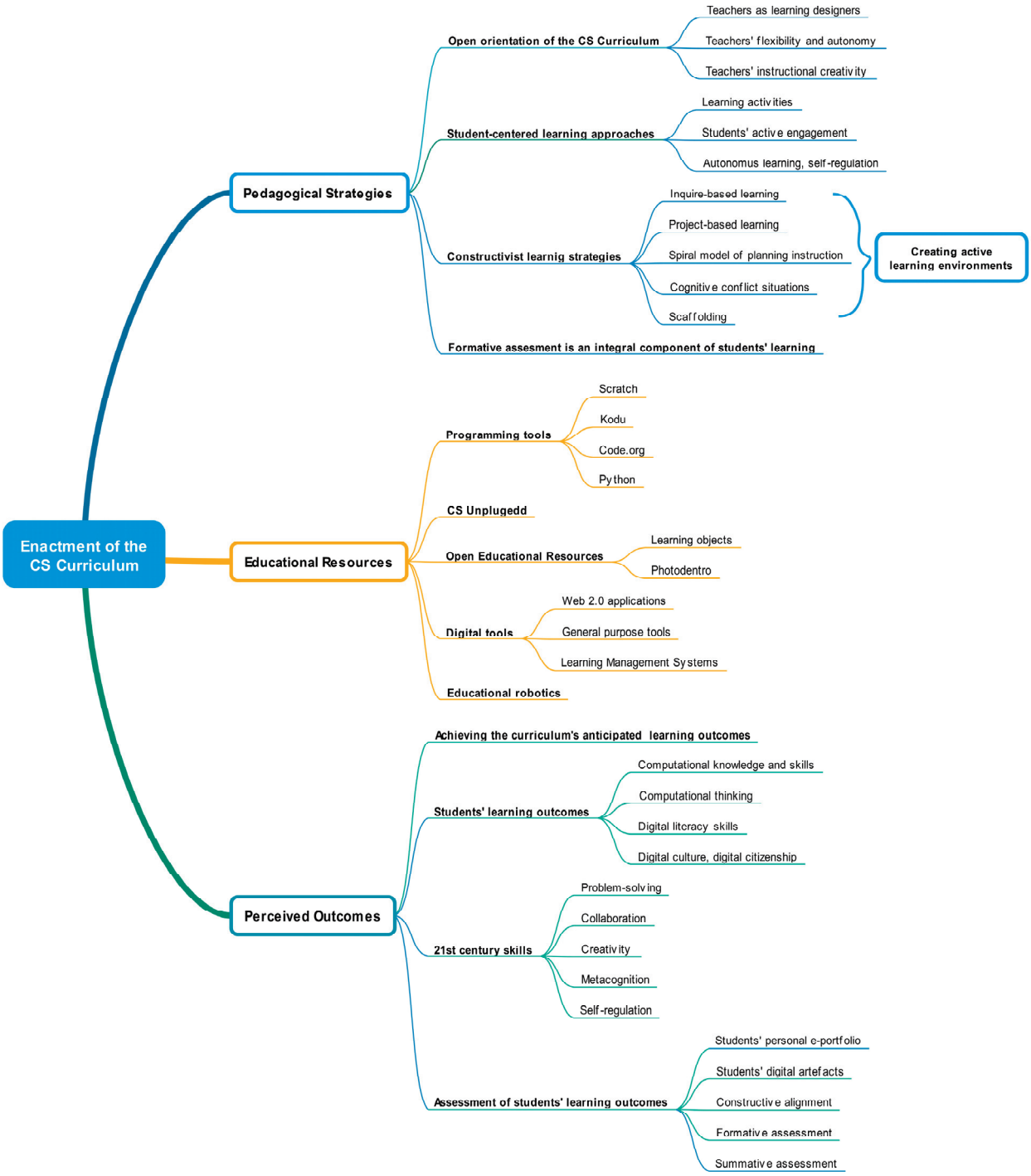


Figure 1. Key dimensions and factors of enacting the CS curriculum in model and experimental lower secondary schools.

Overall, the teachers in the sample appeared ready and well prepared to move from teacher-centred to student-centred pedagogical approaches and to embrace innovative educational practices in their instruction. Notably, however, these findings do not corroborate the existing literature reporting a substantial gap between the official and the enacted curriculum in other subject areas [11,19]. There are three main reasons that justify the strong alignment between intended and enacted curriculum in this study compared to the existing literature.

The first reason is related to the research context and the school culture built in the schools explored in this study. As an alternative to the standard public schools, model

https://doi.org/10.3390/computers15030140

and experimental schools provide a high-performance educational environment that is determined by an open philosophy of pedagogical experimentation, exploration, and integration of new instructional methods, adopts quality standards for students' learning and promotes educational change and innovation. Therefore, these schools offer a friendly environment that facilitates and encourages teachers to explore the critical changes required for the effective implementation of a new curriculum and then to apply new instructional approaches in practice.

The second most critical factor affecting successful curriculum implementation is the role of the educators as transformative agents in model and experimental lower secondary schools. Teachers' agency and effectiveness is rooted in three primary dimensions: (a) they possess high academic expertise and advanced qualifications (i.e., postgraduate and doctoral studies in CS and educational sciences); (b) most of them are actively engaged in continuous professional development, with some of them serving as trainers in professional development programmes for other educators; and (c) the highly motivated staff serving in model and experimental schools create dynamic school ecosystems that promote teachers' collaboration, experimentation, and pedagogical autonomy, thus reinforcing their agency for educational change.

Third, in the case of model and experimental lower secondary schools, the participating teachers had the opportunity to implement the new CS curriculum for three to five years. Undoubtedly, curricula are shaped through educational practice and evolve as teachers reflect on both their instructional experiences and students' learning outcomes. The findings confirm that the adoption of a new CS curriculum and the implementation of educational changes in the schools require sustained time and systematic effort on the part of CS teachers [38].

6.1. Recommendations for Practice

The distinctive features of model and experimental lower secondary schools (namely, student-centred educational orientation, enhanced pedagogical autonomy, and highly qualified faculty) appeared as the key factors for successful implementation of the new CS curriculum. However, they do not constitute inherently non-transferable educational conditions. Rather, they point to a set of school policies and professional practices that could be progressively adopted by standard lower secondary schools. In this sense, model and experimental schools should function not as exceptional cases, but as good practice models illustrating how systemic support, school educational culture, teacher agency, and sustained teacher professional development may facilitate the effective implementation of curriculum changes and innovations, including the new CS curriculum.

Teaching CS in Greek secondary schools entails additional challenges that may hinder the acceleration of curricular change. With regard to standard secondary schools, the present study can contribute to the design of targeted interventions aiming at accelerating the enactment of the new CS curriculum. Strengthening CS teacher professional development, promoting collaborative curriculum design, and integrating pedagogical openness and flexibility within existing frameworks could enable standard schools to similarly cultivate the supportive and student-centred learning environments required for the enactment of the new curriculum.

A recent study in the United States by Sadik and Ottenbreit-Leftwich [26] reported that a major challenge for secondary CS teachers is the need to further develop student-centred instructional strategies—such as problem-based learning and pair programming—as well as skills related to scaffolding and guiding students through effective team-management strategies. Therefore, CS teachers should further develop their pedagogical competencies, including lesson planning, selection of appropriate digital tools and computational

https://doi.org/10.3390/computers15030140

environments, design of appropriate learning activities and programming projects, and management of student work in computer lab settings, as well as assessment of students' learning outcomes and achievements.

Another issue concerning CS teachers' difficulties relates to the nature of CS content in secondary education, which requires continuous adaptation in response to rapid technological developments. In this regard, teacher professional development programmes should also address difficulties associated with emerging or specialized CS topics for which teachers may not have been adequately prepared during their initial university education.

It is important to acknowledge that CS teachers are the primary agents responsible for curriculum enactment as both tutors and learning designers. Consequently, teachers who are able to deeply understand the rationale for curriculum reform and identify the key features of curricular change may serve as catalysts supporting the enactment of a new curriculum in classroom settings. The findings of the present study highlighted the critical importance of designing appropriate professional development programmes and providing sustained support for CS teachers. At the same time, teachers should be encouraged to collaborate, particularly within professional development frameworks that support curriculum enactment. Teacher collaboration constitutes a core component of effective professional preparation, enabling educators to develop a deeper understanding of the objectives and the underlying philosophy of the CS curriculum and, ultimately, to enhance their capacity to effectively implement it in daily classroom practices.

Teachers can also benefit substantially from their participation in online learning communities, helping them to continuously address their professional development needs. Previous studies [25,44–46] have shown that secondary CS teachers perceive online communities as particularly effective since they provide enhanced opportunities to share effective practices with peers, to address needs related to CS pedagogy, curriculum, and content knowledge, and, more importantly, to mitigate professional isolation. To this end, effective online teacher communities provide sustained professional interaction, allowing educators to exchange pedagogical ideas and professional experiences, to share and co-design educational resources (e.g., instructional materials, lesson plans, learning activities, and programming projects), to experiment with alternative pedagogical practices, to discuss and reflect on students' learning challenges and achievements and, eventually to benefit both intellectually and emotionally [46,47].

6.2. Conclusions and Future Research

The implications of this study revealed a range of factors depicting the complex nature of curriculum enactment and the multifaceted roles that teachers must undertake for successful implementation to occur. Thus, the appropriateness of CS curriculum changes and the reality of lower secondary classrooms call for careful design and support of the implementation process.

A limitation of this study is that the results are based on interview data from CS teacher volunteers who were serving in model and experimental secondary schools. Therefore, caution should be taken in generalizing them in standard school settings. Undoubtedly, findings from different educational contexts are valuable towards deepening our understanding of teachers' challenges, pathways, and professional development needs while implementing curricular changes in practice. More importantly, new forms of challenges regarding CS instruction in secondary schools are currently expected to emerge due to the fast growth of artificial intelligence tools and platforms.

Our future research will be directed toward achieving a more comprehensive and rigorous picture of the alignment between the intended and the enacted CS curriculum in Greek lower secondary schools, exploring both teachers' perceptions and their instructional

https://doi.org/10.3390/computers15030140

choices as well as students' learning experiences and outcomes. Therefore, future research will seek to explore larger samples of teachers serving in standard lower secondary schools from various districts of the country. Using multiple data sources for triangulation, such as self-reports, interviews, questionnaires, and observations, we expect to identify the key factors that determine teachers' readiness for the implementation of the CS curriculum in schools.

Author Contributions: Conceptualization, D.Y. and A.J.; methodology, D.Y. and A.J.; validation, D.Y. and A.J.; formal analysis, D.Y. and A.J.; investigation, D.Y. and A.J.; resources, D.Y. and A.J.; data curation, D.Y. and A.J.; writing—original draft preparation, D.Y. and A.J.; writing—review and editing, A.J.; visualization, D.Y.; supervision, A.J.; project administration, A.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author due to privacy reasons.

Acknowledgments: The authors are grateful to the CS teachers who participated in the study presented in this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. ACM. *The CSTA K-12 Computer Science Standards*; ACM: New York, NY, USA; Computer Science Teachers Association: Chicago, IL, USA, 2017.
2. Astrachan, O.; Briggs, A. The CS principles project. *ACM Inroads* **2012**, *3*, 38–42. [CrossRef]
3. ISTE. ISTE Standards for Educators: Computational Thinking Competencies. 2018. Available online: https://www.iste.org/standards/iste-standards-for-computational-thinking (accessed on 10 August 2019).
4. Webb, M.; Davis, N.; Bell, T.; Katz, Y.J.; Reynolds, N.; Chambers, D.P.; Sysło, M.M. Computer science in K-12 school curricula of the 2lst century: Why, what and when? *Educ. Inf. Technol.* **2017**, *22*, 445–468. [CrossRef]
5. Armoni, M.; Gal-Ezer, J. Early computing education: Why? what? when? who? *ACM Inroads* **2014**, *5*, 54–59. [CrossRef]
6. Angeli, C.; Voogt, J.; Fluck, A.; Webb, M.; Cox, M.; Malyn-Smith, J.; Zagami, J. A K-6 Computational Thinking Curriculum Framework: Implications for Teacher Knowledge. *Educ. Technol. Soc.* **2016**, *19*, 47–57.
7. Fluck, A.; Webb, M.; Cox, M.; Angeli, C.; Malyn Smith, J.; Voogt, J.; Zagami, J. Arguing for computer science in the school curriculum. *Educ. Technol. Soc.* **2016**, *19*, 38–46.
8. Gal-Ezer, J.; Stephenson, C. Computer science teacher preparation is critical. *ACM Inroads* **2010**, *1*, 61–66. [CrossRef]
9. Sentance, S.; Csizmadia, A. Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Educ. Inf. Technol.* **2017**, *22*, 469–495. [CrossRef]
10. Voogt, J.; Fisser, P.; Good, J.; Mishra, P.; Yadav, A. Computational thinking in compulsory education: Towards an agenda for research and practice. *Educ. Inf. Technol.* **2015**, *20*, 715–728. [CrossRef]
11. Bergqvist, E.; Bergqvist, T. The role of the formal written curriculum in standards-based reform. *J. Curric. Stud.* **2017**, *49*, 149–168. [CrossRef]
12. Hughes, S.; Lewis, H. Tensions in current curriculum reform and the development of teachers' professional autonomy. *Curric. J.* **2020**, *31*, 290–302. [CrossRef]
13. da Silva, A.L.L.; Mølstad, C.E. Teacher autonomy and teacher agency: A comparative study in Brazilian and Norwegian lower secondary education. *Curric. J.* **2020**, *31*, 115–131. [CrossRef]
14. Sinnema, C.; Nieveen, N.; Priestley, M. Successful futures, successful curriculum: What can Wales learn from international curriculum reforms? *Curric. J.* **2020**, *31*, 181–201. [CrossRef]
15. Sullanmaa, J.; Pyhältö, K.; Pietarinen, J.; Soini, T. Relationships between change management, knowledge sharing, curriculum coherence and school impact in national curriculum reform: A longitudinal approach. *Int. J. Leadersh. Educ.* **2021**, *27*, 1395–1419. [CrossRef]
16. Tao, J.; Gao, X. Teacher agency and identity commitment in curricular reform. *Teach. Teach. Educ.* **2017**, *63*, 346–355. [CrossRef]
17. Park, M.; Sung, Y.-K. Teachers' perceptions of the recent curriculum reforms and their implementation: What can we learn from the case of Korean elementary teachers? *Asia Pac. J. Educ.* **2013**, *33*, 15–33. [CrossRef]

https://doi.org/10.3390/computers15030140

18. Boesen, J.; Helenius, O.; Bergqvist, E.; Bergqvist, T.; Lithner, J.; Palm, T.; Palmberg, B. Developing mathematical competence: From the intended to the enacted curriculum. *J. Math. Behav.* **2014**, *33*, 72–87. [CrossRef]
19. Hewitt, T.W. *Understanding and Shaping Curriculum: What We Teach and Why*; Sage: Thousand Oaks, CA, USA, 2006.
20. Remillard, J.T.; Heck, D. Conceptualizing the curriculum enactment process in mathematics education. *ZDM Math. Educ.* **2014**, *46*, 705–718. [CrossRef]
21. Ni Shuilleabhain, A.; Seery, A. Enacting curriculum reform through lesson study: A case study of mathematics teacher learning. *Prof. Dev. Educ.* **2018**, *44*, 222–236. [CrossRef]
22. van Driel, J.; Bulte, A.; Verloop, N. Using the curriculum emphasis concept to investigate teachers' curricular beliefs in the context of educational reform. *J. Curric. Stud.* **2008**, *40*, 107–122. [CrossRef]
23. Fischer, C.; Eisenkraft, A.; Fishman, B.; Hübner, N.; Lawrenz, F. Adapting to the large-scale advanced placement chemistry reform: An examination of teachers' challenges and instructional practices. *J. Chem. Educ.* **2018**, *95*, 1701–1710. [CrossRef]
24. Harris, R.; Graham, S. Engaging with curriculum reform: Insights from English history teachers' willingness to support curriculum change. *J. Curric. Stud.* **2018**, *51*, 43–61. [CrossRef]
25. Ormond, B.M. Curriculum decisions—The challenges of teacher autonomy over knowledge selection for history. *J. Curric. Stud.* **2016**, *49*, 599–619. [CrossRef]
26. Sadik, O.; Ottenbreit-Leftwich, A.T. Understanding US secondary computer science teachers' challenges and needs. *Comput. Sci. Educ.* **2024**, *34*, 252–284. [CrossRef]
27. Shin, S.; Cheon, J.; Shin, S. Teachers' perceptions of first-year implementation of computer science curriculum in middle school: How we can support CS initiatives. *Comput. Sch.* **2021**, *38*, 98–124. [CrossRef]
28. Yadav, A.; Gretter, S.; Hambrusch, S.; Sands, P. Expanding computer science education in schools: Understanding teacher experiences and challenges. *Comput. Sci. Educ.* **2016**, *26*, 235–254. [CrossRef]
29. Powell, C.; Anderson, R. Changing teachers' practice: Curriculum materials and science education reform in the USA. *Stud. Sci. Educ.* **2002**, *37*, 107–136. [CrossRef]
30. Pajares, M.F. Teachers' beliefs and educational research: Cleaning up a messy construct. *Rev. Educ. Res.* **1992**, *62*, 307–332. [CrossRef]
31. Grundén, E. The planned curriculum—Not just a matter of teachers. *Curric. J.* **2022**, *33*, 263–278. [CrossRef]
32. Thompson, D.; Bell, T.; Andreae, P.; Robins, A. The role of teachers in implementing curriculum changes. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*; ACM: Denver, CO, USA, 2013; pp. 245–250.
33. Falkner, K.; Sentance, S.; Vivian, R.; Barksdale, S.; Busuttil, L.; Cole, E.; Liebe, C.; Maiorana, F.; McGill, M.M.; Quille, K. An international comparison of k-12 computer science education intended and enacted curricula. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*; ACM: Koli, Finland, 2019; pp. 1–10.
34. Samarasekara, C.K.; Ott, C.; Robins, A. Teachers' views on the implementation of a new high school computing curriculum. In *Proceedings of the 22nd Koli Calling International Conference on Computing Education Research*; ACM: Koli, Finland, 2022; pp. 1–10.
35. Zhu, M.; Wang, C. K-12 computer science teaching strategies, challenges, and teachers' professional development opportunities and needs. *Comput. Sch.* **2024**, *41*, 1–22. [CrossRef]
36. P21. Partnership for 21st Century Learning. 2009. Available online: https://www.battelleforkids.org/insights/p21-resources (accessed on 18 January 2026).
37. Bradfield, K.Z.; Exley, B. Teachers' accounts of their curriculum use: External contextual influences during times of curriculum reform. *Curric. J.* **2020**, *31*, 757–774. [CrossRef]
38. Kagan, D.M. Implications for research on teacher belief. *Educ. Psychol.* **1992**, *27*, 65–90. [CrossRef]
39. Creswell, J.W. *Research Design: Qualitative, Quantitative and Mixed Methods Approaches*, 4th ed.; Sage: Thousand Oaks, CA, USA, 2014.
40. Biggs, J. Enhancing teaching through constructive alignment. *High. Educ.* **1996**, *32*, 347–364. [CrossRef]
41. Bruner, J.S. *The Process of Education*; Harvard University Press: Cambridge, MA, USA, 1960.
42. CS Unplugged. 2025. Available online: https://www.csunplugged.org (accessed on 18 January 2026).
43. Phtodentro. Greek National Learning Object Repository. 2025. Available online: https://photodentro.edu.gr/lor/?locale=en (accessed on 18 January 2026).
44. Qian, Y.; Hambrusch, S.; Yadav, A.; Gretter, S. Who needs what: Recommendations for designing effective online professional development for computer science teachers. *J. Res. Technol. Educ.* **2018**, *50*, 164–181. [CrossRef]
45. Tsiotakis, P.; Jimoyiannis, A. Critical factors towards analysing teachers' presence in on-line learning communities. *Internet High. Educ.* **2016**, *28*, 45–58. [CrossRef]

https://doi.org/10.3390/computers15030140

46. Ni, L.; Bausch, G.; Benjamin, R. Computer science teacher professional development and professional learning communities: A review of the research literature. *Comput. Sci. Educ.* **2021**, *33*, 29–60. [CrossRef]
47. Horn, I.S.; Aggül, Y.G.; Li, S.; Hood, C.E. Teachers' subjectivities in responsive instruction: When ambitious practice encounters challenging teaching situations. *Instr. Sci.* **2025**, *53*, 1889–1909. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

https://doi.org/10.3390/computers15030140

computers

MDPI

Article

A Preliminary Usability Study of a Novel Educational Training System to Teach ScratchJr. in School

María Jesús Manzanares [1], Diana Pérez Marín [2,*] and Celeste Pizarro [3]

[1] Department of Financial Economics and Accounting, Faculty of Economics and Business, Rey Juan Carlos University, Móstoles Campus, 28933 Madrid, Spain; maria.manzanares@urjc.es
[2] Department of Computer Science and Statistics, Higher Technical School of Computer Engineering, Rey Juan Carlos University, Móstoles Campus, 28933 Madrid, Spain
[3] Department of Applied Mathematics, Materials Science and Engineering and Electronic Technology, Higher School of Experimental Sciences and Technology, Rey Juan Carlos University, Móstoles Campus, 28933 Madrid, Spain; celeste.pizarro@urjc.es
* Correspondence: diana.perez@urjc.es

Abstract

Teaching programming to children at an early age has been proven to be beneficial. Some research has focused on how to teach programming to children with special needs. According to Human–Computer Interaction, all users should be involved in the design of their systems (including learning systems). However, evaluation procedures with young children are a complex task, which can be even harder when the young children have some special needs. This paper presents a preliminary usability study of a novel educational training system to teach ScratchJr. to young children. 48 neurotypical students between 6 and 7 years and 2 students with special needs in a pilot study were asked to use the system to find out whether they could complete an input/output activity and how it is related to their preferences.

Keywords: teaching programming; ScratchJr.; pre-primary education; neurotypical; neurodivergent

1. Introduction

Human–Computer Interaction [1] studies how to design systems so that they are usable and accessible for everyone, in accordance with the principles of Universal Design [2]. Among interactive systems, special attention is given to those aimed at teaching, and, in particular, at teaching programming [3].

According to Manches and Plowman [4], the teaching of programming at an early age should be a priority. This is because, in early childhood education, children develop the foundations of computational thinking [5], and logical thinking, which enable them to solve problems, design projects, and communicate ideas [6,7]

The resources commonly used for teaching programming in early childhood education include robots such as Cubetto [8,9], Bee-Bot [10], Tale-Bot [11], and KUBO [12].

At the software level, there is a simplified version of Scratch [13] called ScratchJr. [14] which includes fewer blocks that are mainly focused on movement, simple dialogue, and repetition—fundamental concepts typically introduced between the ages of 3 and 7.

Neurotypical children—that is, those who follow a developmental trajectory consistent with the timeframes studied in psychology—are generally able to learn ScratchJr. directly within the application, with guidance from their teachers [15].

However, in the case of children with a disability or learning disorder, the teaching of programming requires additional support and curricular adaptations, as is performed in other subjects, following the principles of Universal Design for Learning (UDL) [16].

In the literature, there are examples of ScratchJr. versions such as ScratchJr. Tactile, a non-digital, tangible version designed to address diversity [17]. There are also examples of tools developed to teach programming to students with autism [18,19], intellectual disabilities [20] or visual impairments [21].

On the other hand, it is worth highlighting the existence of interactive systems that rely on the use of a Pedagogical Conversational Agent (PCA)—that is, an internal or external representation within the system in the form of an animal, robot, or human [22]—which serves as a guide and support for teaching, taking on the role of teacher, student, or companion.

There are examples of agents that have been shown to improve both the performance and satisfaction of neurotypical students [23]. Improvements have even been reported in students' emotional capacity when learning programming with a PCA incorporating mindfulness [24].

In our previous work, teachers were consulted about the possibility of creating an interactive system to train children—both neurotypical and neurodivergent—in the use of ScratchJr., according to their preferences and needs [25].

The difficulty of identifying the needs of some non-verbal neurodivergent children or those with limited attention spans to complete a questionnaire independently was taken into account. This is especially relevant considering that, at these ages, executive functions are generally not yet fully developed, nor are reading and writing skills.

Therefore, a User-Centered Design process [1] was initiated to begin the development of a Pedagogical Conversational Agent (PCA), which the children decided to name Boby, in the role of student, to train in the use of ScratchJr., taking into account classroom diversity.

The first functional prototype of Boby was brought to the classroom during the 2024–2025 school year at a school in Madrid, Spain. It was tested by 48 neurotypical children, aged 6 to 7. A pilot study was conducted with 2 students with special needs to test whether they could also use Boby. The results of this experiment are presented in this article.

Additionally, this article also aims to address research questions such as whether children who are accustomed to using technology will find interaction with Boby more satisfying and taking into account their level of digital competence.

Another issue raised is the relationship between block-based games and interaction with Boby, considering that ScratchJr. is a block-based language and that not all children may enjoy playing with blocks.

The article is structured into five sections: Section 2 presents the related state of the art, Section 3 presents the materials and methods of the experiment, Section 4 presents the results, Section 5 presents the discussion of the results, and Section 6 ends the paper with the main conclusions and future lines of work.

2. State of the Art

2.1. Overview

Research on teaching programming in early childhood education is still in its early stages [26–28]. This section reviews the published work from both methodological perspectives and practical implementations. It is also important to note that, since the children are between 3 and 6 years old, they are generally not expected to create programs as in primary education. Instead, the focus is usually on coding simpler tasks [29], such as performing sequences of movements or planning routes. In this planning—moving from one point to another, for example, on a grid—basic concepts such as conditionals (going left or right) or loops (turning an object a certain number of times) can begin to be introduced.

https://doi.org/10.3390/computers15010017

Furthermore, when teaching coding to preschool-aged children, it is essential to use an appropriate pedagogical approach, as at this age children have only developed very basic reasoning, lacking the ability to abstract or make complex associations. However, they can already perform two tasks simultaneously, making it an ideal time to introduce them to computer science [30].

Bers [31] proposed, at a methodological level, the TangibleK program, which is a detailed curriculum for teachers who wish to teach coding in early childhood education. The theoretical foundations of this approach can be found in the framework of Positive Technological Development, aimed at developing personal and coding skills based on constructionism [32], and Positive Youth Development [33]. The program involves a minimum of 20 h of in-person work divided into six sessions dedicated to engineering design processes, robotics, flows, loops and parameters, sensors and loops, and sensors and branches.

Each session follows a similar format: (1) initial preparation; (2) activities; (3) individual or collaborative work on a small project; (4) technology circle; and (5) assessment. The MECUE methodology [34] is based on the TangibleK program adapted for an unplugged teaching approach without using robots, and on books such as Hello Ruby [35] in which a 5-year-old girl must solve various puzzles using coding skills—for example, dressing herself using the correct sequence of clothing. Other tasks may involve asking children to perform movements as if they were a robot executing the orders provided by the teacher in sequence, as if following a program.

At the practical tools level, ref. [36] distinguished two main approaches: software environments and robots. In the case of software environments, it is emphasized that interaction for children aged 3 to 6 should be primarily tactile, interacting with a digital projector in class or with tablets. One of the most widely used software environments is the ScratchJr. programming language [37]. This language is simplified compared to Scratch but follows the same philosophy, facilitating the transition to using Scratch when the child turns 6. The instructions are also pieces of a puzzle that form the program to move an object, which can also be a cat. However, unlike Scratch, ScratchJr. has fewer instruction blocks and is used on a tablet, as shown in Figure 1.

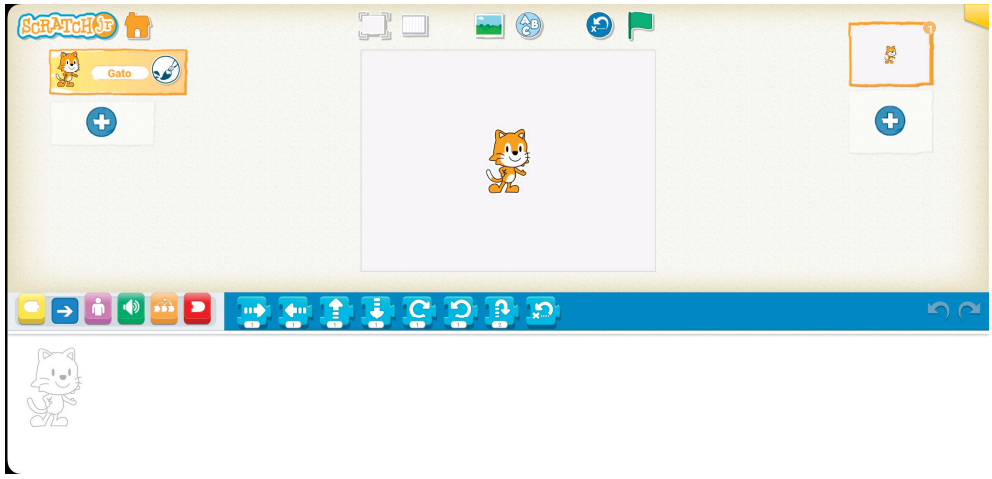


Figure 1. Example of the ScratchJr. environment (source: scratchjr.org).

Figure 2 shows an example of a program to say "Hello" in ScratchJr. on a tablet screen. It can be seen how the instruction with the text "Hello" is selected, and when executed, the cat says "Hello."

https://doi.org/10.3390/computers15010017

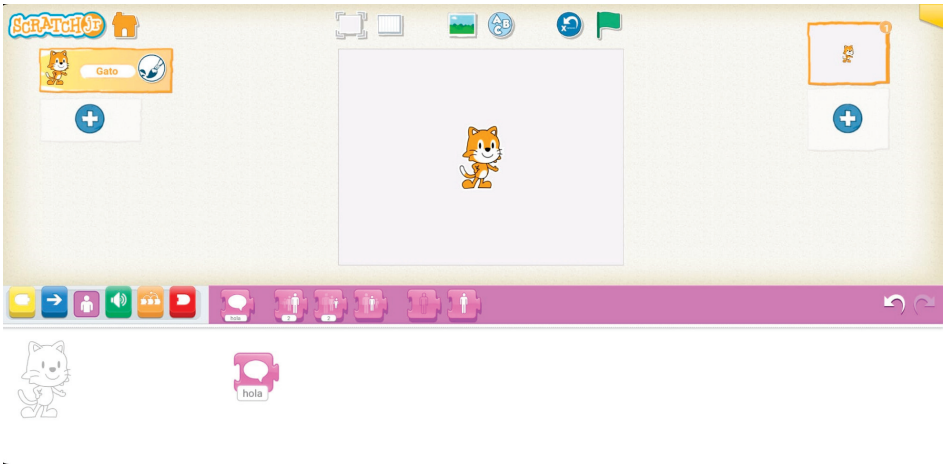


Figure 2. Example of a program in ScratchJr. (source: own elaboration).

In the case of robots, approaches without screens are distinguished, such as Cubetto [8,9], in which children also complete a puzzle—in this case, pieces to be placed on a wooden controller to move the robot on a grid mat forward, turn left, or turn right. It is also possible to combine this with a mixed approach, where one group of children uses Cubetto while other children perform the same actions themselves, following the robot's sequence [36].

Another group of robots is used with buttons, such as KIBO, Bee-Bot, Code & Go, and Code-a-pillar [38], which differ in their design, how children can explore concepts and interact with the robot, and the range of activities that can be performed with the robot. Figure 3 shows an example of the most representative button-based robots for developing programming skills in early childhood.

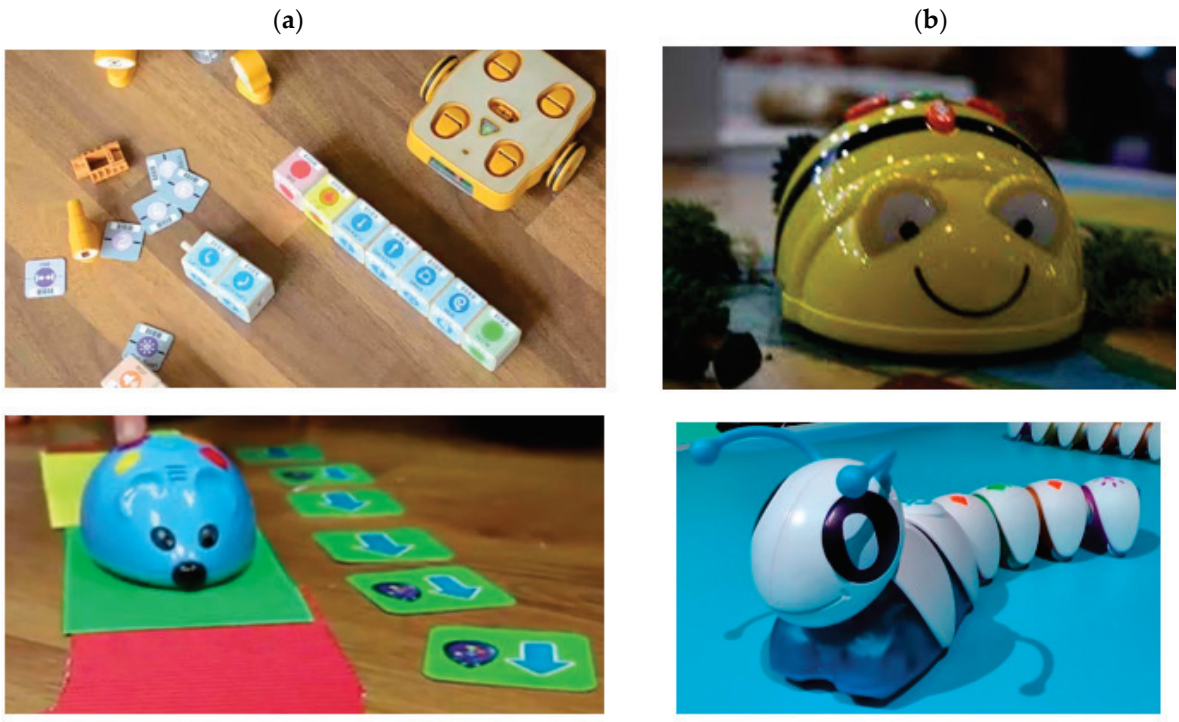


Figure 3. Example of robots: (**a**) KIBO, top left; (**b**) Bee-Bot, top right; (**c**) Code & Go, bottom left; (**d**) Code-a-pillar, bottom right (own elaboration).

It can be observed that, in all cases, the design is friendly, rounded, and features bright colours to attract children's attention. Additionally, three of the four robots are shaped like

https://doi.org/10.3390/computers15010017

animals, which are generally very appealing at these ages. Regarding materials, plastic is typically used to reduce costs. They can also be distinguished based on whether they use insertable code blocks, as in KIBO, or plastic tiles, as in Code & Go. In any case, the benefits of young children being able to immediately see the effects of their interactions with the robot have been validated in multiple studies [31,39–43].

2.2. Teaching Programming to Students with Special Needs

The teaching of programming, as indicated in the various studies found in the literature, is beneficial for children from an early age. However, research on the teaching of programming in Early Childhood Education is still in its early stages [26–28], This section reviews the existing literature from both a methodological perspective and that of practical implementations.

This section presents studies by several authors highlighting the benefits of the use of ICT and robotics for all pupils, and particularly for pupils with special educational needs.

1. One relevant study identified is the work by Lorena Lanzas Llorente entitled "Emotional comprehension work in children with ASD (autism spectrum disorder) through Educational Robotics: a proposed intervention approach", which focuses on the benefits provided by robotics, in this case through the use of a robot called AISOYl Kik-E, for children with ASD (autism spectrum disorder). The study centres on developing the understanding of basic emotions through the recognition of facial expressions. Through this work, the aim is to provide children with an innovative resource that promotes their emotional development and, consequently, supports their educational development.

In this study, the author takes into account the principles of meaningful learning by using the AISOYl Kik-E robot as a motivating element and a tool for enhancing pupils' skills, allowing learners to play an active role in the teaching–learning process [44–46].

Therefore, this study introduces the use of AISOYl Kik-E as a support or facilitator of learning rather than as the object of learning itself [44,47].

The conclusions drawn by the author from this study were as follows

(1) At a general level, the author observed the benefits of Educational Robotics for pupils with special educational needs; in this specific case, for pupils with ASD, as it facilitates learning and enhances interaction, among other aspects,
(2) Robots such as AISOYl Kik-E help children with ASD (autism spectrum disorder) to develop various socio-emotional skills by using visual resources adapted to their needs, which enable repetition, interaction, and emotional comprehension.

2. Other relevant studies identified include the work by [48], in which the authors, in their study on technological tools for inclusive education, highlight the possibilities offered by ICT in addressing classroom diversity. They argue that these tools can facilitate access to education and provide high-quality learning opportunities for pupils with different special educational needs present in mainstream classrooms.

In this study, the authors use various tools already available on the market, such as Araboard, Araword, Pictsonidos, and Pictogram, all integrated within the ARASAAC software version 2024–2028 (project 38-09), for the teaching of different disciplines, such as writing and associating images with their meanings, among others. These tools are particularly aimed at pupils with special educational needs.

The conclusions drawn from this study indicate that ICT represents an approach with significant potential, especially for individuals with motor disabilities, ASD (autism spectrum disorder), and ADHD (Attention-Deficit/Hyperactivity Disorder) within educational contexts.

https://doi.org/10.3390/computers15010017

Studies such as that by [49] reinforce the idea that the use of ICT with all pupils, and particularly with pupils with special educational needs, provides a controllable environment and learning situation. This, in turn, increases learners' motivation and reinforcement, as it promotes attention and reduces frustration in response to errors, enables autonomous work, fosters the development of self-regulation skills, and adapts to the individual characteristics of pupils with special educational needs, thereby facilitating their inclusion.

3. Studies such as that by [50], in their work entitled "Software tools for inclusive education in the early childhood education stage", also focus on the various existing computer-based tools applicable in early educational settings, paying particular attention to how these tools may influence development and learning, especially among pupils with special educational needs. The study conducted by these authors was qualitative in nature, allowing for a more in-depth exploration of existing software tools used in inclusive education, the identification of best practices, and an examination of how their use influenced children with special educational needs.
   The results obtained were as follows:
   1. The authors observed a growing trend in the use of these tools in inclusive education at early educational stages and were also able to verify, through studies conducted by [51], that not all software tools are equally effective for pupils with special educational needs.
      1.1 The existing software tools used with pupils with special educational needs and analysed in the authors' study include:
      1.2 Tinker cad, used to teach programming to secondary school students.
      1.3 Various educational mobile applications employed in different learning processes for pupils in Early Childhood and Primary Education.
   2. Several interactive programmes aimed at the development of logical and motor skills, used with preschool and primary school pupils.
   3. Furthermore, through this qualitative study, the authors were able to confirm that 67% of educational institutions have incorporated at least one software tool designed for inclusive education into their early childhood education programmes.
   4. Finally, at least 72% of pupils with special educational needs showed a notable improvement in learning during their educational process, underscoring the relevance of these tools within the field of inclusive education.

4. Another study identified is "The influence of motivation and cooperation among primary school pupils through educational robotics: a case study" by [52]. This study seeks to explore the relationship between Educational Robotics and the factors influencing pupils' motivation during the implementation of a robotics-based project, with the aim of determining whether Educational Robotics constitutes part of an educational change in the classroom from both a methodological and a procedural perspective.
   The study is focused on an early educational level (primary education) and is addressed to all pupils, both neurotypical and those with special educational needs; however, it does not place particular emphasis on pupils with NEE.
   This research shows how studies by authors such as [53] identify robotics as one of the most important resources within educational technology, as it can provide a constructive learning environment in the classroom.
   The conclusions reached by the author [52], in this study were as follows:
   1. Educational robotics promotes increased motivation and interest among pupils in the classroom, leading to more meaningful outcomes in the teaching–learning process.

https://doi.org/10.3390/computers15010017

2. A cooperative working methodology generates both academic and social benefits for pupils.
3. Additionally, drawing on the work of [54] in his book, the author argues that in order to achieve an effective technological learning environment, it is necessary to take into account all the emotions that may influence pupils. She maintains that for learning to occur, it is essential to generate an emotional response in learners; when this emotion is positive, learning becomes more meaningful, as motivation plays a central role.

5. A brief summary of other studies identified in the literature is presented below:
    1. Floor Robots (Bee-Bot)
    The Bee-Bot robot is part of the new generation of educational robotics designed to introduce young children to programming, specifically pupils in Early Childhood Education and the first cycle of Primary Education. Several studies have employed Bee-Bot, a programmable bee-shaped robot specifically designed for children aged 4–7 years.
        - ✓ According to Angeli and Valanides [55]:
            - ○ The authors examine how working with Bee-Bot influences sequential thinking skills (sequencing, route planning, and debugging) in pupils in the early years of primary education.
            - ○ They conclude that, with appropriate scaffolding, children improve in problem decomposition, step anticipation, and logical reasoning.
        - ✓ According to Di Lieto et al. [56]:
            - ○ An Educational Robotics Lab programme using Bee-Bot and Pro-Bot was implemented with pupils with special educational needs (neurodevelopmental conditions and learning difficulties) integrated within mainstream classrooms.
            - ○ The benefits observed included:
                - Improvements in executive functions (planning, working memory, and inhibitory control).
                - Increased participation and better behavioural regulation during classroom tasks.
                - Inclusive activities in which both pupils with special educational needs and their peers without special educational needs participated, thereby strengthening social interactions.
        - ✓ Review of educational robotics and neurodevelopmental disorders [57]:
            - ○ A critical review of studies involving educational robots with children with ASD, ADHD, intellectual disability, among others.
            - ○ The review includes experiences with Bee-Bot-type robots at Early Childhood and Primary Education levels.
            - ○ The main conclusions indicate that:
                - Robots increase motivation and participation.
                - They can support both academic skills (pre-programming and logical–spatial concepts) and social skills (pair work, turn-taking, and communication).
    2. Kit-Based Robots (KIBO) for Children Aged 4–7 Years
    KIBO is a robot explicitly designed for children aged 4–7 years and is programmed using physical wooden blocks, making it a screen-free system.

https://doi.org/10.3390/computers15010017

- ✓ According to Elkin, Sullivan, and Bers [58]: "Programming with the KIBO Robotics Kit in Preschool Classrooms"
  - ○ The study involved work in early childhood classrooms using KIBO.
  - ○ Results:
    - Children aged 3–5 years were able to create simple programmes, sequence instructions, and debug them.
    - KIBO was naturally integrated with activities such as art, music, and storytelling, which supported the development of language skills and creativity.
- ✓ KIBO as a tool for Computational Thinking and STEAM
  - ○ Documentation and studies from DevTech/KinderLab show that KIBO enables children aged 4–7 years to develop computational thinking, problem-solving skills, and STEAM exploration without the need for screens.
  - ○ Although many studies involving KIBO do not focus specifically on pupils with special educational needs, they do report:
    - High accessibility due to its tangible and manipulable nature.
    - The possibility of adaptation for pupils with attentional, motor, or communication difficulties, owing to the physical, collaborative, and multimodal nature of the activities (sound, movement, and construction).

3. Social Robots and Programming with Children with ASD/Special Educational Needs
   This line of research combines social robotics with the development of social competences, while also incorporating elements of simple programming or "co-programming".
   - ✓ According to Gkiolnta et al. [59]: "Robot programming for a child with ASD"
     - ○ The study presents a case of robot programming involving a child with ASD.
     - ○ The child participates in guided programming activities, with observed improvements in sustained attention, interaction, and enjoyment of the task.

Although many of these studies focus more on social skills than on "pure" programming, in practice robot programming activities (such as deciding sequences and testing them) are used as a vehicle for developing attention, cognitive flexibility, and cooperation.

3. Materials and Methods

3.1. Context

During the 2024/2025 school year, a school in the Community of Madrid, Spain, was asked for permission to attend a validation session of a new interactive system being developed as a future training agent in ScratchJr. This was performed to assess whether, at that stage of development, students were able to interact with the system and to show their level of satisfaction.

Figure 4 shows an example of a system screen, where it can be seen that the Scratch-Jr. blocks are displayed on the left, the question appears at the top, and students must drag the blocks to the center to check whether they have completed the exercise correctly or if they need to try again.

https://doi.org/10.3390/computers15010017



Figure 4. Example of the initial screen of the agent Boby (source: own elaboration).

As can be seen in Figure 4, the initial question is "I have to say Hello! How can I do it?". The idea is that the child helps Boby by dragging the ScratchJr. blocks so that "Hello" is said in the correct position. Initially, no support is provided to see if neurotypical or neurodivergent students can complete the task independently. The option to receive help is available, including interacting with the agent by speaking (microphone) if a block cannot be dragged, and receiving all information in audio for students with visual impairments. For students with autism, the idea of sequence is emphasized using the numbers 1 and 2, and guided information about the task is provided in advance. For students with ADHD, they are allowed to take a pause and receive recommendations, such as going for a walk or using a Bobath ball, to better focus on the system's instructions.

The novel approach of the tool should also be highlighted: it does not aim to teach the student and does not assume the role of a teacher. On the contrary, it takes on the role of a student who needs help, in order to increase motivation and satisfaction for learners who are not only learning for themselves but also helping the agent Boby, following the Learning by Teaching methodology used by agents such as Betty [60].

Figure 5 shows the technical diagram of how the agent works.

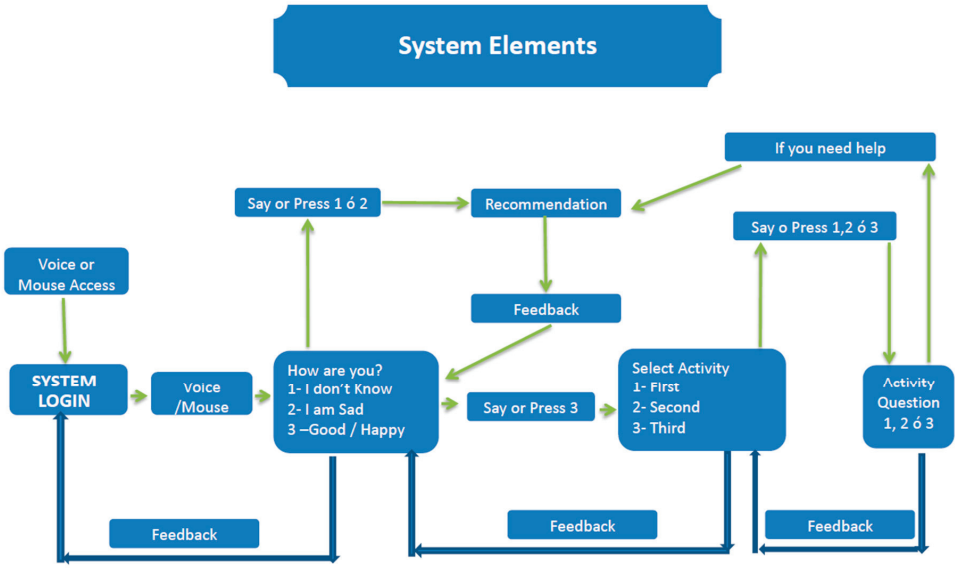


Figure 5. Technical diagram of Boby.

As can be observed, the student accesses the system directly either through voice input or via mouse interaction, without requiring any form of authentication. On the initial screen, or main interface, the system asks the student "How are you?", and depending on the response provided—either verbally or through mouse interaction—the system behaves as follows:

https://doi.org/10.3390/computers15010017

(a) If the student selects options one or two, a screen presenting recommendations on the actions to be performed is displayed. These recommendations are provided both in written form and as audio output.

   a. Once the student has completed the recommended actions, they return to the initial or main screen of the system, either by voice command or via mouse interaction.

(b) If the student selects option three, a screen displaying the number of activities proposed by the system is shown.

   a. When the student indicates—either verbally or through mouse interaction—the activity they wish to perform, the corresponding activity screen is displayed. Instructions for completing the activity are provided both in written form and through audio.

      i. If the child is able to complete the activity independently, they can automatically return—by voice command or mouse interaction—to the main screen containing the set of questions in order to select another activity.

      ii. If assistance is required, the system redirects the student to the recommendations screen. Once the student has completed the recommended actions, they return to the initial or main screen of the system, either verbally or through mouse interaction.

From this main screen, the student may choose to continue working with the system or to exit it, indicating their decision either by voice or via mouse interaction.

A sample dialog of how Boby and the student interact is shown in Figure 6.

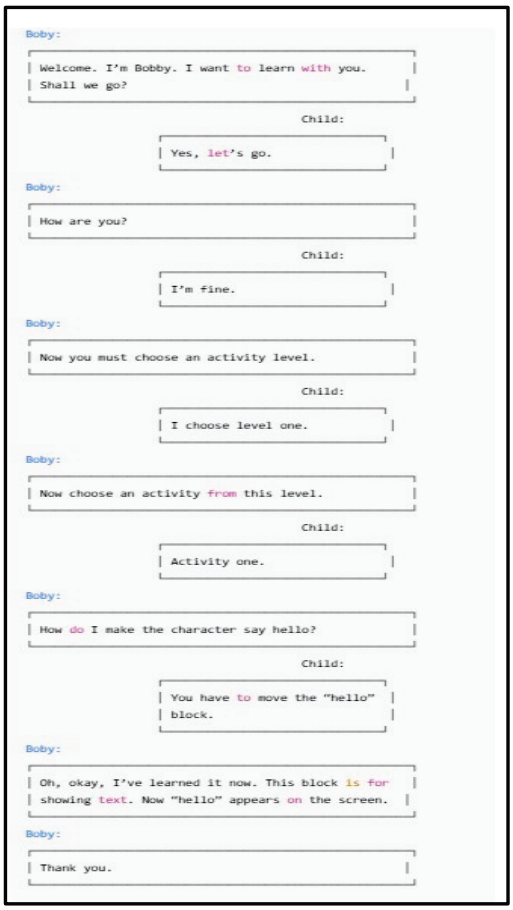


Figure 6. Sample dialog between Boby and a student.

https://doi.org/10.3390/computers15010017

3.2. Sample

Fifty children aged 6–7 years were recruited, of whom 17 (34%) are boys and the remaining 33 (66%) are girls. Two of the children are neurodivergent and have special educational needs. They are distributed across two classes with 25 students each, and in each class, 24 are neurotypical and 1 is neurodivergent with educational support needs.

None of the students have previously learned programming or are familiar with ScratchJr. Forty-three of the 50 students (86%) are able to complete a simple questionnaire and have basic reading skills.

3.3. Research Questions

The main objective of this study is to validate if students are able to use the Boby system to train in the use of ScratchJr., and to analyse which factors influence user satisfaction with the tool.

Research Question 1 (RQ1) is: which factors influence students' satisfaction with Boby? The associated hypotheses are:

H1. *Students who are accustomed to using tablets for learning will find using the Boby system more satisfying.*

H2. *Students who enjoy block-based games will find using the Boby system more satisfying.*

Research Question 2 (RQ2) is: to what extent does the student's reading ability influence their ability to use Boby?

Research Question 3 (RQ3) is: which factors influence students' ability to complete the activity shown in Figure 4, that is, teaching Boby to say "Hello".

3.4. Instrument

To address the research questions, a questionnaire with nine multiple-choice questions was designed and validated by a Special Education teacher, as described in our previous work [25].

The questionnaire is included in Appendix A (in Spanish).

3.5. Procedure

During a session in the computer classroom, students were arranged so that everyone had access to a tablet connected to the Internet to interact with the Boby system (see Figure 4). The classroom tutor and a support teacher for students with special educational needs were present. Additionally, the first author of the article was allowed access to the classroom as an observing researcher.

Initially, all students were asked to individually complete a paper questionnaire validated by the Special Education teacher, as described in our previous work [25]. All neurotypical students were able to complete the questionnaire, as were the two students with special educational needs, with partial support of the teacher.

Subsequently, all students were allowed to interact with the Boby system freely for 30 min. It was observed that all neurotypical students were able to interact with the system without difficulty, and the two neurodivergent students with special educational needs were also able to interact with the system with the support of their teacher.

Finally, all students and their teachers were thanked for their participation in the experiment. The tablets were collected (an email with information about the completed exercise was sent to the researchers), and all data on the classroom tablets were deleted.

https://doi.org/10.3390/computers15010017

4. Results

In this section, the quantitative results of the study are presented. The main outcome variable, Y26 ("Did you like the app?"), is used as an indicator of students' satisfaction with Boby. Three research questions are then addressed: RQ1 examines which background factors (such as usual tablet use and preference for block-based games) are associated with students' satisfaction; RQ2 explores to what extent students' reading ability (Q7) is related to satisfaction; and RQ3 focuses on students' ability to complete the "Hello" activity using ScratchJr. Given the small sample size, the presence of missing data, and the strong imbalance in satisfaction responses, the analyses are based exclusively on descriptive statistics (frequencies and percentages), supported by tables and bar charts in the following subsections.

4.1. Descriptive Overview of the Participants and Variables

Fifty children aged 6–7 years participated in the study. As described in Section 3.2, 17 (34%) were boys and 33 (66%) were girls, and two of the children had identified special educational needs and were considered neurodivergent. None of the students had previous experience with ScratchJr., and 43 out of 50 (86%) were able to complete the paper questionnaire with at least basic reading skills. The distribution of the main sociodemographic and background variables is summarised in Table 1.

Table 1. Sample characteristics and data availability.

Variable	Category	%
Sex	Boys	35.4
	Girls	56.3
Age	6 years	56.3
	7 years	37.5
Class	Class A	46.7
	Class B	53.3
Neurodivergent status	Neurotypical	97.8
	Neurodivergent	2.2
Reading ability (Q7)	0 = reads one word	2.2
	1 = reads some words	15.6
	2 = reads all words	82.2
	3 = advanced reader (if used)	0
Tablet use (Q10)	Mainly for games	66.7
	Mainly for learning	24.4
	Other/mixed	8.9
Likes block games (Q14)	Yes	82.2
	No	8.9
	Other/mixed	8.9
Questionnaire completed	Yes (valid responses)	93.75
	No/missing	0.06

The main outcome variable used in the analyses was item Y26 ("Did you like the app?"), with response options "Yes"/"No". The explanatory variables considered were: (a) Q10, usual use of the tablet (e.g., mainly for games or mainly for learning); (b) Q14, whether the child likes block-based or piece-based games ("Yes"/"No"); and (c) Q7, reading

https://doi.org/10.3390/computers15010017

ability on an ordinal scale from 0 to 3, where higher values indicate greater reading proficiency. In addition, information was collected on whether the questionnaire was completed, which allows the amount of valid data available for the analyses to be estimated.

Given the age of the participants and the classroom context, some items have missing data. In addition, the distribution of the main outcome is strongly skewed: most children reported that they liked the app, with only a small minority selecting "No". This imbalance, together with the small number of children in some categories (for instance, those who do not like block games, or those at the lowest reading level), makes complex modelling approaches difficult to interpret. For this reason, the results are reported using simple, robust descriptive statistics (frequencies and percentages), complemented by bar charts. Percentages are based on valid responses for each variable. Small deviations from the nominal sample size (N = 50) reflect missing data for some items.

4.2. RQ1—Factors Associated with Satisfaction with Boby

Research Question 1 (RQ1) asks which factors are associated with students' satisfaction with Boby, operationalised as answering "Yes" to Y26 ("Did you like the app?"). Among the children with valid data for Y26 and the background variables, 34 out of 40 students indicated that they liked the app, which corresponds to 85% of the usable sample. Descriptively, no subgroup shows a pattern of general dissatisfaction with the system.

In this subsection, satisfaction is described by usual tablet use (Q10) and by preference for block-based games (Q14). The corresponding frequency distributions are summarized in Tables 2 and 3, and visualized in Figures 7 and 8, respectively.

Table 2. Satisfaction with the app (Y26) by usual use of the tablet (Q10): frequencies.

Tablet Use (Q10)	n "Yes, I Liked the App"	n "No"
Mainly for games	23	4
Mainly for learning	7	2
Other/mixed	4	0
Total (valid cases)	34	6

Table 3. Satisfaction with the app (Y26) by preference for block-based games (Q14).

Likes Block Games (Q14)	n "Yes, I Liked the App"	n "No"
Yes	27	5
No	4	0
NA	3	1

4.2.1. Usual Use of the Tablet (Q10)

Table 2 summarizes the distribution of "Yes"/"No" responses to Y26 across the categories of Q10 (usual use of the tablet). For each category, the table reports the number of children who said that they liked the app and the number of those who did not.

As shown in Table 2, satisfaction is high in all three groups. Among students who reported using the tablet mainly for games, 23 out of 27 indicated that they liked the app. Among those who reported using it mainly for learning, 7 out of 9 liked the app. Finally, all 4 students in the "other/mixed" category reported that they liked the app. Overall, 34 out of 40 students with valid data for Q10 and Y26 reported that they liked the app.

Although the proportion of "Yes" responses is slightly higher in the "other/mixed" category, this group is very small and the difference should therefore be interpreted with caution. There is no clear pattern indicating that using the tablet mainly for games or mainly for learning is associated with a markedly different level of satisfaction. Figure 7 presents

https://doi.org/10.3390/computers15010017

these relationships graphically (together with other background variables), highlighting that the bar corresponding to "Yes" is consistently dominant in all Q10 categories.

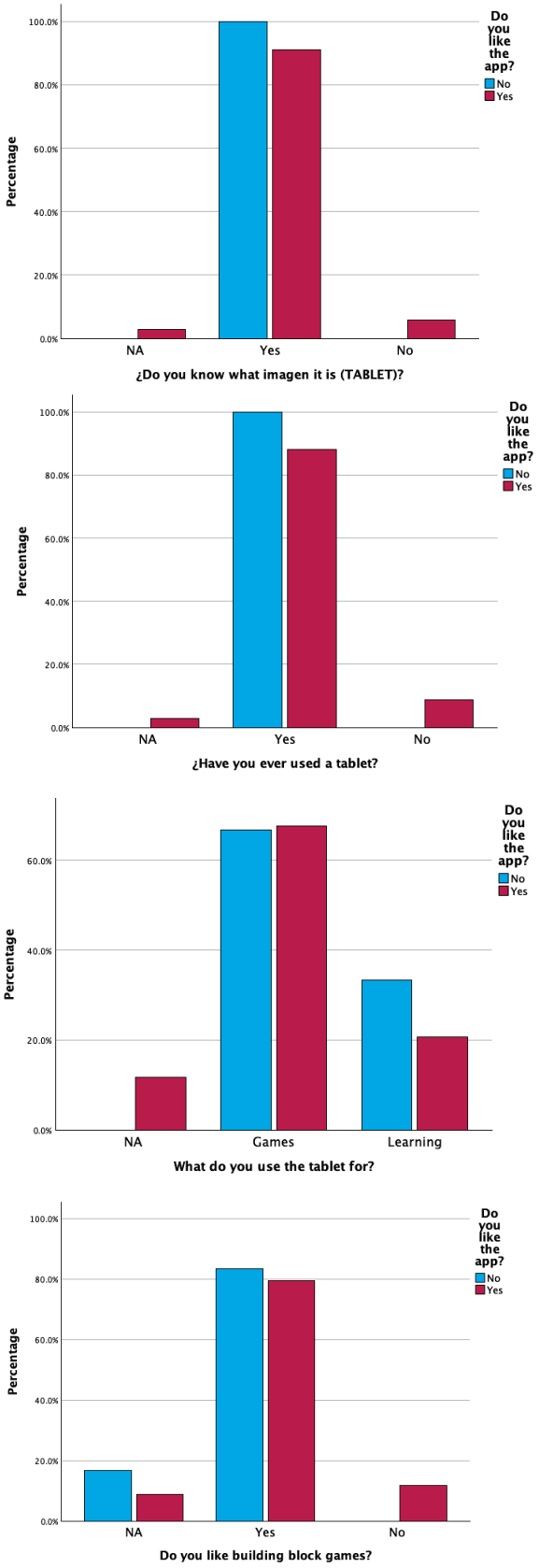


Figure 7. Proportion of "Yes, I liked the app" (Y26) across subgroups of tablet use and previous experience with technology (panels for Q10, Q9, Q8 and Q14).

https://doi.org/10.3390/computers15010017

4.2.2. Preference for Block-Based Games (Q14)

To explore Hypothesis H2 ("Students who enjoy block-based games will find using the Boby system more satisfying"), Table 3 summarises Y26 by Q14. Among children who said that they like block or piece-based games, 27 reported that they liked the app and 5 reported that they did not. In the small group of children who stated that they do not like block games, all 4 reported that they liked the app. For students with missing values or "not applicable" responses on Q14 (labelled as NA), 3 liked the app and 1 did not.

Thus, satisfaction with Boby is high in all Q14 categories, including among those who reported not liking block games. The small size of some subgroups—particularly the "No" and NA categories—means that the observed differences should be interpreted with caution. Figure 8 displays the same information as a bar chart, showing that the percentage of children who liked the app is consistently high and that disliking block-based games does not appear to prevent students from enjoying their interaction with Boby.

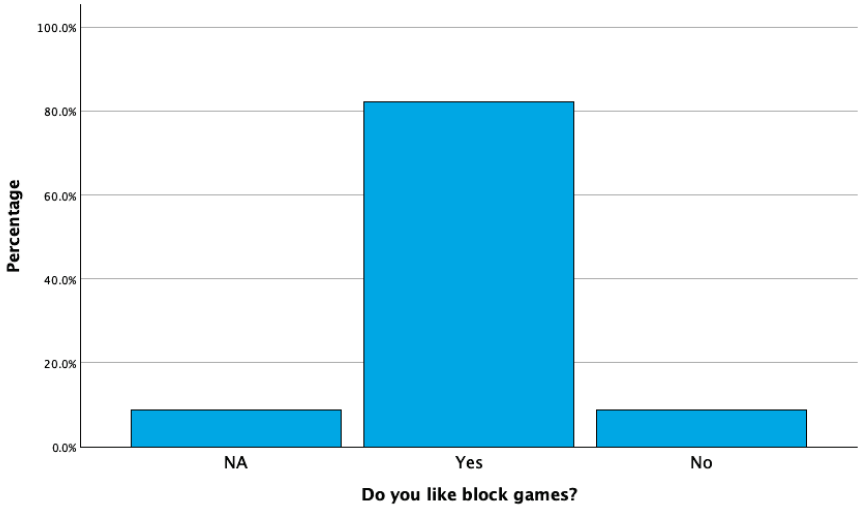


Figure 8. Percentage of children who liked the app (Y26 = "Yes") according to preference for block-based games (Q14).

Taken together, the descriptive analyses for RQ1 suggest that no clear subgroup of children is dissatisfied with the system based on their usual tablet use or their liking of block-based games. Satisfaction scores remain high and relatively homogeneous across the different categories considered.

4.3. RQ2—Reading Ability and Satisfaction with Boby

Research Question 2 (RQ2) examines to what extent students' reading ability (Q7) is related to whether they report liking the app (Y26). For this analysis, data were available for 40 children with valid scores in both variables.

Table 4 shows, for each level of Q7, the total number of children, how many said that they liked the app, and how many did not. The pattern is very stable:

- At the lowest reading level (Q7 = 0, "reads one word"), the only child in this category reported that they liked the app.
- Among those who "read some words" (Q7 = 1), 6 out of 7 children liked the app and 1 did not.
- Among those who "read all words" (Q7 = 2), 27 out of 32 children liked the app and 5 did not.
- No students were classified as "advanced readers" (Q7 = 3) in the current dataset.

https://doi.org/10.3390/computers15010017

Table 4. Satisfaction with the app (Y26) by reading ability level (Q7): frequencies.

Reading Ability (Q7)	Total n in Subgroup	n "Yes, I Liked the App"	n "No"
0 = reads one word	1	1	0
1 = reads some words	7	6	1
2 = reads all words	32	27	5
3 = advanced reader (if applicable)	0	0	0

Overall, 34 out of 40 children with reading data (85%) reported that they liked the app. Figure 9 visualises these percentages, showing that the bars corresponding to each reading level are all high and very similar in height. There is no clear trend suggesting that higher reading ability is associated with greater satisfaction.

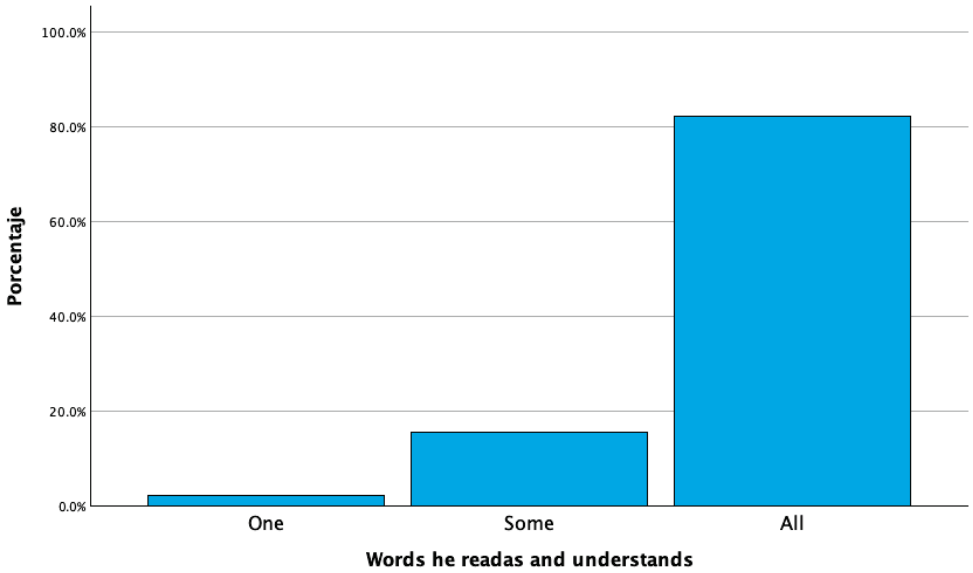


Figure 9. Percentage of children who liked the app (Y26 = "Yes") by reading ability level (Q7).

From a descriptive perspective, these results indicate that high reading ability is not required for children to enjoy using Boby. Children with more limited reading skills were still able to report high levels of satisfaction with the app.

4.4. RQ3—Completion of the "Hello" Activity

Research Question 3 (RQ3) focuses on which factors influence students' ability to complete the activity shown in Figure 4, namely teaching Boby to say "Hello" using the ScratchJr. blocks. For this analysis, log data on completion were available for 42 students, distinguishing between completion without help, completion with help, and non-completion.

Table 5 summarises the distribution of completion outcomes by group. Among neurotypical students with available log data (n = 41), 37 completed the activity (2 without help and 35 with help), while 4 did not complete it. For the neurodivergent student with available log data (n = 1), the activity was not completed. Thus, in total, 37 out of 42 students with valid completion data (88.1%) succeeded in completing the "Hello" activity, although the majority required some degree of assistance.

https://doi.org/10.3390/computers15010017

Table 5. Completion of the "Hello" activity by group and need for help (valid log data).

Group	n Total	n Completed Without Help	n Completed with Help	n Not Completed
Neurotypical	41	2	35	4
Neurodivergent	1	0	0	1

These results indicate that most students were able to use the system to solve the simple programming task, but also that support from teachers or assistants played an important role in enabling completion, particularly for some students. Given the small number of neurodivergent students with log data, the results for this group should be interpreted with extreme caution and are better understood as illustrative cases rather than as generalisable evidence.

4.5. Synthesis

Across the three research questions, the descriptive results show a coherent pattern. First, satisfaction with Boby was very high in almost all subgroups considered (RQ1), and there was no clear descriptive evidence that usual tablet use or preference for block-based games systematically influenced whether children liked the app (see Tables 2 and 3 and Figures 7 and 8). Second, reading ability did not appear to be a limiting factor for enjoyment of the system (RQ2): children with different levels of reading proficiency reported very similar percentages of "I liked the app" (Table 4 and Figure 9). Third, most students successfully completed the "Hello" activity (RQ3), including the two neurodivergent students, who required only minimal support (Table 5).

Overall, within the limits of this small and imbalanced sample, these descriptive findings suggest that Boby was perceived as usable and enjoyable by the large majority of participating children, regardless of their previous experience with technology, their preference for block-based games, or their reading ability.

5. Discussion

Regarding the first research question (P1) about the factors that influence students' satisfaction with Boby, the following hypotheses were explored:

H1. *Students who are used to learning with a tablet will find the use of the Boby system more satisfactory.*

H2. *Students who enjoy block-based games will find the use of the Boby system more satisfactory.*

As shown in the results (Section 4.1), neither of the two hypotheses was confirmed. This means that it is not necessary for students to be accustomed to using a tablet in order to use Boby, nor is it necessary for them to enjoy block-based games.

These results are relevant because they broaden the number of students who can interact with the agent to carry out their programming training using ScratchJr.

In particular, this includes autistic students who might not show interest in playing with blocks [61], but who may still be motivated to use Boby for training in ScratchJr., which is a block-based language.

It also takes into account students who usually do not have access to tablets due to economic constraints in their families or other types of limitations. Specifically, this refers to new regulations such as those in the [62] in Spain, where the use of screens has been legally prohibited for children aged 0 to 3, limited to one shared hour per week for students in the second cycle of Early Childhood Education (ages 3–6) and the first cycle of Primary Education (ages 6–8), one and a half hours per week for students in the second cycle of

https://doi.org/10.3390/computers15010017

Primary Education (ages 8–10), and two hours per week for those in the third cycle of Primary Education (ages 10–12).

There are exceptions to these laws for children with special educational needs, provided that a psychopedagogical report justifies that technology could support their learning [62].

Regarding the second research question, which examined the extent to which a student's reading ability influences their capacity to use Boby, the statistical analysis presented in Section 4.2 also found no evidence that higher reading ability is associated with a greater likelihood of satisfaction when using Boby. This finding is significant because it indicates that students do not need to know how to read in order to use the app, which was designed to be visually accessible through descriptive icons and also includes auditory aids for students who are visually impaired.

These results are consistent with other studies that have gone a step further in exploring the inverse relationship—namely, that the use of an interactive application may even improve reading ability. In particular, studies have shown that young learners can engage successfully with interactive reading applications and that such experiences can positively influence their later non-digital reading skills [63,64].

Regarding the third research question, which explores the factors that influence students' ability to complete the activity shown in Figure 4—that is, teaching Boby to say "Hello"—the results show that both neurotypical and neurodivergent students are able to complete the activity regardless of their ability to use tablets, read, or visually identify the content of images.

In total, 38 out of 48 students were able to complete the activity (79.2%), including the two neurodivergent students, although both indicated that they had needed help. In any case, the assistance provided was minimal, and at no point did the teacher or the researcher complete the exercise for the student; rather, support was given only to allow the student to finish it independently.

When examining the cases in which students did not need help to complete the activity, no significant pattern was observed that could link reading ability or visual image recognition skills. Based on direct observation, it appears instead that this may be more closely related to the students' own attitude—specifically, their preference for completing activities independently and their lower tendency to ask for or request help.

Finally, it should be noted that no student highlighted the agent's "student" behaviour, possibly due to the short duration of the interaction (only a single session). This time limitation should be taken into account when interpreting the results.

It should also be considered that validating the usability of the agent on one simple task (i.e., an activity to say hello, input/output concept) is quite limited to validating the learning outcomes of all programming concepts.

6. Conclusions

The use of a pedagogical conversational agent in the role of a student to train learners in ScratchJr. appears to be effective, regardless of the students' ability to use tablets or play with blocks, when the students are between 6 and 7 years of age.

Overall, 79.2% of the students were able to complete the activity requested by the agent. However, as stated in the Discussion, more research should be devoted to exploring the results with other programming concepts such as sequences or loops.

In the pilot study with the two neurodivergent students, it is noteworthy that both were also able to complete the activity. More sample is needed and research on how students with special needs could use agents such Bobby to learn ScratchJr.

https://doi.org/10.3390/computers15010017

Further exploration is planned to investigate the agent's potential for teaching additional ScratchJr. instructions over longer periods and multiple sessions with more students with special needs, and to subsequently assess to what extent the agent enables both neurotypical and neurodivergent students to program directly in the ScratchJr. environment.

Author Contributions: Conceptualization, M.J.M.; methodology, D.P.M.; software, M.J.M.; validation, D.P.M.; formal analysis, C.P.; investigation, M.J.M.; resources, M.J.M.; data curation, C.P.; writing—original draft preparation, M.J.M.; writing—review and editing, D.P.M. and C.P. All authors have read and agreed to the published version of the manuscript.

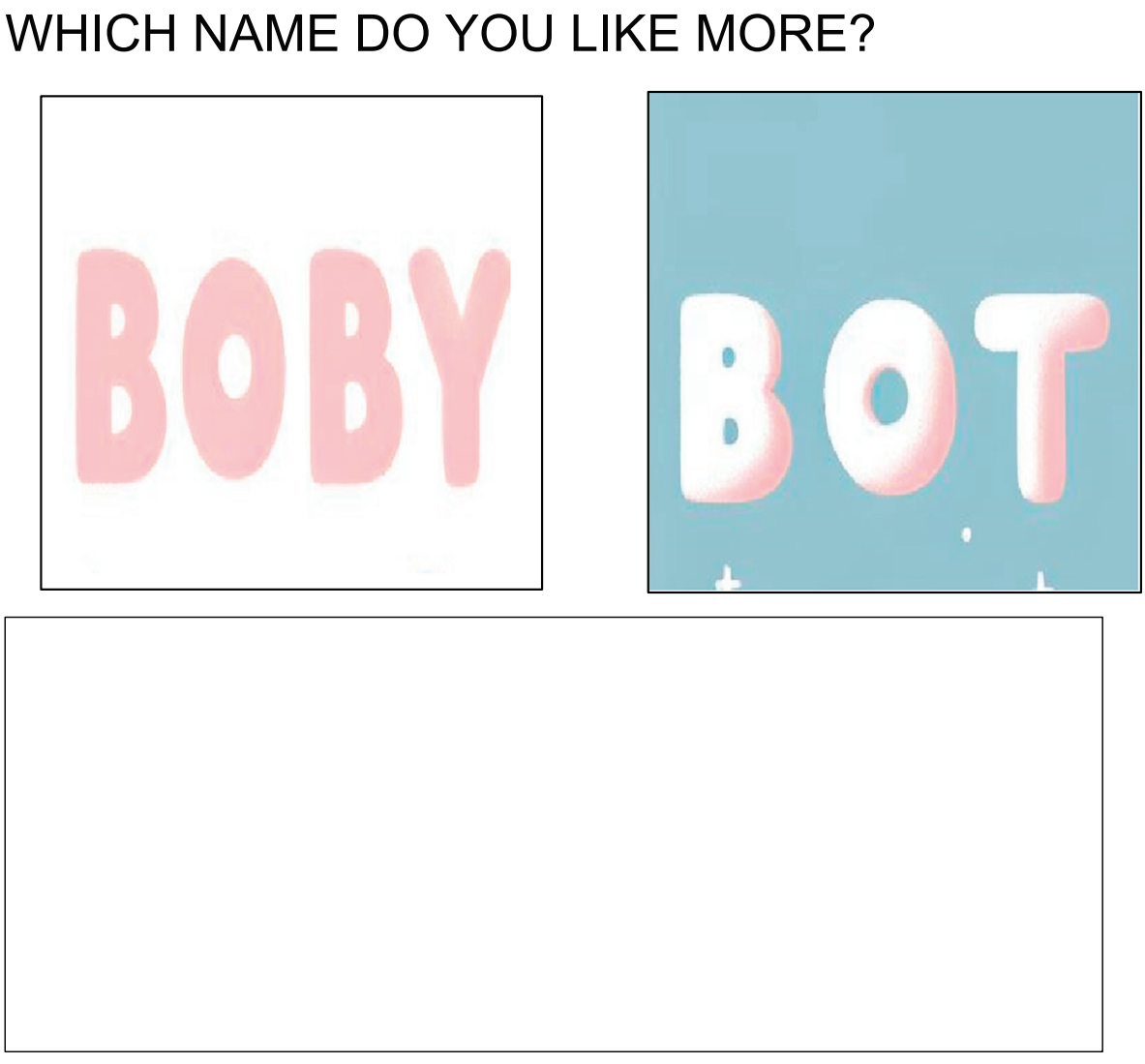
Funding: This research was funded by MICIU/AEI/10.13039/501100011033 and FEDER/UE, grant number PID2022-137849OB-I00.

Data Availability Statement: The data utilized and generated by this study is publicly available at the following University's repository: https://edatos.consorciomadrono.es/dataset.xhtml?persistentId=doi:10.21950/XP2SPQ (accessed on 16 December 2025).

Acknowledgments: We would like to thank all students and teachers involved in the experiment.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A



https://doi.org/10.3390/computers15010017

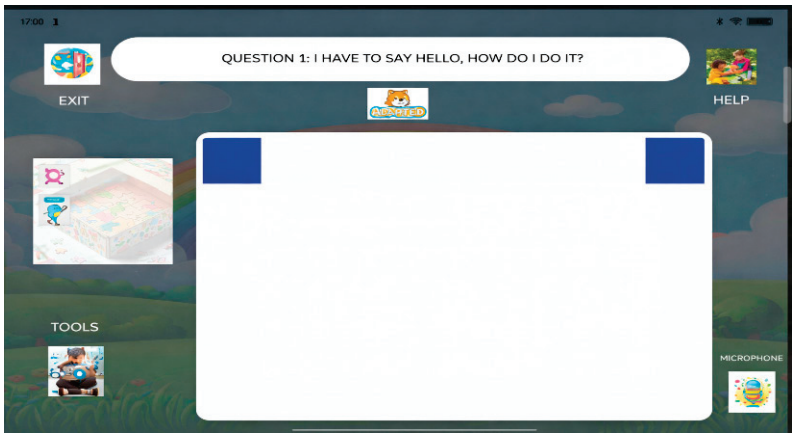
HAVE YOU COMPLETED THE EXERCISE TO SAY HELLO?







https://doi.org/10.3390/computers15010017



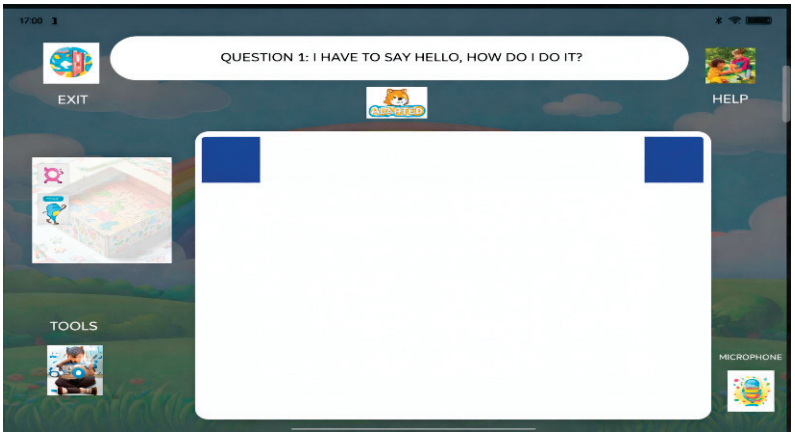
DID YOU DO THE ACTIVITY WITH HELP?





https://doi.org/10.3390/computers15010017

WAS THE ACTIVITY EASY?







https://doi.org/10.3390/computers15010017

MARK THE IMAGES WHERE YOU KNOW WHAT THE CHILDREN ARE DOING









https://doi.org/10.3390/computers15010017

MARK THE IMAGES WHERE YOU KNOW WHAT THE CHILDREN ARE DOING

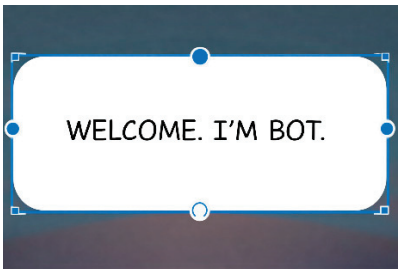


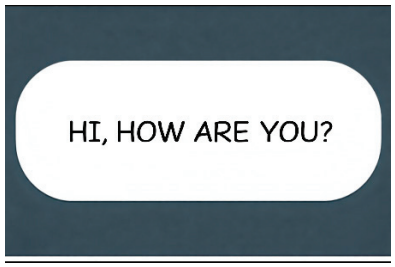






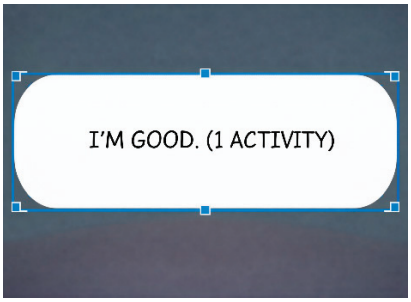
MARK THE WORDS YOU CAN READ





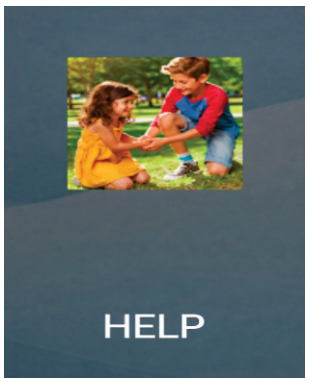


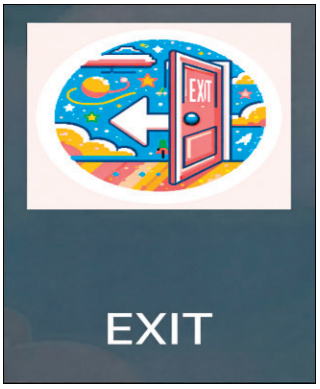




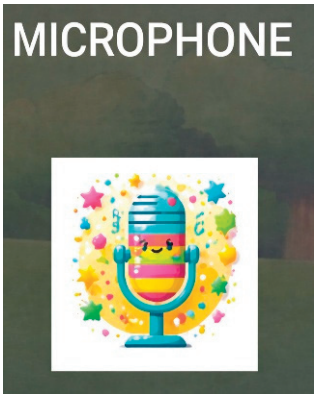
https://doi.org/10.3390/computers15010017

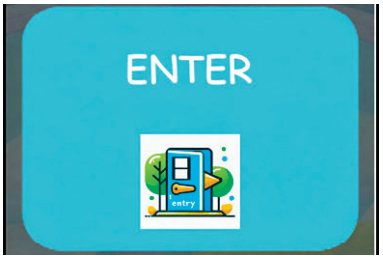
MARK THE IMAGES YOU UNDERSTAND



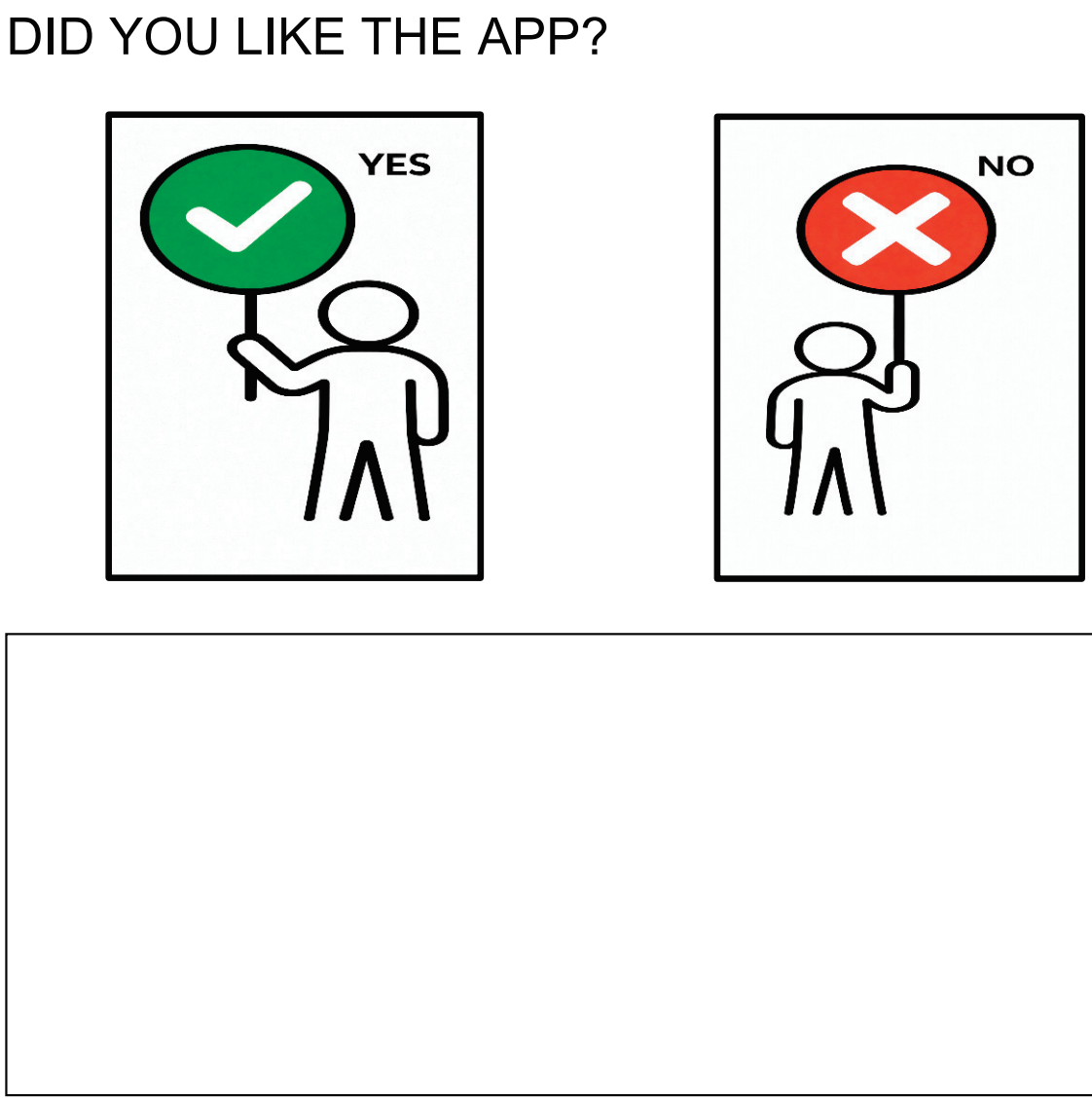








https://doi.org/10.3390/computers15010017



References

1. Abascal, J.; Aedo, I.; Cañas, J.; Gea, M.; Gil, A.; Lorés, J.; Martínez, A.; Ortega, M.; Valero, P.; Vélez, M. La Interacción Persona-Ordenador; Lleida, Spain. 2001. Available online: https://aipo.es/wp-content/uploads/2022/02/LibroAIPO.pdf (accessed on 16 December 2025).
2. Muñoz-Arteaga, J.; Collazos, C.; Granollers, T.; Luna-García, H. Perspectivas en la Interacción Humano-Tecnología; Red HCI-Collab; Lleida, Spain, 2023. Available online: https://hci-collab.uxartetic.com/ (accessed on 16 December 2025).
3. Computer Science Teachers Association (CSTA). Computer Science K–8: Building a Strong Foundation. 2012. Available online: https://csteachers.org/k12standards/ (accessed on 16 December 2025).
4. Manches, A.; Plowman, L. Computing education in children's early years: A call for debate. *Br. J. Educ. Technol.* **2017**, *48*, 191–201. [CrossRef]
5. Wing, J. Computational Thinking. *Commun. ACM* **2006**, *49*, 33–35. [CrossRef]
6. Sánchez-Vera, M. La robótica, la programación y el pensamiento computacional en la educación infantile. *Rev. Infanc. Educ. Y Aprendiz.* **2020**, *7*, 209–234.
7. Caguana-Anzoátegui, L.; Rodrigues-Pereira, M.; Solís-Jarrín, M. Cubetto for preschoolers: Computer programming code to code. In Proceedings of the 2017 International Symposium on Computers in Education (SIIE), Lisbon, Portugal, 9–11 November 2017; pp. 1–5. [CrossRef]
8. Alsina, Á.; Acosta, Y. Conectando la educación matemática infantil y el pensamiento computacional: Aprendizaje de patrones de repetición con el robot educativo programable Cubetto. *Innovaciones Educ.* **2022**, *24*, 1–20. [CrossRef]
9. Ramos Ruiz, J.; Paredes Barragán, P. Desarrollo del pensamiento Computacional mediante Cubetto en Educación Infantil. *IE Comun. Rev. Iberoam. De Informática Educ.* **2023**, *38*, 35–44.

https://doi.org/10.3390/computers15010017

10. Pérez Vázquez, E.; Lorenzo Lledó, G.; Lledó Carreres, A. Aplicación del Robot Bee-Bot en las Aulas de Educación Infantil y Educación: Una Revisión Sistemática Desde el Año 2016. 2021. Available online: https://rua.ua.es/server/api/core/bitstreams/2ed6f7f2-6c22-4f71-8401-1056292a5440/content (accessed on 16 December 2025).
11. Buendía Cueva, G.I.; Tasayco Díaz, A.P.; Menacho Rivera, A.S. Gamificación y tecnología en la educación infantil: Una revisión sistemática. *Rev. InveCom* **2025**, *5*, 1–8.
12. Ruiz Moltó, M.; Arteaga Martínez, B. El pensamiento geométrico-espacial y computacional en educación infantil: Un estudio de caso con KUBO. *Contextos Educ. Rev. De Educ.* **2022**, *30*, 41–60. [CrossRef]
13. Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J.; Silverman, B.; et al. Scratch: Programming for all. *Commun. ACM* **2009**, *52*, 60–67. [CrossRef]
14. Bers, M. El desarrollo de Scratch J.R: El aprendizaje de programación en primera infancia como nueva alfabetización. *Virtualidad Educ. Y Cienc.* **2023**, *26*, 43–62. [CrossRef]
15. Narváez Minda, J.; Torres Navas, E.J.; Analuisa Maguashca, J.; Guerrón Varela, E.R. Software Scratch J.R como recurso pedadógico para la consolidación de nociones espaciales en Educación Infantil. *MIKARIMIN Rev. Multidiscip.* **2021**, *7*, 71–84.
16. Meyer, A.; Rose, D.H.; Gordon, D. *Universal Design for Learning: Principles, Framework, and Practice*, 3rd ed.; CAST: Lynnfield, MA, USA, 2025; ISBN 9781943085392.
17. Smith, J.; Brown, L. Scratch Jr. Tactile: An inclusive approach to programming for young children. *J. Educ. Technol.* **2013**, *10*, 45–60.
18. Elshahawy, M.; Aboelnaga, K.; Sharaf, N. CodaRoutine: A serious game for introducing sequencial programming concepts to children with autism. In Proceedings of the IEEE Global Enginerering Education Conference (EDUCON), Porto, Portugal, 27–30 April 2020.
19. Zubair, M.S.; Brown, D.; Hughes-Roberts, T.; Bates, M. Designing accessible visual programming tools for children with autism spectrum condiction. *Univers. Access Inf. Soc.* **2021**, *22*, 277–296. [CrossRef]
20. Taylor, M. Computer programming with Pre-K through first-grade students with intellectual disabilities. *J. Spec. Educ.* **2018**, *52*, 78–88. [CrossRef]
21. Utreras, E.; Pontelli, E. Introductory programming and young learners with visual disabilities: A review. *Univers. Access Inf. Soc.* **2023**, *22*, 169–184. [CrossRef]
22. Johnson, J.L. Animated Pedagogical Agents: Face-to-Face Interaction in Interactive Learning Environments. *J. Artif. Intell. Educ.* **2000**, *11*, 47–78.
23. Ocaña, J.; Morales-Urrutia, E.; Pérez-Marín, D.; Pizarro, C. Can a Learning Companion Be Used To Continue Teaching Programming to Children Even During The COVID-19 Pandemic? *IEEE Access* **2020**, *8*, 157840–157861. [CrossRef]
24. Morales-Urrutia, E.; Ocaña, J.; Pérez-Marín, D.; Pizarro, C. Cand Mindfulness Help Primary Education Students to Learn How to Program With an Emotional Learning Companion? *IEEE Access* **2021**, *9*, 6642–6660. [CrossRef]
25. Manzanares, M.; Pérez-Marín, D.; Pizarro-Romero, C. Towards the use of pedagogic Conversational Agents to teach block-based programming to all using ScartchJr. In *Actas del XXVII Simposio Internacional de Informática Educativa (SIIE)*; SIIE: Viseu, Portugal, 2025; pp. 107–112.
26. García-Peñalvo, F. La enseñanza de la informática, la programación, el pensamiento computacional en los estudios preuniversitarios. *El Enfoque TACCLE3* **2017**, *18*, 7–17. [CrossRef]
27. Ozturk, H.; Calingasan, L. *Robotics in Early Childhood Education: A Case Study for the Best Practices*; Ozcinar, G., Wong, H., Ozturk, H., Eds.; IGI Global Scientific Publishing: Hershey, PA, USA, 2018; pp. 182–200. [CrossRef]
28. Ching, Y.; Hsu, Y.; Baldwin, S. Developing computational thinkin with educational technologies for young learners. *TechTrends* **2018**, *62*, 563–573. [CrossRef]
29. Morgado, L.; Cruz, M.; Kahn, K. Preschool cookbook of computer programming topics. *Australas. J. Educ. Technol.* **2010**, *26*, 309–326. [CrossRef]
30. Etecé. Niño de 3 Años. Biblioteca Humanidades. 2023. Available online: https://humanidades.com/nino-de-3-anos/ (accessed on 16 December 2025).
31. Bers, M. Beyond computer literacy: Supporting youth's positive development through technology. *New Dir. Youth Dev.* **2010**, *2010*, 13–23. [CrossRef]
32. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*; Basic Books: New York, NY, USA, 1980.
33. Lerner, R.; Almerigi, J.; Theokas, C.; Lerner, J. Positive youth development: A view of the issues. *J. Early Adolesc.* **2005**, *25*, 10–16. [CrossRef]
34. Pérez-Marín, D. Teaching Programming in Early Childhood Education with stories. In Proceedings of the Actas de la international Coference of Education Research and Innovation (ICERI), Seville, Spain, 11–13 November 2019; pp. 9206–9212.
35. Liukas, L. *Hello Ruby: Adventures in Coding*; COBEE: Singapore, 2015. Available online: https://www.helloruby.com (accessed on 16 December 2025).

https://doi.org/10.3390/computers15010017

36. Otterborn, A.; Schonborn, K.; Hultén, M. Investigating Preschool Educators' implementation of Computer Programming in Their Teaching Practice. *Early Child. Educ. J.* **2019**, *48*, 253–262. [CrossRef]
37. Flannery, L.; Silverman, B.; Kazakoff, E.; Bers, M.; Bontá, P.; Resnick, M. Designing ScratchJr.: Support for early childhood learning through computer programming. In Proceedings of the 12th International Conference on Interaction Design and Children, New York, NY, USA, 24–27 June 2013; pp. 1–10.
38. Yu, T.; Roque, N. Designing pedagogic conversational agents: A framework and review. *Int. J. Artif. Intell. Educ.* **2022**, *32*, 1–35.
39. Beraza, I.; Pina, A.; Demo, B. Soft & hard ideas to improve interaction with robots for kids & teachers. In Proceedings of the SIMPAR 2010 International Conference on Simulation, Modeling and Programming for Autonomous Robots, Darmstadt, Germany, 15–18 November 2010; pp. 549–557.
40. Highfield, K. Robotic toys as a catalyst for mathematical problem solving. *Aust. Prim. Math. Classr.* **2010**, *15*, 22–27.
41. Stoeckelmayr, K.; Tesar, M.; Hofmann, A. Pre-primary children programming robots: A first attempt. In Proceedings of the 2nd International Conference on Robotics in Education, Thessaloniki, Greece, 23–25 April 2011; pp. 185–192.
42. Kazakoff, E.; Bers, M. Programming in a robotics context in the pre-primary classroom: The impact on sequencing skills. *J. Educ. Multimed. Hypermedia* **2012**, *21*, 371–391.
43. Bers, M.; Flannery, L.; Kazakoff, E.; Sullivan, A. Computational Thinking and tinkering: Exploration of an early childhood robotics curriculumn. *Comput. Educ.* **2014**, *72*, 145–157. [CrossRef]
44. Bravo Sánchez, F.; Forero Guzmán, A. La Robótica como un recurso para facilitar el aprendizaje y desarrollo de competencias generales. *Teoría De La Educación. Educ. Y Cult. En La Soc. De La Inf.* **2012**, *13*, 120–136. [CrossRef]
45. Ruiz Velasco, E.; Beauchemin, M.; Freyre, A.; Martínez, P.; García, V.; Rosas, L.; Minami, Y.; Velásquez, M. Robótica Pedagógica: Desarrollo de Entornos de Aprendizaje con Tecnología. Virtual Educa. 2006. Available online: https://www.academia.edu/3249497/Rob%C3%B3tica_pedag%C3%B3gica_desarrollo_de_entornos_de_aprendiz (accessed on 16 December 2025).
46. Acuña, A. Robótica y Aprendizaje por Diseño. Educación año xlviii-xlix, 2004, 139–140. Available online: https://es.scribd.com/document/333359840/Robotica-y-Aprendizaje-Por-Diseno (accessed on 16 December 2025).
47. González, A.; Hernández, A.R. AlToy 1: Un robot neo-educativo con emociones. *Rev. Iberoam. De Informática Educ.* **2013**, *51*, 51–62.
48. Romero Martinez, S.J.; Calzada, I.G.; García Sandoval, A.; Lozano Domínguez, A. Herramientas Tecnológicas para la educación Inclusiva. *Rev. Tecnol. Cienc. Y Educ.* **2018**, *2018*, 83–112. [CrossRef]
49. Pérez de la Maza, L. Programa de estructuración ambiental por ordenador para personas del especto autista: PEAPO. In *Las Nuevas Tecnologías en la Respuesta Educativa a la Diversidad*; Soto Pérez, F.J., Rodríguez Vázquez, J., Eds.; Selegrafía, S.L.: Murcia, Spain, 2000; pp. 255–258.
50. Guaña-Moya, J.; Rodrigo Altamiraño-Pazmiño, M. Herramientas de Software para la educción inclusiva en la etapa de educación inicial. *Rev. Enectiva* **2024**, *5*, 1–14.
51. Rozengardt, A. *Lo no Formal en la Atención y Educación de la Primera Infancia*; UNESCO: Paris, France, 2020.
52. Sánchez Sánchez, T. La influencia de la motivación y la cooperación del alumnado de primaria con la Robótica educativa: Un estudio de caso. *Panorama* **2019**, *13*, 117–140. [CrossRef]
53. Merino-Armero, J.M.; Villena-Taranilla, R.; Somoza, J.A.G.-C.; Cózar-Gutiérrez, R. Análisis del efecto de la robótica en la motivación de estudiantes de tercero de Educación Primaria durante la resolución de tareas de interpretación de planos. *Rev. De Estud. Y Exp. En Educ.* **2017**, *3*, 163–173. [CrossRef]
54. Jiménez, L. El Poder y la Ciencia de la Motivación. Cómo Cambiar tu Vida y Vivir Mejor Gracias a la Ciencia de la Motivación. 2017. Available online: https://www.scribd.com/document/544570897/El-poder-y-la-ciencia-de-la-motivacion-PDFDrive (accessed on 16 December 2025).
55. Angeli, C.; Valanides, N. Developing young children's computational thinking with educational robotics: An intervention study. *J. Educ. Comput. Res.* **2020**, *58*, 461–484.
56. Di Lieto, M.; Inguaggiato, E.; Castro, E.; Cecchi, F.; Cioni, G.; Dell'Omo, M.; Laschi, C.; Dario, P. Educational robotics and students with neurodevelopmental disordes: A systematic review. *Heliyon* **2020**, *6*, e05160. [CrossRef]
57. Pivetti, M.; Di Battista, S.; Agatolio, F.; Moro, M. Educational robotis and students with nuerodevelopmental disorders: A systematic review. *Heliyon* **2020**, *6*, e05145. [CrossRef]
58. Elkin, M.; Sullivan, A.; Bers, M. Programming with the KIBO robotics kit in preschool classrooms. *Comput. Sch.* **2016**, *33*, 169–186. [CrossRef]
59. Gkonlnta, E.; Papadopoulos, A.; Giannakoulas, I.; Bamidis, P. Robot programming as an intervention for a child with autism spectrum disorder: A case study. *Int. J. Dev. Disabil.* **2023**, *69*, 424–431. [CrossRef]
60. Leelawong, K.; Biswas, G. Designing learning by teaching agents: The Betty's Brian system. *Int. J. Artif. Intell. Educ.* **2008**, *18*, 181–208. [CrossRef]
61. Elbeltagi, R.; Al-Beltagi, M.; Saeed, N.K.; Alhawamdeh, R. Play therapy in children with autism: Its role, implications, and limitations. *World J. Clin. Pediatr.* **2023**, *12*, 1–22. [CrossRef]

https://doi.org/10.3390/computers15010017

62. Comunidad de Madrid Comunidad de Madrid Inicio. Página no Encontrada. 2025. Available online: https://www.comunidad.madrid/ (accessed on 16 December 2025).
63. Wang, X.C.; Christ, T.; Chiu, M.M.; Strekalova-Hughes, E. Exploring the relationship between kindergarteners' buddy reading and individual comprehension of interactive app books. *AERA Open* **2019**, *5*, 1–17. [CrossRef]
64. Raja, P.; Setiyadi, A.B.; Riyantika, F. The Correlation between perceptions on the use of online digital interactive media and reading comprehension ability. *Int. J. Engl. Lang. Lit. Stud.* **2021**, *10*, 292–319. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

https://doi.org/10.3390/computers15010017

computers

MDPI

Article

Explain Trace: Misconceptions of Control-Flow Statements

Oleg Sychev * and Mikhail Denisov

Software Engineering Department, Volgograd State Technical University, Lenin Ave, 28, 400005 Volgograd, Russia
* Correspondence: o_sychev@vstu.ru

Abstract: Control-flow statements often cause misunderstandings among novice computer science students. To better address these problems, teachers need to know the misconceptions that are typical at this stage. In this paper, we present the results of studying students' misconceptions about control-flow statements. We compiled 181 questions, each containing an algorithm written in pseudocode and the execution trace of that algorithm. Some of the traces were correct; others contained highlighted errors. The students were asked to explain in their own words why the selected line of the trace was correct or erroneous. We collected and processed 10,799 answers from 67 CS1 students. Among the 24 misconceptions we found, 6 coincided with misconceptions from other studies, and 7 were narrower cases of known misconceptions. We did not find previous research regarding 11 of the misconceptions we identified.

Keywords: misconceptions; control-flow statements; introductory programming

1. Introduction

Nowadays, the growing need for software engineering (SE) specialists makes SE educational institutions improve both quantitatively and qualitatively. Mastering programming fundamentals remains one of the key requirements for successful software engineering education. However, this field introduces a lot of new concepts which are difficult to master [1,2]. This attracts a lot of attention from researchers, aiming at studying and improving introductory programming courses.

Most SE teachers witnessed students becoming confused while trying to understand the behavior of a computer program [3] or learn new concepts. Unfortunately, program execution in most of the IDEs is not visible until the user takes steps to visualize it (such as using the debugger), which many worse-performing students do not do, relying on their guesses instead. There is often little time to identify and eliminate the misconceptions of each student personally. So, correcting misconceptions should be a part of the regular educational process, as studying common misconceptions is useful. In software engineering, there have been a number of studies concerning misconceptions [4–7]; different methods have been developed for detecting misconceptions [8]. Some studies were conducted to fix misconceptions during classroom work [9]. Usually, methods for detecting and fixing misconceptions are based on domain knowledge and typical errors in students' mental models.

A review of the related literature showed that little attention has been paid to the systematic study of misconceptions related to control-flow statements, unlike other topics such as the syntax of programming languages [10,11], memory allocation for objects and scalar values [12], passing parameters by value or by reference, and other advanced topics. The misconceptions concerning sequences, loops, and selection statements are mixed with misconceptions about other programming constructs in many studies. This does not allow gathering data aimed specifically at identifying misconceptions of control-flow statements and often makes misconception definitions too general.

The misconceptions described in the literature were formulated differently: some are very specific, and others are vague. The systematic formulation of a set of possible

misconceptions is an important challenge helping to cover the subject. Misconceptions are identified in order to correct students' mental models more easily, so a good misconception formulation should allow for identifying ways to correct them.

So, we see some misconceptions as too broadly formulated to be useful: their formulations may be too abstract or combine several different errors. Such high-level misconceptions are suitable only for superficial diagnostics and cannot be used for planning corrective pedagogical interventions. For example, the misconception "T2: Students misunderstand the process of while loop operation" [12] is too abstract and useful only for giving the student more tasks on while loops; it says nothing about how to correct their understanding of loops in general (e.g., it may relate to repeating the same lines of code, the role of loop conditions, etc.) and of the "while" loop in particular (e.g., the precondition).

Consider an example of composite misconception "M33: Loops terminate as soon as the condition changes to false" mentioned by [5,13]. Although it expresses a well-formed state of the student's mind, it touches on several low-level misconceptions, such as breaking the sequentiality of actions within the loop body and ending the loop without explicit evaluation of the loop's condition.

Obviously, it is impossible to consider and describe all the high-level misconceptions that students have (and may have in the future) about control-flow structures, which reduces the effectiveness of teaching aimed at detecting and removing high-level misconceptions. However, it is possible to reduce high-level errors to low-level misconceptions that relate to elementary rules for executing program code. In this case, the misconceptions should be aimed at explaining one trace line—i.e., one event during code execution. If we continuously correct simple misconceptions and explain to the students why this statement should be executed next, then, over time, we can deal with all higher-level misconceptions. For example, if the control condition of a selection statement is false, the "then" branch should not be executed; an attempt to execute the "then" branch can be attributed to a misconception related to the consequences of the selection statement's condition evaluating to false. An example of a high-level misconception about this is "The 'then' branch is always executed" [13].

Our research questions are:

- RQ1. What undergraduate students' misconceptions of control-flow statements can be identified by interpreting a single line of trace?
- RQ2. How do these misconceptions relate to the misconceptions known in the literature?

Misconceptions concerning sequences, loops, or selection statements are found in many studies intermixed with wrong ideas about similar topics; they are rarely studied on their own. That affects good coverage of the topic, e.g., Sirkiä and Sorva mention "Wrong False" but no kind of "Wrong True" [14]; Albrecht and Grabowski report "order of conditions" but ignore the possibility of skipping conditions [15].

Most of the researchers report only frequent misconceptions, which makes sense for teachers who have limited classroom time. However, the developers of software tutors should take into account less frequent misunderstandings as well to provide students with useful feedback in the maximum range of possible situations. Automatic feedback requires the misconceptions to be specific and precise. Some authors provide too broad formulations of misconceptions such as "Students misunderstand the process of while loop operation" [12], or "Difficulties in understanding the sequentiality of statements" [13], which gives no idea of what is actually wrong and what kind of help the student needs besides re-reading the relevant textbook chapter.

Too few misconceptions about loops have been found; they are mostly concerned with just "loop" or "while loop" [12]. We found no mention of other kinds of loops, such as "for loop" and "do-while loop", and a variant of the latter, "repeat-until loop", which has its condition inverted. The now popular family of "foreach" loops is closely related to the concept of collection and can easily be modeled by simpler loops, so we believe that "foreach" can be studied as a variant of the "for" loop.

In this study, we aimed to close this gap and research specifically the misconceptions about the control flow during program execution. We did it by showing an algorithm and its trace to students and asking them to explain why a particular trace line should be there (for correct traces) or should not be there (for incorrect traces). That limited the number of errors the students could make but let us get an insight into their thinking. As a result, we confirmed some already reported misconceptions, created detailed versions of some broadly generalized misconceptions, and identified misconceptions that we did not find in the relevant literature.

The identified misconceptions can help teachers plan their lessons and develop assessments; they also can be used to foster the development of software tutors providing automatic feedback: for example, to explain deviations from the solution path [16], to create sequences of follow-up questions [17], or to indicate invalid solution states [18].

This article is organized as follows. Section 2 describes related works from two points of view—methods for identifying misconceptions and examples of misconceptions related to selection statements and loops. Section 3 explains how the research on misconceptions was conducted. Section 4 describes the results of the study. In Section 5, we compare our findings with findings of other researchers and briefly discuss the limitations of our approach. Section 6 describes the threats to study validity. The Section 7 emphasizes the most important findings and shows the directions for further research. The list of the misconceptions found is provided in Appendix A.

2. Related Work

Identifying misconceptions is a developed field of study in many areas of education, including software engineering. Misconceptions of many different areas of software engineering education have been studied over time [4,19–21].

Some of those works are devoted to studying the influence of programming languages and teaching methods on understanding the basics of programming [22]. Others analyzed methods for detecting and reducing misconceptions [8,9]. There are two main aspects of the reviewing literature about misconceptions in software engineering education: the methods used and misconceptions in related areas that can be compared to our findings.

2.1. Methods of Studying Misconceptions

Identifying misconceptions requires collecting representative data on students' thinking. Two basic approaches are to study the students' thinking (mainly using structured and unstructured interviews) or the artifacts that students produce (program code, result prediction, code-tracing problems).

2.1.1. Interview

Various forms of interviews were used to study misconceptions such as verbal communication about assignments [23], "explain in plain English" [3], etc. Researchers have used the interview as a method of detecting students' misconceptions [24,25]. In particular, Kaczmarczyk et al. [12] conducted interviews with students, encouraging a student to think aloud. The communication protocols were recorded and handed over to the specialists who analyzed the interview results independently to avoid bias. The assessment results for each answer were discussed, and the researchers came to a general conclusion on the type of misconception the student had.

This method allows in-depth insight into the students' thinking. Still, it requires significant work to process each interview, so the number of participants is usually small, which threatens the validity of the results. One of the ways of increasing the number of participants is by structuring the interviews into a series of free-text responses to predefined questions. Further, it is possible to survey teachers (instead of students) about the misconceptions they encounter [26,27].

2.1.2. Analysis of Code Written by Students

This method relies on the results of code-writing assessments [28–32]. In their study, Ref. [15] used a bank of program code submitted by students to an automated learning environment. The code submissions were tested automatically, and the ones that did not produce the required answer (or the code did not compile) were chosen for further analysis. Each researcher assessed them.

This method allows analyzing bigger banks of code but does not give insight into what the student thought while writing the analyzed code.

2.1.3. Result-Prediction Problems

A result-prediction problem requires the student to imagine the execution of a program (or parametric function) and predict its output [13] or return value [6,7].

The method of mining misconceptions from students' predictions is based on guessing a mental model that, with the given program, would produce the prediction obtained. That is not always possible, so only a small fraction of answers is normally used. The advantage of this method, as well as of the previous method, is a significantly higher number of participating students.

2.1.4. Code-Tracing Problems

To understand how, in the student's opinion, a program (or algorithm) works, the researchers can ask the student to build an execution trace of the given program. Sirkiä and Sorva asked students to simulate the execution of a small program by manually performing all actions of the program in the graphical interface of a visual simulator [5,14]. It concerned the data about values, operators, function calls, parameters passed to functions, call stack, changes in the execution position, etc. The researchers analyzed obtained traces of program code execution by each student. They reported, among other things, patterns of erroneous behavior, probably related to the complexity of the simulator's user interface (UI). To reduce the cognitive load on the student, this approach can be enhanced by using simulators, focusing only on one aspect of the program behavior [33].

2.1.5. Combined Methods

Some studies combine different methods: for example, Velázquez-Iturbide [29] analyzed the code created by the students in conjunction with free-text comments written by them in their code.

2.2. Misconceptions Related to Flow of Control

Kaczmarczyk et al. [12] identified several misconceptions; some of them concern iterations and loops. The closest to our topic is: "Students misunderstand the process of while loop operation". However, that misconception is formulated very broadly and so is of little use.

Albrecht and Grabowski [15] identified 50 misconceptions related to programming. The following misconceptions are related to control-flow statements:

- "spurious/missing text/code fragment";
- "missing condition" (it is unclear whether it applies only to loops or to "if-else" as well);
- "missing loop";
- "order of conditions";
- "code outside of function block";
- "infinite loop".

Swidan et al. [13] studied 11 misconceptions from the big list of misconceptions compiled by Sorva [5], which contained more than 160 items. These misconceptions were better formulated and more detailed. Some of them concern control-flow statements, for example:

- 23-Ctrl "Difficulties in understanding the sequentiality of statements";
- 24-Ctrl "Code after 'if' statement is not executed if the 'then' clause is.";
- 26-Ctrl "A false condition ends program if no 'else' branch exists";
- 27-Ctrl "Both 'then' and 'else' branches are executed";
- 28-Ctrl "The 'then' branch is always executed";
- 30-Ctrl "Adjacent code executes within loop";
- 31-Ctrl "Control goes back to start when a condition is false";
- 33-Ctrl "Loops terminate as soon as condition changes to false" (interrupting iteration/ skipping the rest of iteration).

Sekiya and Yamaguchi [6] conducted a study where students were asked to predict the results of performing small functions involving loops. The closest of the seven identified misconceptions to our research is "NFL: Neglect(ignore) For Loop" (the control condition was not evaluated at the beginning of the loop or after the iteration).

Sirkiä and Sorva [14] identified 11 misconceptions by analyzing student-generated execution traces. Some of them are related to the topics of our research, e.g., "Wrong branch", "Wrong False".

3. Method

Our goal was to collect students' thoughts on how control-flow statements affect program execution. To do this, we asked them in a free-text form why the particular lines of code should or should not be in the specified positions, then marked and analyzed their answers. The scheme of our study is shown in Figure 1.

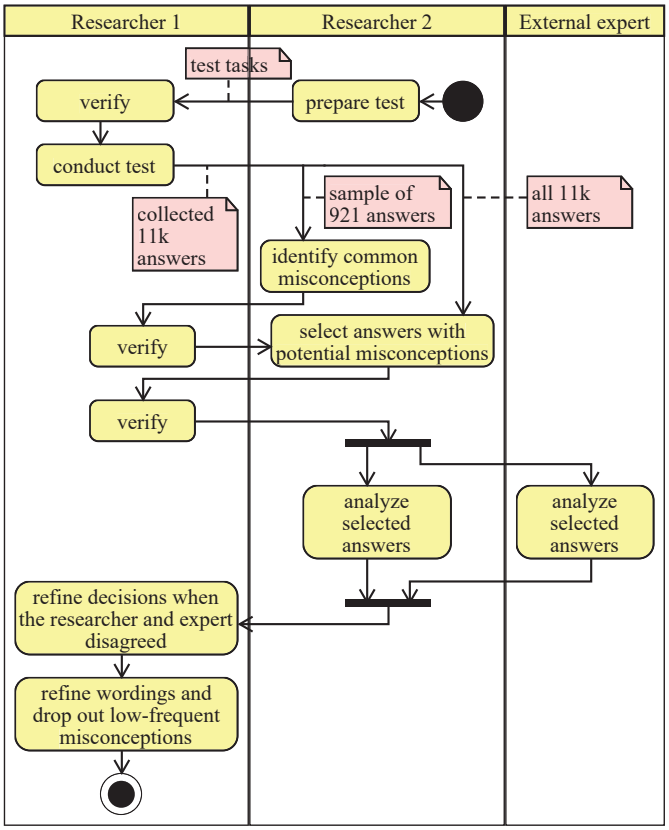


Figure 1. Method flowchart: UML activity diagram of studying misconceptions from students' responses.

3.1. Data Collection

To collect students' thoughts, we conducted a test using questions with free-text answers. Each question showed a small algorithm and its execution trace. One of the lines

in the trace was highlighted: the students had to explain why the highlighted line was correct or incorrect. We created and verified 181 questions on the reasoning for the traces of code execution (see Table 1). The questions can be divided into four categories as follows:

- Selections and sequences (correct traces) (e.g., the question shown in Figure 2);
- Selections and sequences (incorrect traces);
- Loops (correct traces);
- Loops (incorrect traces), see the example in Figure 3.

For each question category, several small algorithms were compiled—various forms of selection statements (one to three "if", "if else" branches, with or without the "else" branch), and loops ("while", "do-while", "for", "foreach"). Some questions about loops contained selection statements within the loop. Some of the algorithms have been used to create both correct and incorrect traces.

For each algorithm, a series of correct execution traces were created automatically by setting values for control conditions of selection statements and loops (we tried to keep traces short but non-trivial). A trace was represented as a sequence of acts of execution. Execution of simple statements was represented as a single act; the beginnings and ends of complex statements (selection statements and loops) and their parts (branches and iterations) comprised separate acts of execution to show nesting in the trace. That allowed us to verify how students understood statement nesting, which was rarely explicitly verified in previous studies. Statement nesting was shown by indentation and curly braces (for example, in Figure 3, the statement "wait" is inside the loop, but the statement "wait_seconds(3)" is not). The acts were supplemented with information about the values of control conditions and how many times they were executed (see Figure 3). Incorrect traces were created manually by modifying correct traces: deleting, duplicating, or moving acts of the original trace. Some correct traces were used in several questions; different lines of that trace were highlighted.

The questions used a pseudo-code notation, which was close to the C programming language the students studied. We did not specify expressions, replacing them with labels to avoid distracting students from the study topic—control-flow statements—and also avoid muddying the results with expression evaluation problems. So we replaced them with words and simple phrases resembling actions and conditions. The only exceptions were control conditions of selection statements and loops, which affected the control flow. The students saw the results of the control conditions' evaluation in the relevant trace lines (for example, see the third trace line in Figure 2). The students were provided with samples explaining the pseudo-code notation of the algorithms and trace lines before the beginning of the study.

Table 1. Test setting and answers collected.

Topic	**Selection**		**Loops**		**Total:**
Algorithms	6		9		**15**
Trace kind	*correct*	*erroneous*	*correct*	*erroneous*	
Traces	7	7	14	12	**40**
Questions	23	54	76	28	**181**
Respondents	67	67	63	47	**67**
Answers given	1539	3602	4254	1404	**10,799**
Answers with misconceptions	32	85	21	42	**180**

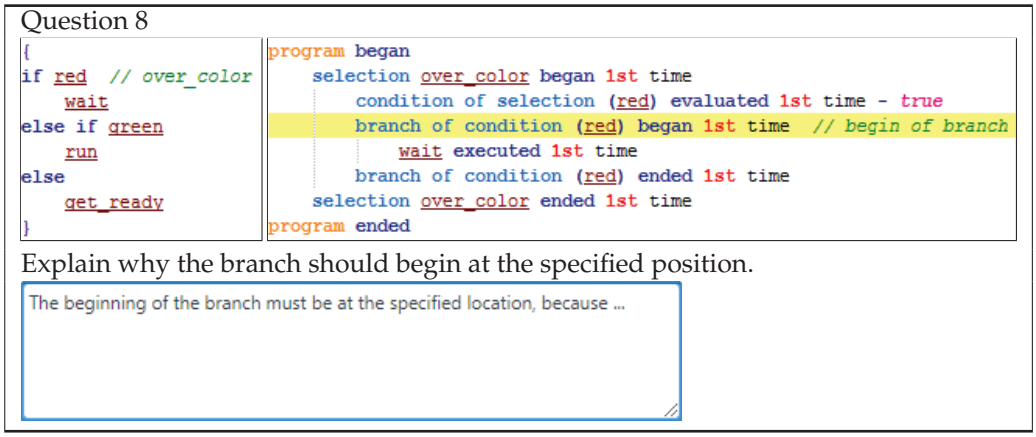


Figure 2. Example of the question about a selection statement with a correct trace.

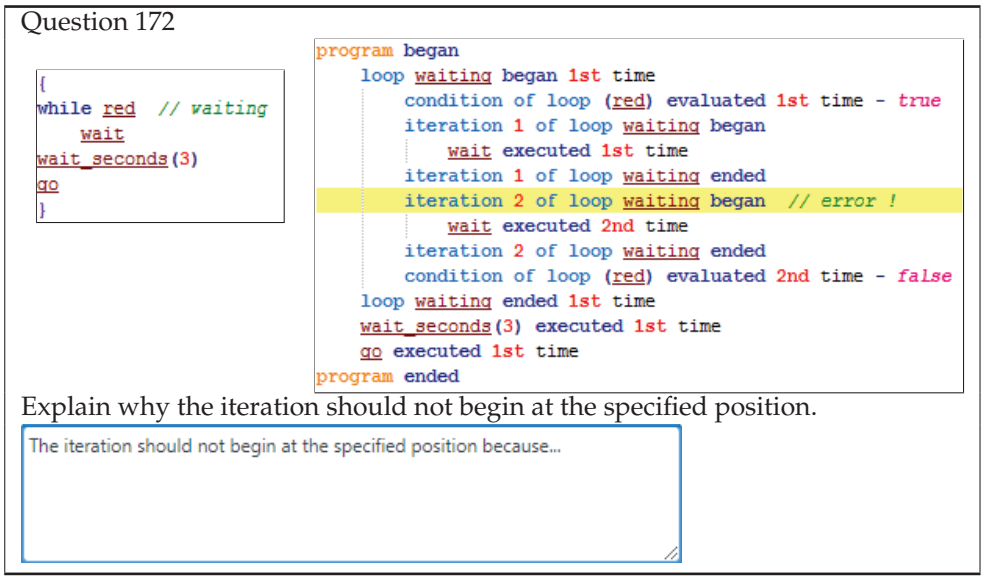


Figure 3. Example of the question about a loop with an incorrect trace.

The questions were used in a summative assignment in the "Programming basics" course for the first-year undergraduate students of Volgograd State Technical University majoring in software engineering. The test was conducted in a remote mode because of the COVID-19 lockdown. We analyzed the anonymized results of that assignment.

A total of 10,799 answers were received from 67 students.

3.2. Data Analysis

The analysis was based on a hybrid inductive–deductive approach [34]. All the researchers had experience in teaching introductory programming courses as professors and teaching assistants teaching the "Programming basics" course for years.

First, we analyzed a sample of responses to 23 questions from different categories (921 responses in total), identifying and clustering frequent errors.

Based on that clustering, a preliminary list of misconceptions that were possible to identify with the created questions was compiled. The findings were formulated in terms of low-level misconceptions about the trace acts ordering, i.e., their mutual position, the absence of a necessary act, or the presence of an extra act. For example, suppose a student places the initialization of a FOR loop after an iteration, i.e., later than its usual location at the beginning of the loop. In that case, we consider this move as both LOOP8 (missing, skipping initialization) and LOOP6 (unexpectedly late initialization) misconceptions (see Table A1). Both of these elementary problems can arise separately because

of the step-by-step nature of the trace construction and should be explained separately. Researcher 1 verified the resulting list and made corrections. The resulting list contained 16 misconceptions about selection, 12 about loops, and 9 other misconceptions.

After that, the collected answers were analyzed. Out of 2370 incorrect answers, we selected for further analysis 258 potentially interesting answers (11%) that could expose misconceptions. The remaining 89% of incorrect answers were excluded as irrelevant because they were incomplete, contradictory, lacked clarity, or were not related to answering the relevant questions (there were even 5 jokes).

Then, the remaining 258 answers were analyzed and assigned the labels from the selected sample with the misconception type from the previously created list. For each answer, the status was determined: the answer demonstrates a misconception, the answer is correct (i.e., answers the question at least partially), or the answer is unclearly formulated. After a discussion, the team excluded about 30% of the responses as not having misconceptions (either correct or irrelevant answers).

The remaining 180 responses with misconceptions (1.7% of all responses collected) were agreed to contain a misconception by both of the researchers. They were independently labeled with the previously agreed misconception labels. The two researchers' agreement about the kind of misconception calculates to be 92% (agreed on 166 responses, disagreed on 14 out of 180). After that, we compared the two lists and produced a final list, flagging the answers on which there was and was not agreement. The answers with no independent agreement were discussed with the research team and with an external expert (also a teacher assistant in an introductory course) to achieve a shared understanding of their interpretation. The Cohen's Kappa metric [35] for the final agreement was 0.918, which is interpreted as almost perfect agreement.

Finally, for each misconception, we counted the number of students who expressed it at least once as a more fair measure of its frequency (the number of questions, answers to which can expose a given misconception, varied). We excluded the misconceptions that were expressed only by one student if they were not mentioned in the previous literature.

4. Results

Table A2 shows examples of students' answers exposing misconceptions.

The final list of misconceptions aligned with the findings from related work is shown in Table A1. It includes 22 misconceptions and two types of terminological substitution:

- 3 misconceptions about sequences (SEQ*),
- 10 misconceptions about selection statements (SEL*);
- 8 misconceptions about loops (LOOP*);
- 1 misconception about concept recognition (RECsl);
- 2 misconceptions about notation (NOT*) or used terms.

Six of the misconceptions we observed align with related work precisely. These are as follows:

- SEL1 matches "28-Ctrl" reported by [5] and is close to "Misc2: Wrong branch" by [14];
- SEL2 is effectively the same as "UI2: Failing to jump upon selection" [14];
- SEL3 matches "e26: order of conditions" [15];
- SEL6 is a generalized form of "27-Ctrl: Both then and else branches are executed" reported by [5];
- LOOP3 is a sub-type of "30-Ctrl: Adjacent code executes within loop" reported by [5];
- LOOP4 is close to "Extra Count" (an extra iteration) by [7].

Other misconceptions we found are specific cases of known misconceptions. For example, "T4: Students cannot trace code linearly" [12] and "23-Ctrl: Difficulties in understanding the sequentiality of statements" reported by [5] are covered in our misconceptions SEQ1, SEQ2, and SEQ3. Another general misconception "T2: Students misunderstand the process of while loop operation" [12] was clarified by LOOP1–LOOP4.

The misconception when the student does not attempt to repeat the control condition and loop body, reported by [6] as "NFL: Neglect(ignore) For Loop", is close to LOOP2 and LOOP5. LOOP1 and LOOP2 are subclasses of "e45: missing condition" [15].

Some known misconceptions occurred rarely (i.e., were demonstrated by only one student):

- RECsl and SEL9 are kinds of "31-Ctrl: Control goes back to start when condition is false" [5,13];
- LOOP5 is a more general misconception than "Missing 0" (skipping the first iteration) by [7].

To the best of our knowledge, nine of the found misconceptions related to understanding how control-flow statements work have not been reported by other researchers. These include the following:

- SEL4, SEL10 are specific to "think aloud"/"write what you think" methods and unlikely to be found by other methods;
- LOOP6, LOOP7, LOOP8 are detectable by the "simulate execution" method too, but they are about "for" and "foreach" loops which received less attention from other researchers than "while" and "do–while" loops;
- SEL7 and SEL8 are only detectable if the researcher asks the students about the beginning and ending of each control-flow statement during execution, not just about executed lines, which was done rarely in the previous literature;
- SEL5, SEL9 can be detected by using the "simulate execution" method, but they were not reported in the literature (to the best of our knowledge); one possible explanation is the low number of tasks using selection statements with multiple branches that are required for detecting SEL5 and increase the possibility of detecting SEL9.

The two common misconceptions found (NOT1sl, NOT2pl) are related to terminology: students used the word "loop" while speaking about selection statements or the entire program. They can only be found using structured or unstructured free-text interviews; other methods do not expose the terms the students use, which may explain why they were not reported in the previous literature.

The misconception "Missed non-loop" reported in [7] was not found in our study.

5. Discussion

In this study, we researched misconceptions about control-flow statements (selection statements: "if-else", loops: "while", "do-while", "for", "foreach") by analyzing almost 10,800 short open-ended answers from students in the "Programming basics" course. The questions showed students an algorithm and its trace and asked them to explain why the particular trace line is correct or incorrect.

The vast majority of the answers we collected were at least partially correct. That indicates a generally good understanding of control-flow statements by first-year undergraduates majoring in software engineering. Many of the erroneous answers were irrelevant or incomprehensible, which indicates that teachers need to pay more attention to teaching students to formulate their thoughts clearly. Given our large initial base of answers, the number of answers exposing misconceptions remained considerable.

- RQ1. What undergraduate students' misconceptions of control-flow statements can be identified by interpreting a single line of trace?

During this study, we formulated a systematic list of 24 control-flow misconceptions regarding a single line of code, including 3 misconceptions about sequences, 10 about selection statements, 8 about loops, 1 about concept recognition, and two cases of misuse of domain terms. The most frequently observed misconceptions were as follows:

1. Not considering the truth of a control condition as the cause for the execution of its branch—demonstrated by 10 students out of 67 participants;
2. Conditions of a selection statement are checked in the wrong order—demonstrated by 9 students;

3. A "for" loop is initialized after its iteration—demonstrated by 8 students;
4. Calling the selection statement a loop—demonstrated by 8 students;
5. Failing to take into account the end of a block—demonstrated by 7 students;
6. Starting a 'then' branch after a false condition—demonstrated by 7 students;
7. Skipping the update step of a "for" or "foreach" loop—demonstrated by 7 students;
8. Two or more branches of one selection statement are executed—demonstrated by 6 students;
9. Action, which is placed after the selection, is considered belonging to its branch—demonstrated by 6 students;
10. Continuing a loop after its control condition evaluates to false—demonstrated by 6 students;
11. Action, which is placed after a loop, is considered belonging to the loop—demonstrated by 6 students.

Out of the most frequently observed misconceptions, five concern behavior of selection statements (primarily multi-branch selection), four concern loops (primarily "for" and "foreach" loops), one concerns sequences, and one concerns terminology usage. This indicates that the areas which most need additional attention during novice programming courses are selection statements with more than two branches and "for" loops which are more complex than "while" and "do-while" loops.

- RQ2. How do these misconceptions relate to the misconceptions known in the literature?

Out of 24 misconceptions that we observed, 13 are similar to the misconceptions from previous research (6 coincide with previously found misconceptions, while 7 are narrower cases of known misconceptions), and 11 were not found by us in the previous literature. The misconception "Missed non-loop" (missing a trace line) that we found reviewing other studies was relevant to our method, but no student demonstrated it during our study.

The newly found misconceptions are as follows:

1. Not considering the truth of a control condition as the cause for the execution of its branch;
2. Checking a condition after executing a branch;
3. Action, which is placed after the selection, is considered belonging to its branch;
4. Exiting a selection without executing a branch or checking all the conditions;
5. Multiple checks of the same condition in one selection statement;
6. Executing a wrong branch after checking a condition;
7. A "for" loop is initialized after its iteration;
8. Skipping the update step of a "for" or "foreach" loop;
9. Skipping the initialization of a "for" loop;
10. Calling the selection statement a loop;
11. Calling the program a loop.

6. Threats to Validity

The main threat to the validity of this research is that it was conducted in one educational institution, and all the participants were undergraduate students majoring in computer science or software engineering. It is possible that K-12 students and non-CS majors will have different distributions of the frequency of these misconceptions. Because of the method we used (free-text answers entered into an automatic test program), some of the information could be lost if the students did not consider it valuable enough to report. The test was conducted in a remote mode, which is different from the classroom situation, so the students could not ask their teachers for help during it. However, this might have allowed them to expose their misconceptions more freely. Another threat to validity is interpreting students' free-text answers. Some of them were worded so vaguely that it was impossible to say what the student meant. We excluded too vague answers from misconception identification, which might have affected the observed frequency of the misconceptions. The test was analyzed anonymously so that the students would not fear

the results affecting their course grades; that did not allow us to interview the participants about their vague answers.

7. Conclusions

This study provides a new perspective on misconceptions in computer science education. First of all, in the previous research, the methods allowing insight into students' thinking (interview, focus groups) did not allow analyzing a large sample of answers, which can lead to missing some of the less common misconceptions. The methods allowing a large number of participants (code analysis, problem solving) allow the researchers to work only with the artifacts of student's work, which limits the number of misconceptions that can be detected. By labeling and analyzing free-text responses to predefined questions, we were able to work with a larger student sample while letting them answer in their own words.

While undergraduate CS majors are common subjects in studies of misconceptions of novice programmers, most of the existing studies span all the misconceptions encountered in CS0/CS1 course. None of them to our knowledge concentrated on studying misconceptions of control-flow statements in particular. That often led to broad formulations of found misconceptions and could cause some of them to be omitted. Our research closes this gap. We created a set of questions requiring students to explain lines of program traces, which were specifically aimed at verifying understanding of control-flow statements. This allowed us to find 11 new misconception and narrow definitions of 7 already-known misconceptions, along with confirming 6 previously known ones.

Among the found misconceptions, the two most frequently observed misconceptions (and three out of the seven most frequently observed misconceptions) concerned multi-branch selection statements. This shows that learning primarily one-branch and two-branch "if-else" statements is not enough and more attention should be given to the behavior of more complex selection structures, including "else if" (or "elif") clauses. Another two out of the seven most frequently observed misconceptions concern the behavior of "for" and "foreach" loops, which is more complex than the behavior "while" and "do-while" loops. "For" loops also need additional attention. A somewhat surprising finding is that students often called any non-linear algorithmic structure a loop, so terminology questions, stressing the difference between iterative and non-iterative algorithmic structures, are needed.

These misconceptions can be used in developing web-based intelligent tutoring systems for teaching programming such as Problets [36], HowItWorks or CompPrehension [17]. Those systems rely on the automatic determining and classifying of students' errors and providing relevant error messages to correct students' misconceptions. What they require from misconception studies is different.The frequency of misconceptions does not matter for automatic tutors because they do not have the time restrictions of human teachers; automatic tutors can address even rare misconceptions to help more students. However, developing intelligent tutoring systems requires fine-grained lists of misconceptions regarding a particular topic, which is in line with the results of this study.

For further research, we plan to analyze the collected answers according to the information they contain: we plan to identify atomic information pieces that should be present in the answer to each question and determine the pieces that were often omitted. We also plan to use these misconceptions to enhance automatic tutors for introductory programming courses with advanced feedback for students. Common misconceptions can also be used to fine-tune adaptive exercises in automatic tutors to make sure each of them was verified during the exercise.

Author Contributions: Conceptualization, O.S.; methodology, O.S.; validation, M.D.; formal analysis, O.S. and M.D.; resources, O.S.; data curation, M.D.; writing—original draft preparation, O.S.; writing—review and editing, O.S.; visualization, M.D.; funding acquisition, O.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research has received no funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Ethical review and approval is not required, because this study involved the analysis of data being based on voluntary participation and having been properly anonymized. The research presents no risk of harm to subjects.

Data Availability Statement: The data is available from the corresponding author upon reasonable request. The data consists of free-text answers, which are not in English language.

Acknowledgments: We thank Fomichev Eugeny for helping us analyze conflicting labeling.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CS Computer science
IDE Integrated Developer Environment
SE Software engineering

Appendix A. Misconceptions and Responses Labeling

Table A1. List of detected misconceptions.

Label	Misconception	Count of Students
SEQ1	Failing to take into account the end of a block: The student thinks that a block or function has not been completed, although it has ended.	7
SEQ2	Action, which is placed after a sequence, is executed inside the sequence: Inside the block, the student executes an action that is placed after the block.	3
SEQ3	Actions are executed in the wrong order: The student executes actions inside a block in a different order than they are placed in the sequence.	1
SEL1	A "then" branch is always executed: The student thinks that a selection-statement branch can be executed without evaluating its condition first.	3
SEL2	Starting a "then" branch after a false condition: The student tries to start a selection-statement branch even though its condition is false.	7
SEL3	Conditions of a selection statement are checked in the wrong order: The student thinks that, in a multi-branch selection statement, conditions can be checked not in the order they are placed in the statement	9
SEL4	Not considering the truth of a control condition as the cause for the execution of its branch: When asked why a branch of the selection statement has started, the student does not mention the value of its control condition as the reason for its execution.	10
SEL5	Checking a condition after executing a branch: The student thinks that a condition can be evaluated after the corresponding branch of the selection statement is completed.	5
SEL6	Two or more branches of one selection statement are executed: The student thinks that more than one branch can be executed during one execution of a selection statement.	6

Table A1. *Cont.*

Label	Misconception	Count of Students
SEL7	Action, which is placed after the selection, is considered belonging to its branch: Inside a selection-statement branch, the student executes an action that is placed after the selection statement.	6
SEL8	Exiting a selection without executing a branch or checking all the conditions: The student thinks that a selection statement can finish before checking all its conditions and/or without executing a branch whose condition is true.	2
SEL9	Multiple checks of the same condition: The student thinks that in a selection statement, a branch condition can be evaluated several times.	4
SEL10	Executing a wrong branch after checking a condition: The student thinks that a different branch can be executed after a selection-statement condition evaluates to true.	3
LOOP1	Missing first condition test in a precondition loop: The student thinks that the first iteration of a precondition loop can begin without evaluating the loop condition first.	4
LOOP2	Continuing a loop without checking its condition: The student thinks that the next iteration of a loop can begin without evaluating the loop condition after the end of the previous iteration.	4
LOOP3	Continuing a loop after its control condition evaluates to false: The student thinks that a loop iteration can begin if the loop condition is false (for "while", "do-while" and "for" loops).	6
LOOP4	Action, which is placed after a loop, is considered belonging to the loop: Inside the body of a loop, the student executes an action that is placed after the loop.	6
LOOP5	Skipping an iteration when the loop condition is true: The student thinks that the computer can proceed to the next action of the loop (e.g., checking the condition again) without executing the loop's iteration.	5
LOOP6	A "for" loop is initialized after its iteration: The student thinks that the initialization step of a "for" loop can be executed after the iteration finishes.	8
LOOP7	Skipping the update step of a "for" or "foreach" loop: The student thinks that the update step of a "for" or "foreach" can be omitted after an iteration finishes.	7
LOOP8	Skipping the initialization of a "for" loop: The student thinks that the initialization step of a "for" loop should not be executed when the loop starts.	5
RECsl	Perceiving a selection statement as iterative: The student thinks that a selection statement can have several iterations during a single execution of the statement.	1
NOT1sl	Calling the selection statement a loop: When explaining the behavior of the selection statement, the student calls it a loop.	8
NOT2pl	Calling the program a loop: When explaining the behavior of the program, the student calls it a loop.	3

Table A2. Examples of labeling students answers.

Misconception Assigned	Question Description	Answer Template	Correct Answer	Student's Answer
SEL1 (q. #139)	A loop having a selection with the target act iterates two times.	The act "well_done" occurs exactly twice in the trace, because...	both times when the selection statement was executed in the loop, its condition evaluated to true.	the action must be performed once in each iteration, so there are two iterations.
SEL4 (q. #8)	A branch's condition has been evaluated to true, and the branch started	The branch should begin here because...	its condition evaluates to true.	it stands after checking the condition.
SEL5 (q. #31)	A branch's condition has been evaluated to false and evaluated again.	The condition "now_green" is wrong in this place, because ...	the selection should check the conditions one by one.	the "else" branch should have been executed first, and only then the condition should be checked.
SEL9 (q. #30)	A branch's condition has been evaluated to false and is evaluated again.	The condition "now_red" is wrong in this place, because...	the selection should check its condition only once.	it must appear before the previous condition.
LOOP4 (q. #168)	Action "go" stands after the end of a "while" loop; the loop tries to end without evaluating its condition.	The end of the loop is wrong in this place, because...	the condition of the loop must be evaluated first.	it should end after the "go" act.
LOOP6 (q. #178)	A "for" loop's iteration begins without checking its condition.	The iteration should not begin here because ...	the condition of the loop has not been evaluated.	2nd initialization was not performed.

References

1. Luxton-Reilly, A.; Simon; Albluwi, I.; Becker, B.A.; Giannakos, M.; Kumar, A.N.; Ott, L.; Paterson, J.; Scott, M.J.; Sheard, J.; et al. Introductory Programming: A Systematic Literature Review. In Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2018 Companion, Larnaca, Cyprus, 2–4 July 2018; pp. 55–106. [CrossRef]
2. Shi, N. Improving Undergraduate Novice Programmer Comprehension through Case-Based Teaching with Roles of Variables to Provide Scaffolding. *Information* **2021**, *12*, 424. [CrossRef]
3. Murphy, L.; Fitzgerald, S.; Lister, R.; McCauley, R. Ability to 'explain in Plain English' Linked to Proficiency in Computer-Based Programming. In Proceedings of the Ninth Annual International Conference on International Computing Education Research, ICER '12, Auckland, New Zealand, 9–11 September 2012; pp. 111–118. [CrossRef]
4. Pea, R.D. Language-Independent Conceptual "Bugs" in Novice Programming. *J. Educ. Comput. Res.* **1986**, *2*, 25–36. [CrossRef]
5. Sorva, J. Visual Program Simulation in Introductory Programming Education. Ph.D. Thesis, Aalto University, Espoo, Finland, 2012.
6. Sekiya, T.; Yamaguchi, K. Tracing Quiz Set to Identify Novices' Programming Misconceptions. In Proceedings of the 13th Koli Calling International Conference on Computing Education Research, Koli Calling '13, Koli, Finland, 14–17 November 2013; pp. 87–95. [CrossRef]
7. Grover, S.; Basu, S. Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE '17, Seattle, DC, USA, 8–11 March 2017; pp. 267–272. [CrossRef]
8. Qian, Y.; Lehman, J. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* **2017**, *18*, 1. [CrossRef]
9. Kennedy, C.; Lawson, A.; Feaster, Y.; Kraemer, E. Misconception-Based Peer Feedback: A Pedagogical Technique for Reducing Misconceptions. In Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '20, Trondheim, Norway, 15–19 June 2020; pp. 166–172. [CrossRef]
10. McCall, D.; Kölling, M. Meaningful categorisation of novice programmer errors. In Proceedings of the 2014 IEEE Frontiers in Education Conference (FIE), Madrid, Spain, 22–25 October 2014; pp. 1–8. [CrossRef]
11. McCall, D.; Kölling, M. A new look at novice programmer errors. *ACM Trans. Comput. Educ.* **2019**, *19*, 38. [CrossRef]
12. Kaczmarczyk, L.C.; Petrick, E.R.; East, J.P.; Herman, G.L. Identifying Student Misconceptions of Programming. In Proceedings of the 41st ACM Technical Symposium on Computer Science Education, SIGCSE '10, Milwaukee, WI, USA, 10–13 March 2010; pp. 107–111. [CrossRef]

13. Swidan, A.; Hermans, F.; Smit, M. Programming Misconceptions for School Students. In Proceedings of the 2018 ACM Conference on International Computing Education Research, ICER '18, Espoo, Finland, 13–15 August 2018; pp. 151–159. [CrossRef]
14. Sirkiä, T.; Sorva, J. Exploring programming misconceptions: An analysis of student mistakes in visual program simulation exercises. In Proceedings of the 12th Koli Calling International Conference on Computing Education Research, Koli Calling 2012, Koli Calling '12, Koli, Finland, 15–18 November 2012; pp. 19–28. [CrossRef]
15. Albrecht, E.; Grabowski, J. Sometimes It's Just Sloppiness—Studying Students' Programming Errors and Misconceptions. In Proceedings of the 51st ACM Technical Symposium on Computer Science Education, Portland, OR, USA, 11–14 March 2020; pp. 340–345. [CrossRef]
16. Kumar, A.N. Generation of Demand Feedback in Intelligent Tutors for Programming. In *Advances in Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 444–448. [CrossRef]
17. Sychev, O.; Penskoy, N.; Anikin, A.; Denisov, M.; Prokudin, A. Improving comprehension: Intelligent tutoring system explaining the domain rules when students break them. *Educ. Sci.* **2021**, *11*, 719. [CrossRef]
18. Mitrovic, A.; Koedinger, K.R.; Martin, B. A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling. In *User Modeling 2003*; Brusilovsky, P., Corbett, A., de Rosis, F., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 313–322. [CrossRef]
19. Pamplona, S.; Medinilla, N.; Flores, P. Exploring Misconceptions of Operating Systems in an Online Course. In Proceedings of the 13th Koli Calling International Conference on Computing Education Research, Koli Calling '13, Koli, Finland, 14–17 November 2013; pp. 77–86. [CrossRef]
20. Lieber, T.; Brandt, J.R.; Miller, R.C. Addressing Misconceptions about Code with Always-on Programming Visualizations. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '14, Toronto, ON, Canada, 26 April–1 May 2014; pp. 2481–2490. [CrossRef]
21. Miedema, D.; Aivaloglou, E.; Fletcher, G. Exploring the Prevalence of SQL Misconceptions: A Study Design. In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research, Joensuu, Finland, 18–21 November 2021*; Association for Computing Machinery: New York, NY, USA, 2021. [CrossRef]
22. Kunkle, W.M.; Allen, R.B. The Impact of Different Teaching Approaches and Languages on Student Learning of Introductory Programming Concepts. *ACM Trans. Comput. Educ.* **2016**, *16*, 3. [CrossRef]
23. Zehra, S.; Ramanathan, A.; Zhang, L.Y.; Zingaro, D. Student Misconceptions of Dynamic Programming. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education, SIGCSE '18, Baltimore, ML, USA, 21–24 February 2018; pp. 556–561. [CrossRef]
24. Herman, G.L.; Kaczmarczyk, L.; Loui, M.C.; Zilles, C. Proof by Incomplete Enumeration and Other Logical Misconceptions. In Proceedings of the Fourth International Workshop on Computing Education Research, ICER '08, Sydney, Australia, 6–7 September 2008; pp. 59–70. [CrossRef]
25. Caceffo, R.; Wolfman, S.; Booth, K.S.; Azevedo, R. Developing a Computer Science Concept Inventory for Introductory Programming. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education, SIGCSE '16, Memphis, TN, USA, 2–5 March 2016; pp. 364–369. [CrossRef]
26. Kelter, R.; Kramer, M.; Brinda, T. Statistical Frequency-Analysis of Misconceptions In Object-Oriented-Programming: Regularized PCR Models for Frequency Analysis across OOP Concepts and Related Factors. In Proceedings of the 18th Koli Calling International Conference on Computing Education Research, Koli Calling '18, Koli, Finland, 22–25 November 2018. [CrossRef]
27. Danielsiek, H.; Paul, W.; Vahrenhold, J. Detecting and Understanding Students' Misconceptions Related to Algorithms and Data Structures. In Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, SIGCSE '12, Raleigh, NC, USA, 29 February–3 March 2012; pp. 21–26. [CrossRef]
28. Gusukuma, L.; Bart, A.C.; Kafura, D.; Ernst, J. Misconception-Driven Feedback: Results from an Experimental Study. In Proceedings of the 2018 ACM Conference on International Computing Education Research, ICER '18, Espoo, Finland, 13–15 August 2018; pp. 160–168. [CrossRef]
29. Velázquez-Iturbide, J.A. Students' Misconceptions of Optimization Algorithms. In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '19, Aberdeen, UK, 15–17 July 2019; pp. 464–470. [CrossRef]
30. Kurvinen, E.; Hellgren, N.; Kaila, E.; Laakso, M.J.; Salakoski, T. Programming Misconceptions in an Introductory Level Programming Course Exam. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '16, Arequipa, Peru, 11–13 July 2016; pp. 308–313. [CrossRef]
31. Sanders, K.; Thomas, L. Checklists for Grading Object-Oriented CS1 Programs: Concepts and Misconceptions. *ACM SIGCSE Bull.* **2007**, *39*, 166–170. [CrossRef]
32. Ardimento, P.; Bernardi, M.L.; Cimitile, M.; Redavid, D.; Ferilli, S. Understanding Coding Behavior: An Incremental Process Mining Approach. *Electronics* **2022**, *11*, 389. [CrossRef]
33. Sychev, O.; Denisov, M.; Terekhov, G. How It Works: Algorithms—A Tool for Developing an Understanding of Control Structures. In Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '21, Virtual Event Germany, 26 June–1 July 2021; pp. 621–622. [CrossRef]
34. Fereday, J.; Muir-Cochrane, E. Demonstrating Rigor Using Thematic Analysis: A Hybrid Approach of Inductive and Deductive Coding and Theme Development. *Int. J. Qual. Methods* **2006**, *5*, 80–92. [CrossRef]

35. Cohen, J. A Coefficient of Agreement for Nominal Scales. *Educ. Psychol. Meas.* **1960**, *20*, 37–46. [CrossRef]
36. Kumar, A.; Dancik, G. A tutor for counter-controlled loop concepts and its evaluation. In Proceedings of the 33rd Annual Frontiers in Education, Westminster, CO, USA, 5–8 November 2003; Volume 1, p. T3C-7. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

computers

MDPI

Review

Predicting Student Performance in Introductory Programming Courses

João P. J. Pires [1], Fernanda Brito Correia [1], Anabela Gomes [1,2], Ana Rosa Borges [1] and Jorge Bernardino [1,2,*]

[1] Coimbra Institute of Engineering—ISEC, Polytechnic University of Coimbra, Rua da Misericórdia, Lagar dos Cortiços, S. Martinho do Bispo, 3045-093 Coimbra, Portugal; a21280231@isec.pt (J.P.J.P.); fernanda@isec.pt (F.B.C.); anabela@isec.pt (A.G.); arborges@isec.pt (A.R.B.)

[2] Centre for Informatics and Systems of the University of Coimbra, Polo II, Pinhal de Marrocos, 3030-290 Coimbra, Portugal

* Correspondence: jorge@isec.pt

Abstract: The importance of accurately predicting student performance in education, especially in the challenging curricular unit of Introductory Programming, cannot be overstated. As institutions struggle with high failure rates and look for solutions to improve the learning experience, the need for effective prediction methods becomes critical. This study aims to conduct a systematic review of the literature on methods for predicting student performance in higher education, specifically in Introductory Programming, focusing on machine learning algorithms. Through this study, we not only present different applicable algorithms but also evaluate their performance, using identified metrics and considering the applicability in the educational context, specifically in higher education and in Introductory Programming. The results obtained through this study allowed us to identify trends in the literature, such as which machine learning algorithms were most applied in the context of predicting students' performance in Introductory Programming in higher education, as well as which evaluation metrics and datasets are usually used.

Keywords: higher education; introductory programming; machine learning; student's performance prediction

1. Introduction

Introductory Programming (IP) continues to be a challenge for higher education (HE) students, with high failure rates [1,2]. Different studies have attempted to analyze and identify possible difficulties associated with learning to program, including aspects such as the complex syntax of programming languages [3] and conceptual knowledge related to how the language itself works [4]. The difficulties reported by students and teachers in these studies included understanding programming structures; dividing functionality into procedures; finding bugs in one's own program; and understanding variables, functions, loop structures, recursion, arrays, pointers and references, structured data types, abstract data types, and input/output handling. Several reasons have been identified to explain these difficulties [5], namely (i) teaching methods; (ii) study methods; (iii) students' abilities and attitudes; (iv) the nature of programming; and (v) psychological effects. This has motivated research into the factors that influence the learning process and how to overcome them [6]. It is possible to find references in the literature to several other factors that influence the learning experience of programming, such as the ability to think abstractly [7], previous experience of programming [8,9], the effect of teaching approaches that take into account students' motivational strategies and learning preferences and styles [10,11], mathematics skills [12], or problem-solving skills [13].

The significant problem of student failure underlines the need for effective interventions, as outlined in several studies that have been carried out with the aim of analyzing and providing possible solutions to this problem [14]. One of the solutions often analyzed

by researchers focuses on analyzing and improving the usual methods of teaching programming in HE [15,16]. Numerous tools and strategies were also developed to contribute to the solution of this problem, and although each one has its merits, none of them has yet completely solved the problem [16]. Among the various strategies studied, the use of methods to predict student performance stands out as a possible solution [17,18]. While several metrics have been used to evaluate student performance, Quille and Bergin [19] introduced the PreSS metric, which focuses specifically on predicting student success in IP courses. The authors propose a relevant approach by developing a prediction model called PreSS, which has been validated and refined over 13 years. The paper discusses the challenges of predicting student success in CS1, such as high attrition rates and the difficulty of identifying students at risk. The authors found that PreSS can predict student success with 71% accuracy and that it can be used to develop interventions to improve student performance. Although these results are positive, 71% is still not ideal and highlights the difficulties that remain in finding solutions.

In this study, student performance is defined as their proficiency in IP, measured through a combination of grades in formal assessments (such as exams and assignments), participation in practical activities, and performance on coding tasks. This definition is consistent with the literature, which emphasizes the importance of assessing not only theoretical knowledge, but also students' practical programming skills [20].

This study delves into this area with the aim of providing a comprehensive and systematic review of the existing literature on the methodologies used to predict student performance in HE, particularly in programming, focusing on machine learning (ML) as a prediction method. It emphasizes the application of these ML algorithms, identifying which algorithms have been used and their performance, and analyzing which evaluation metrics and datasets have been used. ML algorithms are extremely useful techniques that are commonly used to develop models for predicting student performance [21].

Data mining is the extraction of important and potentially useful information or trends from data. Educational data mining (EDM) is the application of data mining techniques in the educational context with educational data. ML is one of the data mining techniques that is widely used in the field of EDM [22,23]. The emphasis on ML in this study is due to its ability to handle large datasets, discover complex patterns, and make predictions with high accuracy. There are other techniques, such as traditional statistical methods, that can also be used in the process of studying student performance. However, although statistical methods are well established and easy to interpret, they are not able to capture complex patterns in the data as effectively as ML techniques. On the other hand, ML can offer greater accuracy and adaptability, with the slight disadvantage of requiring more computational resources and specialized knowledge to implement.

The area of IP has a profound significance for aspiring computer engineers, serving as the fundamental basis on which their entire educational and professional path is built. Aptitude in programming is not just a prerequisite but an essential skill that forms the core of a future engineer's capabilities. This skill lays the foundations for understanding complex algorithms, problem-solving methodologies, and the intrinsic aspects of software development [24]. This systematic literature review aims to address the gap in understanding which ML algorithms should be applied and which are most effective in predicting student performance in IP, by analyzing their performance and examining the evaluation metrics and datasets used. In addition to other similar studies [25], this paper contributes to the field by providing a clear overview of the current state of research and identifying the most promising approaches and areas for future research, which will help in decision-making processes for the further development of the field and the search for solutions.

The remainder of this paper is structured as follows. Section 2 describes the methodology used to conduct this systematic review, highlights the methods identified, and presents the algorithms specified in related works. Section 3 identifies the studies found that address the problem under study. Section 4 summarizes the results and Section 5 presents the answers to the research questions. Section 6 presents the main threats to the validity of

this study and Section 7 identifies future research directions. Finally, Section 8 presents the conclusions.

2. Methodology

This study consists of a review of the current literature to contribute to identifying current trends and major difficulties and to present a comparative analysis of some of the existing studies in the context of predicting student performance in IP courses in HE, using the Kitchenham methodology [26]. This methodology describes some guidelines for developing systematic reviews. The following steps were applied: (a) definition of research questions; (b) selection of data sources; (c) definition of the search string; (d) definition of preconditions; (e) definition of inclusion and exclusion criteria; and (f) data extraction and analysis.

Research Questions—For this study, three research questions (RQ) were defined as follows:

RQ1: What were the most-used ML algorithms proposed by the researchers for predicting students' performance in HE in IP?

RQ2: Which datasets were used, and which evaluation metrics were considered most appropriate for measuring the performance of predictive models in HE in IP?

RQ3: Which ML algorithms seem to better predict students' performance in HE in IP?

Data Sources—The papers analyzed in this study were the result of searches in the search engine Google Scholar.

Search String—The search string used was as follows: ("introduction to programming" OR "programming learning" OR "learning programming" OR "learn to program" OR CS1) AND (novice OR freshman OR beginner) AND ("higher education") AND (aptitude OR performance OR skill OR ability OR proficiency OR facility OR talent) AND ("predict") AND ("machine learning") AND—"video" AND—"ChatGPT".

Preconditions—Two conditions were met as follows: publications from 2015 to 2024 and written in English.

Inclusion and Exclusion Criteria—To select interesting studies, the following inclusion criteria were defined: (i) responses to the research questions established. The exclusion criteria were as follows: (e1) the paper does not refer to IP; (e2) the paper does not focus on HE; (e3) the paper does not refer to the use of ML; (e4) the paper is not available; and (e5) studies others than papers were not considered (like theses and books).

Data Extraction—Initially, the search string was built to find papers related to this problem. Next, some preconditions were appointed for selecting the papers, and then the inclusion and exclusion criteria were defined. Once this was finished, the phase of collecting papers began. Based on the search string and the preconditions, 209 records were identified, and by checking the title, 28 papers ended up remaining. Next, the abstracts of these selected papers were analyzed and checked for their compliance with the proposed research questions and defined criteria for the remaining 18 papers. These 18 papers were read in full, taking into account the research questions and the defined criteria. The search ended with 11 papers to be analyzed. Figure 1 shows the full literature review process steps.

The aim of this study is to explore the different methods that shape the scenario of predicting students' performance in HE institutions, with particular emphasis on the field of IP and focusing on ML algorithms.

The following section gives an overview of the types of datasets used in the state of the art in this specific field and the evaluation metrics. The aim of this analysis is to understand existing strategies, inspire future innovative approaches, and improve forecast methods' accuracy and usefulness.

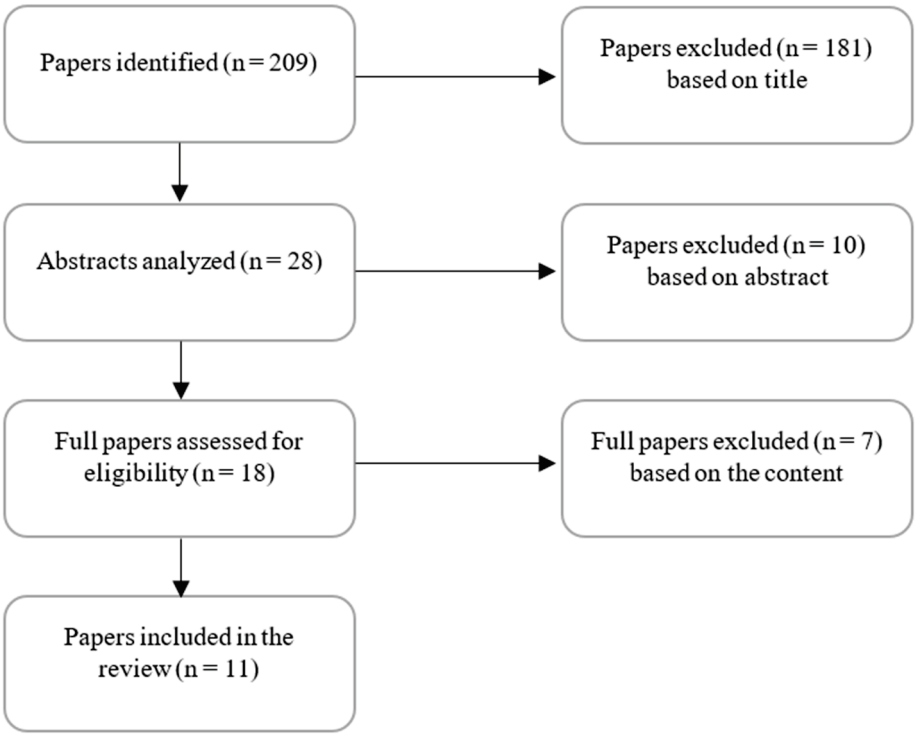


Figure 1. Literature review process steps.

2.1. Datasets

Proper data collection plays a critical role in building a solid foundation for research. This section examines the different datasets used in the papers presented above. Table 1 presents the datasets used in the analyzed papers.

Table 1. Datasets used in the analyzed papers.

Datasets	Papers
Questionnaires	[27–29]
Academic records	[27,28,30–36]
Personal data	[30,32,34,35,37]

In the papers, data were collected from different datasets:

- Questionnaires

Questionnaires consist of a set of questions that, in the context of the subject under study and, as analyzed in the related works, may or may not include questions directly related to programming. The answers to these questionnaires make it possible to extract important information about the students and their theoretical and practical skills.

- Academic Records

Class attendance, course averages, test scores, practical work, and other assessment components are perhaps the most important data for correctly predicting students' performance. In the analyzed papers, using academic records from previous years to train the model and then using academic records from the current year to predict performance was widespread.

- Personal Data

Another type of data commonly used were personal data. These data include information such as age, gender, nationality, and programming experience. These data can identify some patterns that may be associated with academic performance in IP.

2.2. Evaluation Metrics

The correct choice and interpretation of evaluation metrics is essential when analyzing the performance of a model. This section consists of a description of the metrics identified in the related works, which provide information about a model's predictive ability, and are used here to predict students' academic performance in IP in HE. The most important metrics that have been identified are the following:

- Accuracy-is the most common evaluation metric. It works as an indicator of the model's overall performance by determining the percentage of correctly classified instances [38].
- Precision-is an extremely useful metric, in situations where false positives are more important than false negatives because it allows us to evaluate the model and determine what percentage of instances classified as positive are positive [38].
- Recall-is a metric that can be very useful when false negatives are more important than false positives, as it allows us to calculate the percentage of true positive instances that have been correctly classified [38].
- F1-Score-is created when the precision and recall metrics are both critical. This metric combines precision and recall through a harmonic mean, so by maximizing the value of the F1-Score, we simultaneously maximize the precision and recall [38].
- Specificity-is the metric that allows calculating the percentage of truly negative instances that have been classified as negative; in other words, it assesses the ability of the model to correctly identify negative cases [38].
- Sensitivity-is the metric that allows calculating the percentage of truly positive instances that have been classified as positive [38].
- Root Mean Square Error (RMSE)-is commonly used to evaluate model performance. This metric measures the average squared difference between the values predicted by the model and the actual values. The lower the RMSE value, the better the model's performance [38].

3. Related Work

The following provides a detailed overview of the different methodologies identified in this systematic review, highlighting the variety of datasets, algorithms, and evaluation metrics that represent the state of the art in this specific field.

Over the last few years, many studies have been conducted on predicting students' performance in HE, with a particular focus on IP. This section presents the analysis of the papers obtained through the methodology described.

The study carried out in [27] presents a ML model that attempts to predict students at risk of poor performance. The algorithm used in this research was Gradient Boosting, which was trained on students' data obtained from questionnaires and class works conducted during the first two weeks of the semester, with a total of 471 students. The model's expected output is a number between 0 and 1, representing the probability that a given student is at risk of performing poorly.

The study presents a promising approach to predicting student performance with a relatively large database, using a good variation of assessment metrics, and achieving significantly positive accuracy and specificity results. However, there are some significant limitations, such as low recall and precision, which indicate that the model may not adequately identify all students at risk. Improvements can be made by extending the data collection period, increasing the diversity of the sample, and integrating more temporal and contextual data to increase the robustness and generalizability of the model. Table 2 is a summary of the datasets and the algorithm used, along with the metric values obtained.

The work proposed in [28] applies four ML algorithms to create predictive models capable of predicting possible students at risk of failing an IP course. The algorithms used were Decision Tree, K-Nearest Neighbour (KNN), Naïve Bayes, and Support Vector Machine (SVM). The data used to train the models were obtained from an IP course where features were the average entry grade in the course, the result of a questionnaire, practical

assessments, and other class assessments of the full semester. All this information together relates to the four semesters of the academic years 2018/2019 and 2019/2020, with a total of 244 records. The accuracy, precision, and recall metrics were used to evaluate the results of the models. Both Decision Tree and SVM achieved the highest performance, but Decision Tree was twice as fast as SVM at running the model.

Table 2. Datasets, algorithms, and metric values used in [27].

Dataset	Algorithm	Metric	Value
Academic records Questionnaires	Gradient Boosting	Accuracy	82%
		Recall	57%
		Precision	56%
		Specificity	88%

The study shows impressive results with high levels of precision, recall, and accuracy in various algorithms. The sample size seems to be average and realistic. The reason for such positive results may be associated with the fact that the dataset uses information from the entire semester to predict the final grade, thus having a greater number of assessment components already performed, making it easier to predict the final grade. This study does not seem to be recommended if an early prediction is needed to help at-risk students, as it requires the full semester's information. Table 3 is a summary of the datasets and algorithms used, along with the metric values obtained.

Table 3. Datasets, algorithms, and metric values used in [28].

Dataset	Algorithm	Metric	Value
Academic records Questionnaires	Decision Tree	Accuracy	99.18%
		Precision	100%
		Recall	98.7%
	KNN	Accuracy	98.36%
		Precision	99.3%
		Recall	100%
	Naïve Bayes	Accuracy	96.72%
		Precision	97.4%
		Recall	97.4%
	SVM	Accuracy	99.18%
		Precision	100%
		Recall	98.7%

ML techniques are also used to identify high and low performers in the study carried out in [30]. The data considered for the models are from one semester of an IP course at the University of Helsinki with a total of 86 students. Age, gender, grade point average, and programming experience are some information available in the data used. The intended results were the prediction of the final grade of the course. The algorithms used were ADTree, Bayesian Network, Conjunctive Rule, Decision Stump, Decision Table, J48, Naïve Bayes, and PART. The authors tried to predict the students' exam results to understand which algorithm was the best between the models with the best accuracy, Random Forest, and Decision Stump. Random Forest obtained a 90% accuracy and Decision Stump obtained 83%.

Overall, the study showed positive results for practically all the algorithms used. However, the dataset used is very limited, which introduces potential bias due to its reduced diversity. In addition, the fact that the authors relied solely on accuracy as the only evaluation metric may have limited their understanding of the model's performance. In the context of the problem, the use of other metrics, such as specificity and sensitivity, can be very useful in understanding whether or not you are dealing with a case where the

models are unable to generalize. Table 4 gives a summary of the datasets and algorithms used, along with the metric values obtained.

Table 4. Datasets, algorithms, and metric values used in [30].

Dataset	Algorithm	Metric	Value
Academic records Personal data	ADTree	Accuracy	83%
	Bayesian Network	Accuracy	76%
	Conjunctive Rule	Accuracy	86%
	Decision Stump	Accuracy	90%
	Decision Table	Accuracy	76%
	J48	Accuracy	89%
	Naïve Bayes	Accuracy	86%
	PART	Accuracy	76%
	Random Forest	Accuracy	90%

The work carried out in [31] aimed to identify students with learning difficulties at an earlier stage to give them more attention in the form of extra class hours. Identifying students with difficulties was performed by predicting the probability of completing the course using an ML model. Data from 181 students, from the previous year, were used as training data for the model. These data contain the grades obtained in three teaching activities that were carried out every year during the first five weeks of the course: seminar, quiz, and practical work. The problem presented was a binary classification, for which the authors decided to use the Logistic Regression algorithm, where the result was "yes" (1) or "no" (0) to know if the student would pass or fail. The precision, recall, and F1-Score metrics were used to check the quality of the model.

The study presents a relatively different approach to identifying students with learning difficulties early. The high recall rate is a major strength, ensuring that most students who pass are identified. However, the moderate precision may indicate difficulties in predicting students who are unlikely to pass. The sample size is relatively small and may therefore represent a conditioned sample. Possible improvements would be to expand the sample and explore different algorithms or combinations of algorithms to improve model precision. A summary of the datasets and the algorithm used, together with the metric values obtained, is presented in Table 5.

Table 5. Datasets, algorithms, and metric values used in [31].

Dataset	Algorithm	Metric	Value
Academic records	Logistic Regression	Precision	67%
		Recall	92%
		F1-Score	77%

Three ML algorithms to predict potential student grades at an early stage in the semester were used in the study of [32]. The outcome to be expected was a value from 0 to 100 that indicates the student's final grade. With this study, the authors aim to help students identify their likely grades and, if necessary, modify their academic behavior. The data used in the study were collected at the University of Buraimi. The dataset consists of information on 50 students and includes attributes such as gender, first-test grade, and attendance, among others. The performance of the algorithms used was evaluated, based on the accuracy metric and F1-Score. The three algorithms were Decision Table, J48, and Naïve Bayes.

The study presents good results for the three algorithms used. However, the extremely small sample size may result in the models not being able to generalize, which is the main limitation of this study as ML algorithms trained on small datasets may not achieve satisfactory results. It was recommended to use metrics such as specificity or sensitivity to understand whether this generalization capacity exists. The exploration and addition of new data would add further relevance to the study results. Table 6 is a summary of the datasets and algorithms used, along with the metric values obtained.

Table 6. Datasets, algorithms, and metric values used in [32].

Dataset	Algorithm	Metric	Value
Academic records Personal data	Decision Table	Accuracy	83%
		F1-Score	79%
	J48	Accuracy	88%
		F1-Score	88%
	Naïve Bayes	Accuracy	84%
		F1-Score	83%

The study of [33] presents a model developed to try to identify in advance students at risk of failing an IP course by predicting the final exam grade. It used data from assignments and class exercises carried out in the first two weeks of the semester, and these data corresponded to the years 2016, 2017, and 2018. The developed model used the Random Forest algorithm, where the 2016 data with a total of 93 records were used for training, the 2017 data with 94 records were used for validation, and the 2018 data with 102 records were used for testing. The results obtained for Random Forest show a low overall accuracy, but a good sensitivity in predicting students that are at risk. According to the authors, the results obtained were not the best due to the lack of balance in the dataset used.

The authors acknowledge the limitations of the study itself, pointing to the inability of the model to learn on an unbalanced dataset. However, it should be noted that the model appears to have a good ability to identify students who are at risk. Nevertheless, it later fails to predict students who are not at risk. A limitation of the study is the small size of the sample used to train the model. Table 7 is a summary of the datasets and the algorithm used, together with the metric values obtained.

Table 7. Datasets, algorithms, and metric values used by [33].

Dataset	Algorithm	Metric	Value
Academic records	Random Forest	Accuracy	60%
		Sensitivity	77%

Different neural networks to identify at-risk students were tested in the work presented in [34]. The study used historical students' data over the years in IP courses in HE. These data contained personal information and records of different assessment grades given during the semester over the last 7 years. In total, the dataset contained information of 592 students. The authors tested 25 different network topologies and concluded that the Probabilistic Neural Network (PNN) obtained the best results.

The results of this study are very positive. The sample size is one of the largest compared to all the papers analyzed so far. Although in the end the authors only use the accuracy metric, the paper presents the corresponding confusion matrix of the neural network, which allows us to verify the good ability of the model to classify both students at risk and those who are not at risk. The dataset also presents a good balance of classes. This study shows the potential of neural networks. A summary of the datasets and the algorithms used, together with the metric value obtained, is given in Table 8.

Table 8. Datasets, algorithms, and metric values used by [34].

Dataset	Algorithm	Metric	Value
Academic records Personal data	PNN	Accuracy	91%

In [29], the study applied different algorithms for predicting students' performance in programming in HE. The data were obtained from the University of Madras, India, and include information about the students collected through a questionnaire. These data were collected as part of an introductory Python programming course in the first semester of the first year of the course. In total, the data contained records for 490 students. In this study, five different algorithms were tested and evaluated using the recall, precision, F1-Score, and accuracy metrics. The algorithms were the following: Decision Tree, KNN, Naïve Bayes, Random Forest, and SVM. Overall, Naïve Bayes obtained the best results.

The study presents a comprehensive and robust analysis using a variety of machine learning algorithms to predict student performance. The evaluation metrics used are relevant. The sample size also appears to be representative and complete. Considering that the study only uses data from a questionnaire, the results are very good. Table 9 summarizes the datasets and algorithms used, and the metric values obtained.

Table 9. Datasets, algorithms, and metric values used in [29].

Dataset	Algorithm	Metric	Value
Questionnaires	Decision Tree	Accuracy	85.10%
		Precision	86.57%
		Recall	84.89%
		F1-Score	85.72%
	KNN	Accuracy	84.28%
		Precision	85.82%
		Recall	83.46%
		F1-Score	84.62%
	Naïve Bayes	Accuracy	91.02%
		Precision	92.21%
		Recall	90%
		F1-Score	91.09%
	Random Forest	Accuracy	87.55%
		Precision	86.06%
		Recall	88.60%
		F1-Score	87.31%
	SVM	Accuracy	88.77%
		Precision	89.79%
		Recall	88%
		F1-Score	88.88%

The work carried out in [37] applies the J48, Multilayer Perceptron, Naïve Bayes, REPTree, and Sequential Minimal Optimization algorithms to predict students' academic performance in an IP course. The data for the models were collected from a C programming course at the University of Madras, India. The dataset contains 300 records with demographic information about the students. The results obtained were based on accuracy. Out of these algorithms, the Multilayer Perceptron showed the best performance.

Although this study concluded that the five models achieved high accuracy in predicting student performance, relying on accuracy as the sole evaluation metric may have a limited understanding of model performance. More metrics are needed to better under-

stand whether the results are good. Good accuracy does not always mean good prediction. Table 10 provides an overview of the datasets, algorithms, and metrics that were collected.

Table 10. Datasets, algorithms, and metric values used in [37].

Dataset	Algorithm	Metric	Value
Personal data	J48	Accuracy	92.03%
	Multilayer Perceptron	Accuracy	93.23%
	Naïve Bayes	Accuracy	84.46%
	REPTree	Accuracy	91.03%
	Sequential Minimal Optimization	Accuracy	90.03%

The study in [35] takes a different approach from the previous work. Here, a deep learning model was developed and used data, such as the student's identity, professional skills, and academic records, including the code developed by the IP courses in HE students. The main goal is to predict the student's performance in the final exam; it is a regression problem. The authors ran several tests to decide which architecture to use and concluded that a Deep Neural Network (DNN) with four fully connected layers was a good architecture. In this DNN, the input layer had 128 neurons, the second layer had 64, the third layer had 8, and the output layer had only 1. The activation function used was the Rectified Linear Unit (ReLU), and the evaluation metric selected was the RMSE. The results were 12.68 RMSE for the DNN. Finally, a comparison was made with six other algorithms (SVM, Bayesian Ridge, Random Forest, Extra Trees, Gradient Boosting, and Decision Tree), where the SVM stood out as having the best performance, although still below the DNN, with an RMSE of 14.07.

This study explores a new aspect of predicting student performance and shows the great potential of deep learning for the context. A negative point identified was the failure to clarify the size of the dataset used to train the models. The results are very promising and consistent with the literature, suggesting that the way forward could be through deep learning, as these architectures show constant improvements in performance. Table 11 provides a summary of the datasets and the algorithm used, together with the metric values obtained.

Table 11. Datasets, algorithms, and metric values used in [35].

Dataset	Algorithm	Metric	Value
Academic records Personal data	DNN	RMSE	12.68
	SVM	RMSE	14.07

Finally, in [36], the study applied the ID3 and J48 classification algorithms to analyze students' performance in an IP course. The data used to develop the models came from the first semester of an IP course that ran for three months, from September to November 2017, at Usmanu Danfodiyo University in Nigeria. The data consisted of 239 instances with information on class attendance, test grades, and completed assignments. The results of the models were evaluated based on four metrics: accuracy, precision, recall, and F1-Score. The results of both algorithms were similar, so to clarify what was the best one, the authors analyzed the number of instances that were correctly classified; J48 obtained 208 and ID3 obtained 204 out of 239 total instances, so the authors concluded that J48 slightly outperformed ID3.

This study achieved very positive results for both models. The metrics used are relevant and sufficient to understand the quality of the models' predictions, and the sample used is of an acceptable size. Improvements can be made by extending the dataset. A

summary of the datasets and algorithms used, together with the metric values obtained, is shown in Table 12.

Table 12. Datasets, algorithms, and metric values used in [36].

Dataset	Algorithm	Metric	Value
Academic records	ID3	Accuracy	85.4%
		Precision	84.8%
		Recall	86%
		F1-Score	85%
	J48	Accuracy	87%
		Precision	80%
		Recall	87%
		F1-Score	82.8%

4. Results

This section summarizes the results, identifying the main findings and trends found from the literature reviewed, the models and algorithms used, the most-used evaluation metrics and datasets, and the algorithms that gave the best results including the deep learning approach. The identification of patterns provides a more comprehensive overview of the characteristics studied that are most relevant in predicting students' academic success in IP in HE.

In addition to summarizing the key findings, this section highlights the contributions that promise to catalyze future progress in this important area.

4.1. Models

The application of ML models has emerged as a fundamental approach to obtain valuable details about students' performance in IP courses in HE. Table 13 lists the ML models used in the analyzed papers. Each model is presented along with the references for the specific papers in which they were implemented. In total, twenty different algorithms were used in all the analyzed papers. Some of these algorithms were used in more than one paper, and there were studies that applied and compared multiple models.

Table 13. Models used in analyzed papers.

Algorithms	Papers
ADTree	[30]
Bayesian Network	[30]
Conjunctive Rule	[30]
Decision Stump	[30]
Decision Table	[30,32]
Decision Tree	[28,29]
DNN	[35]
Gradient Boosting	[27]
ID3	[36]
J48	[30,32,36,37]
KNN	[28,29]
Logistic Regression	[31]
Multilayer Perceptron	[37]
Naïve Bayes	[28–30,32,37]
PART	[30]
PNN	[34]
Random Forest	[29,30,33]
REPTree	[37]
Sequential Minimal Optimization	[37]
SVM	[28,29,35]

4.2. Most-Used Algorithms

The presentation of the results made it possible to see some patterns and extract some important knowledge. It was possible to identify that the algorithms J48, Naïve Bayes, Random Forest, and SVM tend to be the most used in three or more papers. Other algorithms such as Decision Tree, Decision Table, and KNN are also used, although less frequently, in only two papers. However, although simpler algorithms such as Random Forest and Naïve Bayes are often used, it is important to be aware that more complex algorithms such as DNNs, even if they are used less frequently, can provide superior prediction performance. The paper of [35] that uses a DNN demonstrate this, as does the paper of [34] that uses a PNN on a big dataset and obtains some of the best results identified.

4.3. Most-Used Evaluation Metrics

It was also found that a wide range of evaluation metrics, including precision, recall, and F1-score, are extremely useful and used in this context. However, none of these metrics come close to the levels of use of the accuracy metric, which was used in nine of the eleven papers in the literature review. The predominant use of the accuracy metric can have a significant impact on the way predictive models are evaluated. This suggests that many researchers prefer a simple and straightforward metric, possibly due to its ease of interpretation. However, this may also lead to an underestimation of the limitations of accuracy, particularly in unbalanced datasets where the metric can be misleading.

4.4. Most-Used Dataset

Among the datasets identified, academic records proved to be the most frequently used data for predicting performance in IP in HE, being reported in nine of the eleven papers analyzed. This makes sense since past records of a student's grades allow many conclusions to be drawn about the level of a student's skills and competences.

4.5. Datasets Analysis

When analyzing the different papers selected, constant variations were identified in the types of datasets used (academic records, personal data, and questionnaires) as well as in the size of the dataset itself. While academic records are the most common data source in the reviewed studies, it is important to analyze whether other types of data, such as questionnaires and personal data, can contribute to the accuracy of predictions. More detailed analysis of the datasets is therefore needed to understand the extent to which these variations have an influence on the results.

Comparing the results of predictions based on different data sources, there is no clear evidence that one particular data source leads to better performance than the others. This can be analyzed in the studies carried out in [29,36,37], where each study uses a different data source: questionnaires in [29], academic records in [36], and personal data in [37]. In these three studies, the models were able to achieve good results, regardless of the variation in the type of data. However, it was found that these results can be just as good or even better when several data sources are used together, as can be seen in the studies of [28,34]. In these studies, the use of two types of data resulted in some of the best performances identified in the studies analyzed. This suggests that integrating different types of data can improve the ability to predict student performance. Further research should explore the potential benefits of integrating multiple data sources to improve predictive accuracy.

An analysis of the size of the datasets used to train the different models and the respective quality of the predictions obtained shows that the algorithms with the best performance were in most cases used with the largest datasets, as observed in the studies of [28,29,34,37], where the datasets always have more than 200 instances. This suggests that factors such as data quality may also play an important role in influencing predictive accuracy. In the studies analyzed, there were cases where small datasets achieved good performance [32], but with models trained on so little data, there may be an inability of the

model to generalize to new instances, related to the inability of the model to represent all the variability and details of the population.

4.6. Algorithms with Best Results

Among all the models observed, some of the ML algorithms that showed the best results were the SVM and the Decision Tree, which, in the paper of [28], using questionnaires and academic records datasets, achieved an accuracy of 99.18%. Also noteworthy were Naïve Bayes and J48, which always gave very positive results. Figure 2 shows the accuracy of the models used in the analyzed papers. It is important to note that Figure 2 only shows the cases of algorithms that have been used more than once so that there is a point of comparison and that were evaluated using the accuracy metric.

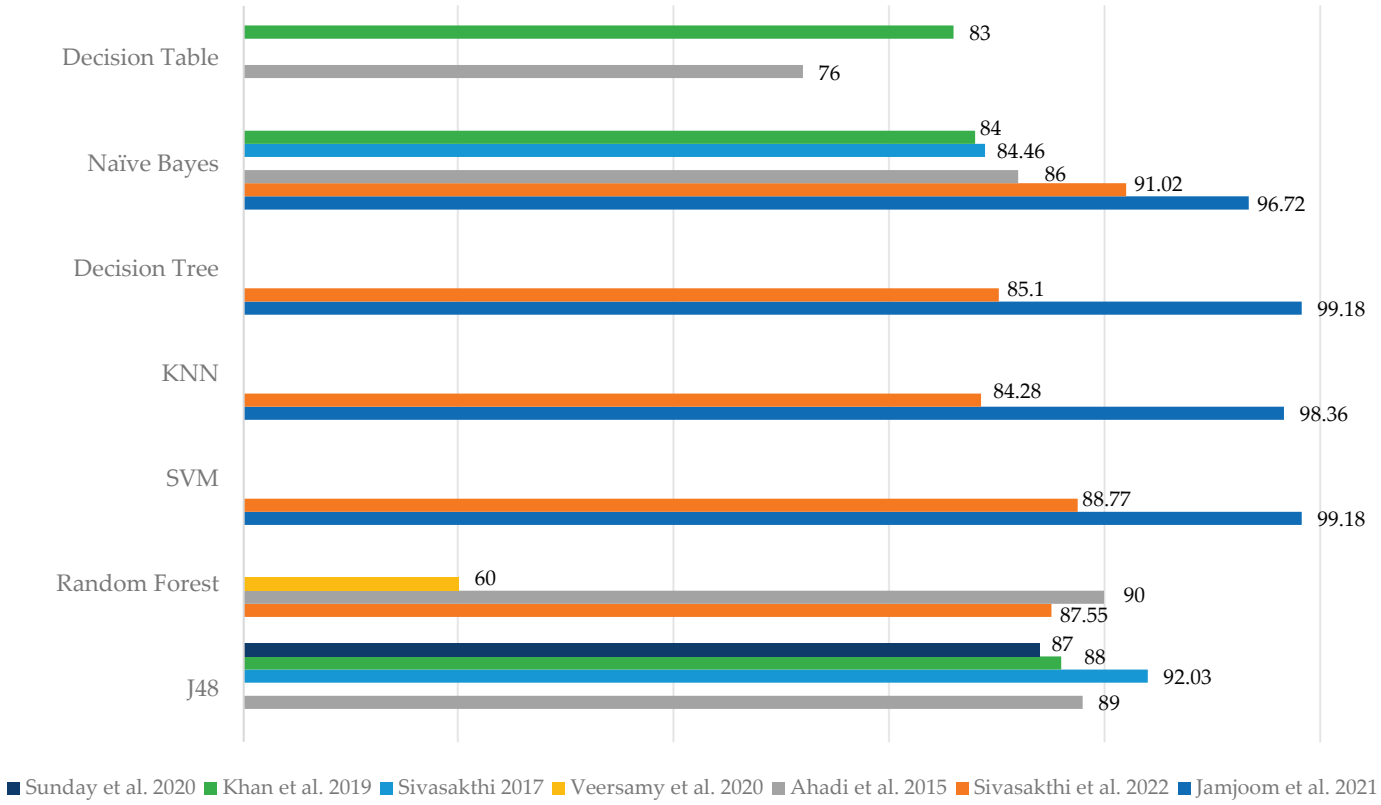


Figure 2. Accuracy of the most-used models. The references used are: Jamjoom et al. 2021 [28], Sivasakthi et al. 2022 [29], Ahadi et al. 2015 [30], Khan et al. 2019 [32], Veersamy et al. 2020 [33], Sunday et al. 2020 [36] and Sivasakthi 2017 [37].

Based on the models in Figure 2 and the corresponding accuracy results, an ANOVA test was performed to determine any significant statistical differences. This test was performed using the IBM SPSS Statistics tool. The ANOVA results show that the *p*-value between the groups was 0.395, indicating that there is no evidence of statistically significant differences between the group means. The F-statistic value confirms this fact, as it reaches a value of 1.137, indicating that the differences between the group means are not large enough to be considered statistically significant.

4.7. Deep Learning Approach

While traditional ML techniques are most commonly used to predict student performance, the application of deep learning methods is still in its early stages. The relative newness and complexity of deep learning may be factors contributing to its limited adoption. In the paper of [35], a DNN was used to predict student performance using personal data and academic records. The DNN performed better than six other algorithms, includ-

ing the widely used SVM. This suggests that DNNs, with their ability to model complex, non-linear relationships in data, have significant potential in this area. The conclusions of the paper of [35] provide a valuable direction for future research in this area.

4.8. Challenges and Gaps

This section highlights the major challenges and limitations identified in the literature review. The following difficulties have a direct impact on the quality of the study of predicting student performance in IP in HE:

- Data Quality Assurance

The foundation of any predictive model is the quality of the underlying data. Ensuring the relevance and accuracy of the data is critical to the success of a study [39]. Data quality assurance involves a careful process of data collection and pre-processing [40]. Given the complexity of educational data, which often include multiple sources and formats, a robust data assurance strategy is essential. Achieving balance in datasets is also a significant challenge, as imbalances can affect results and compromise the predictive ability of the model.

To address data quality concerns, we propose the implementation of two possible solutions: (i) Advanced data cleaning techniques: the implementation of sophisticated data cleaning methods, such as the detection of outliers (an observation that is very different from the others in the series or is inconsistent) and the handling of missing values (one or more missing instances in the data) through imputation techniques (to handle missing data by filling in or estimating the missing values instead of discarding the incomplete records), can improve data quality. (ii) Balancing techniques: applying techniques such as SMOTE (Synthetic Minority Over-sampling Technique) to resolve class imbalances and increase the predictive power and generalizability of the model.

- Selection of the Algorithms

The abundance of available ML algorithms presents a significant challenge in the selection process. While the diversity of options is beneficial, it is difficult to identify the most appropriate algorithm for the given context. Ideally, a comprehensive approach would involve testing multiple algorithms to identify the most effective predictive model in IP in HE. However, time constraints often limit this exhaustive exploration. Finding a balance between exhaustive exploration and time efficiency becomes a critical aspect of the research process. Researchers must carefully weigh the necessary constraints and adopt strategies that allow for a careful selection of algorithms without compromising the overall study schedule. To help with this, research has been undertaken to develop strategies for selecting the most appropriate algorithm for specific tasks [41].

In order to mitigate the challenges of algorithm selection, this study also acts as an attempt to find a solution by carrying out a comparative analysis of different algorithms used in the field of EDM. This approach will allow for a more informed decision to be made about the most appropriate algorithm for predicting student performance.

Another viable solution to this problem, if there are no time constraints, is to apply cross-validation techniques to a group of algorithms. The use of cross-validation methods ensures a robust evaluation and helps to select the model with the best-performing model from the group.

- Evaluation Metrics Application

The selection of appropriate evaluation metrics is a decision that depends on the specific problem at hand [42]. The literature review revealed cases where studies were based on a single metric, potentially limiting the understanding of a model's performance. Multiple metrics are recommended to provide a comprehensive view of the predictive model's strengths and weaknesses. Researchers need to match the metrics chosen to the complexity of the educational context and the nature of the prediction task. This will ensure an evaluation that goes beyond a superficial assessment and provides a deeper

understanding of the model's effectiveness. The simplest solution to this problem is to apply multi-metric evaluation, which uses a range of metrics, such as accuracy, precision, recall, and F1-score, to provide a more comprehensive assessment of model performance.

By addressing these challenges, researchers can strengthen the foundations of their studies and improve the reliability and applicability of ML techniques in predicting student performance in the dynamic landscape of HE, particularly in IP.

5. Discussion

At the beginning of the research, three research questions were identified, which will be answered below.

RQ1: What were the most-used ML algorithms proposed by the researchers for predicting students' performance in HE in IP? The literature review identified several algorithms applicable to the context of this study. However, four algorithms stood out and were used in multiple papers. These algorithms were Naïve Bayes, J48, Random Forest, and SVM, used in five, four, three, and three different papers, respectively. As mentioned above, when presenting the most commonly used algorithms, the fact that these algorithms are widely used does not always mean that they are the best algorithms for this type of problem, as has been shown with approaches using neural networks such as DNN and PNN, which have achieved extremely positive results.

RQ2: Which datasets were used, and which evaluation metrics were considered most appropriate for measuring the performance of predictive models in HE in IP? Regarding the datasets, academic records were the most used; with those, it is possible to observe the student's past performance and understand their ability and academic commitment over time. The fact that it is used most often does not mean that it produces better results, since the analysis carried out showed that the best results were obtained by using several data sources. It is worth noting, however, that in these cases of multiple use, the type of data used were always academic records, which highlights their relevance and impact in the context of prediction. As for the metrics, after the analysis of the papers, it became clear that there was no perfect answer to this question. There is no ideal metric. The choice of metrics always depends on the context in which people work. However, it was possible to conclude that more than one metric should be chosen, usually accuracy and F1-Score. Accuracy is preferred because of its simplicity and ease of interpretation. However, its limitation in unbalanced datasets, where it can be misleadingly high, necessitates the use of additional metrics. The F1-Score, which combines precision and recall into a single metric, is particularly valuable in scenarios where the balance between false positives and false negatives is critical. By using a combination of metrics, it is possible to gain a more complete understanding of model performance. The type of problem (classification or regression) to be applied also influences the metric to be chosen. For example, for regression and deep learning problems, the use of the RMSE metric is recommended.

RQ3: Which ML algorithms seem to better predict students' performance in HE in IP? Twenty algorithms were identified in the papers analyzed. To understand which could be the best, only those algorithms that were applied more than once were considered to compare them, by calculating the average accuracy of the algorithm. This way, it can be concluded that the algorithm with the best average accuracy was the SVM, which was applied in three studies (only in two of those was the accuracy metric used) and obtained an average accuracy of 93.97% using both academic records and questionnaire datasets. The fact that there are not yet many well-established studies on the application of deep learning means that the results found could not be used in this comparison. However, the potential demonstrated in papers such as [35] is great and suggests that this could be an opportunity for future research.

In the literature reviewed, there were situations where the amount of data used were very small, as well as cases of unbalanced datasets, directly affecting the quality of the model, as explicitly stated by the respective authors. In some of the papers analyzed, there

were cases of the use of little-known algorithms that do not appear in any other study, demonstrating that they may not be the most appropriate.

6. Threats to Validity

In this section, we present the main threats to the validity of this work and discuss mitigation strategies. This study, although comprehensive in its review of methodologies for predicting student performance in IP courses, has some limitations that need to be acknowledged.

Firstly, the scope of our literature review is limited by the selection criteria and databases used to identify relevant studies. Although we have attempted to include relevant studies, there is always the possibility that some relevant research may have been missed due to the limitations of our search parameters, such as the search string, the date of publication, or the availability of publications in certain databases.

Secondly, the review focuses specifically on the use of ML algorithms to predict student performance. Although ML techniques are a powerful tool, this focus inherently excludes other potentially valuable methodologies that could also contribute to understanding and improving student outcomes.

Finally, the conclusions reached in this study depend on the performance of the ML algorithms in the papers analyzed. The performance of these algorithms depends on the quality and characteristics of the datasets used, as well as on the evaluation steps applied. The variability in the granularity of the data and the attributes included in the datasets can significantly influence the results and the generalizability of the conclusions. The evaluation methods used and reported are the only way to analyze the results of the papers. Cases where only metrics such as accuracy are used, without further information presented, can sometimes be misleading and directly affect the conclusions reached in this study. Recognizing this type of limitation is crucial to interpreting the conclusions and results of this study.

7. Future Research Directions

In this section, we suggest future research directions for predicting student performance in IP courses in HE.

- Use of deep learning techniques

The literature review revealed a significant gap in the use of deep learning techniques for predicting performance in IP in HE. Although several ML algorithms have been identified, the specific application of deep learning in this educational context remains vastly underutilized. Thus, identifying and exploiting strategies that fully utilize the capabilities of DNNs may represent the next significant step in the evolution of students' performance prediction.

The emerging field of deep learning has demonstrated success in applications ranging from computer vision to natural language processing. The application of these techniques in the education domain remains largely unexplored. By employing more complex models and deeper layers, we can capture richer representations of the patterns present in students' data.

To advance in this area, it is essential to develop a deeper understanding of the specific challenges faced when applying deep learning techniques in an educational context. One obstacle is the need to deal with often complex and diverse datasets, which requires effective pre-processing strategies. In addition, the interpretability of deep learning models can be challenging, and it is crucial to develop approaches that make model decisions more understandable and explainable.

- Integration of various data sources

Another promising direction for future research is to explore using different types of data to predict performance in IP courses in HE. In the analyzed papers, the datasets mainly consist of assessment grades, personal data, or quiz answers. There are also a few studies

that work with the code developed by the students. The development of models that take into consideration the code developed could provide a more comprehensive understanding of student engagement and learning patterns.

The use of aptitude tests in IP could also be a promising strategy for using the obtained results to predict students' success when they start an IP course.

Analyzing different types of information can offer a different view of students' mental and behavioral aspects, allowing for the development of predictive models that do not rely only on traditional metrics.

These approaches have their own challenges, including the complexities of data integration, the development of sophisticated algorithms capable of processing multiple types of data, and the resolution of privacy issues related to the collection and analysis of different kinds of data.

8. Conclusions

Through this review of the available literature, this study examined the methodologies for predicting student performance in IP in HE, with a particular focus on analyzing the ML algorithms applied and the evaluation metrics used. This study describes a systematic literature review in which several papers were selected and analyzed. The results obtained provided valuable insights into the effectiveness of these approaches in an educational context.

The results of this study suggest that the use of ML algorithms, such as SVM, can be a valuable tool for predicting student performance in IP in HE. These algorithms can help identify students at risk of failing, allowing for targeted interventions and improved educational support. However, it is important to note that the performance of the models is directly dependent on the quality of the data used to train and test the model, suggesting that a process of data collection and analysis is essential for successful implementation.

We believe that this study makes a significant contribution to understanding which ML algorithms are used for predicting student performance in IP in HE and which evaluation metrics and data sources are most commonly used. The search for effective solutions to the high failure and dropout rates in IP continues and with the results obtained in this study we hope to assist future research in the process of choosing which algorithms to use in developing a possible solution, as well as the evaluation metrics to use and the types of data to explore that are best suited to this type of problem. By systematically comparing algorithms, metrics, and datasets, we have highlighted their relative strengths and limitations, and thereby clarified their practical utility and potential effectiveness in predicting student performance.

Naïve Bayes and J48 were found to be the most commonly used ML algorithms, with SVM achieving the highest average accuracy. However, deep learning techniques such as DNN and PNN showed significant potential, suggesting a promising direction for future research. In terms of datasets and evaluation metrics, academic records were the most commonly used, often in combination with other data sources to improve results. While accuracy was the most commonly used metric, this study highlighted the need for a combination of metrics to comprehensively assess model performance.

In the quest to improve the prediction of student performance in IP in higher education, some promising directions for future research have been identified. The application of deep learning techniques, which are still relatively unexplored in educational contexts, could be the next step in improving the accuracy of predictions, as well as exploring the use of different types of data, such as student-developed code and the use of aptitude tests, which could provide richer datasets and more nuanced insights into student performance.

Author Contributions: Conceptualization, F.B.C., A.G. and A.R.B.; methodology, F.B.C., A.G., A.R.B. and J.B.; validation, F.B.C., A.G., A.R.B. and J.B.; formal analysis, F.B.C., A.G. and A.R.B.; investigation, J.P.J.P.; resources, F.B.C., A.G. and A.R.B.; data curation, F.B.C., A.G. and A.R.B.; writing—original draft preparation, J.P.J.P.; writing—review and editing, F.B.C., A.G., A.R.B. and J.B.; supervision,

F.B.C., A.G. and A.R.B.; project administration, F.B.C., A.G., A.R.B. and J.B.; funding acquisition, J.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Watson, C.; Li, F.W. Failure rates in introductory programming revisited. In Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference, Uppsala, Sweden, 21–25 June 2014. [CrossRef]
2. Bennedsen, J.; Caspersen, M.E. Failure rates in introductory programming. *ACM Inroads* **2019**, *10*, 30–36. [CrossRef]
3. Lahtinen, E.; Ala-Mutka, K.; Järvinen, H.-M. A study of the difficulties of novice programmers—12 years later. *ACM SIGCSE Bull.* **2005**, *37*, 14–18. [CrossRef]
4. Qian, Y.; Lehman, J. Students' misconceptions and other difficulties in introductory programming. *ACM Trans. Comput. Educ.* **2017**, *18*, 1–24. [CrossRef]
5. Gomes, A.; Mendes, A.J.N. Learning to program-difficulties and solutions. In Proceedings of the International Conference on Engineering Education—ICEE 2007, Coimbra, Portugal, 3–7 September 2007.
6. Jenkins, T. On The Difficulty of Learning To Program. In Proceedings of the 3rd Annual LTSN-ICS Conference, Loughborough University, Loughborough, UK, 27–29 August 2002.
7. Bennedsen, J.; Caspersen, M.E. Abstraction ability as an indicator of success for learning object-oriented programming? *ACM SIGCSE Bull.* **2006**, *38*, 39–43. [CrossRef]
8. Byrne, P.; Lyons, G. The effect of student attributes on success in programming. *ACM SIGCSE Bull.* **2001**, *33*, 49–52. [CrossRef]
9. Luxton-Reilly, A.; Simon; Albluwi, I.; Becker, B.A.; Giannakos, M.; Kumar, A.N.; Ott, L.; Paterson, J.; Scott, M.J.; Sheard, J.; et al. Introductory programming: A systematic literature review. In Proceedings of the ITiCSE '18: 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, Larnaca, Cyprus, 2–4 July 2018.
10. Gomes, A.; Mendes, A. A study on student's characteristics and programming learning. In Proceedings of the ED-MEDIA 2008—World Conference on Educational Multimedia, Hypermedia & Telecommunications, Vienna, Austria, 30 June 2008.
11. Gomes, A.; Mendes, A. A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations. In Proceedings of the 2014 IEEE Frontiers in Education Conference (FIE), Madrid, Spain, 22–25 October 2014. [CrossRef]
12. Tomai, E.; Reilly, C.F. The impact of math preparedness on introductory programming (CS1) success (abstract only). In Proceedings of the SIGCSE '14: The 45th ACM Technical Symposium on Computer Science Education, Atlanta, GA, USA, 5–8 March 2014. [CrossRef]
13. Lishinski, A.; Yadav, A.; Enbody, R.; Good, J. The influence of problem solving abilities on students' performance on different assessment tasks in CS1. In Proceedings of the SIGCSE '16: The 47th ACM Technical Symposium on Computing Science Education, Memphis, TN, USA, 2–5 March 2016; pp. 329–334. [CrossRef]
14. Jokhan, A.; Chand, A.A.; Singh, V.; Mamun, K.A. Increased Digital Resource Consumption in Higher Educational Institutions and the Artificial Intelligence Role in Informing Decisions Related to Student Performance. *Sustainability* **2022**, *14*, 2377. [CrossRef]
15. Sobral, S.R. Strategies on Teaching Introducing to Programming in Higher Education. In *Advances in Intelligent Systems and Computing*; Springer: Cham, Switzerland, 2021. [CrossRef]
16. Gomes, A.; Mendes, A.J.; Marcelino, M.J. Computer Science Education Research—An overview and some proposals. In *Innovative Teaching Strategies and New Learning Paradigms in Computer Programming*; Queiroz, R., Ed.; IGI-Global: Hershey, PA, USA, 2015; pp. 1–29.
17. Köhler, J.; Hidalgo, L.; Jara, J.L. Predicting Students' Outcome in an Introductory Programming Course: Leveraging the Student Background. *Appl. Sci.* **2023**, *13*, 11994. [CrossRef]
18. Liu, Y.; Fan, S.; Xu, S.; Sajjanhar, A.; Yeom, S.; Wei, Y. Predicting Student Performance Using Clickstream Data and Machine Learning. *Educ. Sci.* **2023**, *13*, 17. [CrossRef]
19. Quille, K.; Bergin, S. CS1: How will they do? How can we help? A decade of research and practice. *Comput. Sci. Educ.* **2019**, *29*, 254–282. [CrossRef]
20. Shahiri, A.M.; Husain, W.; Rashid, N.A. A Review on Predicting Student's Performance Using Data Mining Techniques. *Procedia Comput. Sci.* **2015**, *72*, 414–422. [CrossRef]
21. Alhothali, A.; Albsisi, M.; Assalahi, H.; Aldosemani, T. Predicting Student Outcomes in Online Courses Using Machine Learning Techniques: A Review. *Sustainability* **2022**, *14*, 6199. [CrossRef]
22. Silva, C.; Fonseca, J. Educational data mining: A literature review. In *Advances in Intelligent Systems and Computing*; Springer: Cham, Switzerland, 2017. [CrossRef]

23. Bachhal, P.; Ahuja, S.; Gargrish, S. Educational Data Mining: A Review. *J. Physics Conf. Ser.* **2021**, *1950*, 012022. [CrossRef]
24. Scherer, R.; Siddiq, F.; Viveros, B.S. The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *J. Educ. Psychol.* **2019**, *111*, 764–792. [CrossRef]
25. Sobral, S.; Oliveira, C. Predicting Students' Performance in Introductory Programming Courses: A Literature Review. In Proceedings of the 15th International Technology, Education and Development Conference, Online, 8–9 March 2021; pp. 7402–7412.
26. Kitchenham, B. *Procedures for Performing Systematic Reviews*; Keele University: Keele, UK; National ICT Australia: Sydney, Australia, 2004; Volume 33.
27. Silva, M.; Shaffer, E.G.; Nytko, N.; Amos, J.R. A case study of early performance prediction and intervention in a computer science course. In Proceedings of the ASEE Annual Conference and Exposition, Conference Proceedings, Virtual Online, June 2020. [CrossRef]
28. Jamjoom, M.; Alabdulkreem, E.; Hadjouni, M.; Karim, F.; Qarh, M. Early prediction for at-risk students in an introductory programming course based on student self-efficacy. *Informatica* **2021**, *45*, 1–9. [CrossRef]
29. Sivasakthi, M.; Pandiyan, M. Machine Learning Algorithms to Predict Students' Programming Performance: A comparative Study. *J. Univ. Shanghai Sci. Technol.* **2022**, *24*, 1–8.
30. Ahadi, A.; Lister, R.; Haapala, H.; Vihavainen, A. Exploring machine learning methods to automatically identify students in need of assistance. In Proceedings of the ICER '15: International Computing Education Research Conference, Omaha, NE, USA, 9–13 July 2015.
31. Đambić, G.; Krajcar, M.; Bele, D. Machine learning model for early detection of higher education students that need additional attention in introductory programming courses. *Int. J. Digit. Technol. Econ.* **2016**, *1*, 1–11.
32. Khan, I.; Al Sadiri, A.; Ahmad, A.R.; Jabeur, N. Tracking student performance in introductory programming by means of machine learning. In Proceedings of the 2019 4th MEC International Conference on Big Data and Smart City (ICBDSC), Muscat, Oman, 15–16 January 2019. [CrossRef]
33. Veerasamy, A.K.; D'Souza, D.; Apiola, M.-V.; Laakso, M.-J.; Salakoski, T. Using early assessment performance as early warning signs to identify at-risk students in programming courses. In Proceedings of the 2020 IEEE Frontiers in Education Conference (FIE), Uppsala, Sweden, 21–24 October 2020. [CrossRef]
34. Cooper, C. Using Machine Learning to Identify At-risk Students in an Introductory Programming Course at a Two-year Public College. *Adv. Artif. Intell. Mach. Learn.* **2022**, *2*, 407–421. [CrossRef]
35. Shen, G.; Yang, S.; Huang, Z.; Yu, Y.; Li, X. The prediction of programming performance using student profiles. *Educ. Inf. Technol.* **2023**, *28*, 725–740. [CrossRef]
36. Sunday, K.; Ocheja, P.; Hussain, S.; Oyelere, S.S.; Samson, B.O.; Agbo, F.J. Analyzing student performance in programming education using classification techniques. *Int. J. Emerg. Technol. Learn. (iJET)* **2020**, *15*, 127–144. [CrossRef]
37. Sivasakthi, M. Classification and prediction based data mining algorithms to predict students' introductory programming performance. In Proceedings of the 2017 International Conference on Inventive Computing and Informatics (ICICI), Coimbatore, India, 23–24 November 2017; pp. 346–350.
38. Naser, M.Z.; Alavi, A.H. Error Metrics and Performance Fitness Indicators for Artificial Intelligence and Machine Learning in Engineering and Sciences. *Arch. Struct. Constr.* **2021**, *3*, 499–517. [CrossRef]
39. Sessions, V.; Valtorta, M. The effects of data quality on machine learning algorithms. In Proceedings of the 2006 International Conference on Information Quality, ICIQ 2006, Cambridge, MA, USA, 10–12 November 2006.
40. Realinho, V.; Machado, J.; Baptista, L.; Martins, M.V. Predicting Student Dropout and Academic Success. *Data* **2022**, *7*, 146. [CrossRef]
41. Sala, R.; Zambetti, M.; Pirola, F.; Pinto, R. How to select a suitable machine learning algorithm: A feature-based, scope-oriented selection framework. In Proceedings of the Summer School "Francesco Turco", Palermo, Italy, 12–14 September 2018.
42. Liu, Y.; Zhou, Y.; Wen, S.; Tang, C. A Strategy on Selecting Performance Metrics for Classifier Evaluation. *Int. J. Mob. Comput. Multimedia Commun.* **2014**, *6*, 20–35. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

computers

MDPI

Article

Digital Twin-Enhanced Programming Education: An Empirical Study on Learning Engagement and Skill Acquisition

Ming Lu * and Zhongyi Hu

College of Computer Science and Artificial Intelligence, Wenzhou University, Wenzhou 325035, China
* Correspondence: 20210069@wzu.edu.cn

Abstract: As an introductory core course in computer science and related fields, "Fundamentals of Programming" has always faced many challenges in stimulating students' interest in learning and cultivating their practical coding abilities. The traditional teaching model often fails to effectively connect theoretical knowledge with practical applications, resulting in a low retention rate of students' learning and a weak ability to solve practical problems. Digital twin (DT) technology offers a novel approach to addressing these challenges by creating dynamic, virtual replicas of physical systems with real-time, interactive capabilities. This study explores DT integration in programming teaching and its impact on learning engagement (behavioral, cognitive, emotional) and skill acquisition (syntax, algorithm design, debugging). A quasi-experimental design was employed to study 135 first-year undergraduate students, divided into an experimental group (n = 90) using a DT-based learning environment and a control group (n = 45) receiving traditional instruction. Quantitative data analysis was conducted on participation surveys, planning evaluations, and qualitative feedback. The results showed that, compared with the control group, the DT group exhibited a higher level of sustained participation ($p < 0.01$) and achieved better results in actual coding tasks ($p < 0.05$). Students with limited coding experience showed the most significant progress in algorithmic thinking. The findings highlight that digital twin technology significantly enhances engagement and skill acquisition in introductory programming, particularly benefiting novice learners through immersive, theory-aligned experiences. This study establishes a new paradigm for introductory programming education by addressing two critical gaps in digital twin applications: (1) differential effects on students with varying prior knowledge (engagement/skill acquisition) and (2) pedagogical mechanisms in conceptual visualization and authentic context creation.

Keywords: digital twin; programming education; learning engagement; skill acquisition; immersive education; educational technology

1. Introduction

The course "Fundamentals of Programming" is the cornerstone of computer science education, aiming to cultivate students' logical thinking abilities, grammatical application skills, and basic coding skills [1]. It lays a solid foundation for subsequent advanced courses, such as data structures, algorithms, and software engineering. Therefore, the teaching effectiveness of this course has a significant impact on students' long-term academic development and their success in their career paths [2]. However, education statistics present a worrying situation: globally, over 40% of students struggle to complete this course, and in many educational institutions, the dropout rate for this course exceeds 30% [3]. There are three main reasons for these results:

Firstly, students find it challenging to visualize the execution process of code (reported by 32% of students). Programming concepts such as loops, conditions, and function calls are essentially abstract. Students often find it challenging to map lines of code to their tangible results in their minds [4]. For example, beginners may write a "For loop" but cannot master how to iterate over the dataset, leading to confusion during debugging. Secondly, there is a lack of practical application scenarios in learning (reported by 28% of students) [3]. Traditional teaching methods are often limited to isolated coding exercises, such as "calculating the factorial of numbers", which can make students feel disconnected from the real world. This sense of disconnection will weaken students' understanding of the practical value of the course theme, thereby reducing their motivation to engage deeply in the learning process [5].

Third, there is inadequate real-time feedback (reported by 21% of students) [3]. In traditional settings, feedback on code often comes hours or days after submission, in the form of cryptic compiler messages or brief instructor comments. This delay hinders the iterative learning process, as students are unable to associate errors with specific concepts [6] immediately.

These challenges fully demonstrate the urgent need for innovative teaching methods to bridge the gap between abstract programming concepts and concrete, practical outcomes. Digital twin (DT) technology is a promising solution [7]. A digital twin refers to the virtual representation of physical entities or systems, which enables real-time data exchange and simulation [8]. It has been widely utilized in various industries, including predictive maintenance in manufacturing, surgical simulation in healthcare, and urban planning in innovative city development [9]. In education, its potential lies in creating dynamic, interactive environments where students can manipulate virtual models to observe the consequences of their actions—an attribute particularly valuable for programming education [10].

Unlike static simulations or virtual labs, digital twins offer bidirectional data flow: students' code inputs immediately affect the virtual system, which in turn provides contextually relevant feedback [11]. For example, a student writing code to control a virtual robot in a DT environment can see the robot stall if a loop lacks a termination condition, with the system flagging, "Infinite loop detected: Robot motor overload". This immediate, context-rich feedback bridges the gap between code and outcome, making abstract concepts concrete [12].

The existing educational technology research in programming education mainly focuses on two areas: visual programming tools (such as Scratch [13]) and gamified platforms (such as CodeCombat [14]). Although these tools enhance initial engagement, they often lack the complexity and real-world relevance required to develop powerful programming skills. Visual tools simplify syntax but may not adequately prepare students for text-based coding, and gamified platforms often prioritize competition over a deep conceptual understanding. Digital twins, by contrast, can simulate authentic computing environments (e.g., embedded systems, software architectures) within a controlled educational framework, enabling students to experiment with code in scenarios mirroring professional practice [10]. For instance, a DT of a smart home system can teach students how to program sensor inputs to trigger lighting adjustments—an application with clear real-world relevance.

This study addresses three key research gaps:

1. There is a lack of empirical evidence on digital twin integration in introductory programming courses. While DTs have been explored in advanced computer science courses (e.g., software engineering [15]), their application in foundational programming instruction remains underexamined.

2. There is an unclear impact of DT-based environments on learning engagement across diverse student backgrounds. Prior research on educational technology often focuses on average outcomes, but little is known about how DTs affect students with varying levels of previous coding experience or learning styles.
3. The potential of digital twins in cultivating algorithmic thinking and error debugging abilities has not been fully developed. These skills are essential for programming ability; however, there is a lack of quantitative research on how interactive and feedback-rich environments, such as digital twins, affect the development of these skills.

To fill these research gaps, this study aims to provide a theoretical basis and practical guidance for educators who wish to apply digital twin technology in programming education. Specifically, this study attempts to answer the following questions: How will digital twins affect students' engagement (including behavioral, cognitive, and emotional aspects) in the "Fundamentals of Programming" course? What impact will it have on students' acquisition of core programming skills? Are there differences in these impacts among different student groups?

In order to clarify how digital twin technology enhances programming teaching, the relevant terms are introduced according to the logical chain of "core scenario → system architecture → implementation tools" to facilitate understanding in subsequent chapters.

Virtual smart classrooms: They are interactive learning environments created through digital twin technology. In this study, a virtual space was constructed using 3D rendering, which includes interactive objects such as robots and sensors. The execution results of student code can be visually presented through the state changes of these objects, and programming concepts can be visualized through the use of 3D rendering.

Digital twin teaching system (DT teaching system): It is an architecture composed of three interrelated layers—the client layer, the middleware layer, and the server layer. The client layer provides students with an interface to interact with the digital twin environment, including components like a 3D rendering engine for immersive visualization and a code editor for writing code. The middleware layer acts as a bridge, processing code, generating simulations, and enabling real-time communication. The server layer manages data persistence, user progress, and collaborative features. This system can transform abstract programming concepts into visual representations, provide real-time and scenario-specific feedback, and promote interactive and collaborative learning. It supports the operation of the virtual smart classroom.

Digital twin platform (DT platform): It is a platform that uses Unity3D to build a 3D visual interface and combines Python to implement backend logic. In this study, it has created a reproduction scene of a virtual smart classroom. This platform includes a code interface with syntax highlighting and real-time syntax checking capabilities, a real-time simulation function to visually present the program execution process, a feedback system to provide contextual prompts, and a collaborative space for students to interact and learn from each other.

2. Literature Review

2.1. Digital Twin Technology

Digital twin (DT) technology originated in the early 2000s, with NASA's use of virtual replicas to monitor and simulate the performance of spacecraft [16]. Over the past two decades, advancements in cloud computing, the Internet of Things (IoT), and 3D modeling have expanded their applications [8]. DT technology constructs dynamic virtual replicas of physical entities and enables real-time data interaction between physical and virtual spaces [17]. The core of digital twins consists of three parts: (1) physical entities (or logical systems), (2) virtual

replicas, and (3) data bridges for real-time synchronization [18]. This tripartite structure enables iterative testing, predictive analysis, and scenario simulation—features that make DT invaluable to education [7]. In this study, a digital twin (DT) refers to a dynamic virtual replica of physical programming tools (e.g., robots, sensors) that synchronizes with code inputs to simulate real-world behavior within a virtual smart classroom. The virtual smart classroom is defined as a comprehensive teaching environment that integrates multiple DTs and support systems.

In technical education, traditional programming teaching tools, such as CodePen [19] and JSFiddle [20], focus on code editing and basic visual preview but lack dynamic visualization of the underlying mechanisms. This has led to a lack of mechanisms in teaching to reduce the cognitive load of abstract concepts. DT technology can tightly couple the establishment of code syntax and visual results, solving this challenge. DT technology has been used to teach complex systems. In engineering, students interact with DT replicas of machinery to learn about mechanical stress, with the virtual model visualizing wear and tear as parameters change [21]. In robotics, DT technology enables students to program motion algorithms and test them in a virtual environment before deploying them to physical robots, thereby reducing the risk of equipment damage [22]. In computer science, the study of network systems teaches students about network security knowledge by simulating hacker attacks and defense protocols [23].

DT provides unique advantages for programming education [24]:

- Specificity: Abstract concepts (such as variables and loops) are related to observable results (such as virtual counter increments and the gradual movement of robots).
- Security: Students can experiment with risky code (such as infinite loops) without compromising physical systems or data.
- Context: Embedding encoding tasks into meaningful scenarios (such as programming virtual thermostats) enhances perceptual relevance.

2.2. Learning Participation in Programming Education

Learning engagement encompasses three dimensions, behavior, cognition, and emotion, and is a key factor determining the effectiveness of programming course teaching [25]. Among them, behavioral participation is specifically manifested as actively engaging in various coding tasks, such as writing code, troubleshooting program errors, and collaborating with classmates [26]. Cognitive involvement involves in-depth processing of programming logic (such as analyzing the reasons for function failures and linking new concepts with prior knowledge) [27]. Emotional engagement reflects positive attitudes toward coding challenges (e.g., persistence despite errors, enjoyment of problem-solving) [28].

Traditional teaching methods often struggle to balance cognitive, behavioral, and emotional learning engagement [29]. For example, lecture formats frequently rely on passive cognitive participation, with students only listening to explanations and rarely having the opportunity to apply learned concepts in real time. Conventional laboratory exercises, on the other hand, focus on practical actions at the behavioral level. Students often copy preset code snippets without delving into the underlying logical principles. These two forms frequently overlook emotional involvement, as repeated errors and delayed feedback can lead to frustration and anxiety, a phenomenon known as "programming phobia" [30].

Technological enhancement methods attempt to address these gaps but have limitations. Interactive coding platforms, such as LeetCode, enhance behavioral engagement through real-time syntax validation; however, they rarely promote cognitive engagement in complex logic [31]. Video tutorials have improved passive cognitive engagement but lack interactivity and limit behavioral practice [32]. Peer code review platforms can en-

hance students' emotional engagement through social support, but such platforms may not provide the structured feedback necessary for skill development [33]. Digital twin technology possesses the characteristics of immersion and instant response [34], thereby having the potential to mobilize students' enthusiasm for participation at multiple levels.

- Behavioral engagement: Hands-on coding in a virtual environment, where every keystroke affects the DT's state, encourages active participation.
- Cognitive engagement: Real-time visualization of code influence (such as data flowing through loops) facilitates deep processing of logic.
- Emotional participation: The digital twin environment provides a safe experimental space, where even if errors occur, they will only result in virtual system failures rather than real-life failures in public settings. This helps alleviate students' anxiety and cultivates their resilience when facing problems.

2.3. Skill Acquisition in Introductory Programming

The core skills targeted in "Fundamentals of Programming" include mastery of syntax, algorithmic thinking, and proficiency in debugging. Research indicates that these skills develop most effectively through deliberate practice with immediate feedback [35]. This process aligns with the "testing effect" in cognitive psychology, whereby active retrieval and application strengthen memory retention [36].

Traditional teaching feedback mechanisms often hinder students' acquisition of skills [37]. Compiler messages (e.g., "Syntax error on line 5") are technical and decontextualized, failing to explain why the error occurred or how to fix it. Instructor reviews are delayed, disrupting the connection between action (coding) and consequence (feedback). Peer feedback may be inaccurate, as novices often lack the expertise to identify logical errors.

In this study, feedback mechanisms are clarified as a collaboration between the DT (virtual robot's behavioral response) and the DT teaching system (contextual error prompts), which greatly improves learning skills.

Digital twin environments address the above limitations through three key features [10]:

- Real-time syntax validation: As students write code, the DT teaching system (middleware layer) checks for errors (e.g., missing semicolons, undeclared variables) and highlights them in the context of the virtual system's behavior. For example, a missing closing brace in a function may cause a virtual sensor to "malfunction" with the DT annotating, "Function incomplete: Sensor data not processed".
- Algorithm visualization: The DT teaching system (middleware layer) visually illustrates how data flows within the program structure. Students who program sorting algorithms can observe virtual systems rearranging numbers in real time, making it easier to identify inefficiencies (such as unnecessary iterations).
- Root-cause debugging: Students can manipulate the DT's state (e.g., resetting input values, pausing execution) to trace errors back to specific code segments. For instance, a logic error in a conditional statement may cause a virtual door to "open when it should close"; by adjusting the condition and observing the DT's response, students learn to isolate the mistake.

These features are consistent with the cognitive load theory [38], which emphasizes the importance of manageable information presentation in skill acquisition. By allocating information through visual (DT simulation), textual (code), and contextual (feedback) channels, the DT reduces the mental effort required to process complex concepts, thereby releasing cognitive resources for deep learning.

3. Research Design and Methodology

3.1. Research Questions

This study aims to answer the following research questions:

1. Compared with traditional methods, does integrating digital twin technology in "Fundamentals of Programming" teaching significantly improve students' learning engagement (behavior, cognition, emotion)?
2. What impact does a DT-based learning environment have on students' acquisition of core programming skills (syntax application, algorithm design, debugging)?
3. Are there differences in the effectiveness of DT technology across student subgroups with varying prior coding experience?

3.2. Participants

The participants are 135 first-year undergraduate students enrolled in the "Fundamentals of Programming" course at a public university in Wenzhou for the 2024–2025 academic year. This sample comes from three complete classes, and the participants mainly learn the following:

- Computer science (62%, n = 84);
- Information science (23%, n = 31);
- Related fields (such as software engineering, data science) (15%, n = 20).

The experiment selected 89 males (65.9%) and 46 females (34.1%), reflecting the typical gender distribution of computer science majors in the region [39]. The participants were aged between 18 and 21 years old (M = 19.2, SD = 0.8), with 92% of them being between 18 and 19 years old (first-year undergraduate students). All participants reported no formal programming training, confirmed via a pre-course questionnaire. Self-reported exposure to coding (e.g., online tutorials) was minimal, with a mean of 1.2 h/week (SD = 0.7), and no significant differences were found between groups ($p > 0.05$).

To ensure internal effectiveness, the participants were randomly assigned to either the experimental group (n = 90) or the control group (n = 45) after completing the pre-class aptitude test. This test evaluates basic programming skills, including logical reasoning (such as pattern recognition and syllogism), as well as fundamental mathematical skills (such as algebra and problem-solving). The independent sample t-test confirmed that there was no significant difference in the scores of the aptitude test between the two groups, ensuring consistency in the initial ability level.

The basic information of the participants is shown in Table 1. It presents a comparison of baseline data on key features between the experimental group (digital twin teaching group) and the control group (traditional teaching group), with statistical significance mainly reflected in the following aspects.

(1) Verifying group balance

The intergroup differences in all features satisfy $p > 0.05$ (with the lowest being 0.17 for extracurricular coding time), indicating that there is no statistically significant difference between the two groups in core variables such as gender ratio, age distribution, professional composition, basic logical reasoning ability, and extracurricular coding experience. This proves that random grouping is effective, and the initial states of the two groups of students are comparable, eliminating the interference of baseline differences for the attribution of subsequent intervention effects (such as learning engagement and skill improvement).

(2) Controlling confounding variables

The balance of demographic characteristics, such as gender, age, and significance, ensures that these factors do not systematically affect the experimental results (such as potential differences in programming learning due to different professional backgrounds).

There was no significant difference (p = 0.68) in the pre-test logical reasoning score (a key foundational ability for programming learning), indicating that the two groups of students have the same starting point for core learning abilities, and subsequent skill improvement is more likely to be attributed to the digital twin teaching model rather than initial ability differences.

The balance of extracurricular coding time (p = 0.17) eliminates the confusion caused by "extra practice time differences" in the experimental results.

(3) Enhancing research validity

The balance of baseline data is a crucial guarantee of internal validity in quasi-experimental designs. These data support the conclusion that the differences between the experimental group and the control group after intervention, such as skill scores and engagement levels, are more likely to be caused by the digital twin teaching model rather than inherent differences between the two groups of students, thereby strengthening the credibility of the research findings.

This study belongs to the scope of university teaching reform research and is an integral part of the conventional teaching process. Participants in the study were informed of the study details and their informed consent was obtained. They had the right to withdraw from the study at any time without incurring any penalties or adverse consequences. The research data were anonymized using a unique identifier, and all records were securely stored in an encrypted database.

Table 1. Participant basic information form.

Characteristic Indicators	Experimental Group (n = 90)	Control Group (n = 45)	Intergroup Differences (p-Value)
Gender (Male/Female)	57/33	32/13	0.89
Age (M ± SD, years)	19.1 ± 0.7	19.3 ± 0.9	0.23
Professional distribution (%)			
Computer science (CS)	CS: 61.1	CS: 62.2	
Information science (IS)	IS: 22.2	IS: 24.4	0.91
Related fields (RFs)	RFs: 16.7	RFs: 13.4	
Pre-test logical reasoning score (M ± SD)	68.3 ± 7.2	67.8 ± 7.5	0.68
Weekly extracurricular coding time (M ± SD, hours)	1.3 ± 0.6	1.1 ± 0.8	0.17

3.3. Intervention Design

Two groups of students participated in an eight-week course centered on the core theme of "Fundamentals of Programming". The specific arrangement is as follows: In the first two weeks of the course, students mainly learned about the definition and use of variables, the characteristics and applicable scenarios of different data types, and basic input and output operations. Through these contents, they established a preliminary understanding of data processing in programming. Entering weeks 3 to 4, the course focuses on the study of control structures, including the logical judgment and application of if-else conditional statements and the use of loop statements such as for and while, to help students master the skills of implementing different process controls through code. In weeks 5 to 6, the teaching focus shifts to the definition, calling, and parameter passing rules of functions, allowing students to understand how to encapsulate code blocks through functions and improve code reusability and readability. In the last two weeks of the course, namely weeks 7 and 8, students begin to learn the basic concepts, creation, and operation

methods of arrays, as well as basic debugging skills. During the process of identifying and correcting code errors, students further consolidated their prior knowledge and enhanced their programming practical skills. The total teaching time for the two groups of students is the same, at 24 h, with lectures and laboratory practice each accounting for 12 h. The core difference between the two is reflected in the laboratory activities—the control group adopts traditional teaching methods, while the experimental group incorporates digital twin technology.

3.3.1. Architecture of DT Teaching System

As shown in Figure 1, the DT teaching system's architecture comprises three interrelated layers: the client layer, the middleware layer, and the server layer. Each layer is equipped with specialized components designed to achieve real-time interaction, simulation operations, and collaborative learning in the programming teaching process.

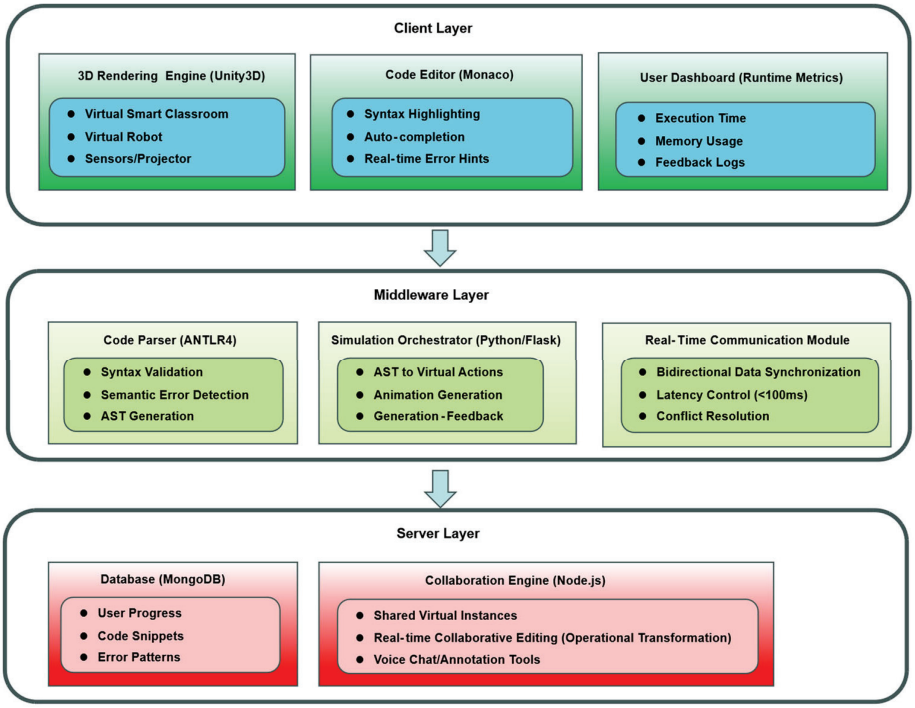


Figure 1. Architecture of the DT teaching system.

(1) Client Layer

The client layer serves as the user interface, providing students with a direct channel to interact with the digital twin environment. The original intention of this layer's design is to create an immersive and intuitive user experience, closely linking the code-writing process with visual feedback.

- 3D Rendering Engine (Unity3D)

  It renders the "Virtual Smart Classroom"—a 3D environment where programming concepts are visualized through interactive objects. It is a simulated space containing virtual devices (e.g., temperature sensors, projectors, and a robotic assistant) that respond to student code. There is a 3D avatar whose movements (e.g., navigation, object manipulation) are controlled by student-written functions (e.g., move_robot (), turn_left ()), making function calls and parameter passing tangible. Virtual sensors display real-time data (e.g., current_temp = 25 °C) to illustrate variables and data flow; the projector animates loop iterations (e.g., counting from 1 to 10) to visualize control structures.

- Code Editor (Monaco Editor)
  It provides a text-based interface for writing, editing, and submitting code (in the C language, aligned with course textbooks) comprising color-coding keywords, variables, and functions to improve readability. It suggests using commonly used functions (such as printf() and scanf()) and variable names to minimize syntax errors. It uses contextual tooltips to emphasize syntax errors (such as missing semicolons) and link them to potential fixes.
- User Dashboard
  It displays runtime metrics and feedback to help students monitor the performance and progress of their code. It measures the time required for code execution to improve efficiency. It visualizes resource consumption (such as a "virtual memory module with a capacity of 60%") for optimized education. It records system messages (such as "Infinite loop detected: robot battery low") to assist with debugging.

(2) Middleware Layer
The middleware layer acts as a bridge between the client and server, processing code, generating simulations, and enabling real-time communication. It ensures that students' input is converted into virtual actions and feedback. It is critical for translating code to virtual actions, addressing the gap between text and visualization.

- Code Parser (ANTLR4)
  It validates code syntax and semantics for simulation purposes by converting text-based code into a structured format. Grammar validation involves checking for structural errors (such as missing curly braces or incorrect loop syntax) and marking them to the client. Logical issues (such as undeclared variables or mismatched parameters) are identified and correlated with virtual system behavior (e.g., undeclared "speed" variable results in no response from robot movement). The valid code is then converted into an Abstract Syntax Tree (AST), a hierarchical representation of the code logic, for processing by the simulation orchestrator.
- Simulation Orchestrator (Python/Flask)
  It maps an AST onto virtual actions, generating animations and feedback according to code logic. It transforms code structures into 3D animations (e.g., for loop nodes triggering rotation counters and printf statements updating a virtual projector). It creates real-time visual effects (such as robot movements, sensor data updates) to visualize code execution. Additionally, it generates contextually rich messages about virtual system behavior (e.g., the "calculate_average()" function lacks a return value, and the sensor displays "undefined").
- Real-Time Communication Module
  It supports bidirectional data transmission between the client and server, ensuring synchronization of code editing, simulation, and feedback. For bidirectional data synchronization, it transfers code submissions from the client to the server and sends simulation results/feedback back to the client. It maintains a latency of less than 100 ms to ensure a sense of "real-time" interaction between code editing and a virtual response. It manages synchronous editing in collaborative mode (e.g., preventing overlapping ping changes to the same line of code).

(3) Server Layer
The server layer manages data persistence, user progress, and collaborative features, supporting long-term learning tracking and peer interaction.

- Database (MongoDB)
  It stores student data, code snippets, and system logs for progress tracking and adaptive learning. It records task completion status, test scores, and standard error patterns (e.g., students frequently miss loop termination conditions). It saves the submitted code for the instructor's review and future reference. It summarizes common errors to inform teaching (e.g., 30% of students encounter difficulties in array indexing) and tailors feedback accordingly.
- Collaboration Engine (Node.js)
  This feature fosters multi-user interaction in the digital twin environment, creating an environment that promotes peer learning and team collaboration. Students can invite their classmates to view and edit in the virtual smart classroom, thereby facilitating collaborative debugging and learning. At the same time, with the help of the Operational Translation (OT) algorithm, the code-editing operations of different users are synchronized to ensure that everyone can see the latest code version in real time. This support includes verbal discussions and visual annotations (such as drawing arrows to highlight code errors) to enhance peer feedback. The system operates continuously in a loop, and students submit their work through the client's code editor. The code parser validates syntax/semantics and generates an AST. The simulation orchestrator converts the AST into virtual actions (animations) and feedback. The client renders the animations and displays feedback to the student. The relevant data (including code, learning progress, error records, etc.) are stored in the server database, and collaborative editing content is also synchronized among users. This architecture design can transform abstract programming concepts into visual representations, provide real-time and scenario-specific feedback, promote interactive and collaborative learning, and solve key challenges in entry-level programming education.

3.3.2. Control Group: Traditional Instruction

The control group followed a conventional teaching model, representative of mainstream programming education.

Lectures: There were instructor-centered sessions using static PowerPoint slides and live code demonstrations via a classroom projector. For example, when teaching loops, the instructor displayed a slide with syntax rules (e.g., "for (initialization; condition; increment) {...}") and demonstrated a simple loop to print numbers 1–10. Students copied code snippets into notebooks for later reference.

Lab activities: There was individual practice using a standard online coding platform. Tasks included replicating predefined code (e.g., "Write a for loop to calculate the sum of 1–50") and solving textbook-style problems (e.g., "Find the maximum value in an array"). Feedback was limited to automated syntax checks and a weekly 10 min meeting with the instructor to review errors.

Materials: A printed course packet with code examples, a textbook, and PDF tutorials with step-by-step instructions for each lab task were provided.

The standard online coding platform used by the control group is fundamentally distinct from the DT teaching system, with differences extending far beyond the absence of a 3D rendering engine and real-time communication tools. Its key characteristics and contrasts with the DT system are as follows:

Core functionality: The control group's platform is a text-based coding environment focused solely on syntax validation and code execution (e.g., compiling and running code snippets). It lacks the DT teaching system's ability to map code logic to tangible virtual behaviors (e.g., robot movements, sensor data changes).

Feedback mechanism: Unlike the DT teaching system's contextual, real-time feedback (e.g., "Infinite loop detected: Robot battery low" linked to virtual robot stalling), the control platform provides only basic, decontextualized feedback (e.g., "Syntax error on line 5") without explaining the logical or practical implications.

Learning scenarios: The control platform uses isolated, abstract tasks (e.g., "Calculate the sum of 1–50"), whereas the DT teaching system embeds coding in meaningful scenarios (e.g., programming a virtual robot to navigate obstacles in a virtual smart classroom), enhancing relevance to real-world applications.

Interaction and collaboration: The control platform supports only individual coding practice with no collaborative features. In contrast, the DT system includes a shared virtual space for synchronous code editing, voice chat, and peer annotation, fostering collaborative problem-solving.

Concept visualization: Critical to programming education, the DT teaching system visualizes abstract concepts (e.g., loop iterations via animated counters), while the control platform relies on text-only output, leaving logic visualization to the student's imagination.

In summary, the control group's platform is a basic tool for code execution, whereas the DT teaching system is an integrated environment that combines virtual replicas (DTs), contextual feedback, scenario-based learning, and collaboration, addressing the limitations of traditional tools in connecting abstract code to practical understanding.

3.3.3. Experimental Group: Digital Twin-Based Instruction

The experimental group received the same lectures and covered the duplicate content as the control group but replaced 50% of lab time (6 h) with DT-based activities. The remaining 50% of lab time involved traditional coding practice to ensure parity in total hands-on experience. The specific differences between digital twin and traditional teaching activities are presented in Table 2, which provides an intuitive comparison of the core differences between the two teaching modes and highlights the innovative aspects of digital twin teaching.

Table 2. Comparison table of digital twins and traditional teaching activities.

Teaching Steps	Experimental Group (Digital Twin Teaching)	Control Group (Traditional Teaching)
Theoretical learning	Same lecturer teaching, consistent content (variables, loops, functions, etc.)	Same lecturer teaching, consistent content
Practical activities (proportion)	50% digital twin tasks 50% traditional exercises	100% Traditional Practice
Core tools	Virtual smart classroom (including code editor, 3D simulator, feedback system)	Text editor Online question answering system
Feedback mechanism	Real-time contextualized feedback (such as "Robot jamming: loop missing termination condition")	Delayed feedback (automatic grammar check, weekly instructor review)
Collaborative support	Virtual shared space (synchronous editing, voice discussion)	Unstructured collaboration, independently completing tasks
Typical task examples	Programming Control of Virtual Robot Obstacle Avoidance (Integrated Application Loop and conditional statements)	Writing a function to calculate the average value of an array (isolated exercise)

3.3.4. Digital Twin Learning Environment

The DT platform uses Unity3D (Version: Unity 2022.3.62f1 LTS, Unity Technologies, San Francisco, CA, USA) to build a 3D visual interface, combined with Python (Version: 3.13.5, Python Software Foundation, Beaverton, OR, USA) to implement backend logic, to create a reproduction scene of the virtual smart classroom—this is an interactive system

controlled by C language code (consistent with the course material content), including virtual robots, feedback prompt boxes, collaborative editing areas, and other interactive elements. The specific interaction scene is illustrated in Figure 2. The core components of the DT platform are as follows:

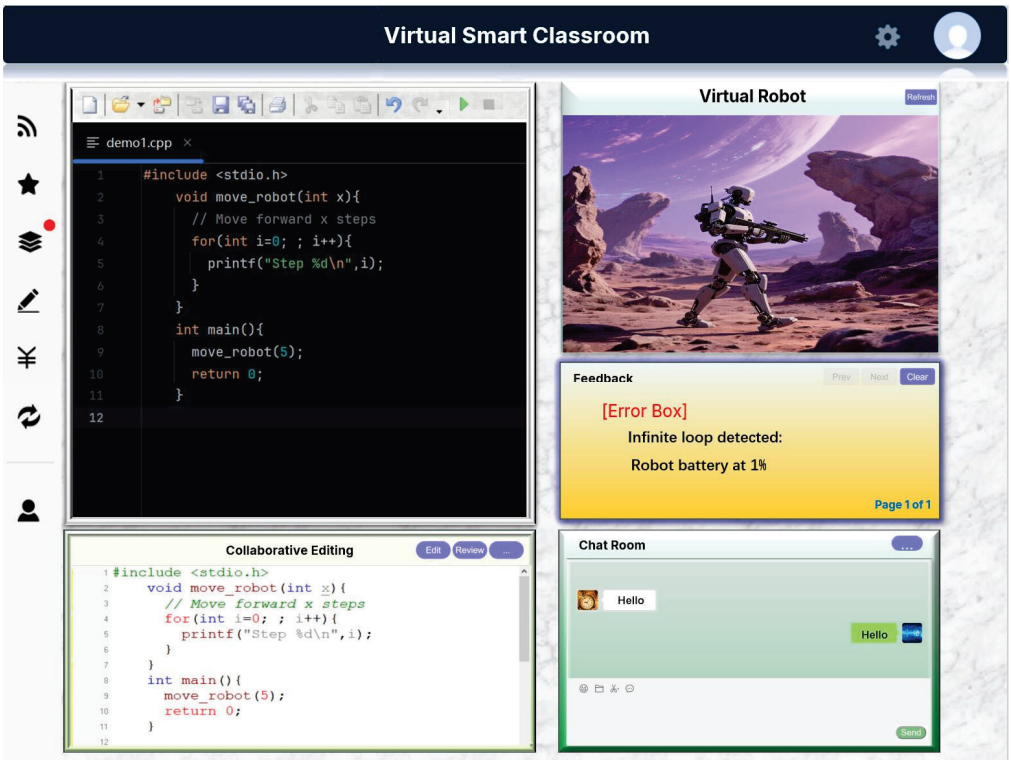


Figure 2. Interactive scene diagram of virtual smart classroom.

Firstly, the code interface, which is a syntax highlighting editor embedded in the digital twin environment, supports writing in the C language. It not only provides an auto-completion function for standard functions such as printf() and scanf() but also has real-time syntax checking capability. Once an error occurs, it will be highlighted in red.

Next is the real-time simulation function. The virtual smart classroom can visually present the program execution process; for example, loop iterations will be displayed in the form of animated loops. For example, the virtual counter corresponding to the "for loop" will increase from 1 to 10, and the changes in variables will be reflected in real time in the virtual memory module. For example, when processing sensor data, the "temperature" variable on the digital display will be updated accordingly, and function calls will trigger visible operations. For example, when executing the "move_robot()" function, the robot assistant will move to the specified position.

Furthermore, a feedback system is in place that can provide contextual prompts based on the digital twin's status. If there is a syntax error, it will prompt "Missing semicolon in line 8: Virtual projector freeze check code structure". When encountering a logical error, it will display "Detected infinite loop: Robot helper battery level is 1%—Add termination condition". For areas that can be optimized, suggestions similar to "Redundant variable 'x' in line 12: Virtual sensor processing delay, simplify code" will also be given. Of course, such feedback is generated by the DT teaching system (especially its middleware layer, including code parsers and simulation choreographers), while DT technology only exhibits behavioral responses (such as stagnation).

Finally, there is the collaborative space, a shared virtual "laboratory area" where students can invite their classmates to view their digital twin instances. It includes features

such as synchronous code editing, voice chat, and annotation tools, including highlighting errors in peer code by drawing arrows.

3.3.5. Technology Justification

The technical components of the DT teaching system and control platform were selected based on educational suitability, classroom feasibility, and the capacity to address traditional tool limitations.

(1) Unity3D for 3D Visualization

Unity3D was chosen as the core engine for rendering the virtual smart classroom, balancing fidelity and accessibility. Its robust 3D pipeline supports interactive elements critical to programming education, including articulated virtual robots, responsive sensors, and context-aware lighting. Compared to Unreal Engine, Unity3D operates efficiently on standard university hardware (eliminating GPU requirements) and offers pre-built educational assets (e.g., robot controllers) that accelerate development by ~40% without compromising pedagogical value. These features enable visualization of abstract concepts (e.g., loop iterations as robot patrol patterns) in classroom-friendly environments.

(2) Python/Flask for Backend Logic

Python with Flask was selected for backend implementation due to its educational compatibility and prototyping efficiency. The middleware layer leverages Python's ecosystem for real-time code parsing (via ANTLR4 integration) and data visualization (e.g., battery consumption graphs). Unlike Java, Python's concise syntax and dynamic typing facilitated rapid iteration of feedback logic, while Flask's modular design ensured sub-500 ms response times between code submission and virtual environment updates.

(3) LeetCode-like Platform for Control Group

The control group used a text-based coding platform to isolate DT-specific advantages. This tool epitomizes traditional programming education, supporting code execution and syntax checking but lacking visualization. The key limitations include abstract error messages (e.g., "Syntax error" without real-world context), static text output, and decontextualized tasks. These gaps highlight the DT teaching system's unique value in bridging code and tangible outcomes.

3.4. Data Collection Tools

The research data were collected at three time points: before the intervention began (week 1), during the intervention (week 4), and after the intervention ended (week 8). A mixed-method research approach was employed to combine quantitative and qualitative measurement methods, thereby verifying the survey results from multiple perspectives.

3.4.1. Quantitative Measures

(1) Participation questionnaire

Adapted from the Student Engagement Tool (SEI) [40], it consists of 18 items measured from three dimensions. Among them, there are six items for behavioral engagement, such as "I actively participated in coding tasks during my laboratory time"; six items for cognitive engagement, such as "I deeply thought about why my code was effective or failed"; and six items for emotional engagement, such as "I feel confident when I try to solve new coding problems". All items are scored on a 5-point Likert scale (1 = strongly disagree, 5 = strongly agree) [41], which demonstrates high internal consistency in this study. The Cronbach alpha coefficient of the total score is 0.89, and the alpha coefficients of each subscale are 0.89. A score that is between 0.82 and 0.87 is consistent with previous validation research results [40].

(2) Programming evaluation

This is a 90 min practical test used to evaluate three core skills, including syntax application (30 points), which involves writing code with a correct structure, such as proper loop syntax and function declarations. The algorithm design (40 points) consists of creating a logical sequence to solve a problem, such as designing a function to calculate the average value of an array. Debugging (30 points) refers to identifying and fixing errors in pre-written code, such as correcting conditional statements that cannot handle edge situations. Two senior programming instructors jointly developed this test, and the feasibility of its content was reviewed and confirmed by a small group of three computer science education experts. To establish reliability among raters, two teachers independently rated 20% of the tests, with an intragroup correlation coefficient of 0.92. The difference in ratings was resolved through discussion.

(3) Previous coding experience investigation

This questionnaire consists of 5 items used to evaluate students' self-reported coding exposure, such as "How many hours do you spend coding outside of class per week?", and their confidence in basic computer skills, as indicated by statements like "I feel comfortable using a code editor". This survey helps to divide students into different subgroups for analysis.

3.4.2. Qualitative Measures

Although quantitative measurement was completed earlier and took a dominant position, the use of qualitative research still has practical significance [42]. After the intervention, a study was conducted involving 12 participants, with 6 in each group. The interactive session lasted for 30–45 min, exploring topics such as perspectives on the learning environment (e.g., "How does DT [or the traditional laboratory] influence your understanding of loops?"), motivation and incentives (e.g., "When using DT, did you find yourself more interested in coding tasks? Why or why not?"), and challenges and successes (e.g., "What is the most difficult part of learning programming, and how has DT [or traditional methods] assisted you?"). The interviews were recorded, transcribed verbatim, and anonymized.

All participants filled out a digital log every week, recording the time they invested in coding tasks (both in and out of class), concepts they found easy or difficult, and the strategies used to solve problems (such as "visualizing the loop process with digital twins" or "reviewing examples from textbooks"). These logs were submitted through secure online forms, with a response rate of 91% (sample size of 123).

3.5. Data Analysis

Utilize SPSS 26.0 (Version: IBM SPSS Statistics 26.0, IBM Corporation, Armonk, NY, USA) [43] to conduct an analysis of quantitative data and undertake the subsequent statistical tests. Compare the participation scores and evaluation outcomes between the experimental and control groups, after the intervention. Check the changes in the participation level and skills of the experimental group before and after the intervention. Investigate the differences in results between the different subgroups and stratify them based on previous coding experience, as evaluated through a pre-survey. Calculate the effect size (Cohen's d) using thresholds for mild (d = 0.2), moderate (d = 0.5), and large (d = 0.8) effects to quantify the actual significance of group differences. Follow the steps below to use thematic analysis to analyze qualitative data [44].

(1) Familiarization: Researchers read transcripts and logs repeatedly to identify initial patterns.
(2) Coding: Labels were assigned to meaningful segments (e.g., "DT visualization clarifies loops", "frustration with compiler messages").

(3) Theme development: Codes were grouped into overarching themes (e.g., "feedback quality", "conceptual clarity").
(4) Validation: Themes were reviewed against the original data to ensure accuracy, with a second researcher independently coding 20% of the transcripts (inter-rater reliability, $\kappa = 0.83$).

3.6. Critical Analysis of Methodology

The study's methodological choices involve deliberate trade-offs to balance rigor and real-world applicability.

(1) The study employed a quasi-experimental design due to practical constraints of maintaining class continuity, which meant not all variables could be strictly controlled. However, this design enhanced ecological validity by reflecting real-classroom settings, with baseline equivalence between groups verified through statistical tests to mitigate selection bias.

(2) Potential novelty effects in DT engagement were addressed by collecting mid-intervention data (week 4), which showed sustained higher engagement in the experimental group, indicating effects were not merely due to initial technology novelty.

(3) The sample size (n = 135) was justified through power analysis using Cohen's d. The effect sizes for key outcomes (e.g., overall engagement d = 1.40, algorithm design d = 1.06) confirmed sufficient statistical power to detect meaningful differences between groups.

4. Results

The experimental data results in this article have high reliability and are supported by various methods and analytical safeguards. This study employed a quasi-experimental design, randomly dividing 135 first-year graduate students into an experimental group (n = 90) and a control group (n = 45). The pre-class aptitude test revealed no significant difference in the baseline logical reasoning ability between the two groups (p = 0.68), ensuring comparability of the groups. This equivalence can minimize the impact of confounding variables, thereby enhancing the credibility of the differences caused by digital twin (DT) interventions after the intervention. The experimental results will be evaluated across four dimensions: student participation, skill mastery, the differences between subgroups, and the qualitative survey results.

4.1. Learning Engagement

The comparison of learning engagement between the experimental group and the control group is shown in Table 3.

Table 3. Comparison of learning engagement between experimental and control groups.

Engagement Dimension	Experimental Group (M ± SD)	Control Group (M ± SD)	*t*-Value	*p*-Value	Effect Size (Cohen's *d*)
Overall	4.12 ± 0.53	3.27 ± 0.61	7.83	<0.001	1.40
Mid-intervention	3.89 ± 0.57	3.21 ± 0.63	5.92	<0.001	1.06
Behavioral	4.23 ± 0.48	3.15 ± 0.59	10.14	<0.001	1.81
Cognitive	4.01 ± 0.55	3.32 ± 0.64	6.37	<0.001	1.14
Emotional	4.16 ± 0.51	3.36 ± 0.57	7.29	<0.001	1.31
On-task time (minutes/hour)	42.3 ± 5.8	28.7 ± 6.2	10.56	<0.001	-

Note: M = mean; SD = standard deviation. Mid-intervention engagement data were collected at week 4. On-task time reflects minutes spent actively coding per lab hour.

The experimental group scored significantly higher in overall engagement at post-intervention (M = 4.12, SD = 0.53) compared to the control group (M = 3.27, SD = 0.61) (t (135) = 7.83, $p < 0.001$, $d = 1.40$), indicating a significant practical effect. This difference emerged by mid-intervention (week 4), with the experimental group already showing higher engagement (M = 3.89, SD = 0.57) than the control group (M = 3.21, SD = 0.63) (t (135) = 5.92, $p < 0.001$, $d = 1.06$), suggesting sustained benefits of the DT environment.

Analyzing student participation in the "Fundamentals of Programming" course from various dimensions, Table 1 reveals significant differences between the experimental group (using digital twin technology) and the control group (traditional teaching). The more obvious observation results are shown in Figure 3. It visualizes the differences between the experimental group and the control group, visually presenting the improvement effect of digital twins on participation in different dimensions and enhancing the persuasiveness of the results.

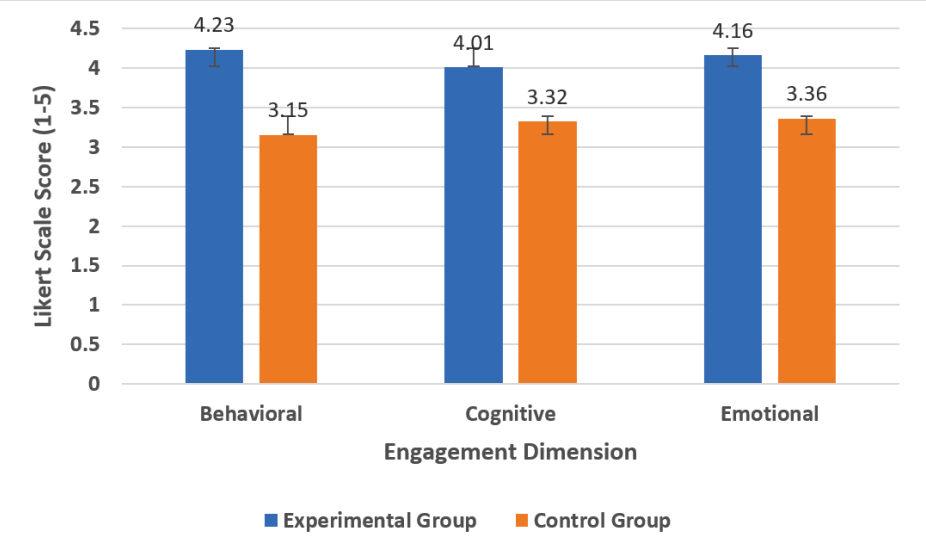


Figure 3. Bar–chart comparing various dimensions of learning engagement.

(1) Behavioral Engagement

The experimental group demonstrated significantly higher active participation, with a mean score of 4.23 (SD = 0.48). In learning logs, it was reflected that they spent more time on tasks during labs, averaging 42.3 min per hour (SD = 5.8). The interactive nature of the digital twin environment motivated their involvement, as they looked forward to labs because they could see their code directly control the movement of virtual robots, which was more engaging than simply typing code into a blank screen.

Control group: The mean score for behavioral engagement was 3.15 (SD = 0.59), and students spent less time on task during labs, averaging 28.7 min per hour (SD = 6.2). Their participation was more passive, mainly focused on replicating predefined code or completing isolated exercises.

(2) Cognitive Engagement

Experimental group: Students demonstrated a deeper understanding of programming logic, with a mean score of 4.01 (SD = 0.55). Learning logs indicated that they were more likely to reflect on errors, such as analyzing why a loop failed by observing the virtual robot getting stuck. They also asked more conceptual questions (e.g., about the importance of parameter order in functions) rather than just procedural ones.

Control group: The mean score for cognitive engagement was 3.32 (SD = 0.64). Students often expressed confusion about why their code did not work and tended to focus on superficial syntax issues rather than exploring underlying logical principles.

(3) Emotional Engagement

Experimental group: This dimension demonstrated the most significant relative improvement (+23.8% from pre-intervention), with a mean score of 4.16 (SD = 0.51). Eighty-two percent of interviewees mentioned that the "low-stakes experimentation" in the digital twin environment reduced coding anxiety. For example, if they made a mistake, the virtual robot would stop, which did not feel like a significant failure, encouraging them to try harder.

Control group: The mean score for emotional engagement was 3.36 (SD = 0.57). Students reported frustration more frequently, such as disliking seeing "error" messages on the screen without clear explanations of what went wrong.

Overall, the digital twin teaching environment has shown significant effects in enhancing students' behavior, cognition, and emotional engagement, particularly in terms of emotional participation. This fully demonstrates its potential to create more positive and practical learning experiences.

4.2. Programming Skill Acquisition

The experimental group achieved significantly higher scores on the post-intervention programming assessment (M = 76.3, SD = 10.2) than the control group (M = 65.7, SD = 11.5) ($t(135) = 4.59$, $p < 0.001$, d = 0.82), indicating a medium-to-large effect size. This advantage was consistent across all three skill dimensions, with the most significant gap in algorithm design.

The comparison of scores in various dimensions of programming skill acquisition is presented in Table 4, which quantitatively illustrates the differences between the two groups in core skills, supporting the conclusion that "digital twins enhance skill acquisition".

Table 4. Comparison table of scores for various dimensions of programming skills.

Skill Dimension	Experimental Group (M ± SD)	Control Group (M ± SD)	Mean Difference	*t*-Value	*p*-Value	Effect Size (*d*)
Grammar Application (30 points)	25.2 ± 3.8	22.1 ± 4.3	+3.1	3.87	<0.001	0.69
Algorithm Design (40 points)	31.5 ± 5.2	24.8 ± 6.1	+6.7	5.86	<0.001	1.06
Debugging Ability (30 points)	23.7 ± 4.1	18.8 ± 4.7	+4.9	5.02	<0.001	0.91
Comprehensive Score (100 points)	76.3 ± 10.2	65.7 ± 11.5	+10.6	4.59	<0.001	0.82

(1) Syntax application

The experimental group scored higher (M = 25.2, SD = 3.8) than the control group (M = 22.1, SD = 4.3) (t (135) = 3.87, $p < 0.001$, $d = 0.69$). Instructors noted that the DT's real-time syntax validation helped students internalize the rules: "Students in the DT group rarely missed semicolons by week 6, because they had learned to associate the red underline with the robot freezing."

(2) Algorithm design

The most substantial gap occurred in this dimension (experimental M = 31.5, SD = 5.2; control M = 24.8, SD = 6.1) (t (135) =5.86, $p < 0.001$, $d = 1.06$). The DT's visualization of data

flow appeared critical: "Watching the robot follow my algorithm step by step helped me see where the logic broke down" (Student E8). For example, 78% of the experimental group correctly designed a function to sort sensor data, compared to 49% of the control group.

(3) Debugging ability

The experimental group outperformed the control group (M = 23.7, SD = 4.1 vs. M = 18.8, SD = 4.7) (t (135) =5.02, $p < 0.001$, $d = 0.91$). DT users were better at tracing errors to their root cause, as noted in learning logs: "Used the DT to pause the code—saw the variable was not updating because I forgot to increment it" (Student E15). Control group students more often relied on trial and error: "I changed random things until the code worked" (Student C12).

The radar chart comparing programming skill scores is shown in Figure 4, which compares the performance of different skill dimensions and highlights the improvement of core skills, such as algorithm design, facilitated by digital twins.

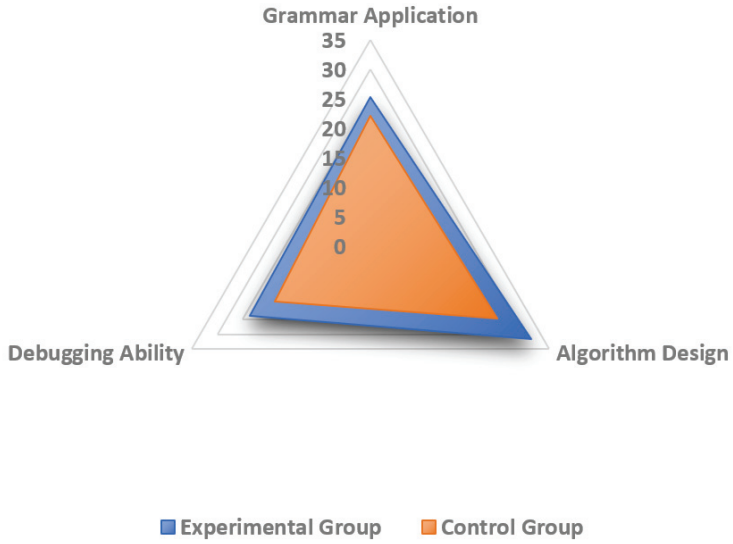


Figure 4. Radar chart comparing programming skill scores.

4.3. Subgroup Differences

Based on their pre-class aptitude test scores, students were divided into three subgroups: low (minimum 30%, $n = 41$), medium (moderate, 40%, $n = 54$), and high (maximum 30%, $n = 40$). Analysis of variance revealed that group–subgroup interactions were significant ($p < 0.05$) in terms of participation and skill level, with the most pronounced benefits observed among students with lower abilities. The comparison of skill improvement among students with different ability levels is shown in Table 5, highlighting the significant improvement effect of digital twins on low-ability students and reflecting the value of educational equity.

Table 5. Comparison table of skill enhancement among students with different ability levels.

Ability Grouping	Experimental Group (Pre-Test → Post-Test)	Increase Amplitude (EG)	Control Group (Pre-Test → Post-Test)	Increase Amplitude (CG)	Effect Size (d)
Low ability ($n = 41$)	52.7 → 64.0	+21.3%	51.9 → 56.4	+8.7%	0.02
Intermediate ability ($n = 54$)	67.3 → 77.8	+15.6%	66.8 → 73.0	+9.2%	0.05
High ability ($n = 40$)	78.5 → 87.0	+10.8%	79.1 → 85.0	+7.5%	0.15

The line charts illustrating skill improvement for students with different abilities in the experimental group are presented in Figure 5. In contrast, those for students with varying

skills in the control group are shown in Figure 6. These displays of changes before and after intervention demonstrate the improvement effect of digital twins on vulnerable groups, supporting the conclusion of "promoting educational equity".

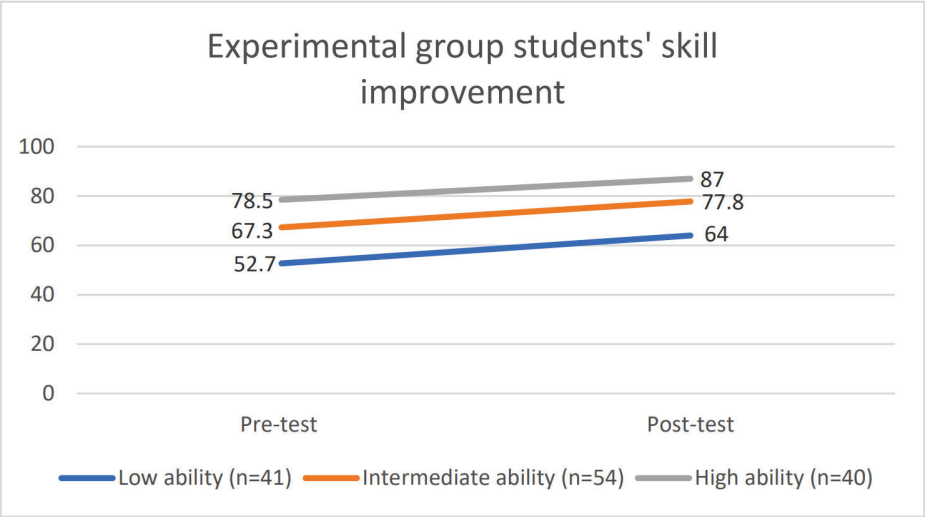


Figure 5. Experimental group students' skill improvement.

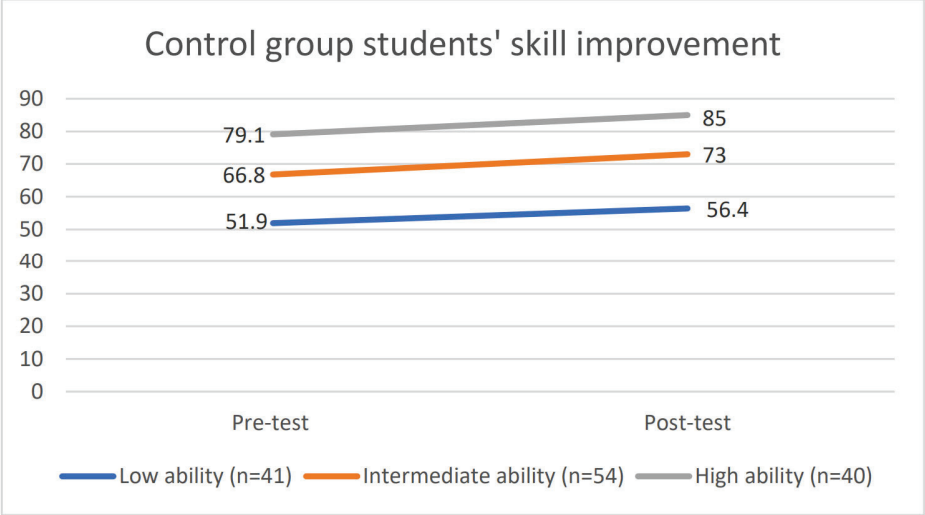


Figure 6. Control group students' skill improvement.

For the group of low-qualified students, the evaluation scores of the experimental group increased by 21.3% (from a pre-intervention average score of 52.7 to a post-intervention average score of 64.0). In comparison, the evaluation scores of the control group increased by 8.7% (from a pre-intervention average score of 51.9 to a post-intervention average score of 56.4). After testing (F (1,39) = 5.24, p = 0.02, η^2 = 0.12), a significant difference was found between the two groups. The role of digital twins in helping to understand abstract concepts is evident in the interview, as stated by low-talent student E22: "Before seeing robots counting with my own eyes, I had only a vague understanding of loops—now I finally understand its meaning."

In the performance of students with moderate abilities, the experimental group improved by 15.6% (average score before intervention: 67.3, post-intervention: 77.8), while the control group improved by 9.2% (average score before intervention: 66.8, post-intervention: 73.0). After testing, there was a certain difference (F (1,52) =3.98, p = 0.05, η^2 = 0.07). This type of student values the feedback provided by digital twins to improve their skills, as

student E9 said, "Digital twins indicate that my function is inefficient, so I learned to simplify it."

For the group of high-talent students, both subgroups showed a performance improvement; the experimental subgroup showed a 10.8% increase in scores (with an average score of 78.5 before intervention and 87.0 after intervention). In comparison, the control subgroup showed a 7.5% increase (with an average score of 79.1 before intervention and 85.0 after intervention). After testing (F (1,38) =2.11, p = 0.15, η^2 = 0.05), the difference between the two groups was not significant. However, these talented students often use the collaborative capabilities of digital twins to explore more advanced concepts, as student E4 said, "We have designed a more complex robot path by integrating various functions—something that cannot be attempted in traditional laboratories."

4.4. Qualitative Findings

The thematic analysis of interviews and study logs identified three key themes.

Theme 1: DT visualization bridges the gap between abstraction and concreteness.

Students in the experimental group often noticed that seeing code come to life in the virtual smart classroom helps them grasp abstract concepts more effectively. For example, a student struggling with functions explained: "I did not get why we need parameters until I programmed the robot to move different distances—changing the number in the function made it go farther" (Student E17). Control group students more often described concepts as "just words on a page" (Student C5).

Theme 2: Real-time feedback enhances learning iteration.

DT users emphasized the value of immediate, contextual feedback: "When the DT said 'sensor data not read', I knew to check my input function instead of guessing" (Student E8). Control group students, relying on delayed feedback, reported frustration: "I would forget what I was thinking when I wrote the code by the time I got it back" (Student C14).

Theme 3: Low-stakes experimentation reduces anxiety.

Emotional engagement was closely tied to the DT's sense of a safe environment. As one student put it: "In traditional labs, I was scared to try new code because I might break something. In the DT, I can mess up and just reset—so I take more risks" (Student E19). This kind of adventurous behavior often leads to deeper learning as students explore edge cases that they would not otherwise test (such as "What if the loop runs zero times?").

5. Discussion

5.1. Enhancing Engagement Through Digital Twins

The findings confirm that digital twin technology has a significant impact on learning engagement in introductory programming, with substantial effects on behavioral, cognitive, and emotional dimensions. These results align with self-determination theory [45], which posits that engagement is driven by three psychological needs, autonomy, competence, and relatedness, all of which are fulfilled by the DT environment. This section discusses the mechanism of increasing participation through digital twins. Figure 7 presents the mechanism of growing participation through digital twins, providing a visual theoretical framework to transform abstract theories into visual models that effectively explain the inherent logic of digital twins in increasing participation.

This concept diagram illustrates how the digital twin environment can enhance student engagement by meeting the three core needs of self-determination theory.

Autonomy: This is achieved through "independent task selection" and "custom virtual scenes", cultivating greater behavioral engagement as students master their learning. The interactivity of DT enables students to guide their learning, from selecting robot actions to attempting alternative solutions. This sense of control does not exist in traditional "copy

and paste" laboratories, as it increases behavioral engagement: "I am not just doing what the teacher says—I have to decide what the robot does" (Student E7).

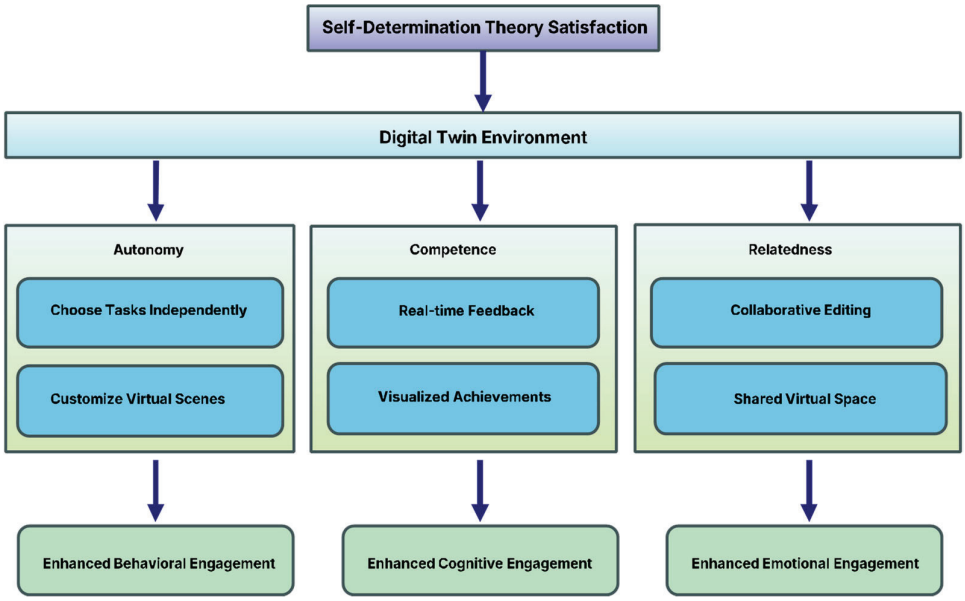


Figure 7. Conceptual diagram of mechanisms for digital twin-enhanced engagement.

Competence: This is supported by "Real-time Feedback" and "Visualized Achievements", strengthening cognitive engagement through clear progress validation. The instant feedback in DT technology provides clear evidence of progress (such as "the robot is moving correctly—functioning properly!") and cultivates a sense of mastery. This increases cognitive engagement, as students become motivated to tackle more complex tasks once they feel capable of doing so.

Relatedness: This is facilitated by "Collaborative Editing" and "Shared Virtual Space", boosting emotional engagement via social connection. The DT's collaborative space allowed students to share challenges and successes, fulfilling the need for social connection. This reduced feelings of isolation, often reported in programming courses, enhancing emotional engagement [46].

Notably, emotional engagement showed the most significant relative improvement, suggesting that DTs address a critical pain point in programming education: anxiety. The safe, consequence-free environment of the DT reduced the fear of failure, encouraging persistence. As a student once said, "I used to give up when the code did not work, now I think, ' Let me try to fix it', because DT told me where to look" (Student E11). This finding is consistent with relevant research, which has shown that positive emotional experiences generated during the learning process are often associated with higher memory retention rates.

5.2. Skill Development Mechanisms

The excellent performance of the experimental group in skill mastery fully demonstrates the significant effectiveness of digital twin (DT) technology in cultivating students' practical programming abilities. Three mechanisms can explain this result.

(1) Instant and situational feedback

The real-time feedback loop of DT technology, where code changes immediately alter the state of the virtual system, enables rapid knowledge correction. This is consistent with the iterative learning model [47], which emphasizes that skill mastery requires frequent and timely feedback. For example, students who declare variables incorrectly in a DT will

see the virtual sensor display "undefined", which serves as a direct prompt to reexamine the variable syntax. This is in stark contrast to traditional environments, where feedback is often delayed and disconnected from its context, thereby disrupting the connection between action and outcome.

(2) Visualization of abstract logic

A DT's animation of algorithm flows (such as loops and data processing) helps students develop computational thinking and the ability to break down problems into logical steps. This is especially beneficial for students with a low ability, as it provides an alternative to pure symbolic reasoning. This supports the cognitive load theory [35], which suggests that allocating information through visual and textual channels can reduce mental labor and free up resources for deeper learning.

(3) Authentic problem-solving

A DT's simulation of real-world scenarios (such as programming for the virtual smart classroom) makes coding tasks relevant and motivates students to develop transferable skills. For example, when debugging the motion algorithm of a robot assistant, not only is grammar knowledge required but logical reasoning ability also is—this is the core ability in the field of professional programming. This is in stark contrast to traditional textbook exercises, which often appear rigid and deliberate, making it challenging to cultivate skills that can be transferred to practical scenarios [48].

5.3. Addressing Equity in Programming Education

The significant progress of low-talent students suggests that digital twins may help narrow the achievement gap in programming education. Traditional teaching often fails to benefit students with limited abstract reasoning abilities, as they struggle to transform symbolic codes into mental models [49]. The multimodal representation of DT technology (visual and interactive) offers an alternative approach to understanding, aligning with the design of a universal learning framework [50], which advocates for multiple representation methods to accommodate diverse learners.

For example, a student with low spatial reasoning ability may struggle to visualize cyclic behavior in their mind. However, they can master it by observing the iterations of virtual robots. This ease of operation is of great significance, as programming education has long excluded students with different learning needs [51]. Digital twins (DTs) create a more inclusive learning environment by offering various participatory concepts.

It is worth mentioning that high-talent students can also utilize digital twins to explore advanced applications, but the improvement is relatively small. This suggests that digital twins can cater to the diverse needs of learners, laying a foundation for beginners while providing advanced project resources for more experienced students.

5.4. Limitations

Several limitations are worth noting when interpreting the research results in this study. Firstly, the expected duration of the intervention is relatively short, only 8 weeks, which suggests that students' long-term retention of skills may be insufficient. Although the digital twin group has shown immediate results, further research still needs to track students' performance in subsequent courses (such as "data structures") to evaluate the transferability of skills. Secondly, there are specific obstacles related to technology and cost. This DT platform not only requires moderate computing power support, such as 8GB of memory and dedicated graphics cards, but also requires two developers to spend about 200 h on development, which may limit its promotion and application in resource-limited institutions. In addition, the research sample only comes from one university, whose

demographic data reflect trends in the region, such as gender distribution. Therefore, the research findings may be difficult to generalize to other scenarios, such as online courses or institutions with more diverse student populations. Additionally, a potential novelty effect suggests that the initial increase in participation by the digital twin population may be partially due to the novelty of the technology. Therefore, it is necessary to conduct longitudinal research to investigate whether students' learning motivation remains high after long-term use.

5.5. Future Research Directions

To overcome the limitations of existing research and deepen research results, future exploration can be carried out from multiple aspects: Firstly, we should focus on the impact of digital twin-based learning models on long-term outcomes, specifically tracking their roles in advanced programming course performance and career preparation, to clarify the sustained value of this learning model. Secondly, efforts should be made to develop digital twin platforms that rely on networks or mobile devices. These platforms should minimize hardware requirements so that educational institutions with relatively scarce resources can easily use them and expand the application scope of technology. In addition, artificial intelligence can be combined with digital twin technology to achieve personalized feedback, such as dynamically adjusting the learning difficulty based on students' error patterns, providing targeted concept reviews for students' weak links, and improving the learning efficiency of different learners. Finally, to comprehensively consider the universal applicability of the research conclusions, existing research results still need to be validated in online courses, vocational training programs, and various institutions with a more diverse student composition.

6. Conclusions and Recommendations

6.1. Conclusions

Digital twin technology has had a significant positive impact on learning participation in the "Programming Fundamentals" course, as reflected in multiple dimensions, including behavior, cognition, and emotion. Emotional engagement showed the most significant improvement, as the DT's secure environment reduced coding anxiety. The learning environment built on digital twins (DTs) can effectively enhance students' mastery of core programming skills, especially in algorithm design and code debugging. This is due to the real-time visualization of the code running process, the feedback mechanism that fits the scene, and the integration of real-world problem-solving tasks. It is worth noting that low-talent students benefit more significantly from digital twin technology, which means that digital technology is expected to narrow the gap in academic performance and provide strong support for the fairness of programming education.

6.2. Teaching Recommendations

In terms of course integration, 30% to 50% of traditional laboratory time can be replaced with activities based on digital twins, with a focus on conceptual visualization, such as using virtual sensors to explain variables and making abstract variable concepts more intuitive through changes in sensor data. Simultaneously, conducting interactive debugging, such as tracking loop errors through robot movements, links logical issues in the code with the actual actions of the robot, making the debugging process easier to understand. It is also necessary to integrate project-based learning, guide students to write virtual system code related to the real world, and enhance the practicality of learning. In differentiated teaching, the adaptability of digital twins should be utilized to tailor tasks according to the needs of different learners. For students with low talent, a simplified virtual

system should be provided, such as one that involves only one robot action, supplemented by guided feedback to help them gradually master the knowledge. For high-talent students, complex scenarios such as open challenges with multi-sensor integration are designed to stimulate their curiosity and innovation ability. For instructor training, it is necessary to equip educators with the ability to design digital twin activities that align with course objectives, such as linking robot motion with the concept of circulation, to make teaching activities more targeted. We also need to learn how to interpret the data generated by digital twins, such as analyzing common error patterns, to identify students' misunderstandings and adjust teaching accordingly and promptly. At the same time, this should also promote collaborative learning in the digital twin environment such as guiding students to conduct peer code review activities to cultivate their cooperation and communication skills.

6.3. Final Remarks

Digital twin technology represents a transformative approach to "Fundamentals of Programming" education, addressing key challenges in participation and skill acquisition. By creating an immersive interactive learning environment that reflects the real-world programming environment, DT bridges the gap between abstract code and tangible results. This not only enhances current learning outcomes but also cultivates the perseverance and algorithmic thinking necessary for long-term development in the field of computer science. With the continuous advancement of educational technology, digital twins have outlined an inclusive, interactive, and practical blueprint for programming education, enabling students not only to write code proficiently but also to thrive and excel in this technology-driven world.

Author Contributions: Conceptualization, M.L. and Z.H.; methodology, M.L.; software, M.L.; validation, M.L. and Z.H.; formal analysis, M.L.; investigation, M.L.; resources, Z.H.; data curation, M.L.; writing—original draft preparation, M.L.; writing—review and editing, M.L.; visualization, M.L.; supervision, Z.H.; project administration, Z.H.; funding acquisition, M.L. All authors have read and agreed to the published version of the manuscript.

Funding: M.L. has been awarded funding from the Collaborative Education Project of the Ministry of Education of China (Project No. 220903880215602) and the Teaching Research and Reform Project of Wenzhou University (Project No. jg2024046). This includes the article processing fee (APC) and revision of the English manuscript.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. (Due to privacy or ethical limitations, this data is not publicly available).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Nicolajsen, S.M.; Nielsen, S.; Carlsen, L.M.; Brabrand, C. Programming education across disciplines: A nationwide study of Danish higher education. *High. Educ.* **2024**, 1–27. [CrossRef]
2. Lee, H.Y.; Lin, C.J.; Wang, W.S.; Chang, W.C.; Huang, Y.M. Precision education via timely intervention in K-12 computer programming course to enhance programming skill and affective-domain learning objectives. *Int. J. STEM Educ.* **2023**, *10*, 52. [CrossRef]
3. Groher, I.; Vierhauser, M.; Sabitzer, B.; Kuka, L.; Hofer, A.; Muster, D. Exploring diversity in introductory programming classes: An experience report. In Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training, Pittsburgh, PA, USA, 22–24 May 2022. [CrossRef]
4. Giannakoulas, A.; Xinogalos, S. Studying the effects of educational games on cultivating computational thinking skills to primary school students: A systematic literature review. *J. Comput. Educ.* **2024**, *11*, 1283–1325. [CrossRef]
5. Zhan, Q.; Wang, J.; Pan, X.; Ding, Y.; Liu, Y. Teaching Model Design of Computer Programming Courses for Digital Media Technology Students. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 7085914. [CrossRef]

6. LoSchiavo, F.M. How to Create Automatically Graded Spreadsheets for Statistics Courses. *Teach. Psychol.* **2016**, *43*, 147–152. [CrossRef]
7. She, M.; Xiao, M.; Zhao, Y. Technological Implication of the Digital Twin Approach on the Intelligent Education System. *Int. J. Hum. Robot.* **2023**, *20*, 2250005. [CrossRef]
8. Tao, F.; Cheng, J.; Qi, Q.; Zhang, M.; Zhang, H.; Sui, F. Digital twin-driven product design, manufacturing and service with big data. *Int. J. Adv. Manuf. Technol.* **2018**, *94*, 3563–3576. [CrossRef]
9. Deren, L.; Wenbo, Y.; Zhenfeng, S. Smart city based on digital twins. *Comput. Urban. Sci.* **2021**, *1*, 4. [CrossRef]
10. Tarng, W.; Wu, Y.J.; Ye, L.Y.; Tang, C.W.; Lu, Y.C.; Wang, T.L.; Li, C.L. Application of Virtual Reality in Developing the Digital Twin for an Integrated Robot Learning System. *Electronics* **2024**, *13*, 2848. [CrossRef]
11. Dihan, M.S.; Akash, A.I.; Tasneem, Z.; Das, P.; Das, S.K.; Islam, M.R.; Islam, M.M.; Badal, F.R.; Ali, M.F.; Ahamed, M.H.; et al. Digital twin: Data exploration, architecture, implementation and future. *Heliyon* **2024**, *10*, e26503. [CrossRef]
12. Fan, G.; Liu, D.; Zhang, R.; Pan, L. The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: A comparative study with traditional pair programming and individual approaches. *Int. J. STEM Educ.* **2025**, *12*, 16. [CrossRef]
13. Maloney, J.; Resnick, M.; Rusk, N.; Silverman, B.; Eastmond, E. The scratch programming language and environment. *ACM Trans. Comput. Educ.* **2010**, *10*, 1–15. [CrossRef]
14. Kroustalli, C.; Xinogalos, S. Studying the effects of teaching programming to lower secondary school students with a serious game: A case study with Python and CodeCombat. *Educ. Inf. Technol.* **2021**, *26*, 6069–6095. [CrossRef]
15. Dong, Z.; Han, X.; Shi, Y.; Zhai, W.; Luo, S. Development of Manipulator Digital Twin Experimental Platform Based on RCP. *Electronics* **2022**, *11*, 4196. [CrossRef]
16. Hananto, A.L.; Tirta, A.; Herawan, S.G.; Idris, M.; Soudagar, M.E.M.; Djamari, D.W.; Veza, I. Digital Twin and 3D Digital Twin: Concepts, Applications, and Challenges in Industry 4.0 for Digital Twin. *Computers* **2024**, *13*, 100. [CrossRef]
17. Iliuţă, M.-E.; Moisescu, M.-A.; Pop, E.; Ionita, A.-D.; Caramihai, S.-I.; Mitulescu, T.-C. Digital Twin—A Review of the Evolution from Concept to Technology and Its Analytical Perspectives on Applications in Various Fields. *Appl. Sci.* **2024**, *14*, 5454. [CrossRef]
18. Ye, W.; Liu, X.; Zhao, X.; Fu, H.; Cai, Y.; Li, H. A Data-Driven Digital Twin Architecture for Failure Prediction of Customized Automatic Transverse Robot. *IEEE Access* **2024**, *12*, 59222–59235. [CrossRef]
19. Holman, T. The best of CodePen. *Net* **2016**, *283*, 76–81.
20. Wielemaker, J.; Lager, T.; Riguzzi, F. SWISH: SWI-Prolog for Sharing. *arXiv* **2015**. [CrossRef]
21. Pitzalis, R.F.; Giordano, A.; Di Spigno, A.; Cowell, A.; Niculita, O.; Berselli, G. Application of Augmented Reality-Based Digital Twin Approaches: A Case Study in Industrial Equipment. *Int. J. Adv. Manuf. Technol.* **2025**, *138*, 3747–3763. [CrossRef]
22. Filipescu, A.; Simion, G.; Ionescu, D. IoT-Cloud, VPN, and Digital Twin-Based Remote Monitoring and Control of a Multifunctional Robotic Cell in the Context of AI, Industry, and Education 4.0 and 5.0. *Sensors* **2024**, *24*, 7451. [CrossRef]
23. Erceylan, G.; Akbarzadeh, A.; Gkioulos, V. Leveraging digital twins for advanced threat modeling in cyber-physical systems cybersecurity. *Int. J. Inf. Secur.* **2025**, *24*, 151. [CrossRef]
24. Al Hakim, V.G.; Yang, S.H.; Wang, J.H.; Lin, H.H.; Chen, G.D. Digital Twins of Pet Robots to Prolong Interdependent Relationships and Effects on Student Learning Performance. *IEEE Trans. Learn. Technol.* **2024**, *17*, 1883–1897. [CrossRef]
25. Fredricks, J.A.; Blumenfeld, P.C.; Paris, A.H. School engagement: Potential of the concept, state of the evidence. *Rev. Educ. Res.* **2004**, *74*, 59–109. [CrossRef]
26. Toukiloglou, P.; Xinogalos, S. Effects of Collaborative Support on Learning in Serious Games for Programming. *J. Educ. Comput. Res.* **2025**, *63*, 126–146. [CrossRef]
27. Gebre, E.; Saroyan, A.; Bracewell, R. Students' engagement in technology rich classrooms and its relationship to professors' conceptions of effective teaching. *Br. J. Educ. Technol.* **2014**, *45*, 83–96. [CrossRef]
28. Hong, W.; Zhen, R.; Liu, R.D.; Wang, M.T.; Ding, Y.; Wang, J. The longitudinal linkages among Chinese children's behavioural, cognitive, and emotional engagement within a mathematics context. *Educ. Psychol.* **2020**, *40*, 666–680. [CrossRef]
29. Silva, L.; Mendes, A.; Gomes, A.; Fortes, G. What Learning Strategies are Used by Programming Students? A Qualitative Study Grounded on the Self-regulation of Learning Theory. *ACM Trans. Comput. Educ.* **2024**, *24*, 9. [CrossRef]
30. Berry, C.; Walcott, M. Reducing Caribbean's Students' "Code-Phobia" with Programming in Scratch. In Proceedings of the 2019 ACM Conference on International Computing Education Research, Toronto, ON, Canada, 12–14 August 2019. [CrossRef]
31. Vishnu, S.; Sahil; Garg, N. Unveiling the Role of GPT-4 in Solving LeetCode Programming Problems. *Comput. Appl. Eng. Educ.* **2025**, *33*, e22815. [CrossRef]
32. Bao, L.; Xing, Z.; Xia, X.; Lo, D. VT-Revolution: Interactive Programming Video Tutorial Authoring and Watching System. *IEEE Trans. Softw. Eng.* **2019**, *45*, 823–838. [CrossRef]
33. Wang, Y.; Liang, Y.; Liu, L.; Liu, Y. A multi-peer assessment platform for programming language learning: Considering group non-consensus and personal radicalness. *Interact. Learn. Environ.* **2016**, *24*, 2011–2031. [CrossRef]

34. Yun, H.; Jun, M.B.G. Immersive and interactive cyber-physical system (I2CPS) and virtual reality interface for human involved robotic manufacturing. *J. Manuf. Syst.* **2022**, *62*, 234–248. [CrossRef]
35. Yu, J.H.; Chang, X.Z.; Liu, W.; Huan, Z. An online integrated programming platform to acquire students' behavior data for immediate feedback teaching. *Comput. Appl. Eng. Educ.* **2023**, *31*, 520–536. [CrossRef]
36. Saville, B.K.; Pope, D.; Lovaas, P.; Williams, J. Interteaching and the Testing Effect: A Systematic Replication. *Teach. Psychol.* **2012**, *39*, 280–283. [CrossRef]
37. Roediger, H.L.; Karpicke, J.D. The power of testing memory: Basic research and implications for educational practice. *Perspect. Psychol. Sci.* **2006**, *1*, 181–210. [CrossRef]
38. Kirschner, P.A.; Sweller, J.; Kirschner, F.; Zambrano, R.J. From Cognitive Load Theory to Collaborative Cognitive Load Theory. *Int. J. Comput.-Support. Collab. Learn.* **2018**, *13*, 213–233. [CrossRef]
39. Zourmpakis, A.I.; Kalogiannakis, M.; Papadakis, S. The Effects of Adaptive Gamification in Science Learning: A Comparison Between Traditional Inquiry-Based Learning and Gender Differences. *Computers* **2024**, *13*, 324. [CrossRef]
40. Appleton, J.J.; Christenson, S.L.; Furlong, M.J. Student engagement with school: Critical conceptual and methodological issues of the construct. *Psychol. Sch.* **2008**, *45*, 369–386. [CrossRef]
41. Matosas-López, L.; Leguey-Galán, S.; Regaña, C.B.; Piris, N.P. University and Quality Systems. Evaluating faculty performance in face-to-face and online programs: A comparison of Likert and BARS instruments. *Int. J. Educ. Res. Innov.* **2024**, *22*, 18. [CrossRef]
42. Braun, V.; Clarke, V. Is thematic analysis used well in health psychology? A critical review of published research, with recommendations for quality practice and reporting. *Health Psychol. Rev.* **2023**, *17*, 695–718. [CrossRef]
43. Moon, S.; Yeo, G.; Kim, Y.; Oh, J. A study on the consumer behavior and attitude toward low-sodium convenience store foods. *Nutr. Res. Pract.* **2024**, *18*, 567–585. [CrossRef]
44. Faul, F.; Erdfelder, E.; Lang, A.G.; Buchner, A. G*Power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behav. Res. Methods* **2007**, *39*, 175–191. [CrossRef] [PubMed]
45. Ryan, R.M.; Deci, E.L. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *Am. Psychol.* **2000**, *55*, 68–78. [CrossRef] [PubMed]
46. Goetz, T.; Frenzel, A.C.; Pekrun, R.; Hall, N.C.; Lüdtke, O. Between- and within-domain relations of students' academic emotions. *J. Educ. Psychol.* **2007**, *99*, 715–733. [CrossRef]
47. Robins, A.; Rountree, J.; Rountree, N. Learning and Teaching Programming: A Review and Discussion. *Comput. Sci. Educ.* **2003**, *13*, 137–172. [CrossRef]
48. Mccracken, M.; Wilusz, T.; Almstrum, V.; Diaz, D.; Guzdial, M.; Hagan, D.; Kolikant, Y.B.; Laxer, C.; Thomas, L.; Utting, I. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education, Canterbury, UK, 1 December 2001. [CrossRef]
49. Bosse, Y.; Gerosa, M.A. Difficulties of Programming Learning from the Point of View of Students and Instructors. *IEEE Lat. Am. Trans.* **2017**, *15*, 2191–2199. [CrossRef]
50. Morgan, A. Enhancing Access in an Online Course Using Universal Design for Learning (UDL) and Scenario-Based Learning (SBL). *TechTrends* **2024**, *68*, 904–913. [CrossRef]
51. Fisher, A.; Margolis, J. Unlocking the clubhouse: Women in computing. In Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, Navada, NV, USA, 19–23 February 2003. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

computers

MDPI

Article

Teach Programming Using Task-Driven Case Studies: Pedagogical Approach, Guidelines, and Implementation

Jaroslav Porubän [1,*], Milan Nosáľ [2], Matúš Sulír [1] and Sergej Chodarev [1,*]

[1] Department of Computers and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia; matus.sulir@tuke.sk
[2] ValeSoft, s.r.o., Vodárenská 636/3, 040 01 Košice, Slovakia; milan.nosal@gmail.com
* Correspondence: jaroslav.poruban@tuke.sk (J.P.); sergej.chodarev@tuke.sk (S.C.)

Abstract: Despite the effort invested to improve the teaching of programming, students often face problems with understanding its principles when using traditional learning approaches. This paper presents a novel teaching method for programming, combining the task-driven methodology and the case study approach. This method is called a *task-driven case study*. The case study aspect should provide a real-world context for the examples used to explain the required knowledge. The tasks guide students during the course to ensure that they will not fall into bad practices. We provide reasoning for using the combination of these two methodologies and define the essential properties of our method. Using a specific example of the Minesweeper case study from the Java technologies course, the readers are guided through the process of the case study selection, solution implementation, study guide writing, and course execution. The teachers' and students' experiences with this approach, including its advantages and potential drawbacks, are also summarized.

Keywords: computer science education; programming; software engineering; computer-aided instruction; task-driven methodology; case study approach

1. Introduction

The efficient teaching of programming is a difficult task. Often, students know the principles and processes, but they are not able to use them effectively in practice. There are even cases where professional developers have problems with applying basic principles, such as inheritance or encapsulation in object-oriented design. As a reference, the work of Hadar analyzes the reasons why professional object-oriented programmers make bad decisions although they possess the required knowledge [1]. A common reason is the lack of understanding of the context and background of a given principle. A student or a programmer has to see the problem and then come to a solution on one's own. By realizing the nature of the problem and obtaining the solution only through his own work, he will better remember the solution and even the problem to which the solution applies.

1.1. Task-Driven Teaching

The task-driven teaching method is based on the constructivism teaching theory [2]. It is based on acquiring knowledge through the experience of solving tasks. A related approach, task-centered instructional strategy, starts by presenting a series of real-world tasks that gradually form the taught content [3].

Explaining the theory in lectures is undoubtedly necessary; however, it does not provide all the understanding necessary for the application of knowledge in practice. According to Bonwell et al., active involvement of students in the learning process increases its effectiveness [4]. While people may remember the mistakes of others, they never forget their own. *Learning by doing* includes learning from one's own mistakes. Very few people can understand theory so well they will not make any mistakes when it comes to applying it in practice.

1.2. Case Studies

A case study investigates phenomena in a real-life context [5]. The case method of teaching provides a realistic situation or project to students and allows them to apply their knowledge to devise solutions to the presented problems [6]. According to Coppit, providing students with meaningful development experiences in computer science courses is essential to produce graduates who can enter the software industry and be productive [7]. Therefore, we use case studies as the basis for our method.

In general, many researchers discovered that using case studies has a positive effect on students' learning (e.g., [8,9]). However, there are not many guides available on how to design, prepare, and execute a case study in the context of computer science.

1.3. Task-Driven Case Studies

In this paper, we describe *task-driven case studies*—a specific implementation of the case study technique [6]—and share our experience with it. Our special task-driven case studies are case studies combined with the task-driven methodology. This means that the case study guides a student through multiple tasks corresponding to the learning objectives of the course. An essential part of them is study guides—written guides that describe the case study and specify tasks to be solved.

We provide guidelines on what a good task-driven case study for a course related to programming should look like and how it is prepared. These guidelines and experience may be found useful even in other types of case study implementation, i.e., without task-driven methodology, and for teaching in general.

Our previous conference article briefly described teachers' experience with our approach, analyzed the results of a survey with students, and outlined potential drawbacks [10]. In the current paper, we summarize and complement these findings, thoroughly describe a novel teaching approach devised as a combination of case studies and the task-driven approach, and provide guidelines for other educators supplemented by specific examples.

2. Background

Existing research related to task-driven case studies can be divided mainly into the topics of problem-based learning, case studies, task-driven methodology, and game-based teaching.

2.1. Problem-Based Learning

O'Grady provides an overview of some of the existing literature about problem-based learning [11]. Problem-based learning (PBL) focuses on students so they take responsibility for their learning themselves. In this aspect, PBL resembles case studies. However, in comparison with our method, PBL gives only little guidance to students. They are given a problem, and it is their responsibility to analyze and solve it. Our method guides them through a solution with tasks. Similar to challenge-based learning [12], we also draw from PBL to support students' critical thinking, self-directed learning, and long-term retention.

A more recent scoping review of PBL research articles by Ghani et al. identifies learning behaviors required for effective PBL application and can be used as a helping guide for designing better case study tasks [13].

Pérez et al. report their positive experience of introducing complex, more industrial-like projects at their university [14]. Cico et al. report that project-based learning is currently the most dominant approach to teaching software engineering [15]. Case studies enable teachers to focus more on the process of solving the project than on the end result.

2.2. Case Studies in Teaching

Many works concern the use of case studies in teaching. One of the more recent works by Grimes provides guidelines for the design of continuous case studies [16]. Continuous cases represent a type of case study that is split into segments that gradually unfold the

story represented by the case. The guidelines have many similarities with our suggestions; for example, they suggest aligning case study segments with course learning outcomes or assessing students continuously. On the other hand, the guidelines are quite generic and do not assume some specific form of task design based on the case study.

Zhang et al. explore case-based teaching of a digital image processing course [17]. They find significant improvement in several learning aspects, including students' learning interest and motivation. They also propose basic principles of case design that are quite generic.

Ouh et al. present the application of case-based teaching in a postgraduate software architecture course [18]. Instead of a large case study, they use smaller bullet cases and mini-cases mainly as a way to facilitate discussion. In our case, however, a large and more detailed case study is used and split into simpler tasks intended to be solved by students on their own.

Varma et al. present a short analysis of using case studies in software engineering courses [19]. They also provide a few general basic guidelines on how to prepare such a case study. In another work, Garg et al. provide an experimental comparison of the lecture–examination learning model and the case study method [8]. They claim that case studies help to nurture the higher-order cognitive skills of application, analysis, evaluation, and synthesis. In their later work, they discuss the advantages of using case studies as assessments [20]. In our experience, we came to quite the same conclusions, and that is the reason why case studies are used as assessments in our courses. In comparison with them, we incorporate also guidance through the case study using tasks.

A case study in a software engineering course is described also by Jia [21]. They also argue for case studies' advantages in comparison with conventional teaching approaches. Additionally, they also express the importance of the clear specification of teaching goals that we argue for in Section 4.

Burge et al. introduce a more business-oriented case study into a software engineering course [22]. In their work, they describe the case study and analyze the advantages of using it in a course. Many other works argue for using case studies and large projects as assessments. For example, experience with a case study is presented by Hilburn et al. [23]. Martin [24] and Meyer discuss the shortcomings of using introductory toy examples in comparison with large projects [25]. Coppit argues that providing students with meaningful development experiences in software engineering courses is essential to produce graduates who can enter the workforce and be productive [7]. These works identify the issue with most common academic projects so far—that projects may present a highly abstracted view of reality and students usually focus on the project outcome (product) and not the process. As a solution to these issues, the works argue for using case studies.

2.3. Task-Driven Teaching Methodology

The task-driven methodology belongs to the constructivism teaching theory. It aims to divide one standard assessment and one single task into multiple tasks. According to Yu et al., this approach is supposed to make learning more effective and faster because the knowledge could be covered naturally and the learning motivation could be maximized through full engagements [26]. In their work, Yu et al. describe their experience with a task-driven methodology.

A meta-analysis by Xie et al. shows that constructivist methods have a significant effect on improving learning in the area of mathematics compared with the traditional transmission acceptance method [27]. A similar effect can be expected in computer science as well.

Peng et al. argue that by using tasks, students' views of learning will be influenced unconsciously [28]. In the past, they used to take learning as a task; now they learn in light of the needs of the tasks, changing from passive acceptance of knowledge to active pursuit of knowledge. Peng et al. describe their specific implementation of the task-driven

methodology in one of their courses, and based on their experience, they provide some basic guidelines and advice about the application of the method.

Xue et al. describe the application of the task-driven methodology in information technology courses in detail [2]. While our work focuses more on how to devise tasks and how the tasks collaborate with case studies, Xue et al. aim more at how the task-driven methodology can be effectively executed in information technology courses.

A study by Liu et al. shows that proper usage of tasks can result in learning benefits for students [29]. Students' creative thinking is cultivated; they can work independently and at their own pace. The methodology makes hierarchical teaching more convenient through different hierarchies of tasks. Similarly, we argue the same benefit by using tasks with layered hints to provide control of the task difficulty.

Rosenberg-Kima et al. compare the effectiveness of two task-based approaches with explicit instructions in an empirical study with programming novices [3]. The whole-task strategy presents instructions in the context of the whole task, so the integration of individual components is necessary from the beginning. The part-task strategy defines distinctive components that are taught separately, while their integration is completely omitted or postponed until the tasks are finished. According to the results of this controlled experiment, students using the whole-task strategy achieved better performance and learning transfer. This empirically confirms our intuition that incremental implementation with a solution that is continually executable benefits students.

Liang et al. present a combination of task-driven and objective-oriented hierarchical education methods that they use in their Linux curriculum [30]. Objective-oriented hierarchical education enhancement allows students to solve different tasks according to their objectives in their studies. If the student's objective is to become a system administrator, he would be solving different tasks compared with a programmer. This enhancement would apply to our method as well; however, as it is expressed in Section 9, our task-driven case studies are already quite expensive to prepare. Making multiple versions according to different objectives would be very costly. A viable application is to provide different additional tasks since those are only specified as feature requirements in the problem terminology, and they do not have to be implemented by a teacher.

Dong uses a so-called graded task-driven methodology [31]. A graded task-driven methodology employs tasks graded based on their difficulty. Its purpose is to provide better students with enough challenges and worse students with tasks that they would be able to solve. These graded tasks are very similar to our idea of multilevel hints to tasks.

Birnbaum and Langmead argue that a task-driven approach is suitable for teaching programming to digital humanities students [32]. They view it as an analogy of the task-based method of teaching foreign languages, where, instead of focusing on comprehensive knowledge of grammar and vocabulary, the learning is organized around communication tasks. They suggest that this approach is suitable for students whose main area is not computer science. While our method is focused on computer science students, its principles are in line with the suggestions provided by Birnbaum and Langmead, including the important role of the case study domain, or incremental development. Therefore, it can be adapted to other study programs as well.

In comparison with all of the aforementioned works about task-driven methodology, our task-driven case studies differ in combining the task-driven methodology with case studies. In this combination, we wanted to give students a complex real-world environment in a case study combined with proper guidance through tasks to teach them best practices.

2.4. Games in Teaching and Automatic Feedback

In many of our courses, we implement games as case studies. If possible, we recommend taking this approach, as games are a proven tool for motivating students and encouraging their engagement in the learning process [33,34]. Zhan et al. review and analyze multiple recent studies that show the positive effects of gamification on the learning process [35]. Papadakis documents another study with evidence of the positive motivational effects of

games on students in learning programming [36]. Jordaan provides evidence that even board games can improve students' engagement in learning programming [37].

Tay et al. examine and describe game design elements needed to inhibit students' motivation and engagement in the learning process [38]. We refer to them for some useful guidelines on the design of a game case study. Useful information on this topic can be also found in [39,40].

Cavalcanti et al. summarize a review of recent works in providing automatic feedback to students in an online environment [41]. As suggested at the end of Section 8.1, task-driven case studies open space for utilizing automatic feedback and obtaining its benefits.

3. Task-Driven Case Studies' Pedagogical Framework

As was explained earlier, task-driven case studies are a teaching approach that combines case studies with the task-driven methodology. The approach consists of the following three main components:

1. **Case study**, a realistic project providing an opportunity to showcase most of the learned topics.
2. **Tasks** defined in the context of the selected case study and corresponding to the learning objectives of the course.
3. **Written study guides**, explaining the case study, leading students through the tasks, and providing enough context and explanation.

3.1. Reasons to Use Tasks

By using the task-driven approach, it is possible to change students' view of learning [28,30]. Instead of perceiving learning as a task, they start learning because of the tasks. This view changes their attitude from passively accepting knowledge to pursuing it for the needs of the tasks. Such a learning model better mimics industrial practice.

A task-driven case study is built upon relatively fine-grained tasks. These tasks define what a student should do to continue in the case study. More specifically, programming courses contain implementation tasks, defining what needs to be implemented. They guide students through the process of solving the case study.

The following are the three main reasons why we use tasks:

1. We want to make sure that students are active during lessons. Both the teacher and the students are reminded that practical lessons should revolve around the students working and not the teacher lecturing.
2. We want the student to use best practices while working. Finer-grained tasks help us to keep the student on the right track. In learning, the process is more important than the result.
3. Instead of only showing the process and the students repeating after the teacher, we use task-driven case studies to make them try the process by themselves. Then it is harder to forget about the details.

In our method, case studies are combined with the task-driven methodology, incorporating tasks to guide students to active participation in practical lessons. The whole case study with the tasks is presented to students using written study guides.

3.2. Role of Written Study Guides

The written study guides are an integral part of our method. They contain all the important information for solving the case study or, in general, to fulfill the goals of the practical lessons. The following are the three main reasons why we recommend preparing written study guides for task-driven case studies or even for task-driven teaching in general:

- A written form of the guide allows students to work at their own speed. If there is something unclear, they can get back to it anytime.
- Students can continue working at home without any impedance. If a student is unable to come to the lesson, he misses only individual consultations with a teacher. To a

large degree, the lesson can be simulated by the study guide. This proved to be very useful during COVID restrictions.
- A study guide can be shared between universities or learning facilities.

Study guides substitute the teacher in explaining the case study (or the contents and tasks of the lesson) to the students. The teacher can focus on explaining only the more problematic aspects of the case study and individually helping the students with their problems.

The tasks represent the main component of the study guides—each lesson should have at least one task. However, they are not the sole content of them. Besides the tasks, the study guide for each lesson provides the students with a problem definition and explanation of the problem context, objectives of the corresponding lesson, background, reasoning, further reading, etc. These concepts will be explained in the rest of the paper.

3.3. Course and Case Study Relationship

To prepare a good case study for a course, its author has to realize the connection between the course itself and the case study. Figure 1 demonstrates such a relationship.

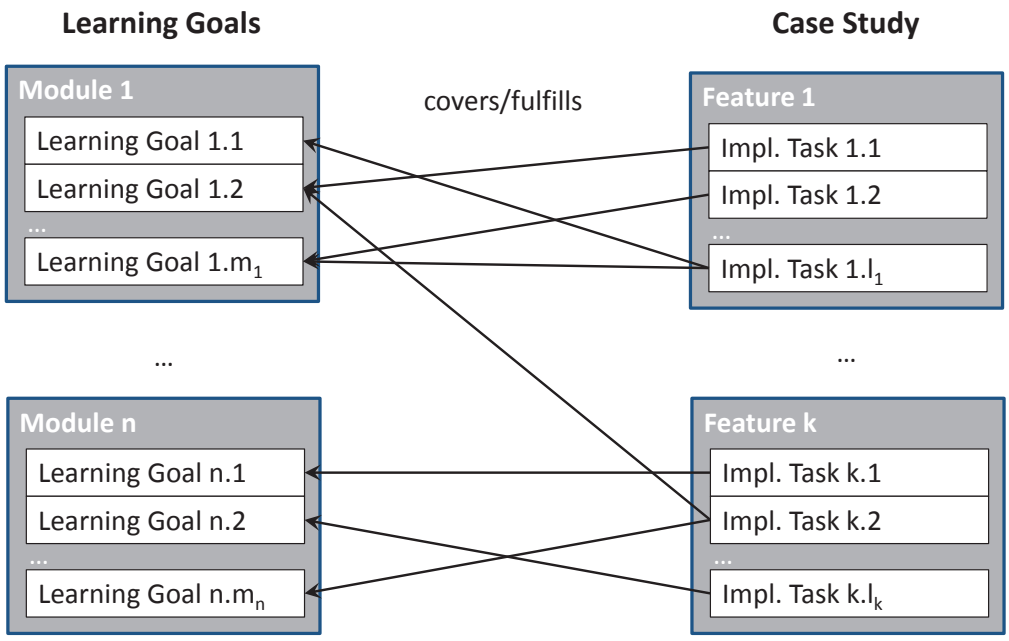


Figure 1. The relationship between goals and tasks in case studies.

Each course has a set of learning goals (displayed on the left side of Figure 1). For example, the Programming Basics course could have goals such as "understand the meaning of conditional statements" and "know how to use iterations". Learning goals are the core of the course. The course is usually divided into modules, where each module contains a set of learning goals for one lesson of the course.

Everything in the course should revolve around these goals, so the case study has to concentrate on them too. One very important issue is mapping the learning goals to the case study tasks.

A case study solution (right side of Figure 1) consists of multiple functional features of an implemented product. Each of these features is implemented by solving a sequence of implementation tasks. A task is a specification of some implementation goal that has to be achieved by a student to continue in the case study, e.g., "perform empty game field generation" or "shuffle the field items randomly". Each implementation task covers and fulfills one or more learning goals. Each goal can be covered by multiple tasks. If there is a

goal that is not covered by any task, we risk that the goal material will not be exercised in full and therefore not be comprehended well.

According to our experience, it is a good practice to map each feature completely into one module only so that the feature can be runnable after the lesson. However, this is not always possible, but as we argue later, it is useful to have an executable solution after each module, and mapping like this can help to achieve that.

3.4. Task-Driven Case Study Lifecycle

For preparing a good case study, it is useful to understand the case study lifecycle. We have identified a lifecycle of the task-driven case study as it is presented in Figure 2.

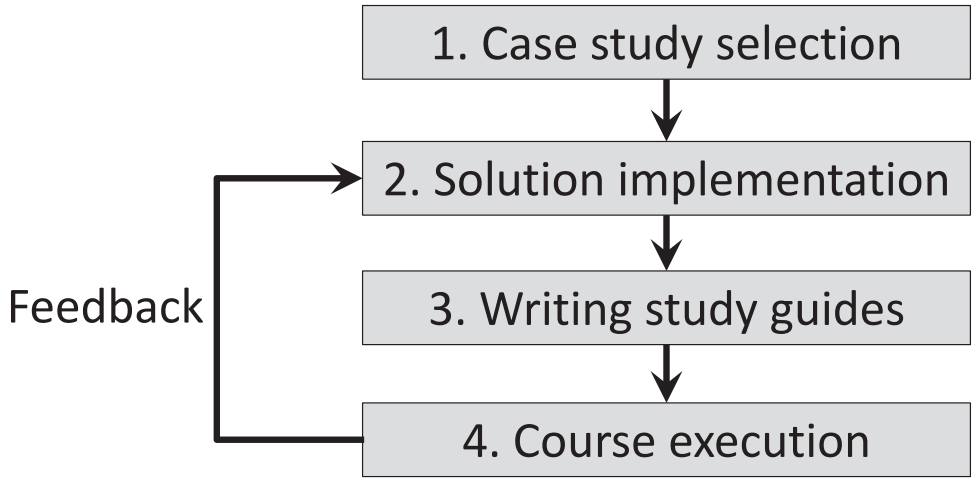


Figure 2. The task-driven case study lifecycle.

The lifecycle starts with the case study selection phase. In this phase, the authors need to find a suitable theme and project that is appropriate for the course. The selected project should offer enough space to fulfill the learning goals. After the selection phase, we suggest starting with the design and implementation of a solution to refine our understanding of the details and find potential problems. The solution implementation phase is followed by the writing study guides phase. After the study guides are finished, it is time to run the course. During the execution, it is essential to collect feedback from the students and subsequently include the suggestions and observations in the study materials and update the implemented solution.

3.5. Guidelines

In this paper, we use our experience to extract guidelines that apply to our approach or even to case studies or the task-driven methodology in general. The guidelines will be backed with examples from our most mature case study—the Minesweeper from the Java technologies course.

The guidelines are structured based on the task-driven case study lifecycle phases. We discuss each of them in detail in the following sections. Here, however, we provide only a summary of them in the form of lists as an overview or a checklist for course authors.

Selection of the Case Study

- Know the learning goals.
- Select a well-known domain.
- Find an interesting topic.
- Keep the project in industrial quality.
- Show the whole process.

- Prepare for incremental implementation.
- Support individual approach.

Solution Implementation

- Implement code first.
- Divide code into goals.
- Evaluate goals' coverage.
- Find goal dependencies.
- Do the review.

Writing Study Guide

- Always tell objectives.
- Describe the current state and expected increment.
- Show context and reasoning.
- Specify tasks clearly and precisely.
- Refine tasks with comments.
- Add supplementary tasks.
- Provide further reading.
- Use suitable software support.
- Verify continuity and consistency of the guide.
- Review the guide.
- Start with a minimal version and improve it.

Course Execution

- Monitor progress continuously.
- Favor individual achievements.
- Examine understanding.
- Get feedback from the "battlefield".
- Update the case study.

4. Selection of the Case Study

In the *case study selection* phase, the author should find a project that would be appropriate for a case study. A good subject of a case study can have a positive effect on students' motivation, satisfaction, and learning efficiency. The following are the guidelines that stemmed from our experience.

4.1. Know the Learning Goals

As a first step, it is really important to identify the learning goals for students in the course. Clear, sound, and specific learning goals are a basis for preparing a course, and they should be prepared upfront. Goals help to select a proper case study project. According to our experience, without having explicitly stated goals, it is hard to prepare tasks that would cover all the required topics in the course.

Example: For our case, the Minesweeper case study for the Java technologies courses, we have written down a list of learning goals that would lead to understanding the Java language and all the fundamental Java technologies. The following list presents an excerpt of the goals of the Java technologies course:

- Students are familiar with Java language basics.
- Students can create a project and a class in the NetBeans IDE.
- Students understand the role of interfaces in object-oriented programming.
- Students can implement existing interfaces in Java.
- Students can create their own interfaces in the Java language.
- Students are familiar with the role and types of collections in the Java language.
- Students can work with the generic `ArrayList` collection.

- Students can use `String` class methods for working with strings.
- Etc.

These goals are specific and can be used later to check whether the implementation of the case study covers all the necessary topics in the course.

4.2. Select a Well-Known Domain

We experienced that a proper domain of the case study project should be a domain well known to the students. When the students are supposed to learn to program in a Java language, they should not be bothered with understanding a domain (unless it was stated as a learning goal). A well-known domain helps to save time and energy in favor of learning the course topics.

Example: You can imagine that, for example, the problem of luggage transport at an airport is not a very good choice. Most of the students never had to (or will never have to) deal with that problem, and some of them probably never even traveled by plane. That is the reason why we picked a Minesweeper game. Thanks to the Microsoft Windows operating system, it is one of the most known games. We could barely find a student in the Java technologies course who had never played Minesweeper.

4.3. Find an Interesting Topic

Motivating students is one of the most challenging parts of education. Without students willing to learn, teachers may use state-of-the-art methodologies and approaches, but they will hardly obtain the expected results. Using the words of Mingins et al. [42], "The course should be fun!".

We have to impress students with their assignments, ideally making them work with impressive programs, like those that handle graphics. Teaching can capitalize on this "wow effect". "Trying to get students excited is pedagogy, not demagogy" [25].

In this context, it is important to know the target group. Our experience proved that the easiest way to get today's students' attention is to make them implement their own game—we use game-based learning [43]. Many of our students claim that they came to study Informatics only because they wanted to learn how to make games.

Example: That is why we chose the Minesweeper game. In some of the other courses, games are used too. For example, we used the N-Puzzle game in the .NET programming course and Alien Breed clone in the object-oriented programming course.

4.4. Keep the Project in Industrial Quality

We suggest implementing a project that is of industrial quality. Students need to feel the touch of the real world, of the industry. Many times, students' motivation can be killed if they have to solve problems that seem to be too trivial or unrealistic.

Another benefit of an industrial-quality product is its motivational aspect. Students can flaunt their solutions to their friends and family. In this context, the domain also plays a role. For example, a game has a greater impression on non-IT people than a console or desktop application.

The projects should also implement best industrial practices.

Example: The solution to our Minesweeper case study is comparable to the professional Minesweeper game (see Figure 3). Students can see that they are able to implement a program comparable to the industry.

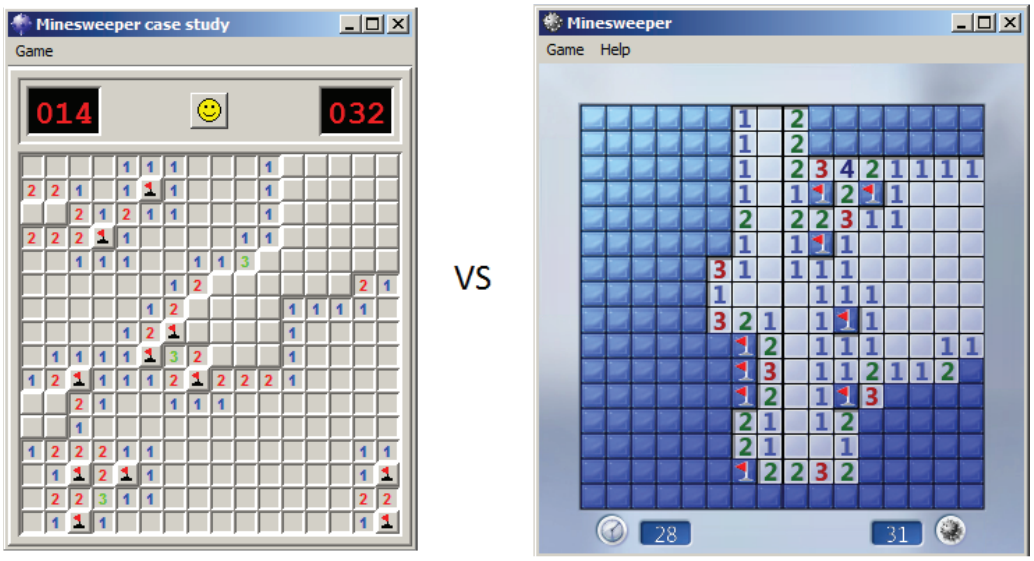


Figure 3. Our case study solution (**left**) compared with Microsoft Minesweeper (**right**).

For example, our Minesweeper case study has a modular design that strictly separates individual user interfaces: the console (textual) and the graphical user interface.

4.5. Show the Whole Process

We want to teach students the whole process of implementing a good software system. Showing only partial solutions and expecting that they will somehow understand the glue that put these together is naive. Garg et al. [8] talk about receiving feedback that students felt that software engineering is a theoretical and unexciting subject and is perhaps of very little use in the future. The reason for this feedback may be found in the lack of experience related to a project [19,44]. We, on the other hand, had very good feedback on our case studies that guided them through the whole process of implementing the project (see Section 8).

Students usually focus on the project outcome (product) and not on the process of the project implementation. However, the process is a part of the course during which they should learn how to apply theoretical knowledge. It is not enough to show them what they should apply. As Garg et al. say, the competence of a student in applying skills or the synthesis of learning in real-world scenarios should be established from a case study carefully designed by an experienced professional [20]. Throughout solving the case study, students have to be guided to use best practices, not only to implement a solution that will work. Instead of merely showing them the best techniques and expecting them to somehow become programming masters, the case study shows them how a masterpiece is done (in an ideal case, when the case study really covers a masterpiece). The case study shows them how the work and reasoning go through the whole process.

The tasks should not focus only on the implementation phase itself. To abstract the problem from implementation details, in some courses, students first write pseudocode in natural language and draw diagrams (flowcharts, UML, etc.) as necessary. In some courses, the first one or two weeks are dedicated to the analysis, while in other ones, the beginning of each lab can be focused on the analysis of the problem. The details depend on a specific course and are a part of the study guide.

To be able to show the whole process, the project size has to be picked very carefully. Students should implement a reasonable part of the project themselves, not only a small part. We want them to see the creation of a comprehensive solution (especially important when teaching a programming paradigm, e.g., object-oriented programming). Therefore, a final solution cannot be a large application or just its small part. It has to be something smaller yet comprehensive—not only, for instance, a sorting algorithm.

Example: In our Minesweeper case study, we chose the Minesweeper game because it is small enough to be implemented by a student in the time span of the course, and yet the result is a whole game that is comparable with industrial Minesweeper.

4.6. Prepare for Incremental Implementation

In our experience, it is very good for students' motivation if, after each lesson (module), the partial solution is runnable. Students can immediately see the results and test their implementation. If the students work for 10 weeks and see no result, they may lose the courage and motivation to continue. "*What if I have made some error in the beginning and the whole implementation is now incorrect?*" Seeing the results gives them the feeling that it is going the right way. Moreover, it gives them satisfaction with the work they had performed during that particular lesson. Therefore, we suggest choosing a case study that supports such incremental implementation.

Example: In the Minesweeper case study, the first case study lesson introduces the Minesweeper game. We are designing the game core together with students using standard UML notations. During this, they are led to our source code skeleton implementing best practices from object-oriented programming. In the next lesson, students implement the game logic—the game field and all its behavior. So far, their solution is not playable. However, we do not wait anymore, and in the third lesson, they start implementing a simple console user interface that would be able to present the current state of the field. Although, after the third lesson, they cannot play the game, they can already run the game and see whether their field generation works as it should. Since that moment, they are always able to run the game and see the increment they add during each particular lesson.

4.7. Support Individual Approach

Most of the previous advice aimed to keep students on the right track when implementing the case study. We argued that we have to guide them so they can see how a professional solution is created. However, students' creativity should be nurtured too. Therefore, we recommend selecting a project that would provide students with enough space for their individuality. Students should follow the guide in creating the main program, but then, they must have the opportunity to enrich it with their own ideas and special features.

Example: In our Minesweeper case study, we suggest that students implement some additional features that the main project does not have. An example may be implementing the support for a simultaneous click of both mouse buttons to open also adjacent tiles and not merely the one that the mouse points to. We also support their own ideas on how to make the game even more interesting.

5. Solution Implementation

After a good subject for the case study was selected, in the next step, the teacher should implement the case study himself—the *solution implementation* phase. It helps the teacher to get familiar with the case study and the study solution, with all its details and problems. The sample solution will verify if the selection was really appropriate.

5.1. Implement Code First

If the teachers start writing a study guide and tasks without first trying to implement them, they risk that the students will stumble upon some unseen issues that may complicate the learning process. We always implement the case study first ourselves. It helps us to see every detail of the solution, make the best design decisions, and provide sound reasoning about them to the students. There should not be any open problem in the main project that the teachers would not be aware of and to which they would not have an answer. The implementation-first approach also helps the teachers to avoid guiding the students the wrong way. It is better if they find out upfront that there is some problem than if the students find it out during the course.

Example: We implemented our own Minesweeper game before giving the case study to students. Before using the case study approach in the Java technologies course, the Minesweeper game was one of the examples that we used to show the students more complex examples implemented in Java. Then we decided to prepare a case study and reworked the old solution to incorporate all the topics of the course and to reflect our best knowledge.

5.2. Divide Code into Goals

It is recommended that the teacher divide the source code into learning goals they achieve. In the *case study selection* phase of the case study creation process, we recommended preparing a list of learning goals. These goals specify what a student has to learn in the course. Now the source code of the implementation should be matched to these goals. We recommend doing it in two steps. First, the teacher identifies implementation objectives needed to implement the case study and maps the code to them. The implementation objectives represent the tasks, such as students have to implement the generation of the game field and students have to implement the presentation of the game field. Second, the teacher maps the implementation objectives to learning goals. For example, the implementation objective "students have to implement the presentation of the game field" would be mapped to the learning goal "students understand and know how to use data streams".

We recommend explicitly marking the division in the code as well—-by using either comments or preferably source code annotations (attribute-oriented programming [45]). Having the source code of the case study explicitly annotated by its implementation objectives (and through them to learning goals) helps with the maintainability and evolvability of the case study.

Example: To explicitly record implementation objectives in the code of the Minesweeper case study, TODO comments marked with the task identifiers were used. Each task represented an implementation objective. We started using TODO comments because of the tool support—see the automatically generated list of TODO comments in an IDE in Figure 4. However, current IDEs have better support for source code annotations since they are first-class citizens of the language. The following code is an excerpt showing a TODO comment marking the `getColumnCount()` method as a part of the task identified as "getters" in the second module (lesson) of the course.

```
//TODO: Task 2 - getters
/**
* Returns column count of the field.
* @return column count.
*/
public int getColumnCount() { ...
```

An alternative using source code annotations (e.g., a custom `@Task` annotation) is presented in this second listing:

```
/**
* Returns column count of the field.
* @return column count.
*/
@Task(module = "02", id = "getters")
public int getColumnCount() { ...
```

In this example, the `getters` task can be easily identified in the source code. The task itself then needs to be mapped to a learning goal. In our example, it would be the goal of the learning principles of implementing the Java Beans convention. To illustrate the indirect mapping through tasks, Figure 5 shows relationships between the "Swing User Interface" feature implementation tasks of the Minesweeper game and the learning goals in the Java technologies course that spans multiple case study modules. The diagram is an instantiation of the diagram presented in Figure 1. It shows how implementation

tasks relate to learning goals. The coverage of tasks in the code is represented by code annotations (or legacy TODO comments).

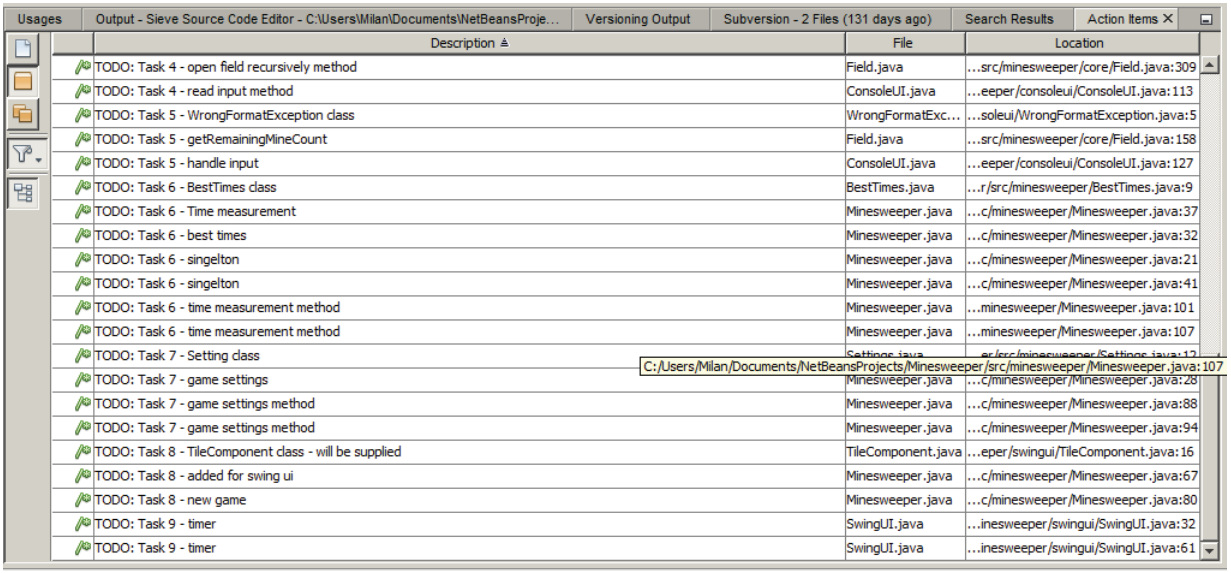


Figure 4. The projection of code division to goals using TODO comments in Minesweeper.

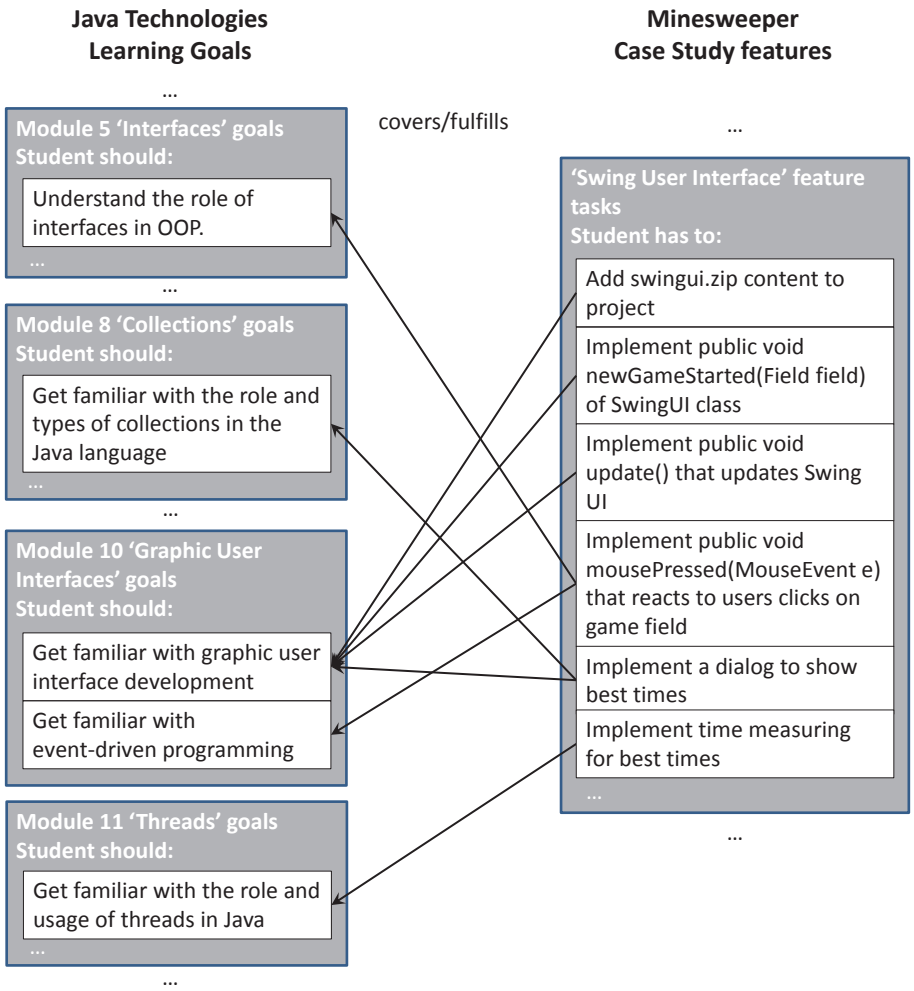


Figure 5. Relation between implementation tasks and learning goals in the Minesweeper case study.

5.3. Evaluate Goals' Coverage

It is important to verify that every goal is addressed in the source code. In this process, IDE tools can be used to check the explicit code division according to implementation tasks.

It is also crucial to go through every goal and consider the difficulty of its tasks too. Some goals may not be that important, and they may include a lot of code and, therefore, a lot of work for students, while other important goals may be covered only by simple implementation problems. The idea is to balance the code with the importance of its goal.

Each learning goal has to be covered in the case study—not vice versa—not every piece of code has to be relevant to learning goals. In such cases, these pieces of source code can be given to students ready-made. If students cannot learn anything relevant to the course from implementing that code, it is usually a waste of time to make them do it.

Example: A simple help in checking the code coverage is, for example, the Action Items window in the NetBeans IDE (previously called the Tasks window). In Figure 4, there is a screenshot showing the list of tasks currently present in the Minesweeper teacher's solution. It is easier to check the goals this way since the goals' source code can be scattered throughout the whole project.

In the Minesweeper case study, the students are given the code skeleton of the game core—it is the result of object-oriented design and does not address any specific Java technology. Thus, its creation would fit in the OOP (object-oriented programming) course, but not in the Java technologies course.

5.4. Find Goal Dependencies

Learning goals and their tasks need to be divided to form a list of modules. One module defines one lesson. Modules should be balanced so that all the lessons are approximately of the same difficulty. The order of the tasks for a lesson is very important. In our experience, the best time to prepare the order of the lessons and their tasks is in the *solution implementation* phase of the case study creation. Usually, the order will follow the order in which the implementation is made by the teacher.

The most important aspect in the ordering of goals is their implementation dependencies. If the code of one task cannot be implemented unless a code of some other task is present, then the other task has to be fulfilled first.

For the sake of having continuity, it is also useful to verify whether the tasks for one goal are not scattered too much in the implementation process. For example, when the students are exercising interfaces, they should not be forced into other topics until they are finished with interfaces.

One should also consider the principle of incremental implementation from the *case study selection* phase. If there is no implementation dependency, the goals should be ordered in a way that each learning module will end with a runnable partial solution. From the motivational viewpoint, this aspect has a higher priority than the continuity aspect.

After preparing the order of the goals, we recommend aligning the lectures with the case study (practical lessons). It is really important to ensure that each topic exercised in the case study was already discussed in the lectures.

Example: In the Java technologies course, the lectures are aligned with case study modules so that the students always have the theoretical knowledge that is needed for the current module of the case study. The alignment is depicted in Figure 6.

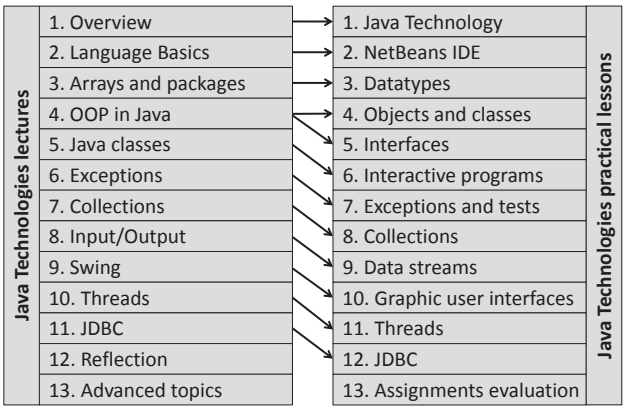


Figure 6. Alignment of lectures and practical lessons in the Java technologies course.

5.5. Do the Review

After finishing the teachers' sample solution, the best practice is to get it reviewed by other professionals. This may mean other teachers who are teaching the course or even experts from the industry. The feedback from them can significantly improve the case study.

6. Writing Study Guide

We recommend preparing written study guides instead of merely presenting the problem and the case orally for a few reasons that have already been mentioned in Section 3. First, the case study is usually non-trivial, and therefore, it is better to have its written specification, so the students can get back to the parts they did not understand anytime they want. Second, they can work at home, having a copy of a study guide or accessing it online. Third, the quality of case study specification does not differ even when there are multiple teachers with different knowledge and skills. Fourth, this way, a case study can be easily shared between universities or other teaching facilities.

Writing a study guide for a case study follows the *solution implementation* phase. However, the teacher does not have to wait until the completion of the implementation of the whole solution. He can implement a part of the solution for the first lesson, and then write a study guide for the first lesson, and only then continue with the next one.

To better mimic the process instead of only a specification, we suggest that the study guide for each module should consist of a sequence of steps. Each step would bring a student closer to the finish. Steps consist of the problem context for the step and *tasks* that have to be implemented.

Example: To get a better notion of how such a study guide looks, a specific example of one lesson from the Java technologies course can be found at https://kurzy.kpi.fei.tuke.sk/tjava-en/student/06.html (accessed on 30 August 2024).

6.1. Always Tell Objectives

Students should know what is their objective for the current lesson. The best way to begin a study guide for a lesson is to start it by listing the objectives. This can include learning goals, but also the implementation objectives that specify project features or their parts that should be implemented during the lesson. Sometimes the students are afraid of some important topics because they consider them complex and difficult. By telling the students more and providing concrete information, we want to show them that there is nothing to be afraid of.

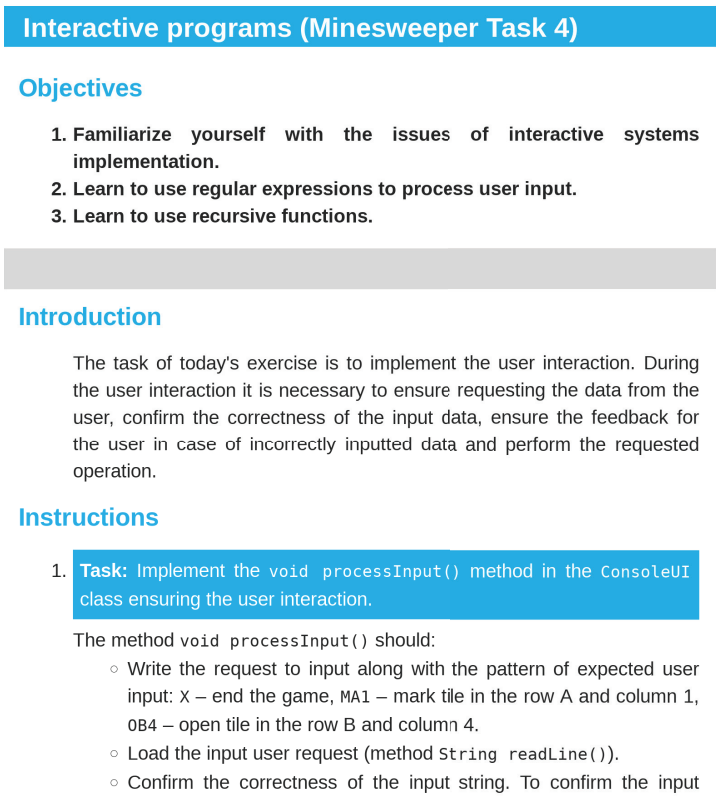
After the lesson, students can also verify that they did not miss any important point by checking whether they fulfilled and understood each goal/objective.

Last but not least, we found it very useful to link the learning goals in the study guide to the materials used in lectures. This way, students can easily get to the relevant information from the lectures if it is needed.

Example: Examples of learning goals are presented in Section 4 when we discussed making a list of the learning goals for the case study. However, there are also implementation objectives. These are specific to each case study. For example, in the following list, there are some implementation objectives from the Minesweeper case study.

- Students have to implement the generation of the game field.
- Students have to implement the presentation of the game field.
- Students have to implement a time-measuring feature for the game.
- Students have to implement the settings feature.
- Students have to implement a graphical user interface in Swing.
- Etc.

In Figure 7, there is a beginning of a study guide page for one of the lessons of the Java technologies course. Each goal (objective) is mapped to one or more steps in the instructions part, which contains explanations for students along with implementation tasks. In the example, the first task fulfills goal numbers 1 and 2 since it implements user interaction and uses regular expressions to achieve this.



Interactive programs (Minesweeper Task 4)

Objectives

1. **Familiarize yourself with the issues of interactive systems implementation.**
2. **Learn to use regular expressions to process user input.**
3. **Learn to use recursive functions.**

Introduction

The task of today's exercise is to implement the user interaction. During the user interaction it is necessary to ensure requesting the data from the user, confirm the correctness of the input data, ensure the feedback for the user in case of incorrectly inputted data and perform the requested operation.

Instructions

1. **Task:** Implement the `void processInput()` method in the `ConsoleUI` class ensuring the user interaction.

   The method `void processInput()` should:
   - Write the request to input along with the pattern of expected user input: `X` – end the game, `MA1` – mark tile in the row A and column 1, `OB4` – open tile in the row B and column 4.
   - Load the input user request (method `String readLine()`).
   - Confirm the correctness of the input string. To confirm the input

Figure 7. An excerpt from a study guide demonstrating the relationship between goals and tasks.

6.2. Describe the Current State and Expected Increment

After listing the objectives, a good start is to recapitulate the current state of the solution (the state in which the implementation should be after previous lessons) and to show the desired increment to it. Of course, it is not always necessary to recapitulate the complete current state of the solution. Many times, an abstraction (e.g., a UML class diagram) focused on the current problem will suffice.

Example: We use class diagrams in each module to show the difference in the program structure before and after finishing the current lesson. In Figure 8, there is a class diagram

of the Minesweeper case study in the fifth module. The yellow classes represent the current state, and the red ones are the increment for the fifth lesson. This helps the students to understand how the implementation goals will be projected into the program structure.

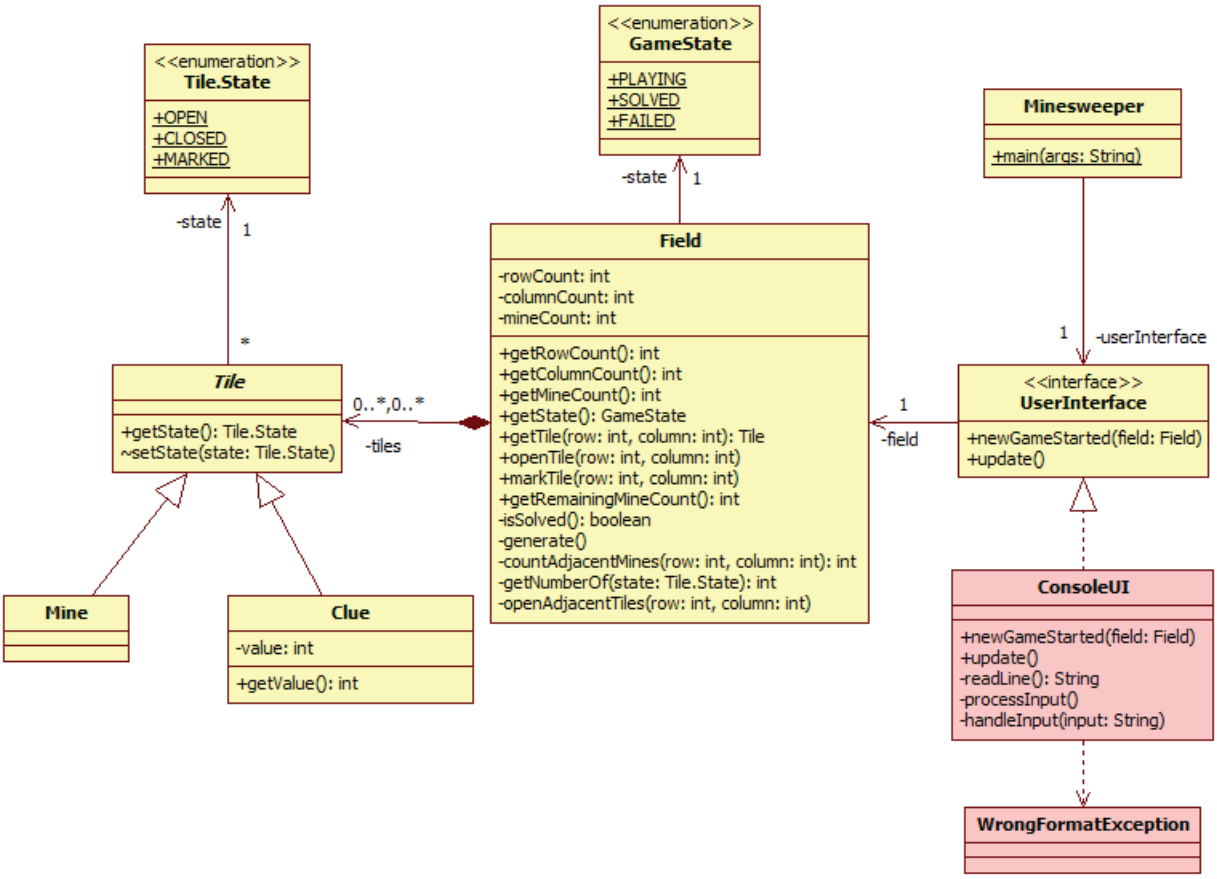


Figure 8. A snapshot of the Minesweeper class diagram in the fifth module.

6.3. Show Context and Reasoning

It would not be a case study if it did not provide a clear context and reasoning. In the study guide, there has to be a part that should explain to students the reasoning and motivation for the work they will be doing in the current step of the lesson. It is also the specification that can be used to verify the correctness of the student's increment implementation.

Example: In the Minesweeper case study, each step starts with an explanation of the current situation, the problem context, and the reasoning behind it. This part of the step is intertwined with the tasks that lead to solving the problem—each step of the module starts with the context and reasoning, and after that, the tasks follow.

6.4. Specify Tasks Clearly and Precisely

Each step provides some context to the current problem, motivation, and reasoning. A step consists of this context and of tasks that guide students in solving the problem.

The most important and hardest part of the case study creation is the clear and precise specification of the tasks. Task specifications should not be too long. If a task is too complex to be explained in a few sentences, then it should be divided into subtasks. Additionally, multiple shorter tasks are used to control the students' implementation process so they will not fall into bad practices.

The level of abstraction for task specification depends on the context. Usually, at the beginning of the course, the tasks are described more thoroughly. For instance, precise pseudocode is provided and the students can simply rewrite it to the given programming language. Near the end of the course, tasks tend to be described more briefly since students have already gained experience.

Tasks for a single lesson should be balanced. They do not have to be of the same difficulty, but it is important to keep the difficulty generally non-decreasing.

Example: Following is an example of a task from our Minesweeper case study that tells the student to implement a method that will be a part of the Minesweeper marking tiles feature. This is one of the simpler tasks from earlier lessons of this introductory Java course.

```
Task (id = markTile)

Implement the void markTile(int row, int column) method in the Field
class. This method allows marking/unmarking tiles specified by the
row and column. In~case the tile is closed (Tile.CLOSED), its state
will be marked (the state will change to Tile.MARKED). If~a tile is
marked (Tile.MARKED), its state will be changed to closed (Tile.
CLOSED). Rows and columns are numbered from 0.
```

When the students would get to this task, they should already have a case study code skeleton with a bit of their code. In the `Field` class, students would find the following `markTile(int row, int column)` method stub:

```
/**
* Marks tile at specified indices.
* @param row row number
* @param column column number
*/
public void markTile(int row, int column) {
    throw new UnsupportedOperationException("Method markTile not yet
    implemented");
}
```

The following is an example of a possible solution to this task:

```
public void markTile(int row, int column) {
    final Tile tile = tiles[row][column];
    if (tile.getState() == Tile.State.CLOSED) {
        tile.setState(Tile.State.MARKED);
    } else if (tile.getState() == Tile.State.MARKED) {
        tile.setState(Tile.State.CLOSED);
    }
}
```

This task is quite simple. However, with the case study progress, the tasks are growing in difficulty and provide less guidance, as the students are expected to build upon the already-passed tasks. For example, one of the latest tasks in the course requires the user to handle best scores persistence through a database connection, as follows:

```
In the BestTimes class define a private void insertToDB(PlayerTime
playerTime) method that will store a PlayTime object in the database.
```

The students are also provided with a couple of general notes about creating database connections in Java and about prepared statements with links to official documentation, but otherwise, the students have to work on their own to come up with a correct implementation.

6.5. Refine Tasks with Comments

We suggest adding comments with hints to tasks to refine their difficulty. The tasks need to be challenging even for above-average students. However, in such a case, the

students that are below the average could become overwhelmed. To keep the tasks challenging for the better students while not making it too hard for the others, we recommend using hints in comments that would help with solving the tasks.

Students can start solving the task without looking at the comments at all (our study guides are rendered as HTML pages, and therefore, the hints can be hidden by default). Then, if a student finds the task too difficult, he can take a look at hints. There may be even levels of the hints that can be later taken into consideration when the solution is graded. For example, without using comments, a student would obtain a full number of the points, and then looking at each comment (hint) would be penalized by subtracting some points. Multilevel hints would allow for the finer-grained difficulty of tasks.

Example: The following is an example of a hint for the "markTile" task that was presented above. It advises using the implementation of the `openTile(int row, int column)` method for inspiration. The `openTile(int row, int column)` method does a very similar job and is provided to students in the Minesweeper code skeleton at the beginning of the course. Most of the students should not have any serious problems in solving the task without this hint. However, below-average students or programming novices may struggle with it, and this hint should refer them to the right direction even before they will need to ask the teacher for help.

```
When implementing the void markTile(int row, int column) method, you
can use the implementation of void openTile(int row, int column)
method as an inspiration.
```

6.6. Add Supplementary Tasks

The aforementioned tasks and comments should cover all the learning goals. After that, the students can face open problems in additional features with the experience they obtained from the main project. To support individuality, some supplementary tasks are added to each lesson, recommending features that are not covered by the standard tasks. Additionally, students are also asked to come up with their own ideas about additional features.

Supplementary tasks provide a space to try out the learned principles in a less restrictive environment. They are specified in the problem terminology. Only a feature is requested, but we do not add any information about implementation. Students have to solve the problem themselves now without our guidance.

Example: In the Minesweeper case study, the implementation of new features is suggested, such as support for a new state in marking tile—using the question mark to signify that the user is not sure whether there is a mine or not.

6.7. Provide Further Reading

A module can be concluded with a list of links to additional literature and resources. There can be links to blogs, tutorials, documentation, books, or even scientific papers—anything that might be interesting for the students who want to know more.

Example: In the Java technologies course, we recommend the book *Head First Java* for further reading. For the advanced study, *Effective Java* is recommended. For particular topics, students are usually provided with links to specialized tutorials and blogs.

6.8. Use Suitable Software Support

In the final stage, the study guide should be available online for students in some format, usually as a website. Since it is already time-consuming to devise the study guide itself, the technical requirements should not further complicate this process and distract the teacher from focusing on writing the content. Instead of hand-crafting HTML code or designing the guide in desktop publishing software, the teachers can utilize static website

generators that automatically produce websites from simple text documents written in formats such as Markdown or reStructuredText [46].

Example: At our university, we developed a specialized document generator for task-driven case study guides. This program transforms a set of Markdown files enriched with metadata representing the goals, tasks, etc. into a static website and possibly other formats, such as PDF. Utilizing custom software for this task also opens possibilities to check for inconsistencies, such as goals without any corresponding tasks.

6.9. Verify Continuity and Consistency of the Guide

After writing the study guides, they need to be checked for inconsistencies. All the tasks have to be correct, and they cannot contradict each other. Additionally, the coverage of the solution has to be checked. If a student follows the study guide and solves the main tasks (not additional), his solution has to be complete. If some implementation detail is forgotten, a student will probably face frustration—-he followed the guide, and yet the solution is not complete or, in a worse case, not even working. Explicit task marking in the source code can help to ensure that all the tasks in the code are stated in the study guide too. Unmarked code that does not belong to any task has to be given to students ready-made.

6.10. Review the Guide

The study guide should be reviewed too. However, for reviewing the guides, we use colleagues from different courses and older students. Their task is to try solving the case study and give us feedback about the form of the guide, the used terms, the style of writing, and so on. Older students can tell us whether the study guides are clear and easily understandable. With colleagues from different courses, we can unify our terminology.

The older students can provide the teacher with feedback about the overall difficulty of the case study. They can warn the teacher that the case study is too difficult for the target audience (since they are not so "distant" from it as the teacher's peers). This can be dealt with by providing students with more ready-made source code (a skeleton of the program, or some complex method), hints, or more elaborated task specifications.

6.11. Start with a Minimal Version and Improve It

This point applies both to the solution implementation and to study guide writing. We recommend first preparing a simple version of the source code and instructions that the students can already use and then gradually extending and improving them. This is similar to the concept of a minimum viable product in software engineering.

Regarding specific time frames, they are highly dependent on the specific course, teachers' experience with the subject, and many other factors. We managed to produce a minimal version of materials for one course in as little as two weeks of very intensive work. An upper limit is usually thirteen weeks, i.e., one semester. Therefore, it is possible to develop the materials continuously if the interconnection between individual lessons is carefully analyzed upfront.

A whole course can be implemented by one person or a small group of teachers. In the latter case, close cooperation and synchronization are necessary. In our experience, if the whole course consists of one large case study, two teachers are the upper practical limit. If multiple smaller case studies are provided, the number of teachers can be larger.

7. Course Execution

When the study guides are reviewed and ready, the case study can be put to use in a course. Students are given the guides, and they have to solve the case study. The case study is divided into modules to match the course lessons. We recommend beginning each lesson by introducing the current objectives and the problem context. Then students should be left to work on their own on the solution by solving tasks. The study guide should not substitute teachers but rather help them to get more time to approach students individually.

7.1. Monitor Progress Continuously

Based on our experience and literature [47], it is necessary to monitor students' progress continuously. They need to keep up with the plan that is determined by the study guides. When they start getting behind the plan, it usually gets only worse, and then often comes a moment when they feel overwhelmed and give up (or try to cheat by obtaining the implementation from their peers). Each module should be designed to be solvable in the time of the lesson, plus some homework time. Doing two or three modules at the same time might stress them so much that they will give up. However, we had also an opposite experience when good students finished the whole case study in a week or two because they found it so interesting.

Using multiple partial deadlines might help to keep the students working continuously. Requiring to deliver partial solutions after each lesson also mimics the industry, so it has the benefit of modeling the practice.

This monitoring also helps teachers to find out about students' errors as early as possible and to help them with misunderstood topics.

7.2. Favor Individual Achievements

To obtain better grades, students should be encouraged to exercise the skills that they obtained by working on the supplementary tasks. Performing only the mandatory tasks should be a minimum for passing the course. If a student wants a better grade, he should be proactive. This grading motivates students to try to apply the gained skills on their own.

Better students might find a case study too easy. In those cases, we let them work on their own projects, but first, they have to finish a case study to see the best practices (they are usually able to finish it in a few weeks instead of the whole semester). As a reward for this additional work, these students do not have to take the standard exam, but they rather have to prove their understanding of the principles by showing their application in their own projects.

Example: In the Java technologies course, we reward finishing the Minesweeper case study without any supplementary tasks by half of the maximum points available. The rest of the points can be obtained through additional tasks and custom features.

7.3. Examine Understanding

By finishing the case study, students exercised all the taught principles in practice. That does not necessarily mean that they have understood the topics and techniques. Therefore, the understanding has to be evaluated during the exams. Since the students were working on a comprehensive project the whole course, it is possible to assign them tasks concerning this project without risking that they will lose too much time comprehending the tasks' context.

Example: For our exams, an incomplete modified version of the Minesweeper solution is used. Students get to implement some missing part where they have to use the skills they should have acquired from the case study. The tasks are not the same as those in the case study, but they are similar in character. The following is an example of the exam task:

```
Implement the processInput() method that will process input from the
console using regular expressions. After~the implementation, the~game
 should be playable. After~each printing of the field, the~game would
 require typing some input. The~input is in the same format as in the
 case study: (X) EXIT, (MA1) MARK, (OB4) OPEN.
```

7.4. Get Feedback from the "Battlefield"

Reviews from the "training grounds" (colleagues, experts from industry, and older students) are very important, but these reviews will not discover all the issues with the case study. The best feedback is from the target audience.

Example: One of the important feedback entries we received from the students in the Java technologies course was the information that the task of implementing a method that counts adjacent mines to a tile is too difficult for many students (algorithmically). It did not look so difficult to us or our colleagues.

7.5. Update the Case Study

Feedback should be used to increase the quality of the case study and to keep it up-to-date. This update means going back to the *solution implementation* phase, modifying the teacher's solution, and then updating the guides as well. We highly recommend keeping the teacher's solution up-to-date with the guides, not only to update the study guides. An example of the benefits this brings is the fact that if a new teacher joins the course, it is much easier for him to get familiar with the case study.

Example: After finding out that the implementation of counting the adjacent mines is too difficult for the students, we decided to provide a hint to the task. If necessary, we could also include a note in the teachers' version of the study guide about discussing the algorithm with students.

8. Experience

We have been successfully using case studies in our courses for over 18 years. Our first task-driven case study started in 2006. Task-driven case studies have been used in several different courses at our university: the Minesweeper case study in the Java technologies course, N-Puzzle in the .NET platform programming, the Share Me case study in the distributed programming course, Karel the Robot in the C programming course, and others. Our point of view as teachers is presented here and supplemented by a summary of a survey that we conducted with our object-oriented programming students.

8.1. Teachers' View

Students are more active. Before the introduction of task-driven case studies, the teacher spent a large part of the practical labs explaining the instructions to the students. This resulted in less time allocated to the students' practical experience. However, with task-driven case studies, the whole lesson is focused on solving tasks.

Individual progress speed is a consequence of having the study guide with a detailed description of the case study and its tasks. Students follow the guide and ask a teacher for advice when a problem with the realization of the case study arises. The guide is accessible to students not only during the lessons but also during the rest of the semester. A student who is slower than average can continue with the study at home—and vice versa—an ingenious student does not have to wait for the teacher's explanation and can progress faster. This way, we can nurture better students while not forgetting about the rest.

The teacher has more time for an individual approach. Since, in an ideal case, the study guide gives the students all the necessary information, it partially substitutes the role of the teacher. This does not remove the need for a teacher though. The teachers act rather as consultants, and they can spend the time saved by the guide discussing the specific problems the individual students face.

The guide narrows the gap in the expertise of teachers. Because of the large number of students in our courses, there are usually multiple teachers per course. Since each teacher has a different set of qualities and abilities, we could not guarantee exactly the same conditions for everyone. The written guide gives all the students the same basic environment.

The case study presents the whole process. Since the case study is a whole project, not only an explanatory example, we can incorporate professional methods and tools commonly used in industry. This offers the students the feeling of methods and tools used for the design, testing, versioning, etc.

Students see showpieces, not toy examples, and therefore, we can show them the best practices. The case study and tasks lead them through our reasoning, so they understand and follow our design decisions (which were verified by multiple reviews).

The teacher monitors the students' progress, so they will not fail to deliver the assignment because of procrastination. Since the case study is divided into relatively balanced modules according to the number of practical lessons in the course, the teacher can easily determine the portion of the project that is already done. Instead of vague and ambiguous statements such as "I am working on a database layer", they are asked about which task they already fulfilled.

Teachers can share study guides with other universities and facilities. If the case study is good, other educators might be interested in using the same case study in their courses. Thanks to written study guides and pre-made teachers' solutions, the task-driven case study is highly reusable.

Task-driven case studies open space for tools in the process of teaching. All of the students have to work on the same project that has a certain main skeleton that they have to follow. Standard, non-supplementary tasks can be therefore checked automatically by a tool; in case of standard mistakes and errors, the feedback can be provided to students; etc.

Although the method has many benefits and is worth applying in education, we are aware of its imperfection. The drawbacks of the method are outlined in Section 9.

8.2. Students' View

To evaluate the advantages and shortcomings of our approach from the students' point of view, we conducted a survey with students that were using a task-driven case study during one of their courses.

8.2.1. Objective

Our main goal was to answer the following research question: how do students perceive the task-driven case study approach when applied to their course?

We used a mixed-method research approach and gathered both quantitative and qualitative data, with a main focus on the quantitative part.

8.2.2. Method

An online questionnaire was constructed using Google Forms. It consisted of 13 single-choice, 1 multiple-choice, and 3 open-ended questions. For a list of questions, see Table 1.

The questionnaire was sent to all second-year computer science bachelor's degree students who took the object-oriented programming course, where the task-driven case study was used. They also had previous experience with this method from the programming course. In the given semester, a portion of the students were using a more mature case study called Text Game, specifically an adventure with a textual command-based interface. The rest of the students used the new Alien Breed case study, which is a game with 2D isometric graphics. Participation in the survey was voluntary; we received 112 responses out of 180 invitations.

For single-choice and multiple-choice questions, absolute and relative numbers of responses for each answer were computed using spreadsheets. The responses to the free-form questions were read by one of the authors, gradually grouped into categories, and then summarized textually.

Table 1. Survey questions and results.

#	Question	Answer	No.	%
1	Do you like learning by implementing a game?	**Yes**	103	92
		No, I would rather implement something else	9	8
2	How do you like the implemented game?	Poor (is it still a game?)	0	0
		Below average (I would never play it)	18	16
		Average	56	50
		Above average (I would definitely try it)	36	32
		Excellent (from now on, I will play only this game)	2	2
3	Would you show your game to your friend or a family member?	**Yes**	87	78
		No	25	22
4	In practical lessons, you prefer to implement:	**One large project (as in Alien Breed or Text Game)**	84	75
		Multiple simple independent tasks	28	25
5	Do you think you understood programming principles better by implementing one large project?	**Yes**	91	81
		No	21	19
6	From the point of view of assignment organization, you prefer:	**Study guide that leads me through the process**	97	87
		To get the assignment in the beginning and to solve it on my own	15	13
7	Did you have problems with dependencies between lessons, i.e., that you had to solve a previous lesson to be able to continue?	Yes	48	43
		No	64	57
8	Was the difficulty of the tasks in the case study balanced?	Yes	48	43
		No, some were too easy and some too difficult	64	57
9	Were the tasks too easy?	Yes, most of the time, I just needed to repeat what was written	3	3
		No, I usually had to think more about it	109	97
10	Were the tasks described clearly enough?	**Yes, usually I understood the task without the help of the teacher**	57	51
		No, often I had to ask the teacher or colleagues for help	55	49
11	Would you like learning with study guides for a case study in future courses? *	**Yes**	103	92
		No	8	7
12	Do you think that working with a study guide:	Limits me because I cannot do what I want	22	20
		Does not limit me; I still have enough space for my individuality	90	80
13	When did you implement the tasks for a given lesson?	I programmed mostly before the lesson	12	11
		I programmed during the lesson	24	21
		I programmed after the lesson	76	68
14	Which properties of studying with study guides for a case study do you consider most important (choose max. 3)?	I implemented a large project	69	62
		I implemented a game	34	30
		Thanks to the online study guide, I could work at my own speed	81	72
		I had a study guide that lad me to good practices	55	49
		I worked incrementally, but the game was always playable	42	38
15	What did you like about practical lessons in the OOP course?			
16	What did you dislike about practical lessons in the OOP course?			
17	What would you change or improve about practical lessons in the OOP course?			

* One of the respondents did not choose any response for this question.

8.2.3. Results

The results for individual questions are displayed in Table 1. In the rest of the section, the results concerning the individual aspects related to task-driven case studies will be summarized.

Motivation, particularly intrinsic motivation, is an important aspect of learning efficiency [48]. Although the task-driven case study approach is not limited to games, they represent an interesting topic for a case study. Indeed, 92% of the students stated that they like learning by implementing a game in contrast with other types of applications, as can be seen in question 1. In question 3, the majority of the participants (78%) responded that they would show their game to a friend or family members, which could be one of the motivational factors to implement the application with their best effort.

Question 3 also emphasizes the need to implement a project in **industrial quality and in a well-known domain**. Here, we analyzed also responses for the Text Game and Alien Breed case studies individually. While, for the former, only 67% of the students would present their solution to other people, for Alien Breed, this number reached 81%. Text Games are less known nowadays, while Alien Breed represents a genre of graphical action games that many people recognize. Students also believe that graphical applications are more relevant to their future jobs in the industry. Similar results were achieved in question 2, where the students also preferred Alien Breed: 41% perceived their creation as above average or excellent, and only 11% in the case of Text Game.

The importance of **showing the whole process** and **not using toy examples** was confirmed by questions 4 and 5. Three-quarters of the students preferred one large project instead of multiple small independent tasks. A total of 81% thought that they understood programming principles better by using this approach.

Questions 6 and 11 were focused on **task-oriented study guides**. A vast majority of the students (87%) preferred guides leading them through the process of project implementation in contrast with receiving only instructions at the beginning. Furthermore, 92% of the participants would like to use task-driven study guides in future courses.

Since **dependencies between lessons** are present in task-oriented case studies, students should finish previous modules to continue with the next one. For 43% of the students, this presented a problem sometimes, while 57% did not encounter any difficulties with lesson dependencies. This should be taken into account by the instructors, e.g., by reminding the students of this fact periodically and by continuously checking the progress.

In questions 8 to 10, the **quality of tasks and their descriptions** was evaluated. More than half (57%) of the students did not consider the tasks well balanced. However, this is not a crucial property of task design, although it can make solving them more manageable. Only 3% thought the tasks were too easy, and they just needed to read the provided study guides leading them through the whole process and repeat the instructions in their own projects without considerable thinking effort. According to half of the students (51%), the instructions were clear enough to proceed without help, while the rest had to ask the teacher or colleagues additional questions. We think that the clarity of tasks could be improved by providing multilevel hints (additional hints with gradually more detailed information) in the future.

We were also interested in whether the study guides did not limit the students' **creativity**. According to the results of question 12, the majority of the students (80%) thought that the guide still left enough space for their individuality. This could be attributed also to the possibility of own extensions and enhancements in addition to the mandatory tasks.

The importance of **individual progress speed** was evaluated in question 13. Since the study guides with complete instructions are available online, the students can program not only during the lesson (21%) but also before it (11%) or after it (68%). The last option was popular because students often prefer to work in their home environment with their own equipment, and they are frequently unable to fully finish all tasks during the time of the lesson itself.

Summing up, the task-driven case study method is popular among students, with a vast majority desiring to apply it also in their future courses. Care must be taken mainly to describe the tasks clearly and provide additional hints if necessary.

9. Potential Drawbacks

There are also some potential drawbacks to using task-driven case studies. Along with the drawbacks that we experienced, our approaches to decrease their effect are presented.

Case study costs—one of the most significant reasons against using a task-driven case study is the costs of preparing and evolving such a case study. Preparing a good task-driven case study is time-consuming.

One of the problems causing the increase in costs is the relationship between the teacher's solution and the study guides. Working on the two resources simultaneously is more complicated due to synchronization. We have proposed a solution using source code annotations to define tasks in the teacher's solution and to generate the contents of the study guides from these annotations. More about this work can be found in [49].

Task difficulty—balancing the difficulty of the tasks is really hard. They cannot be too challenging so that all students would be able to solve them. On the other hand, the tasks cannot be so easy that the case study would not become a simple tutorial. Our experience is that sometimes a good difficulty can be achieved only after trying a case study during a course and obtaining feedback from students (in Section 8, our experience with the find adjacent mines method was mentioned).

In Section 6, we mentioned comments with hints that can help you in dealing with this issue.

Creativity—in our approach, all students work on the same case study, and the whole process is controlled by the study guides and teachers. From this view, it may look like the creativity of students is impeded.

We deal with this issue by trying to find a balanced ratio of mandatory tasks and additional tasks.

Strong dependencies—the lessons usually strongly depend on each other. If a student misses a lesson for some reason (e.g., sickness), he cannot continue without solving the missed lesson. It is not possible to start solving the current lesson unless the source code from the previous lesson is finished. In our experience, when students decide to take a longer break, it is really hard for them to get working again.

To keep them working continuously through the course, their progress at every lesson is monitored, and when we see that they are behind, we try to find out why and help them catch up.

Instant execution—in some case studies, it is not possible to achieve program executability after every lesson. In our experience, a runnable version after each lesson acts as a motivational factor. Having an executable program allows them to check for possible mistakes and differences between the requirements and their solutions.

When there is no way to make some lesson content executable, we suggest providing students at least with unit tests that would give them feedback about whether their solution to the lesson's task is correct. Besides the advantage of executability, unit tests mimic the real industrial process.

To deal with this issue, there is an alternative to the incremental approach. The differential approach gives a whole solution to students from the very beginning. However, the solution is provided as a black-box library. Throughout the lessons, they are given new libraries, each one missing more from the implementation. The students' task is to implement the missing part. This way, the solution is runnable after each lesson. However, this approach has its issues too. Here, part of the motivational factor is lost since, after each lesson, they do not see anything new—the solution is always the same, and changing is merely a ratio of their implementation to the library.

In practice, we use a rather combined approach. Students are building the solution incrementally, but to speed up the parts that would make a lesson inexecutable, students are provided with some code written by a teacher. When one builds a real-scale application, he cooperates on it with other developers, and the teacher's source code mimics this.

Plagiarism—plagiarism and cheating are not exclusive problems of the case studies. However, since there is usually only one case study per course, it is easier for students

to obtain a solution from older students or their peers. It would be possible to prepare multiple case studies to teach the same goals, but yet one good case study is an expensive and challenging task for a teacher.

Furthermore, case studies sometimes make it more difficult to decide what constitutes plagiarism. Students' solutions use the same skeleton, and some methods are so simple that it is not easy to write completely new implementations even when a student works alone.

To some degree, it is possible to apply traditional preventive and repressive measures for plagiarism, such as more frequent checking of the students' progress or automated similarity checks.

10. Conclusions

While both case studies and the task-driven methodology are well-known teaching approaches, their combination is our contribution. This paper represents an introduction to task-driven case studies. It identifies its lifecycle and shows the proper relationship between the case study and the goals of a course. In our experience, case studies combined with the task-driven methodology have proven themselves a worthwhile approach to teaching. We have backed our experience also with a survey with students, who appreciated also the study guides that are an integral part of the method.

An important part of the paper is guidelines for preparing and executing a task-driven case study. These guidelines are extracted from our experience with using task-driven case studies over multiple years. They should be useful for teachers that start with our method, but many of them can be used also in the context of teaching in general.

Although we were able to devise an effective approach to teaching, there is still a lot of space for improvement. Section 9 mentioned multiple problems with task-driven case studies. We have already made some progress in the case study costs, which was published in one of our previous papers [49]. However, there is still an important problem with the definition of tasks, which was identified by the survey too. In this matter, we plan to experiment with multilevel hints. Additional quantitative evaluation is also one of the tasks for future research.

Author Contributions: Conceptualization, J.P. and M.N.; methodology, M.N.; software, J.P., M.N., and S.C.; validation, M.S. and M.N.; formal analysis, M.S.; investigation, M.N.; resources, J.P.; data curation, M.N.; writing—original draft preparation, J.P., M.N., M.S., and S.C.; writing—review and editing, J.P., M.N., M.S., and S.C.; visualization, M.N.; supervision, J.P.; project administration, J.P.; funding acquisition, J.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by VEGA No. 1/0630/22 Lowering Programmers' Cognitive Load Using Context-Dependent Dialogs.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The anonymized results of a survey with students are available in the public repository at https://doi.org/10.17605/OSF.IO/WT7C4 (accessed on 30 August 2024).

Conflicts of Interest: Author Milan Nosál' was employed by the company ValeSoft, s.r.o. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Hadar, I. When intuition and logic clash: The case of the object-oriented paradigm. *Sci. Comput. Program.* **2013**, *78*, 1407–1426. [CrossRef]
2. Xue, J.; Zhang, L. Application of task-driven approach in information technology education. In Proceedings of the International Conference on Electrical and Control Engineering 2011, 2011, ICECE 2011, Yichang, China, 6–18 September 2011; pp. 2024–2026. [CrossRef]
3. Rosenberg-Kima, R.B.; Merrill, M.D.; Baylor, A.L.; Johnson, T.E. Explicit instruction in the context of whole-tasks: The effectiveness of the task-centered instructional strategy in computer science education. *Educ. Technol. Res. Dev.* **2022**, *70*, 1627–1655. [CrossRef]

4. Bonwell, C.C.; Eison, J.A. *Active Learning: Creating Excitement in the Classroom*; ASHE-ERIC Higher Education Report No. 1; School of Education and Human Development, The George Washington University: Washington, DC, USA, 1991.
5. Anwar, S. Use of engineering case studies to teach associate degree electrical engineering technology students. In Proceedings of the 31st Annual Frontiers in Education Conference, Washington, DC, USA, 10–13 October 2001; Volume 3, pp. 8–10. [CrossRef]
6. Nilson, L.B. *Teaching at Its Best: A Research-Based Resource for College Instructors*; John Wiley & Sons: Hoboken, NJ, USA, 2010.
7. Coppit, D. Implementing large projects in software engineering courses. *Comput. Sci. Educ.* **2006**, *16*, 53–73. [CrossRef]
8. Garg, K.; Varma, V. A Study of the Effectiveness of Case Study Approach in Software Engineering Education. In Proceedings of the 20th Conference on Software Engineering Education Training, Dublin, Ireland, 3–5 July 2007; CSEET '07; pp. 309–316. [CrossRef]
9. Daun, M.; Salmon, A.; Tenbergen, B.; Weyer, T.; Pohl, K. Industrial case studies in graduate requirements engineering courses: The impact on student motivation. In Proceedings of the 2014 IEEE 27th Conference on Software Engineering Education and Training (CSEE&T), Klagenfurt, Austria, 23–25 April 2014; pp. 3–12. [CrossRef]
10. Porubän, J.; Nosáľ, M. Practical experience with task-driven case studies. In Proceedings of the 2014 IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, 4–5 December 2014; pp. 367–372. [CrossRef]
11. O'Grady, M.J. Practical Problem-Based Learning in Computing Education. *Trans. Comput. Educ.* **2012**, *12*, 10:1–10:16. [CrossRef]
12. Leijon, M.; Gudmundsson, P.; Staaf, P.; Christersson, C. Challenge based learning in higher education– A systematic literature review. *Innov. Educ. Teach. Int.* **2021**, *59*, 1–10. [CrossRef]
13. Abdul Ghani, A.; Fuad, A.; Yusoff, M.S.B.; Hadie, S.N.H. Effective Learning Behavior in Problem-Based Learning: A Scoping Review. *Med Sci. Educ.* **2021**, *31*, 1199–1211. [CrossRef] [PubMed]
14. Pérez, B.; Rubio, A.L. A Project-Based Learning Approach for Enhancing Learning Skills and Motivation in Software Engineering. In Proceedings of the 51st ACM Technical Symposium on Computer Science Education, New York, NY, USA, 11–14 March 2020; SIGCSE '20; pp. 309–315. [CrossRef]
15. Cico, O.; Jaccheri, L.; Nguyen-Duc, A.; Zhang, H. Exploring the intersection between software industry and Software Engineering education - A systematic mapping of Software Engineering Trends. *J. Syst. Softw.* **2021**, *172*, 110736. [CrossRef]
16. Grimes, M.W. The Continuous Case Study: Designing a Unique Assessment of Student Learning. *Int. J. Teach. Learn. High. Educ.* **2019**, *31*, 139–146.
17. Zhang, X.; Zhang, B.; Zhang, F. Student-Centered Case-Based Teaching and Online–Offline Case Discussion in Postgraduate Courses of Computer Science. *Int. J. Educ. Technol. High. Educ.* **2023**, *20*, 6. [CrossRef]
18. Ouh, E.L.; Irawan, Y. Applying Case-Based Learning for a Postgraduate Software Architecture Course. In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, Aberdeen Scotland, UK, 15–17 July 2019; pp. 457–463. [CrossRef]
19. Varma, V.; Garg, K. Case studies: The potential teaching instruments for software engineering education. In Proceedings of the Fifth International Conference on Quality Software, 2005, QSIC 2005, Melbourne, VIC, Australia, 19–20 September 2005; pp. 279–284. [CrossRef]
20. Garg, K.; Varma, V. Case Studies as Assessment Tools in Software Engineering Classrooms. In Proceedings of the 22nd Conference on Software Engineering Education and Training, Hyderabad, India, 17–20 February 2009; CSEET '09; pp. 8–11. [CrossRef]
21. Jia, Y. Improving software engineering courses with case study approach. In Proceedings of the 5th International Conference on Computer Science and Education 2010, Hefei, China, 24–27 August 2010; ICCSE 2010; pp. 1633–1636. [CrossRef]
22. Burge, J.; Troy, D. Rising to the Challenge: Using Business-Oriented Case Studies in Software Engineering Education. In Proceedings of the 19th Conference on Software Engineering Education and Training, Turtle Bay, HI, USA, 19–21 April 2006; pp. 43–50. [CrossRef]
23. Hilburn, T.; Towhidnejad, M.; Nangia, S.; Shen, L. A Case Study Project for Software Engineering Education. In Proceedings of the 36th Annual Frontiers in Education Conference, San Diego, CA, USA, 27–31 October 2006; pp. 1–5. [CrossRef]
24. Martin, F. Toy Projects Considered Harmful. *Commun. ACM* **2006**, *49*, 113–116. [CrossRef]
25. Meyer, B. Software engineering in the academy. *Computer* **2001**, *34*, 28–35. [CrossRef]
26. Yu, D.; Wang, Q. Task-Driven Method in Practical Teaching of Software Engineering. In Proceedings of the Third Pacific-Asia Conference on Circuits, Communications and System 2011, Wuhan, China, 17–18 July 2011; PACCS 2011; pp. 1–3. [CrossRef]
27. Xie, C.; Wang, M.; Hu, H. Effects of Constructivist and Transmission Instructional Models on Mathematics Achievement in Mainland China: A Meta-Analysis. *Front. Psychol.* **2018**, *9*, 1923. [CrossRef] [PubMed]
28. Peng, W.; Jingjing, X. The implementation and harvests of task-driven in basic computer education at university. In Proceedings of the International Conference on E-Health Networking, Digital Ecosystems and Technologies 2010, Shenzhen, China, 17–18 April 2010; EDT 2010; Volume 2, pp. 311–314. [CrossRef]
29. Liu, H.H.; Su, Y.S. Effects of Using Task-Driven Classroom Teaching on Students' Learning Attitudes and Learning Effectiveness in an Information Technology Course. *Sustainability* **2018**, *10*, 3957. [CrossRef]
30. Liang, L.; Deng, X.; Liu, Q. Task-driven and objective-oriented hierarchical education method: A case study in Linux curriculum. In Proceedings of the IEEE International Symposium on IT in Medicine and Education 2008, Xiamen, China, 12–14 December 2008; ITME 2008; pp. 316–318. [CrossRef]

31. Dong, Y. A Graded Task-driven Methodology for Computer Science Education. In Proceedings of the Second International Workshop on Education Technology and Computer Science 2010, Wuhan, China, 6–7 March 2010; ETCS 2010; Volume 3, pp. 654–656. [CrossRef]
32. Birnbaum, D.J.; Langmead, A. Task-Driven Programming Pedagogy in the Digital Humanities. In *New Directions for Computing Education*; Fee, S.B., Holland-Minkley, A.M., Lombardi, T.E., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 63–85. [CrossRef]
33. Martinez, L.; Gimenes, M.; Lambert, E. Entertainment Video Games for Academic Learning: A Systematic Review. *J. Educ. Comput. Res.* **2022**, *60*, 1083–1109. [CrossRef]
34. Mayer, R.E. Computer Games in Education. *Annu. Rev. Psychol.* **2019**, *70*, 531–549. [CrossRef] [PubMed]
35. Zhan, Z.; He, L.; Tong, Y.; Liang, X.; Guo, S.; Lan, X. The effectiveness of gamification in programming education: Evidence from a meta-analysis. *Comput. Educ. Artif. Intell.* **2022**, *3*, 100096. [CrossRef]
36. Papadakis, S. Evaluating a Game-Development Approach to Teach Introductory Programming Concepts in Secondary Education. *Int. J. Technol. Enhanc. Learn.* **2020**, *12*, 127–145. [CrossRef]
37. Jordaan, D.B. Board Games in the Computer Science Class to Improve Students' Knowledge of the Java Programming Language: A Lecturer's Perspective. In Proceedings of the 2nd International Conference on Education and Multimedia Technology, New York, NY, USA, 2–4 July 2018; ICEMT '18; pp. 1–4. [CrossRef]
38. Tay, J.; Goh, Y.M.; Safiena, S.; Bound, H. Designing digital game-based learning for professional upskilling: A systematic literature review. *Comput. Educ.* **2022**, *184*, 104518. [CrossRef]
39. Díaz, J.; López, J.A.; Sepúlveda, S.; Ramírez Villegas, G.M.; Ahumada, D.; Moreira, F. Evaluating Aspects of Usability in Video Game-Based Programming Learning Platforms. *Procedia Comput. Sci.* **2021**, *181*, 247–254. [CrossRef]
40. Lindberg, R.S.N.; Laine, T.H.; Haaranen, L. Gamifying programming education in K-12: A review of programming curricula in seven countries and programming games. *Br. J. Educ. Technol.* **2019**, *50*, 1979–1995. [CrossRef]
41. Cavalcanti, A.P.; Barbosa, A.; Carvalho, R.; Freitas, F.; Tsai, Y.S.; Gašević, D.; Mello, R.F. Automatic feedback in online learning environments: A systematic literature review. *Comput. Educ. Artif. Intell.* **2021**, *2*, 100027. [CrossRef]
42. Mingins, C.; Miller, J.; Dick, M.; Postema, M. How We Teach Software Engineering. *JOOP* **1999**, *11*, 64–66.
43. Nuci, K.P.; Tahir, R.; Wang, A.I.; Imran, A.S. Game-Based Digital Quiz as a Tool for Improving Students' Engagement and Learning in Online Lectures. *IEEE Access* **2021**, *9*, 91220–91234. [CrossRef]
44. Shaw, M. Software Engineering Education: A Roadmap. In Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA, 8–13 November 2000; ICSE '00; pp. 371–380. [CrossRef]
45. Nosáľ, M.; Sulír, M.; Juhár, J. Source code annotations as formal languages. In Proceedings of the 2015 Federated Conference on Computer Science and Information Systems (FedCSIS), Lodz, Poland, 13–16 September 2015; pp. 953–964. [CrossRef]
46. Diaz, C. Using static site generators for scholarly publications and open educational resources. *Code4Lib J.* **2018**.
47. Leinonen, J.; Denny, P.; Whalley, J. A Comparison of Immediate and Scheduled Feedback in Introductory Programming Projects. In Proceedings of the 53rd ACM Technical Symposium on Computer Science Education—Volume 1, New York, NY, USA, 2–5 March 2022; SIGCSE 2022; pp. 885–891. [CrossRef]
48. Kian, T.W.; Sunar, M.S.; Su, G.E. The Analysis of Intrinsic Game Elements for Undergraduates Gamified Platform Based on Learner Type. *IEEE Access* **2022**, *10*, 120659–120679. [CrossRef]
49. Porubän, J.; Nosáľ, M. Generating Case Studies from Annotated Sources Codes. *J. Comput. Sci. Control Syst.* **2013**, *6*, 81–86.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

computers

MDPI

Article

Gamified Project-Based Learning in Vocational Education and Training Computer Science Courses

Belkis Díaz-Lauzurica *,† and David Moreno-Salinas †

Department of Computer Science and Automatic Control, Universidad Nacional de Educación a Distancia (UNED), 28040 Madrid, Spain; dmoreno@dia.uned.es

* Correspondence: bdiaz216@alumno.uned.es

† These authors contributed equally to this work.

Abstract

Active methodologies place the student at the core of the teaching–learning process, with the teacher becoming a companion and guide. Among these methodologies, gamification is demonstrating great capacity to attract students and promote interest, being of particular relevance in STEM subjects. While gamification and Project-Based Learning (PBL) have been extensively studied independently, their integration into Vocational Education and Training (VET) computer science courses remains underexplored, particularly regarding approaches where students develop games themselves rather than merely incorporating game elements or playing serious games. This work presents a novel gamified PBL approach specifically designed for VET Programming education, with three distinctive features: (i) students develop a complete game based on graph theory and Object-Oriented Programming, with each student working under personalised conditions and constraints; (ii) a custom-developed software tool that simultaneously serves as a pedagogical scaffold for students to validate their solutions iteratively and as an automated evaluation platform for teachers; and (iii) empirical validation through action-research with first-year VET students, employing mixed-methods analysis including qualitative observations and descriptive quantitative comparisons. The approach was implemented with first-year Web Application Design students in the Programming subject, where students developed a game integrating graph theory algorithms, Object-Oriented Programming, and Markup Language. Despite the small sample size (10 students), qualitative observations and descriptive analysis indicated promising results, and grade distributions were comparable to those in more accessible subjects. Teacher diary observations, follow-ups, and questionnaires documented sustained engagement, peer collaboration, and strategic problem-solving throughout the project phase. These preliminary findings suggest that gamification through game development, particularly when supported by automated tools enabling personalised conditions and iterative validation, represents a promising approach for teaching and learning Programming in VET contexts.

Keywords: active learning; project-based learning; gamification; vocational education; programming; personalised learning; automated assessment

1. Introduction

The rapid evolution of technology, social relations and the labour market requires the teaching learning process, and education itself, to adapt to this rapid transformation, providing individuals that can fully and confidently join society and the labour market [1,2].

This is especially important in STEM (Science, Technology, Engineering, and Mathematics) areas, where the constant evolution of the content taught requires the use of flexible and adaptable methodologies in which students can actively contribute and participate.

Among the different methodologies that can be implemented, active methodologies stand out. These methodologies place students at the core of the teaching–learning process, making them the protagonists of the process and placing the teacher in a guiding or accompanying role. These methodologies have proven to be particularly well-suited to STEM students, where the practical application of theoretical concepts is essential for proper training [3]. Furthermore, the complexity and difficulty of some STEM subjects lead to high dropout and absenteeism rates among students [4], a critical problem that these methodologies can mitigate by increasing students' interest.

This dropout problem is more prevalent in the early years of Vocational Education and Training (VET), especially in technological fields such as computer science. Some of these students choose computer science courses in search of job opportunities or due to family pressure, rather than authentic interest or true vocation. In addition, many of them have had poor previous educational experiences in Compulsory Secondary Education (CSE). All of this makes it difficult to create a suitable learning environment in the classroom [5,6]. Therefore, the application and development of adequate methodologies is of the utmost importance in order to engage students, impart the content in an interesting and practical way, and make the learning process adaptable to any student profile.

Active methodologies have shown positive results in previous works [7–9], enhancing interest, motivation, and important skills, such as Computational Thinking [10–12]—findings also reported in previous work by the authors [3,13,14].

Although gamification and Project-Based Learning have been extensively studied independently, three critical gaps persist in VET computer science contexts.

Most gamification research in Programming education falls into two categories: either studies adding game elements (points, badges, leaderboards) to traditional instruction [15,16] or the employment of serious games where students learn by playing purpose-designed educational games [17]. A third approach, gamified Project-Based Learning in which students create games themselves as the learning activity, remains comparatively underexplored in VET contexts, despite its potential to combine authentic motivation with development of technical skills.

The personalisation of Programming projects is uncommon in documented gamified interventions [15,18,19]. Identical requirements of all students simplifies assessment but facilitates superficial copying and discourages deep engagement with adaptable algorithmic thinking. Consequently, personalisation appears pedagogically valuable, requiring students to understand principles deeply enough to adapt algorithms to specific scenarios rather than merely replicating example code. However, it creates substantial assessment burdens for teachers when each student works under unique constraints. The practical challenge of scaling personalised gamified projects while maintaining assessment rigour and teacher workload sustainability has not been adequately addressed in VET Programming education literature, as far as the authors know.

Automated assessment tools for Programming typically evaluate predefined problems with single correct answers. Extending automation to creative, open-ended projects with personalised conditions and multiple valid solutions represents a technical and pedagogical challenge not well documented in VET contexts. The question of how technological tools can simultaneously support student learning (providing iterative feedback during development) and teacher evaluation (enabling efficient assessment of diverse implementations) remains an open research problem, particularly for complex projects integrating multiple advanced concepts.

https://doi.org/10.3390/computers15020082

This work addresses these gaps by presenting a novel gamified Project-Based Learning approach specifically designed for VET Programming education, with three distinctive contributions that extend beyond the existing literature.

First, a methodological framework employing gamified Project-Based Learning with systematic personalisation. Unlike approaches using game elements or serious games, students develop a complete functional game, with each student working under unique conditions. This personalisation requires adaptive algorithmic thinking and integration of multiple advanced concepts (graph theory, OOP, Markup Language) using industry-standard tools (such as Java and Eclipse) rather than introductory block-based environments. The value added by gamified Project-Based Learning, compared to other gamification variants, lies in authentic motivation through creative authorship combined with rigorous technical skill development. Students are building purposeful artefacts that demonstrate deep understanding through adaptation to personalised constraints.

Second, a custom-developed software tool with dual functionality addressing the personalisation assessment challenge. The tool serves simultaneously as pedagogical scaffold (enabling students to test solutions step-by-step, visualise algorithmic decisions, compare their strategies against optimal solutions, and iteratively refine implementations) and as an automated assessment platform (allowing teachers to evaluate personalised projects efficiently/automatically). This dual-purpose design enables personalised projects at scale by reducing teacher evaluation burden while maintaining assessment rigour.

Third, mixed-methods empirical evidence from authentic VET implementation documenting both the approach's potential and its challenges. Through action-research combining qualitative observations (teacher diary, follow-ups, interviews, questionnaires, project artefacts) with descriptive quantitative comparisons (grades across subjects and academic years), this study provides preliminary evidence that gamified PBL with personalisation and automated tool support can engage VET students and support learning of traditionally challenging concepts in first-year courses.

The rest of the paper is organised as follows. In Section 2 the theoretical foundation, positioning relative to existing work, and research exploratory propositions are presented. In Section 3 the methods, data collection instruments, and evaluation are given; and the software tool is described in Section 4. The results obtained are detailed and discussed in Section 5 and in Section 6, respectively. Finally, the conclusions are described in Section 7.

2. Theoretical Foundation and Justification

There are multiple methods, approaches and tools that can be applied in computer science subjects to improve the motivation and interest of VET students. Some approaches use hardware–software platforms [20], robotics kits [21], or educational software and visual Programming environments [22–25]. Some of these latter environments are specially designed for novice programmers, such as Scratch, Code.org, Alice, Greenfoot, Blockly Games and App Inventor, whose advantages are analysed in different works [26–30].

Active methodologies allow the inclusion of these methods, approaches and tools during the class. The roots of these methodologies, in which the students have to take an active role in their own learning, reside in the constructivist approach that comes from the works of Vygotsky, Bruner, Piaget and Dewey [31,32]. Some of the most known are Flipped Classroom (FP), Project-Based Learning (PBL) and gamification, which are briefly introduced.

- Flipped Classroom has previously been applied in a VET context [33,34], and, briefly, it consists in inverting the teaching order. The theoretical content is acquired at home with material provided to the students, and the classroom time is dedicated to exercises and practical work [35].

https://doi.org/10.3390/computers15020082

- Project-Based Learning (PBL) is perhaps the best known and extended active methodology [36]. It consists in acquiring the required knowledge by working in projects and practical activities [37,38]. This methodology fits well with STEM education and VET students [39–41], and it has an important and positive impact on the classroom environment [42]. This methodology is highly adaptable, having given rise to many other closely related but slightly different methodologies, such as Design Thinking (DT) [43–47] or Cooperative Learning (CL) [48].
- Gamification introduces game-like elements into the teaching–learning process or uses games to promote interest and engage students. These activities may include new technologies for blended learning [49], game elements [16], or educational games [17] and virtual reality [50–52]. There are three main approaches for the application of gamification: the use of game elements such us rewards, points, etc.; the learning through playing serious games; and gamified Project-Based Learning, in which the students develop the game [18,19]. This methodology has shown very good results in VET [15].

These methodologies share the objective of increasing the interest and motivation of students by acquiring the necessary competences through their own experience. Gamification and PBL are well-suited for VET profiles and Programming learning [36,53], and particularly for the teaching–learning of Object-Oriented Programming, given the abstraction skills required and OOP's principles (e.g., encapsulation, inheritance, and polymorphism), as indicated in [54]. However, there is a lack of literature about the use of these active methodologies in VET, and particularly in computer science courses.

The work at hand presents a gamified PBL teaching–learning intervention in the first courses of computer science in VET, particularly in the Programming subject. For this purpose, a project for the design and development of a game based on the "cops and thieves" narrative is proposed. Students should design the game using graph theory, OOP, and Markup Language, with personalised conditions, and they should implement the logic for both game roles: the thief attempting to reach the destination and the police attempting interception. In addition, to support the students' learning and the teacher work, a software tool has been designed and developed in order to test and evaluate the solutions automatically. This tool allows the students to check their solutions and improve them, and the teacher to automatically assess the games presented by the students.

To contextualise the contributions of this work, it is essential to position it within the broader landscape of research on gamification, Project-Based Learning, and Programming education in VET contexts. The literature identifies three primary approaches to gamification in education as previously introduced [15,18,19]: game elements integration (adding mechanics such as points, badges, and leaderboards to traditional instruction), serious games (purpose-designed games where students learn by playing), and gamified Project-Based Learning (students create games themselves). While the first approach is widely studied and the second has shown effectiveness in VET contexts [15,49], the third approach is less common but potentially more powerful for Programming education. In this latter approach, the technical skills are developed by designing and implementing a functional game using professional tools. This work employs gamified Project-Based Learning, but extends existing implementations in several key ways.

Most Programming projects, gamified or not, described in the literature provide identical requirements to all students [18,26,28,30,55], which simplifies assessment but may discourage deep engagement. In contrast, this work implements individualised game conditions for each student, requiring students to understand principles deeply enough to adapt algorithms to their specific scenario. To our knowledge, no existing work on gamified Project-Based Learning in VET contexts systematically personalises project conditions while

https://doi.org/10.3390/computers15020082

simultaneously providing automated tools to support both learning and assessment under these varying constraints.

Gamified projects with personalised conditions create two main challenges. Students need iterative feedback to debug complex implementations but teachers cannot manually test every student's unique conditions multiple times. This way, final assessment becomes time consuming when each submission requires evaluation under different rules. The custom-developed tool in this work addresses both challenges through a dual functionality, serving simultaneously as pedagogical scaffolding (enabling students to test solutions step-by-step, visualising routes and comparing strategies against optimal solutions) and as automated assessment platform (allowing teachers to evaluate personalised projects efficiently, reducing evaluation time). While automated assessment tools exist, they typically evaluate predefined problems with single correct answers. Extending automation to creative, open-ended projects with personalised conditions and multiple valid solutions represents a technical and pedagogical advance.

Furthermore, most gamified Programming education focuses on introductory concepts using visual, block-based environments like Scratch, Alice, or Blockly [13,22,24,25], which are excellent for novices but insufficient for VET students preparing for industry careers. This work demonstrates that gamified Project-Based Learning can be extended to first-year VET courses integrating multiple advanced concepts simultaneously while maintaining the technical rigour appropriate for VET. The game development context provides motivation with a narrative for the use of theoretical Programming concepts. This is particularly important in VET, which presents distinctive challenges that can limit commitment and motivation, such as high early dropout rates, diverse prior preparation, external work pressures, and wide variation in intrinsic motivation [5,6]. This work provides exploratory findings, based on action research with a small sample, suggesting that the integration of advanced concepts through gamified Project-Based Learning, project personalisation, and a custom tool for testing and assessment can engage VET students and support learning. Although these findings are constrained by sample limitations and should be interpreted with caution, they establish feasibility and identify patterns that warrant larger-scale investigation.

Building on this positioning within the existing literature, the specific contributions of this work are threefold: (i) A gamified PBL approach employing gamification to develop a game rather than using game elements or serious games, with personalisation to ensure that each student works under unique conditions. This personalisation prevents superficial copying while requiring adaptive algorithmic thinking. The approach integrates multiple advanced concepts (graph theory, OOP, Markup Language) within a single coherent project. (ii) A custom-developed software tool with dual functionality: Pedagogical scaffolding, enabling students to test solutions step-by-step, visualise algorithmic decisions, and compare their strategies against optimal solutions computed by the tool, supporting self-directed debugging and iterative refinement. Automated assessment, allowing teachers to evaluate personalised projects by checking automatically each student's unique project and conditions and running comprehensive correctness checks automatically, reducing evaluation time. (iii) Mixed-methods action-research evidence from a VET implementation, combining qualitative data (teacher diary observations, semi-structured student interviews, etc.) with descriptive quantitative comparisons across subjects and academic years. Despite sample size limitations, converging evidence from multiple sources suggests that the approach can engage VET students, sustain participation through complex project development, and support learning of traditionally challenging concepts.

Therefore, considering mixed-methods action-research, mainly based on qualitative evidence, the following exploratory propositions are formulated:

https://doi.org/10.3390/computers15020082

- The implementation of a gamified PBL approach in introductory computer science courses in VET enhances student motivation, interest and can mitigate absenteeism.
- The design of a tool for testing the solutions, comparing results, and checking the advances carried out by the students, promotes student engagement and involvement, while facilitates the evaluation of projects with different and personalised conditions.

3. Material and Methods

This section describes the methodological design and implementation of the gamified Project-Based Learning intervention, pedagogical approach, and empirical evaluation.

3.1. Educational Context and Intervention Design

The method is focused on the teaching–learning process and practical work, based on a gamified Project-Based Learning approach. Inductive methods and qualitative research [56] are used, focussing on the teaching–learning process rather than on the results. During each session, the teacher continuously implements adaptations and improvements based on ongoing observations of the learning experience. This is done in combination with continuous planning, action, observation, and replanning of the contents imparted during the intervention. Regarding the approach, the teacher approach of the action-research was selected, as the research focuses on the teacher's experience in regard to understanding of the context, systematisation, analysis, and study of reality [57].

For the project, the students should design a game based on graph theory and searching algorithms to simulate two different roles playing by turns, with different conditions and constraints for each student.

This gamified project is designed and structured based on a competence-based teaching unit, and it focuses on learning by doing, following the PBL methodology. The project is divided into different steps or activities, from simple to complex concepts, in order to meet adequately all the requirements and to acquire the corresponding competences and learning results. These steps are similar to those described in previous works by the authors [3,13,14], where different active methodologies were applied.

Along with the project development, activities and exercises are proposed to reinforce the learning of the most complex concepts. As a final step, the game must be presented in class. This allows for the students to see other solutions, obtain new ideas, show the advantages and disadvantages of other alternatives, discuss in class the different solutions and options, and improve their own solution if there is enough time, or for the second-chance evaluation. Evaluation criteria are also given in advance.

In order to facilitate the evaluation by the teacher, and also to serve as a support and a comparison tool for the students, a software application has been designed and developed. This software allows each entry graph to be simulated with multiple conditions and constraints, providing different solutions and alternatives for each setup. Therefore, the educational software tool fulfils two main needs: the teacher can automatically and rapidly check the solutions provided by the students, and the students can test their partial or final solutions, to check if they obtain the desired and/or required output. This software is explained in detail in Section 4.

3.1.1. Course Curriculum and Prerequisite Knowledge

The intervention was implemented in the Programming subject, a first-year course within the Web Application Design (WAD) degree of VET. The Programming subject spans the entire academic year (September to June), with 8 h of weekly instruction divided into theory and practical sessions. This is one of the first subjects in which students learn Pro-

https://doi.org/10.3390/computers15020082

gramming. By the time of the gamified project implementation (the final four weeks of the course), the students have completed eight months of Programming instruction covering:

- Object-Oriented Programming fundamentals: class design, inheritance, polymorphism, encapsulation, interfaces (Comparable), communication between objects.
- Data structures: arrays, collections framework, iterators.
- Algorithms: sorting algorithms, search algorithms, searching for minima and maxima.
- Control structures: conditionals, loops, recursion.
- File I/O: reading from and writing to text files, CSV files, basic serialisation.
- Database basics.

The students had not studied graph theory yet, so this was new content introduced during the project; Flipped Classroom was used for teaching it. The students reviewed the theory at home, and they applied it in class by working on the project. The new concepts imparted were graph theory concepts (graphs, vertices, edges, weighted graphs, shortest path algorithms), application of Dijkstra's algorithm and Depth-First Search (DFS), and integration of these algorithms within a system.

The project addressed specific learning outcomes from the Programming subject curriculum, such as the design and implementation of algorithms for problem-solving (search and sorting algorithms), the application of Object-Oriented Programming principles (agent class hierarchy, polymorphic behaviours, management of structured and unstructured data formats, graph representation, XML configuration, CSV output, debugging and validation of complex programs), and communication of technical solutions effectively (project presentation).

The gamified project thus served as a summative assessment integrating the entire year's Programming curriculum while introducing graph theory as an advanced extension topic.

It is important to remark that the project was not introducing Programming from scratch; rather, it integrated previously learned OOP and data structures with new algorithmic concepts (graph theory) in a complex applied project.

3.1.2. The Gamified Project: "Cops and Thieves" Game Development

Against this curricular background, the gamified project required students to develop a complete functional game by creating a simulation of a pursuit-evasion scenario (thief trying to reach a destination, police attempting interception) on an undirected weighted graph representing an urban street network.

The technical specifications to follow were based on graph representation and algorithm development: implement undirected weighted graph data structure (adjacency list or matrix); implement Dijkstra's algorithm to compute shortest path from start to goal; implement Depth-First Search (DFS) to find all possible paths between two nodes; and use graph algorithms to determine thief routes and police strategies.

The main requirements for the game implementation were:

- Object-oriented design: separate classes for Graph, Node, Edge, Agent, Thief (extends Agent), Police (extends Agent), GameController, etc.
- Use of basic concepts of inheritance and polymorphism for agent behaviours.
- Markup language (XML) to store graph configuration, game state, and results.
- CSV output file recording the complete route taken, time per segment, obstacles encountered, final state.
- Graphical interface displaying graph, agent positions, and game progression (optional but encouraged).

https://doi.org/10.3390/computers15020082

The game mechanics were personalised for each student, although within a general framework in which the thief had to navigate from a starting node to a goal node while avoiding police and obstacles.

This general framework is as follows. The thief starts with an initial speed that decreases when encountering police officers or obstacles. If the speed reaches 0, the thief is captured. The police officers are initially positioned along Dijkstra's shortest path, and they must intercept the thief. When surveillance cameras detect the thief's position (at certain nodes), the police officers reposition strategically. The game should indicate who wins, considering all constraints.

The starting and goal nodes, obstacle placements, number of police officers, camera locations and initial speeds, were different for each student. This personalisation ensured that while all students implemented the same general game structure, their specific solutions could not be directly copied, requiring genuine understanding and adaptation.

The project design reflected several deliberate pedagogical choices grounded in constructivist learning theory and active learning principles such as the integration of multiple concepts, gamification through development, and scaffolded progression.

Each stage built upon previous work, with daily follow-up exercises ensuring progressive learning before advancing to more complex concepts. The custom software tool (detailed in Section 4) provided immediate feedback, enabling iterative refinement cycles central to learning through debugging and problem-solving. While the projects were individually evaluated, the students were encouraged to discuss approaches, debug together, and learn from peer presentations, supporting social constructivist principles regarding collaborative knowledge construction.

3.1.3. Gamification Elements and Game-Based Design

To clarify the gamification approach employed in this intervention, it is essential to explicitly identify the game elements integrated into the project and to distinguish this approach from alternative gamification strategies.

There exist three types of gamification, as stated in Section 2, which consist in the integration of game elements to traditional learning activities, serious games to learn by playing, and gamification through development. In this latter case, the students create the game themselves as the learning activity, and this is the type of gamification that was implemented in the present work. Students are not earning points for completing exercises or playing an educational game designed by others; rather, they are building a complete, functional game from scratch. The gamification lies in the inherently engaging nature of game development as a creative, purposeful, and personally meaningful activity.

The project employs the "cops and thieves" narrative as thematic framing, a universally recognisable pursuit-evasion scenario that provides intuitive context for abstract algorithmic concepts. This narrative serves multiple pedagogical functions, i.e., the pursuit scenario gives purpose to pathfinding algorithms, the students must implement logic for both roles, requiring them to think on multiple strategies, and the narrative transforms a dry algorithmic exercise into a scenario with stakes and outcomes that matter within the story.

Other specific game elements are the existence of characters or roles, a clear goal of victory for both roles (which have competing objectives), and the use of game rules with personalised constraints and obstacles that can modify the difficulty of the challenge.

Beyond these explicit elements, the project incorporates several additional characteristics of game-based learning, which are enhanced by the use of the custom- designed tool:

https://doi.org/10.3390/computers15020082

- Iterative experimentation: Like games, the project encourages trial and error experimentation without severe penalties. Students can test their implementations repeatedly using the software tool, learning from failures and refining strategies.
- Mastery through practice: The progression from simple requirements to complex integrated functionality mirrors game-level design, where players master basic mechanics before facing advanced challenges. Early milestones (implement basic Dijkstra) scaffold toward complex achievements (full police strategy with dynamic repositioning).
- Visible progress and achievement: Students can visualise their game working (or failing) through the tool's graphical interface, providing evidence of progress similar to game advancement.
- Personalised challenge: The unique conditions for each student create individualised difficulty curves, similar to adaptive difficulty in games. This personalisation prevents both boredom (too easy) and frustration (impossibly hard) by calibrating challenge to ensure all students work in their zone of proximal development.

Then, when the students develop the game, they can experience intrinsic motivation from creative authorship, purposeful application of knowledge, and playful problem-solving. In addition, the narrative context makes the abstract concrete. These psychological mechanisms are precisely what gamification seeks to activate [18].

This gamification approach integrates seamlessly with Project-Based Learning (PBL) methodology [36,38,40,41], i.e., the game development is the project. Students must define requirements, design architecture, implement functionality iteratively, test and debug systematically, and present and defend their solution.

3.2. Assessment and Evaluation Methods

The students were clear from the outset about the assessment criteria, to ensure that they were met before the project was submitted. In addition, they were encouraged to try different possibilities and alternatives, as well as to discuss them with their classmates in order to promote collaborative work and learn other ways of working. In order to adequately measure student learning and project outcomes, a distinction was made between formative assessment during the project development phase and summative assessment of final deliverables:

- Formative assessment: Daily follow-ups. Throughout the four-week project development period, the students participated in daily follow-up activities designed to ensure progressive mastery and identify difficulties early. These follow-ups served formative assessment purposes, detecting misconceptions early, providing feedback before the students invested significant time in incorrect approaches, and encouraging consistent daily engagement with project concepts. The students received immediate verbal feedback and, when widespread errors appeared, concepts were re-explained to the class. Passing 80% of the daily follow-ups was mandatory for a positive evaluation.
- Summative assessment: Project evaluation. The final project was evaluated based on multiple criteria organised in a detailed rubric (available to the students from the project outset). Some of the criteria evaluated were the technical correctness of the solution (Dijkstra implementation, DFS implementation, thief and police strategies, etc.), code quality (OOP design, readability, error handling), presentation and communication (clear explanation and response to questions), and innovation and extensions (creative solutions, graphical interface or features beyond minimum requirements).

The students used the tool to test their implementations iteratively, validate correctness, and debug errors. The tool provided immediate feedback on algorithmic correctness, enabling self-directed learning.

https://doi.org/10.3390/computers15020082

Projects were submitted via the educational platform as compressed files containing source code, executable jar file, sample configuration files, and CSV output. The teacher used the custom software tool to test the solutions and identify errors efficiently, allowing the teacher to focus manual evaluation on code quality, design decisions, and conceptual understanding demonstrated during presentations.

The students who did not pass the initial project evaluation (due to technical errors, incomplete functionality, or failing the follow-up threshold) were offered a second-chance evaluation opportunity two weeks later. Between these attempts, interviews were carried out where teacher and student reviewed the evaluation report together, identified specific errors, discussed conceptual misunderstandings, and outlined correction strategies. Then, the students corrected their errors, improved their implementations, and resubmitted. The same rubric and evaluation process were applied, but with emphasis on demonstrating learning from initial mistakes. This second-chance structure reflects the VET educational philosophy of emphasising mastery learning and providing multiple pathways to success rather than single-attempt assessment.

Assessment validity and reliability was ensured by several factors:

- Alignment with learning outcomes: The rubric criteria were directly mapped to the course learning outcomes, ensuring content validity.
- Transparency: The students received the complete rubric at the project start, enabling them to assess it before submission and to understand the expectations clearly.
- Multiple data sources: Evaluation combined automated technical testing, manual code review, and oral presentation assessment, triangulating evidence of understanding.
- Consistency: The automated tool ensured consistent technical evaluation across all the students, i.e., the same tests, run under each student's conditions, with objective comparison criteria.
- Inter-rater reliability concern: All the evaluations were conducted by a single teacher (the course instructor), eliminating inter-rater reliability issues but introducing potential for individual bias. To mitigate this, evaluation by the tool provided an objective image of technical errors, and the presentations were observed by multiple attendees (other students), providing implicit social accountability.

3.3. Data Collection Instruments and Analysis Procedures

The instruments and procedures employed to collect and analyse data for understanding the intervention's impact on student learning, engagement, and outcomes were the following:

- Java was used as the object-oriented language, using Eclipse and its tools as the development environment. The use of the Swing interface was permitted.
- To determine the students' initial knowledge, an initial questionnaire was done. This questionnaire was based on previous validated questionnaires, also validated by other teachers. A final questionnaire was also provided to evaluate the students' perception about the intervention.
- A diary was kept to record the relevant events of each class in terms of attendance, date, objectives, content and activities carried out. The observation was done as outlined in [58], with all the events organised and categorised.
- The project grade was considered one of the main indicators of the students' performance and knowledge acquisition.
- The rubrics allowed the students to check their projects before the submission, and to evaluate the project by the teacher.

https://doi.org/10.3390/computers15020082

- The educational platform was used for additional communication with the students, to submit the project and the additional exercises, and to receive feedback about their results and the daily follow-ups.
- Semi-structured and unstructured interviews were conducted after the initial evaluation so that the students could explain the main problems they had and start a discussion with the rest of the class. These interviews were of great help for the students who failed in the first evaluation attempt, to face the second chance evaluation with more warrantees and to solve the problems detected in their projects.

3.3.1. Qualitative Analysis: Data Collection and Procedures

Given the action-research nature of this study and the emphasis on understanding student experiences and learning processes in depth, qualitative data collection and analysis procedures were implemented to ensure the credibility and trustworthiness of the findings [59–61].

Throughout the intervention, a structured teacher diary was maintained, following established protocols for systematic classroom observation [58]. Each class session (8 h per week over 4 weeks of project development) was documented using a standardised template capturing the following: date, attendance, and session objectives; content covered and activities performed; observable student behaviours related to engagement (e.g., voluntary participation in discussions, time working on tasks, questions asked); technical difficulties encountered and how students addressed them; emotional responses (e.g., expressions of frustration, satisfaction, confusion); notable incidents or breakthroughs in understanding; and teacher reflections and preliminary interpretations.

Observations were recorded descriptively with minimal initial interpretation, separating observation from inference. To enhance reliability, key observations were corroborated through multiple sources where possible. For instance, claims about peer collaboration were verified by confirming with students during informal conversations.

Post-evaluation interviews were conducted following a semi-structured protocol with core questions about the main difficulties encountered (technical concepts, time management, understanding requirements), about collaboration with class-mates, about concepts acquired (graph theory, OOP, problem-solving strategies), and about their opinion about the methodology followed. Informal conversations were also carried out about how and when the software tool helped them and influenced their approaches. For students who failed the initial evaluation, additional questions explored specific errors and their understanding. Interviews were conducted by the course instructor in a conversational tone to reduce power dynamics.

The code, documentation, and output files submitted by the students served as artefacts for analysis. These were examined to analyse the correctness of algorithmic implementations (Dijkstra, DFS), code organisation and use of OOP principles, the handling of personalised conditions, and completeness relative to requirements.

Daily follow-up responses submitted via the educational platform provided additional artefacts documenting progressive understanding of concepts throughout the project period.

Qualitative data from diary entries, follow-ups, interview notes, and artefact observations were analysed using a hybrid deductive–inductive coding approach [59,62,63]:

1. Based on research questions and theoretical frameworks (constructivism, engagement theory, Computational Thinking), an initial coding scheme was developed with broad categories: engagement behaviours, cognitive processes, tool usage, affective responses, and learning outcomes.

https://doi.org/10.3390/computers15020082

2. Through iterative reading of diary entries, follow-ups and interview notes, subcategories emerged from the data.
3. All diary entries and interview notes were coded systematically using the refined scheme. Each observation or statement was assigned one or more codes. When observations fit multiple categories, all relevant codes were applied.
4. Coded data were organised to identify patterns and relationships.

To ensure credibility and reduce bias, the data were triangulated across multiple sources [60–64] by combining observations, interviews, and artefact analysis; comparing teacher observations with student self-reports and actual project performance; and examining patterns across multiple class sessions rather than isolated incidents. Dependability was ensured through systematic documentation following established protocols [58], with all diary entries dated and organised chronologically. Confirmability was pursued by maintaining awareness of potential confirmation bias and actively seeking disconfirming evidence.

3.3.2. Quantitative Analysis: Limitations and Statistical Interpretation

Regarding the quantitative analysis, the statistical independence of the data and the distribution of the data were considered, with incidental sampling. The data distribution was first analysed to determine whether the samples followed a normal distribution, for the use of parametric or non-parametric tests [65,66]. The Shapiro test was used to determine the normality of the distribution. In the case of non-normality, the Mann–Whitney U test or the Kolmogorov–Smirnov test could be used. In the first case, normality was not assumed for the samples, and in the second case, it was determined if the samples came from similar distributions. Additionally, the Kruskal–Wallis test was used to compare the scores between the different groups, without assuming normality in the distribution of the samples.

Given the common characteristics of VET contexts outlined in Section 2 and the specific group studied (see Section 5.1), several limitations must be explicitly acknowledged regarding the quantitative analysis presented in this study. The small sample size ($N = 10$ students who completed the project in 2025, with similar numbers in previous cohorts) significantly limited the statistical power of quantitative tests and the generalisability of the findings.

Normality testing with samples smaller than 20 observations is inherently unreliable, as the Shapiro–Wilk and similar tests lack sufficient power to detect departures from normality in small samples [65,66]. Similarly, while non-parametric tests such as Mann–Whitney U and Kruskal–Wallis are appropriate when normality assumptions are violated, their statistical power is substantially reduced with small sample sizes, increasing the probability of failing to detect true differences when they exist.

Moreover, the interpretation of p-values in the context of small samples requires particular caution. A non-significant result ($p > 0.05$) with $N = 10$ does not constitute evidence of equivalence between groups; rather, it indicates insufficient power to detect differences if they exist. Therefore, the absence of statistically significant differences between subject grades, or across different academic years, should not be interpreted as definitive proof that the groups perform identically, but rather as descriptive observations of similar patterns within the limitations of the available data.

For these reasons, the statistical tests in this study are presented primarily as descriptive indicators of trends and patterns rather than as definitive inferential conclusions. To complement p-values and address their limitations with small samples, effect sizes (Cohen's d) are also reported where comparisons between groups are made. Effect sizes provide information about the magnitude of differences independent of sample size, offering a more complete picture of practical significance [65].

Importantly, the qualitative observations derived from the daily classroom interactions, systematic teacher diary entries, student interviews, and rubric-based project evaluations

https://doi.org/10.3390/computers15020082

constituted the primary evidence for the claims about student engagement, motivation, and learning processes. These qualitative data provided rich, contextualized insights into how the students interacted with the gamified PBL approach and the software tool, capturing nuances that quantitative metrics alone cannot reveal. The quantitative data served to complement and contextualize these qualitative findings, providing additional perspective rather than serving as the sole basis for our conclusions.

This mixed-methods approach, with the emphasis on qualitative depth over quantitative breadth, aligned with the action-research methodology employed, and it is appropriate for exploratory studies in specific educational contexts where large sample sizes are impractical or impossible to obtain [57,59,60,64].

4. Custom-Developed Software Tool Description and Functioning

This section provides a detailed technical description of the custom software tool developed. This tool's description serves several essential purposes:

- The tool is not merely an incidental implementation detail but a core component of the intervention. It functions both as a pedagogical scaffold (enabling students to test solutions iteratively) and as an assessment instrument (enabling automated evaluation under personalised conditions). Understanding the tool design and capabilities is necessary to evaluate the validity of research findings and the replicability of the approach.
- The tool represents a methodological contribution that other VET educators could adapt for their own gamified Programming projects. Sufficient technical detail enables replication.
- Personalised project conditions (unique for each student) create a significant assessment burden without automation. The tool's design demonstrates how automated testing can scale to complex, open-ended projects with multiple valid solutions under varying constraints, a challenge not well-addressed in the existing Programming education literature. This technical solution is itself a research contribution.

Regarding the use of the tool in the teaching–learning process, in general terms, it is incorporated as follows:

- The teacher creates different configurations for each student. Reviewing and checking the suitability of the different solutions according to different configurations depends largely on the teacher's expertise and experience. The application allows the teacher to know the exact optimal solution for each particular situation.
- When the student wants to verify if something is working well, the teacher can give advice or corrections. In addition, the feedback provided by the tool is direct and fast, ensuring that the student knows what is happening without knowing the direct answer to the solution (the teacher knows the solution in advance).
- Students do not have to wait for the teacher to be available, nor does the teacher have to review the code of each student. The students can continue searching for the solution, avoiding frustration from not knowing what is happening or from waiting for the teacher's advice.

4.1. Underlying Theory: Graphs

The application has been developed for its use with first-year VET students. The idea behind the game is to use different graph theory algorithms to determine the possible paths to follow and choose the most appropriate one based on the options and restrictions set for each student.

https://doi.org/10.3390/computers15020082

At the start of the project, the students did not have knowledge of graph theory, so it was necessary to give them an introduction and the necessary materials. An initial session was given to them to introduce the basic concepts.

For the application developed, undirected graphs weighted with positive numbers were used, in which the vertices of an edge were adjacent to each other, and self-edges, i.e., edges of the type (v, v), were not considered. It was also assumed that all the nodes had some connection, i.e., there were no isolated nodes. All documentation and theory were given to be reviewed at home, following the Flipped Classroom approach.

There are several techniques for solving optimisation problems. The requirement for the students was to solve these optimisations with lists, hash maps, and arrays, which were mainly studied in class.

To find the shortest path, the students were instructed to use the Dijkstra method, a widely known algorithmic method of greedy Programming. The use of a stack in this method would have simplified the algorithm, but there was insufficient time to explore its use in class. As a result, they resorted to recursive logic without using this structure.

Algorithms such as Depth First Search (DFS) or Breadth First Search (BFS) can be used to traverse graphs. DFS was chosen because of its lower memory storage requirements, and it was used to search for all possible routes from the source node to the destination.

4.2. Tool Description

Although the application is easily adaptable to different requirements, it was designed based on the concepts to be learnt by students and the requirements that would be asked in the Programming subject.

Thus, in a general setup, the students start with an unweighted graph, with nodes and distances between them, and a narrative based on the "cops and thieves" game. Their basic requirements and functioning are described next.

Before starting the thief's strategy, the police officers have to be placed on the shortest route from the starting point to the final point, according to Dijkstra's algorithm. The thief starts at a certain speed from the starting point, which must be reduced each time he/she encounters a police officer or an obstacle. When the speed reaches 0, the thief has been caught and the game ends.

The speed at which the thief travels depends on whether he/she is driving, running or walking (the speed for each means of transport can be personalised for each student). If there is no route other than that of a police officer, the thief must pass where the police officer is, which is reflected in the decrease in speed (proportional to the means used). If the thief is travelling and encounters an obstacle, the thief must modify the transport and the speed accordingly or find an alternative route.

Therefore, the thief must assess which routes to take depending on road conditions, whether there are police officers present and/or whether speed has been reduced too much. To ensure that each solution is different and, therefore, innovative in its design, each student should start from different conditions, which are set out as follows:

- Obstacles (points) that cannot be passed.
- Each student has a different route and starting point with different conditions (starting speed; speed at which they are travelling; obstacles that reduce the speed; etc.).
- Some of the points through which the thief has passed are known to police officers.
- Time taken to reach certain points (depending on different conditions for each student).

Then, the results to be obtained can be summarised as:

- Determine the shortest path between the origin and destination according to Dijkstra.

https://doi.org/10.3390/computers15020082

- Show the thief's entire route (all the points she/he has passed through), complying with the arrival time at the final point. The coordinates of the route are saved in a CSV file.
- Implement the police strategy: the route that police officers must take to get ahead of and surround the thief.

For a technical description of the tool, the reader is referred to Appendix A.

4.2.1. Execution Modes

There are two alternatives to obtain the solutions with the application: step-by-step or in a single step from the beginning to the end.

The step-by step execution is equivalent to running the application in stages, and its main objective is to guide the student towards the solution. At this level, different nodes can be specified as the starting or ending point, in order to analyse what happens along the way. The initial configuration is established and the possible routes are obtained: the Dijkstra path; the route analysis; and the recommended route. The Dijkstra route appears highlighted in the graph and is where the police officers will initially be positioned. From here, according to the recommended route, the next node to which the thief should move is chosen. In a second stage, this node is defined as the new origin, and the outputs are also obtained, adapted to this new route, and so on, at each step. This level of execution is reserved solely for analysing possible routes, obstacles, the Dijkstra route and the initial position of the police officers. It does not include the position of cameras to relocate police officers or the thief's speed rate.

Through the step-by-step execution level, the students can understand which part of their code is failing. In this way, two important objectives can be achieved: (i) The student receives quick, visual feedback, as they see the execution through an example and step-by-step. (ii) The teacher can quickly understand and explain what is happening, helping the student to locate the part of the code that needs to be fixed. A detailed view of the interface is shown in Figure A5 in Appendix A.

If the student requests a more complete explanation, including the overall police strategy, the second level of execution must be performed, in which the game is played through once. Then the one-step execution (from beginning to end) directly provides the final results, revealing the winner and the path chosen by the players (police and thief). In addition to what is obtained in the step-by-step execution, it shows how the police strategy ends and who wins the game. Both the thief and the police develop different movement strategies: (i) The thief chooses the route with the shortest weighted length due to obstacles, but considering that he/she will always try to avoid the Dijkstra route. (ii) In the case of the police station, they will have information at all times about which nodes the police officers are initially located in and will readjust their positions only if the thief passes through a node with a camera. At that point, the police officers will be relocated, using the following protocol: The police officer closest to the node is sent to the thief's location; the location left by the previous police officer will be occupied by the one closest to that position, and so on. If there are not enough police officers, an attempt will be made to place them near the thief.

Therefore, the pedagogical design of these interface features reflects deliberate choices to support learning processes. The step-by-step execution mode works under the constructivist principle of scaffolded exploration, enabling students to build understanding incrementally rather than confronting complex complete solutions immediately. The Dijkstra route visualization provides an external representation of shortest-path computation, which can facilitate its understanding by students. The comparison between student strategies and optimal solutions creates cognitive dissonance, i.e., when outputs differ, students must reconcile discrepancies by examining their logic, promoting deeper engagement with

https://doi.org/10.3390/computers15020082

algorithmic principles than passive instruction alone would achieve. In Section 6.3 the impacts on student usage patterns and learning outcomes are discussed more deeply.

4.2.2. Evaluation of Students' Solutions

Students can check their solutions by comparing their results with those obtained within the application. To do this, they upload the files where they saved their solutions, specifying the final route of the thief according to their conditions.

To evaluate the solutions, a complete and/or step-by-step simulation is carried out with the conditions imposed on each student, in order to compare with the results of the application. The following are assessed:

- Whether the obstacles and starting conditions were taken into account correctly.
- The speed the thief ends up with, taking into account the sightings. A maximum number of speed decreases to win or lose is determined, which must be configurable.
- The distance travelled, time, and number of speed decreases. To do this, the speed index is calculated based on the distance travelled (speedIndex = number of decreases/total distance travelled).
- Additional considerations regarding the route, to determine the winner of the game.

In a CSV file, the system outputs are recorded on each line. For example, the first line stores the final route of the pursued, and the next line stores the times per section. Subsequently, the lines below store the total time of the route; the speed with which it ends; whether it wins or loses; the routes chosen, speed and mode of transport for each section, etc.

Finally, the grade for the project is given based on a rubric using the feedback obtained by the application and the teacher observations. Given the personalised nature of the challenge and the multiple concepts addressed, the project evaluation can be passed considering several combinations of developments and achievements. Table 1 shows a summary of the combinations of achievements that can lead to passing or failing. The final grade will also depend on the rubric and, in specific cases, on the assessment of the student's code.

Table 1. Possible combinations of achievements that can lead to passing or failing.

Combination	Correct Route or Equivalent	Final Speed over a Given Threshold	Final Speed Higher Than Tool	Solution Correct
1	✓	✓	✓	✓
2	✓	✓	×	✓
3	✓	×	×	×
4	×	✓	✓	✓ †
5	×	✓	×	×
6	×	×	✓	×
7	✓	×	✓	×

† In Option 4, the possibility that an obstacle has not been considered for justified unforeseen reasons is taken into account, in which case the solution is approved.

5. Results

The project and software tool were used in the subject Programming (PR) of the first year of a Web Application Design (WAD) course. In this section, the group and the results obtained with the project development are described.

https://doi.org/10.3390/computers15020082

5.1. Group

Twenty-four adult students began the course, with only one female student, who dropped out at the beginning of the course. The cultural level was diverse: some were university students from the arts field, others came from other vocational training courses or secondary school, and some were alternating work with study. The atmosphere in the classroom was good in terms of discipline, although there was a clear lack of study. Because some students worked, there was a significant initial dropout rate, usual in subjects such as Programming, as they could not devote enough time to the studies. Others remained because of parental pressure, without having a real vocation for the course. The dropout rate was high at the beginning of the course, with 13 students remaining, of whom there was still a possibility of dropout in the last stage of the course, with 10 students finally working on the project. They found Programming logic complex, having difficulties in understanding loop conditions, recursion and file management, as well as basic object-oriented concepts, which they should have already learned. The pass rate per term was very low, and they made excessive use of code-generating programmes to replace what they should be doing and practising themselves. None of them had previous experience with coding before starting the course.

Regarding research ethics and participant awareness, all the students enrolled in the Programming subject participated in the pedagogical intervention as part of their regular coursework. The gamified project and tool usage were curricular components required for course completion, not optional research activities. The students were informed at the beginning of the course that their experiences might contribute to educational research and that anonymous data (grades, anonymized project artefacts, observation notes) could be used for research purposes. They were informed that they could decide to withdraw their data at any moment. All the students provided consent for research participation. Individual interviews were voluntary and all data were anonymized. Institutional ethics protocols for educational action-research were followed, which permit research on regular curricular activities with student awareness and anonymization.

5.2. Qualitative Analysis Results

Monitoring of class structure and progress showed that several students faced recurring difficulties in understanding and adapting Dijkstra's algorithm, as well as in applying OOP concepts. Some findings throughout the students' work in the project were:

- Although they made progress in implementing the Dijkstra algorithm and identifying routes, some problems persisted in understanding the algorithm and the logic behind its implementation.
- Understanding other concepts, such as the use of HashMap, the management of obstacles and sightings in the game, and the correct database structure were areas that needed improvement and reinforcement.
- Often the implemented methods did not work as expected, and error handling was not adequately addressed.
- Monitoring in class, with constant feedback, was necessary to help the students' progress, but assessment methods had to be reinforced to ensure that all the students understood the algorithms and their practical application.
- The schedule was established to maintain a constant review of progress and to make a final presentation of the project in class.

The follow-ups, the teacher's diary and the final questions conducted with the students were then analysed.

https://doi.org/10.3390/computers15020082

5.2.1. Follow-Ups Structure

Twenty-nine two-hour sessions were allocated to the project, of which 26 were assessable follow-ups for each student and three were designed for assignments (one mid-term, one final) and for the presentation (one). The follow-ups were structured as described in Table 2. The sessions were devoted almost exclusively to the specified problem/concept, although any issues pending resolution from previous follow-ups were addressed again as needed.

Table 2. Number of sessions and description of the follow-ups for the Programming project in 2025 (N = 10 students).

Concept	Sessions	Description
Class structure	2	They had to form classes to model the problem. The class structure included modelling both the attributes and methods specific to the classes, as well as the relationships between them.
Dijkstra	5	The algorithm had already been tested in sessions prior to the development of the project, and they only had to adapt it to the new game context. Even so, they found it difficult to adapt it.
Thief strategy	6	They should have included DFS implementation, obstacles, sightings, selection of the best route and exporting results to CSV.
Mid-term submission	1	The initial solutions were tested with the tool.
Database import	5	They had to create a database, take the data from it and fill in the classes, in order to continue working in memory. Those who had not finished exporting to CSV continued with that task.
Bidimensional arrays use	2	They should have computed statistics.
XML import	6	Using the knowledge from the Markup Language subject, they modelled the data to fill in the main class structures of the game.
Final submission	1	Algorithm refinement for the submission.
Presentation	1	Oral defence of the project.

To examine whether the project-based approach supported concept assimilation, the students' performance on theoretical examinations (conducted earlier in the course) was compared with their performance on daily follow-up assessments during the project phase. Both assessments addressed the same conceptual domains, allowing for observation of whether practical application within the project context was associated with improved performance compared to examination performance on theoretical knowledge of the same topics. For simplicity, average grades and general thematic concepts are described, considering that the grades range from 0 (minimum) to 10 (maximum), with 5 required to pass.

The average exam grade for file handling (CSV, XML) was 3, while the follow-ups yielded an average of 7, suggesting significant improvement in knowledge acquisition. Feedback and observations collected in the diary showed that work with CSV files proved more accessible. Although the results with XML were also good, more observations and comments were documented, including problems with data parsing, as well as multiple instances of feedback regarding menus, data input, and output formatting.

Regarding database usage, the exam grade was 2 while the follow-ups averaged 9, suggesting substantial improvement. The most frequent comments and feedback regarding these concepts concerned time management issues during implementation.

https://doi.org/10.3390/computers15020082

Finally, concerning algorithms, only recursion, sorting, and minimum/maximum searching algorithms were considered, since graph theory algorithms were new content introduced specifically for the project. The average exam grade was 2, while the project follow-ups averaged 6, suggesting improvement, though more moderate than for previous concepts. Despite this improvement, the evidence collected showed that the students encountered difficulties when modifying algorithms to add complexity, such as incorporating weights, obstacles, or recomputing distances dynamically.

Therefore, working on the project allowed the students to put into practice concepts taught throughout the course, which, given their exam marks, had not been fully understood initially. However, applying these concepts practically enabled them to improve their understanding substantially. Moreover, most of the exercises and project components were adequately completed in the majority of cases. Thus, the qualitative evidence gathered during the follow-ups suggests that the students assimilated concepts better through the practical work of the project. It should be noted, however, that these concepts had been previously covered during the course, so working with them was not entirely new content but rather reinforcement and application in an integrated context.

5.2.2. Class Diary Analysis

The class diary was created and maintained by the course instructor throughout the intervention, following the structured observation protocols detailed in Section 3.3.1. Each session was documented using a standardised template capturing attendance, objectives, activities, observable student behaviours, difficulties encountered, and teacher reflections.

From the content collected and described in the class diary, a word count was generated. After excluding words without significant meaning (of, in, the, etc.), the count showed those words that had the most relation to the context and the importance/frequency of each word.

These words were related to theoretical concepts, project elements and learning aspects, and they appeared in different contexts (positive or negative). For example, for Dijkstra's algorithm, some of the students correctly implemented it, so this was a positive context included in the diary. Other students did not understand it, so it was recorded as a negative context. This process was followed for all the relevant words. According to the word count, it suggests a priori that the word *Feedback* had an important weight during development, followed by *Dijkstra*, among others.

Additionally, a grouping was made according to the semantics of the words and the known context, as shown in Table 3. The Frequency column indicates the number of times a word appeared.

From Table 3, the possible categories in which the described actions could be classified are summarised by colours in Table 4. The Frequency column is the sum of the occurrences of the words classified under that colour.

Taking into account the frequency of the possible categories, some conclusions and findings about importance and time consumption issues can be drawn. Algorithms played the main role and consumed a large part of the development. Daily explanations also played an important role, as well as the development of strategies for both the thief and the police. The least time was spent on the menu to organise data input and class design. An intermediate position had import from different sources. Additionally, occurrences of pending tasks exceeded those of completed ones.

https://doi.org/10.3390/computers15020082

Table 3. Semantic grouping of words from the analysis of the class diary.

Word	Frequency	Word	Frequency	Word	Frequency
algo	5	done	9	one	5
algorithm	14	injuries	10	speed	5
edge	7	thief	10	seen	6
still	21	menu	5	XML	8
sighting	5	method	5		
sightings	11	minimum	5		
database	14	not even something	5		
well	9	node	15		
calculating	6	nodes	9		
class	5	obstacles	17		
CSV	8	pending	5		
data	11	but	10	Total	471
must/should	17	police	6	The frequency count includes only the categorised words shown coloured [†]	
DFS	7	police officers	11		
Dijkstra	21	recommendation	6		
distance	8	relationship	5		
spy	5	fill in	8		
strategy	7	feedback	51		
structures	10	review	13		
explains	5	route	8		
lack	7	routes	15		
works	6	tracking	14		
hashmap	6	time	12		
there is/are	5	transport	5		

[†] Words appearing without colour highlighting (e.g., 'one', 'algo') were present in the diary but did not fit the semantic categories defined in Table 4 and were therefore not included in the categorised frequency totals. Note: The original diary and classroom interactions were conducted in Spanish. The words presented in Table 3 have been translated to English for this manuscript. The semantic grouping and frequency analysis were performed on the original Spanish text to preserve linguistic accuracy, with the categories subsequently being translated.

Table 4. Possible categories identified from semantic grouping (see colour patterns in Table 3).

Colour	Meaning	Frequency
	Feedback and advice using the application	89
	Algorithms (Dijkstra, DFS)	121
	Things to be done	59
	Something finished and working	35
	Strategies	87
	System inputs	13
	Database: import from different sources	46
	Memory structures	21
Only categorised words are counted. Total: 471		

5.2.3. Code Group Creation

Based on previous information, the code group creation was made using Atlas.ti. Code groups were created taking into account the analysis of the word list; then, relationships between them were detected. It can be seen in Figure 1 that *Dijkstra Algorithm* and *Thief Strategy* contained more events than the other categories. The first number indicates the number of occurrences of that code. The second refers to the links between that code and others.

The process followed for each category is explained on the basis of the *Dijkstra Algorithm* group. The codes identified in this group are shown in Figure 2. Again, the first number indicates times classified, and the second gives the relationship with the other groups.

Upon analysis of the network of difficulties in this group, around 30 were counted. Among the most difficult concepts were the computation of the minimum, temporal and final distances in the routes. The correlation of each of these aspects is shown through the *Dijkstra Algorithms* network of Figure 3.

Figure 3 visualises the relationships between coded observations within the Dijkstra Algorithm group, illustrating how specific technical difficulties (e.g., computing minimum distances, handling temporal values) contributed to broader patterns of algorithmic chal-

https://doi.org/10.3390/computers15020082

lenges. This network analysis supported the qualitative finding that Dijkstra's algorithm represented the most time-consuming challenge.

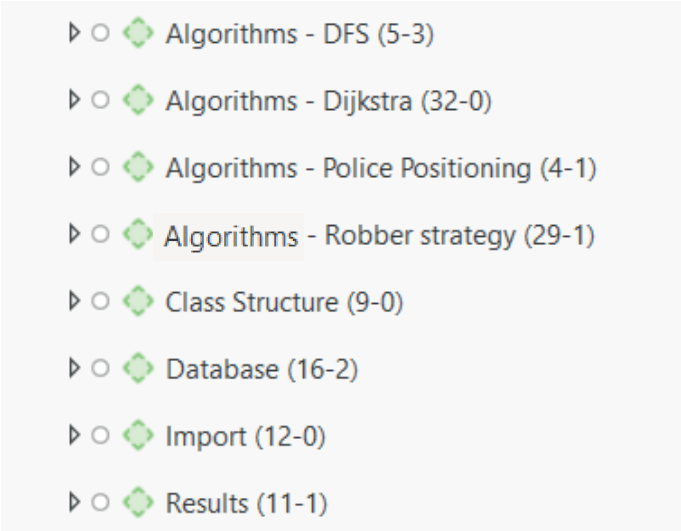


Figure 1. Qualitative coding structure: Main code groups created from word frequency analysis. Numbers indicate (occurrences–relationships with other codes).



Figure 2. Qualitative coding structure: Detailed codes within *Dijkstra Algorithm* group. Numbers indicate (occurrences–relationships with other codes).

Similar analyses could be carried out within the other groups. In the thief's strategies, there were more successes than difficulties, the most important difficulties being the handling of sightings and obstacles. In terms of positive aspects, the calculation of Dijkstra nodes, distance calculation, speed update, and means of transport were generally positively resolved. With regard to the database, the students had difficulties in the conception and design of the Entity-Relationship Model (ERM). The creation of the physical structure of tables and their modification, as well as the import from the database, were performed correctly. The same was true for XML, where there were no significant problems.

Therefore, based on daily monitoring and word clouds, an initial semantic grouping was made, from which possible categories or groups of codes emerged. By creating these groups and associating the codes, it was possible to design links between codes that

reflected the semantics described above and obtained networks representing these links, in a similar way as shown in Figure 3 for the *Dijkstra Algorithm* group.

The comparison between the Dijkstra and DFS algorithm groups showed that the Dijkstra group had a higher number of occurrences than the DFS group. Note that Dijkstra was the first graph theory algorithm they learned, and they needed more time to understand it. Therefore, with the experience gained with Dijkstra, DFS could be understood and implemented with less difficulty. In terms of strategies, the thief strategy took longer than the police strategy. In that sense, the most important delays in implementation were detected mainly in the thief strategy and in the Dijkstra algorithm.

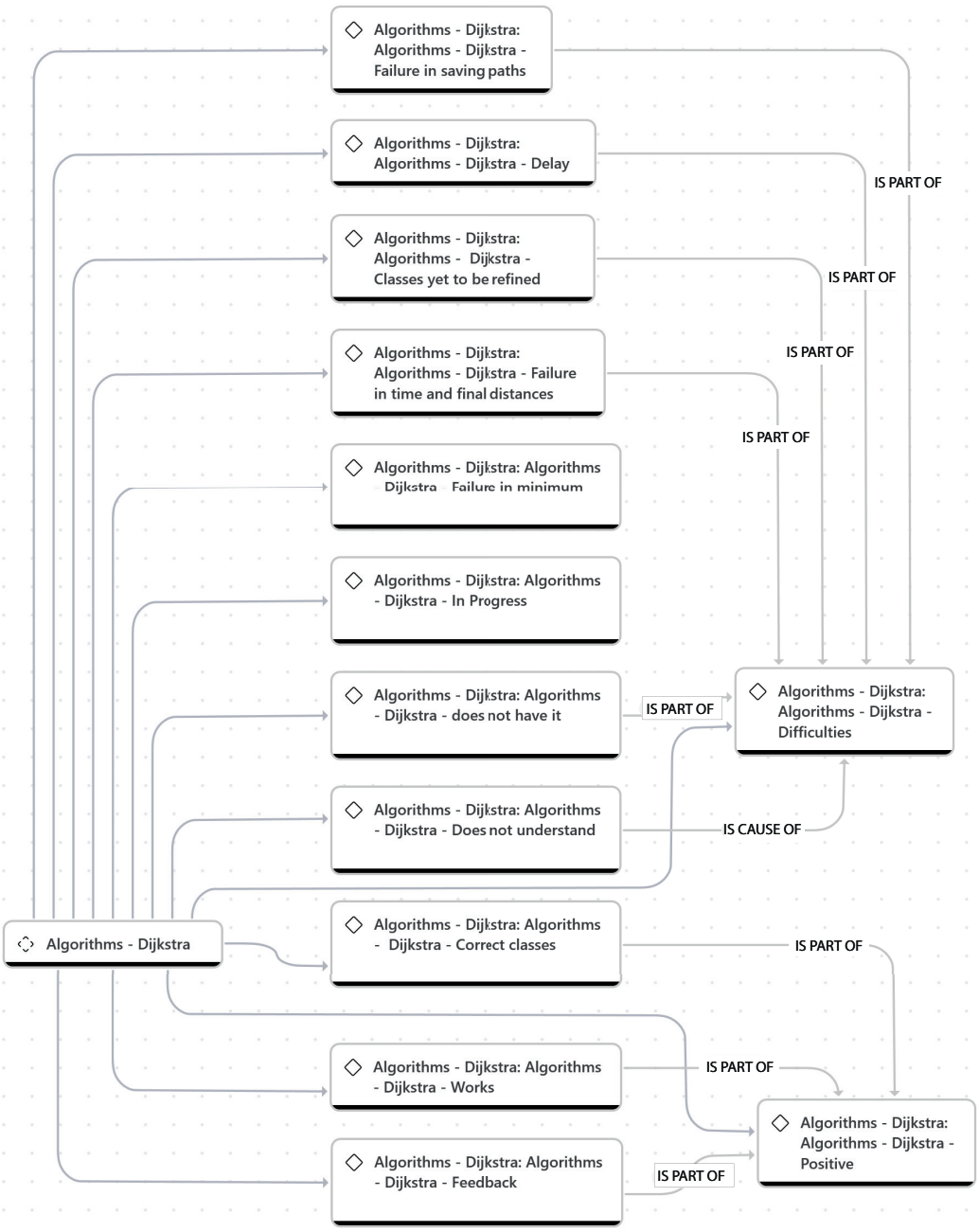


Figure 3. Network diagram showing relationships between codes within the *Dijkstra Algorithm* group.

The fact that more occurrences were recorded in the thief's strategy than in the police's strategy may be due to the fact that the police's strategy depended, to a certain extent, on the former. In general, once the Dijkstra route had been calculated, they positioned the police correctly on the route, although they had difficulties in calculating the routes

https://doi.org/10.3390/computers15020082

themselves. There were no major complications with the import of data from a database or XML. The same happened for exporting the results to CSV. They showed difficulties in the design and conception of tables, ERM and data structures. In each process, rapid feedback through the application played an important role in understanding the purpose of the game and, therefore, the final goal to be achieved. The process followed made it possible to compare the initial conclusions obtained from the word list with the coding of the document, obtaining the conclusions summarised in Table 5.

Table 5. Summary of conclusions obtained from the qualitative analysis of class diary entries (N = 10 students, 29 sessions).

	Word Cloud/Word List	**Document Coding. Highlighted Difficulties**	**Document Coding. Successes**
Most time-consuming challenges	Algorithms	It was determined that they had more difficulties with Dijkstra.	DFS took less time.
	Feedback	Very intensive process to record each feedback.	Important during the process.
	Strategies	The thief's strategy was predominant and very costly for them.	They understood where to get to.
Medium consumption challenges	Pending tasks	Calculation of the minimum, temporal and final distances in the Dijkstra routes. Thief's strategy when there are sightings and obstacles.	Calculation of speed decreases and transport update.
	DB, XML, import	Design conception and ERM.	Table modifications. Import from DB/XML.
	Completed aspects	Calculation of the best routes and the strategy to move the police.	Export data to CSV.
Least time-consuming challenges	Creation of structures (classes and methods) and menu.	Modifications to classes that had not been conceived in the initial design were recorded.	Menu design took little time.

5.2.4. Questionnaires

Once the intervention was finalised, an assessment was carried out using a questionnaire with 13 questions: 8 rated on a 5-point Likert scale and 5 open-ended questions.

The questions with numerical ratings and their corresponding mean scores are shown in Table 6.

Table 6. Questions rated numerically and their mean score (5-point Likert scale).

Question	**Item**	**Mean (M)**
Q1	Assessment of your initial knowledge of the subject	1.50
Q2	Do you feel that you have learned?	3.67
Q3	Was the content appropriate?	3.83
Q4	Are the tasks to be completed explained clearly?	3.50
Q5	Have you been informed of the results of the tasks on time?	4.14
Q6	Has the feedback helped you to improve?	3.33
Q7	Was the teacher's willingness to explain doubts adequate?	4.17
Q8	Assess the classroom atmosphere	4.17

The students provided an overall positive assessment. From the results obtained, several conclusions can be drawn about their perceptions:

- Learning value: Having started with minimal prior knowledge (Q1), the students felt they learned significantly (Q2), with content perceived as appropriate (Q3).
- Teacher responsiveness: The teacher's willingness to return results promptly (Q5) and explain doubts thoroughly (Q7) was rated highly favourably.
- Feedback effectiveness: Regarding task explanations and feedback utility (Q4, Q6), although the students rated these aspects positively, the responses suggest that these areas could be reinforced in future implementations by gathering satisfaction ratings at multiple time points throughout the project. The students acknowledged that the explanations were generally good, yet some expressed that they did not fully understand certain concepts. This perception may reflect the inherent complexity of the topics covered rather than inadequate explanation quality.
- Classroom atmosphere (Q8): The students perceived that activities took place in a positive and supportive learning environment.

The open-ended questions addressed broader aspects of the learning experience: Q9—Do you think you would have had time to learn additional content that was not taught, such as graphical interfaces?; Q10—Of all the work you have done, which has been most useful for your learning?; Q11—What was done well? Please explain; Q12—What would you improve about the subject?; Q13—Any additional comments and/or self-criticism?

When asked what was done well during the course (Q11) and which work was most useful (Q10), 83% of the students rated the final project as the most valuable learning activity. Additional responses emphasised positive aspects such as teaching OOP from the beginning, the practical nature of classes, and the reinforcement of fundamental concepts through repetition. Some of these comments remarking positive aspects were "Start Object-Oriented Programming from the beginning", "Progressive advancement of knowledge", or "Repeat the fundamental content many times to achieve a good solid foundation". However, there were also some more critical comments as for example "I think that in this module, a lot has been 'delegated' to the students' individual work, leading them to make an effort and acquire (if the student made an effort) a lot of knowledge. By this, I do not mean that the teacher has neglected her duty."

To identify areas for improvement, the responses to (Q9) and (Q12) were analysed. Regarding additional content (Q9), all the students indicated they did not believe they had time to learn more content beyond what was taught. Responses regarding areas for improvement (Q12) varied, with the students emphasising concerns about the course length and the sequencing of topics. Examples of the answers provided were "I would add more hours (to the subject)", "I know there isn't enough time, but try to include some graphics, even if it's just for a week, to see what it's about and how it would be done", "The organisation of the topics. Start with loops, conditions, etc."

Question (Q13) assessed the students' self-perceptions regarding their own performance and engagement. The responses revealed fluctuations in motivation at certain points during the course, with some students noting that the need to balance work and study responsibilities affected their consistency. Examples of some of these answers were "The first term, although it was difficult to get into the world of Programming, was relatively easy to pass, which led me to become overconfident. This meant that I had to catch up on my studies, which took its price", "Spending more time at home and avoiding disconnecting at certain points in the subject', or "A considerable lack of study and moments when I almost gave up on the course".

To examine the relationship between student perceptions (questionnaire responses) and academic performance (project grades), Spearman's rank correlation coefficient was

https://doi.org/10.3390/computers15020082

calculated. The resulting coefficient was $\rho = 0.42$, indicating a moderate positive correlation between the students' self-reported learning perceptions and their actual project grades. This could suggest that students who reported more favourable learning experiences tended to achieve higher project grades. However, it is important to note that correlation does not imply causation. Multiple factors could explain this relationship, including that students who performed better may have perceived their learning more positively, or that genuinely stronger learning experiences contributed to better performance. Further analysis would be necessary to establish causal relationships beyond this correlational evidence.

5.3. Quantitative Analysis Results

The statistical analysis presented in this section must be interpreted with awareness of the small sample sizes involved ($N = 10$ in the 2025 cohort). As detailed in Section 3.3, these sample sizes limited statistical power, making non-significant results difficult to interpret definitively. Therefore, the statistical tests serve primarily to provide descriptive context for the qualitative findings rather than as standalone evidence for claims about effectiveness. Effect sizes (Cohen's *d*) are reported alongside *p*-values to indicate the magnitude of the observed differences, independent of sample size.

The distribution of the students based on the grade obtained in the project is shown in Figure 4, and it is detailed in Table 7.

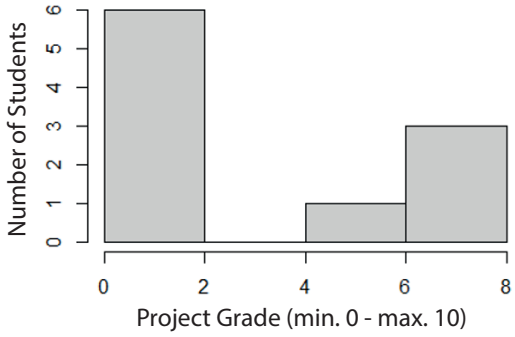


Figure 4. Distribution of students based on the grade obtained in the Programming project for 2025 cohort ($N = 10$).

Table 7. Project grades details of Programming for 2025 cohort ($N = 10$).

Min.	Median	Mean	Std. Dev.	Max.
0.00	0.00	3.11	3.64	8.00

As mentioned in Section 5.1, there was a high drop-out rate at the beginning of the course, with only 10 students working on the project. It should be noted that only 4 out of 10 passed the initial project evaluation, shown in Figure 4. In this sense, it is important to remark the amount of contents and requirements needed for the game, which, together with the daily updates, required important effort and the dedication of the students.

In any case, after the first assessment of the project and examination, interviews were conducted with the students so that they could comment on the main difficulties encountered and the teacher could explain the mistakes made. This allowed them to resolve the problems in the second-chance evaluation, with 8 out of 10 students finally passing.

In order to provide a broader context for these outcomes, the results of the same group in the project of another subject, Markup Language, and the results of other groups in previous years in Programming and Markup Language subjects were also analysed. Part of these latter results can also be found in [14].

https://doi.org/10.3390/computers15020082

5.3.1. Comparison with Markup Language Results

The performance of the same group described in Section 5.1 was analysed in the context of the Markup Language subject during the same year. This subject was generally considered more accessible and straightforward by the students, and they were also required to complete a project at the end of the course, albeit one of considerably less scope and complexity than in the Programming subject. This project consisted of developing a web page with different requirements and functionalities. Moreover, the same 10 students remained in the Markup Language subject. The distribution of the students based on the grade obtained in the Markup Language project is shown in Figure 5 and detailed in Table 8.

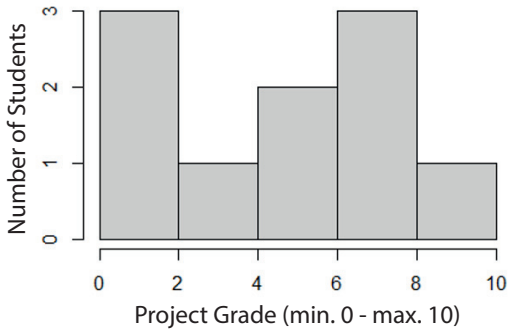


Figure 5. Distribution of students based on the grade obtained in the project of Markup Language for 2025 cohort (N = 10).

Table 8. Project grades details of Markup Language for 2025 cohort (N = 10).

Min.	Median	Mean	Std. Dev.	Max.
0.00	5.50	4.80	3.71	10.00

In this case, 50% of the students passed the project evaluation. After this evaluation, a process similar to that used for the Programming subject was followed. Once the examination had been completed, interviews were conducted with the students so that they could explain their main problems and understand their mistakes. Then, in the second-chance evaluation all the students passed the subject.

As shown in Table 9, the results obtained with the same group in the subjects of Programming and Markup Language were compared. Given the small sample sizes (N = 10), the statistical power was limited, and the results should be interpreted descriptively. Thus, effect sizes (Cohen's d) are reported to complement p-values. The results imply a small effect $\approx$ 0.2, medium $\approx$ 0.5, or large $\approx$ 0.8.

Table 9. Normality and distribution comparison tests for Programming (PR2025, N = 10) and Markup Language (ML2025, N = 10) projects in 2025 cohort.

Test	Cohort	p-Value	Cohen's d	Interpretation
Shapiro–Wilk	PR2025	0.001018	-	Data not normally distributed
Shapiro–Wilk	ML2025	0.1725	-	Data normally distributed
Mann–Whitney U	PR2025 vs. ML2025	0.2823	0.46	No significant difference in project performance
Kolmogorov–Smirnov	PR2025 vs. ML2025	0.5908	-	Distributions are similar

For both distributions of grades, the Shapiro–Wilk normality test was carried out to determine if the data were normally distributed. Since one of the two data sets did not

follow a normal distribution, the Mann–Whitney U test was used. Based on this test, it suggests that there is no significant difference between the data sets. To determine whether the data came from the same type of distribution, the Kolmogorov–Smirnov test was applied. The result suggests that the data were governed by similar distributions.

These facts are very positive, since they suggest that similar results are obtained in subjects of significantly different complexity. This rationale is expanded in Section 6.

5.3.2. Comparison with Other Groups and Years

In order to analyse the results obtained in greater detail, the marks obtained in the practical projects carried out in the Programming and Markup Language subjects in 2024 and Markup Language in 2023 were taken into account. In Figure 6 the students' distributions based on the grades obtained in the projects of 2024 in Programming (a) and Markup Language (b) are shown, and in Figure 7 the distribution of 2023 for Markup Language. The details of the data distributions are shown in Table 10.

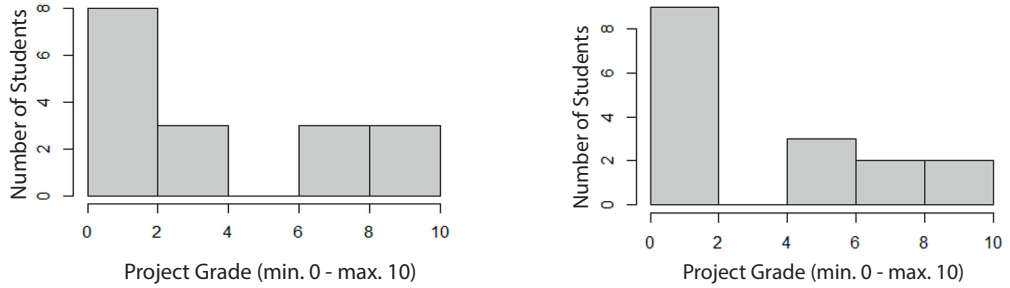


Figure 6. Distribution of students based on the grade obtained in the project for 2024 cohorts: (**a**) Programming (N = 17); (**b**) Markup Language (N = 16).

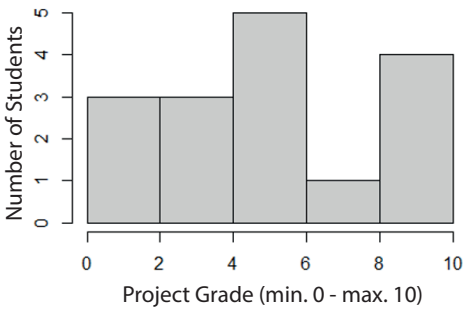


Figure 7. Distribution of students based on the grade obtained in the project for 2023 Markup Language cohort (N = 16).

Table 10. Project grades details for Programming and Markup Language subjects: PR2024 (N = 17), ML2024 (N = 16), and ML2023 (N = 16).

Cohort	Min.	Median	Mean	Std. Dev.	Max.
PR2024	0.00	4.00	3.64	3.87	9.80
ML2024	0.00	2.00	3.35	2.89	9.50
ML2023	0.00	5.00	5.12	3.26	10.00

The analysis of the results was grouped on the basis of the subjects, comparing across these courses the results of the Programming and Markup Language subjects separately.

In Table 11 the data analysis for the Markup Language subject across three courses is described and compared. The Shapiro–Wilk normality test was carried out to determine if

the data followed a normal distribution. In this case, the grades for 2025 and 2023 followed a normal distribution, whereas the data for 2024 did not.

Initially, the courses were compared with each other. For 2025 and 2023, as both followed a normal distribution, Student's *t*-test could be used, the result of which indicates that there were no significant differences between the two data sets. For comparing the 2025 and 2024 courses, as well comparing the 2024 and 2023 courses, since one of the distributions did not follow a normal distribution, the Mann–Whitney U test was carried out, showing in both cases that there was no significant difference in the performance. The Kolmogorov–Smirnov test was also applied to these two latter cases, showing that in both cases the distributions were similar. Finally, the three groups were compared using the Krustal–Wallis test, to determine if there were significant differences among the data sets, with the result showing that there was no significant difference in the data sets.

Table 11. Normality and distribution comparison tests for Markup Language projects across three academic years: ML2023 (N = 16), ML2024 (N = 16), and ML2025 (N = 10).

Test	Cohort	*p*-Value	Cohen's *d*	Interpretation
Shapiro–Wilk	ML2025	0.1725	-	Data normally distributed
Shapiro–Wilk	ML2024	0.002134	-	Data not normally distributed
Shapiro–Wilk	ML2023	0.1881	-	Data normally distributed
T-Student	ML2025 vs. ML2023	0.6373	0.027	No significant difference in project performance
Mann–Whitney U	ML2025 vs. ML2024	0.2972	0.132	No significant difference in project performance
Mann–Whitney U	ML2024 vs. ML2023	0.1831	0.187	No significant difference in project performance
Kolmogorov–Smirnov	ML2024 vs. ML2023	0.1184	-	Distributions are similar
Kolmogorov–Smirnov	ML2025 vs. ML2024	0.187	-	Distributions are similar
Krustal–Wallis	ML2023-ML2024-ML2025	0.3513	-	No significant difference in data

Similarly, in Table 12 the data analysis for the Programming subject across two academic years (2024 and 2025) is described and compared. The Shapiro–Wilk normality test showed that the data for both 2024 and 2025 did not follow a normal distribution. The Mann–Whitney U test showed that there was no significant difference in the performance. The Kolmogorov–Smirnov test is also applied, showing that the distributions are similar. Finally, the groups were compared using the Krustal–Wallis test, to determine if there were significant differences among the data sets, with the resulting that there were no significant differences among the data sets. The negligible effect size (d = 0.15) combined with the non-significant tests suggested similar grade distributions across years, though the small sample sizes limited the statistical power for detecting modest differences if they existed.

Table 12. Normality and distribution comparison tests for Programming projects across two academic years: PR2024 (N = 17) and PR2025 (N = 10).

Test	Cohort	*p*-Value	Cohen's *d*	Interpretation
Shapiro–Wilk	PR2025	0.001578	-	Data not normally distributed
Shapiro–Wilk	PR2024	0.004474	-	Data not normally distributed
Mann–Whitney U	PR2025 vs. PR2024	0.3883	0.038	No significant difference in project performance
Kolmogorov–Smirnov	PR2025 vs. PR2024	0.6335	-	Distributions are similar
Krustal-Wallis	PR2024-PR2025	0.3696	-	No significant difference in data

https://doi.org/10.3390/computers15020082

The results shown in Tables 11 and 12 suggest consistency. It is important to emphasise once again that all these cohorts followed a project-oriented methodology in both subjects, in which the students showed great commitment and dedication, as also registered in [14]. At the academic level, there was no significant difference in the grades for the Markup Language and Programming projects. The Markup Language projects took half the time of the Programming projects, and the Markup Language subject had four hours per week in 2023 and 2024 and three hours per week in 2025. The fact that the grades for the projects in the different modules were comparable is positive because:

- Normally, Markup Language grades were better during the course, compared to Programming, and the students tended to pass almost all the evaluations of Markup Language.
- In contrast, the students typically experienced greater difficulty with Programming assessments throughout the course, with higher failure rates on individual evaluations compared to Markup Language.

Therefore, with respect to 2025, obtaining comparable results in the Programming and Markup Language project is considered an important achievement in terms of: (a) involvement in the classroom (the results assessed were the challenges set in the classroom) and demonstrating that they were at least able to tackle them; (b) the results themselves, because, although the students did not attain high marks, at least they managed to take the right approach to each daily challenge. In addition, the fact that the Markup Language project lasted two weeks less meant that it was not possible to carry out more ambitious challenges in Markup Language than those that were carried out in the Programming project. The custom-developed software application was important during the academic course 2025, because they saw graphically and explicitly why they had made mistakes and what path they should follow or correct. These results will be analysed in greater detail in the next section, Section 6.

6. Discussion

The profiles of Vocational Education and Training students are very diverse, with different backgrounds and motivations. Different levels of education and different personal situations lead to high levels of absenteeism and dropout rates, which may impact on the classroom environment [6].

This study examined the implementation of a gamified Project-Based Learning approach combined with a custom software tool for teaching Programming and graph theory in first-year VET students. For the group studied, the students' attitude was positive and the classroom atmosphere was good. Despite their previous lack of knowledge of the contents of the subject, the students who participated in the project showed considerable interest and autonomy. Similarly to what is indicated in [4,46], the students became involved in the development of the project despite the complexity of many of the concepts, maintaining regular and consistent attendance. However, some of them continued to have difficulty with some of these concepts, similar to what is indicated in [54].

The findings provide valuable insights into both the potential and challenges of applying active methodologies in vocational computer science education, particularly with students presenting diverse academic backgrounds and varying levels of motivation. Next, the results of the present research are discussed from different perspectives.

6.1. Interpretation of Academic Results, Student Engagement and Dropout Rates

The academic results obtained must be interpreted within the specific context of the Programming subjects in VET. Although the results of the initial project evaluation may appear concerning at first glance due to the low grades, several factors contextualise them

https://doi.org/10.3390/computers15020082

positively. The complexity and scope of the project, which required knowledge about graph theory, OOP, Markup Language, and algorithmic thinking, represented a substantial challenge for first-year students with no prior Programming experience. Second, qualitative evidence (teacher observations, knowledge acquisition improvement comparing exams and follow-ups, student perceptions as given in the questionnaires) and the progression from 4 students passing the first project evaluation to 8 students (80% of those who remained in the subject) passing after the second evaluation suggests the effectiveness of the iterative learning process and the importance of post-evaluation interviews and feedback mechanisms.

The comparison with the Markup Language subject results also suggests the effectiveness of the approach. The Markup Language subject was generally perceived as more accessible by the students, and it required less complex abstract thinking than Programming. Despite this, the statistical analysis suggested no significant differences between the distributions of grades in both subjects. This is particularly meaningful considering that the students usually had greater difficulty in passing Programming than in passing Markup Language: the Programming project lasted two more weeks than the Markup Language project and involved significantly greater complexity; and the Programming project required the integration of multiple advanced concepts simultaneously. Thus, the students achieved results in a traditionally difficult subject that were comparable to those in a simpler and more accessible subject.

These results align with previous findings [3,13,14] that suggested that active methodologies can level subjects of varying difficulty by increasing participation and providing practical contexts for applying theoretical concepts.

These results, combined with qualitative observations of sustained engagement, voluntary time extension beyond class hours, spontaneous peer collaboration, and student interest, suggest patterns consistent with enhanced learning motivation. However, grades alone cannot definitively establish motivation levels since successful performance could alternatively reflect prior ability, external pressures, or other factors. The qualitative evidence provides a stronger basis for claims about motivation than grades alone.

In this sense, the students showed great commitment to the project. In the documentation and public presentation of the project, they stated that it helped them to integrate the knowledge taught throughout the course (OOP, arrays, files, databases, control structures, collections, etc.). This is even more significant considering that there was a clear lack of interest and commitment to studying at the beginning of the course. In a post-evaluation questionnaire, the students unanimously stated that the task or work that most helped them understand the complex concepts was the project. They also commented that during its implementation they gained a better understanding of the use of collections, arrays and files, and different algorithms. This supports the first research exploratory proposition regarding enhanced motivation and interest.

The statistical comparisons across three academic years (2023, 2024, and 2025 for Markup Language, and 2024 and 2025 for Programming) indicated no significant differences in project performance for either the Programming or Markup Language subjects. Although this may initially suggest that the new software tool did not dramatically improve outcomes, this interpretation would be oversimplified. All cohorts followed project-based active methodologies [3,14], with the 2025 intervention specifically incorporating the gamification element and software tool.

The consistency of the results across the years actually suggests that active, project-based approaches yield reliable outcomes regardless of specific technological implementations. Moreover, the addition of gamification and the software tool maintained effectiveness

https://doi.org/10.3390/computers15020082

while adding complexity to the learning experience; showing that the methodology seems to be robust across different student cohorts with varying profiles and backgrounds.

In addition, the qualitative observations from the teacher's diary documented higher levels of classroom discussion, student-to-student knowledge sharing, and visible engagement during the 2025 implementation. However, it is important to note that these outcomes were not fully captured by grade distributions alone. The students were observed discussing strategic approaches, debugging solutions, and exhibiting curiosity about alternative implementations. This deep involvement suggests that learning outcomes may have been enhanced in ways not reflected purely in numerical grades.

This finding emphasises the importance of mixed-methods approaches in educational research [59–61,64]. Quantitative data provide important benchmarks, but qualitative observations of classroom dynamics, student interactions, and engagement patterns offer essential complementary insights into pedagogical effectiveness.

6.2. Dropout Rates and Student Retention Challenges

The high dropout rate at the beginning of the course, passing from 24 initial students to 10 remaining by the project phase, represents a significant concern that requires careful consideration. Understanding the actual reasons for dropout, based on student perspectives rather than speculation, is essential for improving future implementations.

This attrition rate, while consistent with rates in typical VET computer science courses [4–7], may raise questions about whether the gamified PBL approach adequately addressed retention issues as proposed.

It is important to remark that several factors may have contributed to the dropout, some of them stated by the students themselves, such as that for a significant proportion of students the initial course selection is often driven by external factors (job market advertising, family pressure) rather than genuine interest; or that some students had negative previous educational experiences in Compulsory Secondary Education. Furthermore, in the past there was an entrance exam to be admitted in VET courses, but it has been eliminated in recent years. This means that there is no minimum level of knowledge required to access computer science VET courses, which can be an insurmountable barrier for some first-year students.

Similarly, some of the students who dropped out had jobs, making it impossible for them to combine their studies with their work at the beginning of the course. Others students never provided any explanation.

All dropouts occurred at the beginning of the course, while the project was developed in the last month. Thus, the above suggests that dropout had multiple causes, mainly unrelated to the specific pedagogical approach:

- External constraints beyond pedagogical control, such as work-life conflicts, which no pedagogical intervention alone can address. Institutional support (flexible scheduling, financial aid, asynchronous or distance learning options) would be required.
- Early-stage course selection issues. These students departed during introductory Programming content, not specifically due to the project. Better pre-enrolment orientation might reduce this attrition source.
- Pre-existing self-efficacy challenges due to prior negative schooling experiences. While active methodologies might eventually build confidence, early withdrawal suggests that these students needed more intensive scaffolding and encouragement during initial struggles.

Notably, among the students who remained engaged through the project phase, retention was notably higher, and none dropped out during the final month, when implementation of the intervention was carried out. This suggests that while the approach may

not prevent early-stage attrition driven by fundamental course-fit issues, it does appear to sustain engagement among students who remain. The daily follow-ups, requiring demonstration of progress and understanding, may have actually worked as a retention mechanism for committed students. However, careful analysis must be carried out in future interventions to confirm that these daily follow-ups do not accelerate the departure of those lacking sufficient motivation.

These findings partially support the first exploratory proposition, since the gamified PBL approach enhanced motivation and engagement for the students who remained, but it did not eliminate early-course absenteeism driven by deeper motivational or circumstantial factors. The proposal does not seek to delve into those issues that fall outside the scope of the methodology used. This suggests that interventions targeting initial course selection and early orientation may be needed to complement pedagogical innovations to fully address retention challenges.

6.3. The Role of the Software Tool

The custom-developed software tool emerged as a critical component of the learning ecosystem, functioning simultaneously as a pedagogical support and an assessment instrument. The tool's dual functionality, enabling students to validate their solutions step-by-step while allowing the teacher to automate evaluation, addresses the common challenge in Programming education of providing immediate, meaningful feedback without overwhelming teacher workload.

Beyond its technical functionality described in Section 4, understanding how the students actually used the custom-developed tool and whether usage patterns correlated with project outcomes provided important insights into the tool's pedagogical effectiveness.

Tool usage data were collected through several sources, among which the responses to interviews, iterative project versions, and teacher observations stand out. During post-evaluation informal interviews, the students were asked about when and how they used the tool, which features they found most helpful, and whether they encountered difficulties using it. The step-by-step execution was the most important and useful feature for the students. Examining code provided by the students and comparing early vs. final implementations revealed patterns of iterative refinement potentially guided by tool feedback.

During the implementation of the project and the use of the tool, some common errors were identified and solved. These common errors were mainly Dijkstra's path determination, and determining the appropriate route for the thief by choosing non-optimal paths or calculating the shortest routes. These errors were quickly identified, thanks to the graphical visualisation of the routes and step-by-step execution. The tool's analysis of the routes also made it possible to verify the proper handling of restrictions and obstacles. Similarly, the complete execution made it possible to verify the implementation of the police strategy. While correlation does not prove causation, qualitative evidence from student interviews supports the interpretation that the tool provided genuine pedagogical value by externalising algorithmic processes and enabling iterative refinement, suggesting meaningful relationships between tool usage and outcomes.

As mentioned, the step-by-step execution feature proved to be particularly useful for debugging and understanding algorithmic logic. The students could visualise how their strategies worked compared to optimal and alternative solutions, identify specific failures or errors, and iterate on their approaches. This immediate visual feedback mechanism aligns with constructivist learning principles [31,32] by allowing students to build understanding through experimentation and reflection. The graphical representation of routes, police positions, and game outcomes transformed abstract algorithmic concepts into concrete,

https://doi.org/10.3390/computers15020082

visible consequences of Programming decisions. These findings support the second research exploratory proposition about the use of the software tool to facilitate the learning of Programming concepts.

However, teacher observations and informal interviews revealed that the effective use of the tool was initially difficult for the students. Some attempted to reverse-engineer solutions by repeatedly testing variations rather than working with the underlying algorithmic principles. This behaviour underscores an important limitation: technological tools alone do not guarantee effective learning. Teacher guidance was essential for helping the students interpret tool feedback effectively, especially initially. The post-evaluation interviews proved to be essential for helping the students understand why certain approaches failed and how to interpret the tool's feedback constructively. This is in line with the literature on educational software environments [22–25,29,30], which emphasises that pedagogical support must accompany technological tools to maximise their learning potential.

6.4. Gamification as Game Development: Pedagogical Advantages, Challenges and Implications

The specific approach to gamification employed in this study required students to develop the game itself rather than merely adding game elements or playing serious games [15,18]. This approach presents both unique advantages and challenges by creating a functional "cops and thieves" game based on graph theory, OOP, and visual interfaces, and demanding the use and integration of multiple Programming skills simultaneously. Moreover, the narrative of the game can be easily adapted depending on the student's interest.

This integration aligns well with the constructionist learning paradigm [32], where students construct knowledge by creating meaningful products. The game development context provided authentic motivation because the students were not solving abstract algorithmic problems but were implementing strategies for a concrete practical scenario. The narrative element (fugitive versus police) added an intuitive and simple framework for understanding graph theory, pathfinding, and optimisation concepts that might otherwise seem disconnected from real-world applications, supporting the first exploratory proposition.

However, this approach's complexity may constitute both a strength and a limitation. It successfully engaged students who remained in the course, but the steep learning curve and multiple simultaneous technical demands may overwhelm some students. This latter fact was reported by some students, who declared feeling overwhelmed by the scope, particularly when combining OOP, graph theory concepts (which were new to them), file I/O operations, and interface design.

The personalised conditions assigned to each student (different obstacles, starting points, constraints) emerged as a particularly effective design decision. This customisation prevented direct copying while encouraging the students to understand principles deeply enough to adapt them to unique scenarios. This addressed a common challenge in Programming education where students may rely excessively on shared or generated code. This customisation of scenarios for each student was possible thanks to the use of the developed software tool, allowing automatic evaluation and providing feedback information, supporting the second research exploratory proposition.

According to the findings obtained, gamified PBL approaches can successfully bridge the gap between theoretical computer science concepts (graph theory, algorithms) and practical Programming skills, which has been a constant challenge in vocational education [36,39–41]. The project-centred approach is in line with VET students' preference for practical and application-oriented learning.

In addition, the development of supporting technological tools, even relatively simple applications, can substantially enhance both learning and teaching efficiency in VET. The software tool developed does not require extensive infrastructure or specialised equipment;

https://doi.org/10.3390/computers15020082

it operates on standard computers with Java installations. This accessibility is crucial for VET institutions that may lack resources for educational technology.

Furthermore, most documented gamified Programming education employs visual, block-based environments such as Scratch, Alice, or Blockly, which are designed for novices and younger learners [22–30]. Although these environments serve important pedagogical purposes, VET students require experience with text-based, professional development tools. This work demonstrates that gamified Project-Based Learning can be extended beyond introductory block-based environments to first-year VET courses using industry-standard languages (Java) and professional development environments (Eclipse), maintaining engagement while developing skills directly transferable to professional practice.

Despite the above, the results also highlight the necessity of robust support structures. The daily follow-ups, availability for consultation, post-evaluation interviews, and the second-chance evaluations were essential for the approach success. These support mechanisms require significant teacher time and commitment, and they may limit scalability without adequate institutional support and appropriate teacher–student ratios.

6.5. Limitations and Methodological Considerations

Several limitations must be acknowledged when interpreting these findings and their implications for VET Programming education.

The most significant limitation is the small sample size resulting from high early course attrition, retaining only 10 students through the project phase. This substantially limits the statistical significance and generalisability of quantitative analyses. Even robust non-parametric tests lack sufficient power to detect moderate effects, and the risk of false negative errors is considerable.

The statistical comparisons presented in Tables 9, 11 and 12 should therefore be interpreted as descriptive observations of patterns rather than definitive hypothesis tests. The finding of "no significant differences" between Programming and Markup Language grades ($p = 0.2823$, Cohen's $d = 0.46$) indicates a small-to-medium effect size. This does not prove equivalence; rather, it suggests that within the constraints of the available data, grade distributions followed similar patterns. Similarly, comparisons across academic years (2023–2025) reveal Cohen's d values representing negligible effects. While none reached statistical significance, the pattern of effect sizes suggests potential practical evidences that a larger sample might detect.

Therefore, the quantitative data reveal suggestive patterns consistent with the qualitative observations, but do not constitute definitive statistical proof. Future research with larger samples would be necessary to confirm these preliminary quantitative trends.

The qualitative component of this study provides richer evidence for understanding student experiences and learning processes than grades alone can capture. These qualitative data revealed patterns of increased classroom discussion, peer collaboration, strategic problem-solving, and sustained engagement throughout the project phase among students who remained in the course. However, the qualitative analysis also has limitations. The primary observer was the course instructor, introducing potential confirmation bias despite efforts at systematic documentation and triangulation. Interviews were conducted by the same instructor who would grade the second chance evaluations, potentially influencing student responses despite efforts to create a comfortable atmosphere. The absence of independent observers or inter-rater reliability checks limits the generalisation of qualitative interpretations. Future iterations of this research would benefit from involving external observers, student self-reports, or peer evaluations. Despite these limitations, the systematic procedures employed, triangulation across multiple data sources, and explicit acknowl-

https://doi.org/10.3390/computers15020082

edgment of constraints strengthen the trustworthiness of qualitative findings within the bounds of action-research methodology [57,60].

The high early dropout rate (58% attrition) introduced substantial selection bias. The students who completed the project likely possessed greater intrinsic motivation, stronger foundational skills, or more favourable external circumstances than those who dropped out. Consequently, the observed engagement and learning outcomes may reflect characteristics of persistent students rather than effects of the pedagogical approach itself.

This self-selection process also limits the generalisation of findings. The approach may work well for motivated students, but its effectiveness for at-risk students, precisely the population where gamification theoretically offers greatest benefit, remains uncertain.

As an action-research study conducted within real educational practice, this work lacked experimental controls. There was no control group receiving traditional instruction, no random assignment to conditions, and no pre-/post-assessments of specific conceptual knowledge or skills. The comparisons with previous academic years provided historical context but did not control for cohort effects, evolving teacher expertise, or broader institutional changes. These limitations are inherent to action-research designs, which prioritise ecological validity and practical improvement over experimental rigour [57]. The trade-off is appropriate for exploratory research in VET settings where experimental manipulation would be impractical or ethically problematic. However, it means that causal claims must be expressed cautiously, and findings should be viewed as preliminary evidence warranting further investigation rather than definitive conclusions.

The dual role of teacher–researcher introduces additional methodological constraints beyond those already discussed. The simultaneous position as instructor, evaluator, observer, and analyst created potential for confirmation bias, investment bias, and evaluator influence on student behaviour. Systematic documentation procedures, triangulation across data sources, and deliberate attention to disconfirming evidence (e.g., student reports of frustration, initial tool usage difficulties, feelings of being overwhelmed) provided partial mitigation, but the absence of independent external observers limited confidence in qualitative interpretations. All classroom observations, interview analyses, and interpretive conclusions reflected a single perspective. This positionality is appropriate for action-research generating contextualised practitioner knowledge, which is valuable for informing similar educational contexts rather than context-independent generalisable truths. The teacher–researcher perspective enables insights into pedagogical decision-making and implementation challenges difficult to access through external observation alone. However, the findings should be interpreted with awareness of these inherent constraints. Future iterations would benefit from incorporating external observers, student self-assessment instruments independent of teacher evaluation, or collaborative analysis with educational researchers not directly involved in instruction.

Despite these limitations, the convergence of multiple data sources strengthens confidence in the core findings. The quantitative grade patterns, though not statistically powerful, aligned with the qualitative observations. The student interviews corroborated the teacher diary entries. The project evaluation demonstrated the application of graph theory and OOP concepts. This triangulation, while imperfect, provided more robust evidence than any single data source alone. This study offers promising preliminary evidence that gamified PBL combined with automated tool support can engage VET Programming students and support learning of complex concepts. The evidence is sufficient to warrant continued development and more rigorous evaluation, but not sufficient to make strong generalisable claims about effectiveness across diverse VET contexts.

https://doi.org/10.3390/computers15020082

6.6. Generalisability, Transferability, and Scalability Considerations

The intervention described was highly contextualised within a specific educational setting, i.e., a first-year VET Programming course with a small cohort (10 students) as an end-of-year summative project. While these contextual factors do not invalidate findings within action-research methodology [57,60,61], they necessarily limit claims about broader applicability. The findings should be viewed as exploratory evidence of feasibility and potential effectiveness rather than definitive proof of generalisable outcomes across all VET Programming contexts.

Several core aspects appear potentially transferable. The pedagogical principles (gamification through game development, personalised project conditions, and tool-supported iterative validation) do not depend fundamentally on specific institutional characteristics. The "cops and thieves" narrative is culturally accessible, though alternative scenarios (delivery route optimisation, network pathfinding) could be equally effective. However, successful transfer requires careful attention to prerequisite knowledge, curricular sequencing, and alignment with institutional realities regarding student profiles, curricular structures, and available resources.

Scalability to larger cohorts presents pedagogical and technical considerations. With 10 students, individual daily follow-ups were manageable; with 30–40 students, this may become impractical without modification. Potential adaptations include: (i) shifting to small-group checkpoints (3–4 students) rather than individual reviews; (ii) implementing structured peer review mechanisms; (iii) leveraging the software tool more extensively for automated preliminary feedback, with teacher intervention reserved for complex conceptual issues; and (iv) employing teaching assistants for technical troubleshooting while instructors focus on conceptual guidance.

The software tool scales relatively well technically. Adding personalised conditions for additional students requires minimal effort (primarily configuration file creation), and the tool can evaluate projects from any number of students. However, deployment requires moderate technical expertise. An instructor capable of teaching first-year VET Programming subjects possesses sufficient knowledge to use the tool, modify configurations, and perform basic troubleshooting. Meaningful adaptation (alternative game scenarios, modified algorithms) requires deeper competence in graph algorithms and software architecture. Future work could address this barrier through more modular, configuration-driven architectures enabling common adaptations without code modification.

Thus, although the specific implementation is contextually bound, the underlying approach appears transferable to diverse VET Programming contexts with appropriate adaptation. Successful transfer requires the above-mentioned factors to be taken into account. Claims about broader applicability should be tempered by acknowledgment of these contextual dependencies, with findings viewed as promising preliminary evidence warranting replication across varied settings rather than as prescriptive universal solutions.

6.7. Future Work

Longitudinal studies tracking the subsequent academic performance and professional outcomes of students could assess whether gamified PBL experiences translate into longer-term benefits beyond immediate course completion.

Experimental designs incorporating control groups or comparison conditions (traditional instruction, game-element gamification, serious games) could more definitively isolate the specific contributions of game development approaches compared to other active methodologies.

It may be possible to expand the capabilities of the software tool to support additional game scenarios, algorithms, and Programming concepts. This way, the tool could broaden

https://doi.org/10.3390/computers15020082

its applicability across different units within Programming courses. Its modular architecture facilitates such extensions. In this sense, it will be important to investigate further the specific features of the software tool that the students found most valuable. Tracking studies or detailed usage analytics could provide insights into how students actually interact with the tool during learning.

The exploration of hybrid approaches to introduce students more gradually into complex game development projects may reduce early overwhelming feelings while preserving engagement benefits. A progressive sequence from simpler gamified exercises to full game development could optimise the challenge level throughout the learning process.

Finally, addressing the early dropout problem requires interventions beyond pedagogical methodology alone. Research examining pre-course orientation programs, realistic course preview opportunities, and early-alert systems identifying students at risk could complement pedagogical innovations to improve overall retention rates in VET.

7. Conclusions

This work presented the design, implementation, and evaluation of a gamified Project-Based Learning approach for teaching Programming and graph theory to first-year VET students, distinguished from the existing literature by three novel contributions. First, a methodological framework employing gamified Project-Based Learning with systematic personalisation: each student develops a complete game under unique conditions (distinct obstacles, starting points, constraints), preventing superficial copying while requiring adaptive algorithmic thinking and integration of multiple advanced concepts (graph theory, OOP) using industry standard tools. Second, a custom-developed software tool with dual functionality serving simultaneously as pedagogical scaffold (enabling students to test solutions iteratively, visualise algorithmic decisions, and compare against optimal strategies) and automated assessment platform (allowing teachers to evaluate personalised projects efficiently despite varying conditions). Third, mixed-methods empirical evidence from authentic VET implementation documenting both the approach's promise, with qualitative observations indicating sustained engagement and peer collaboration, and its challenges, including high early dropout and methodological constraints limiting statistical inference. These contributions extend the existing gamification and PBL literature.

The results suggest that gamified PBL can effectively engage VET students in complex Programming tasks such as graph theory and OOP. Despite the project's considerable scope, 80% of the students, from those who remained engaged in the subject, successfully completed it by the second evaluation. The qualitative evidence (diary, follow-ups, questionnaires) indicates there was an improvement in student learning and suggests a correlation between motivation and results obtained. The statistical analysis, although complementary and limited, potentially suggests no significant differences between student performance in the Programming subject and the more accessible Markup Language subject, diverging from typical patterns where Programming learning results are substantially worse than those from Markup Language. This suggests that the gamified approach successfully improves motivation and provides effective learning for difficult concepts, although this should be confirmed by further studies with larger samples.

The custom software tool proved to be instrumental by enabling students to validate solutions step-by-step and also by allowing the teacher to efficiently evaluate diverse implementations under personalised constraints. The students could visualise algorithmic decisions, compare strategies with optimal solutions, and iteratively refine their approaches. However, the effectiveness of the tool depended critically on complementary support structures including the daily follow-ups, post-evaluation feedback sessions, and teacher guidance in interpreting results.

https://doi.org/10.3390/computers15020082

Important limitations must be acknowledged. The high early-course dropout rate indicates that pedagogical innovations alone cannot address fundamental issues of course selection, prior educational experiences, and external pressures. However, engagement was high among students who persisted. This suggests that comprehensive interventions targeting orientation and early student support should complement pedagogical improvements.

Preliminary evidence on the consistency of results across several academic years suggests the robustness of project-based active methodologies. These results may indicate that gamification and the software tool do not produce dramatically higher grades. However, qualitative observations showed enhanced classroom dynamics and deeper engagement, highlighting the importance of mixed-methods research capturing both quantitative and qualitative outcomes.

Therefore, gamified Project-Based Learning, supported by appropriate technological tools and pedagogical structures, offers a viable approach for teaching complex Programming concepts in VET contexts. While these findings are contextually situated within a specific VET Programming course with a small cohort, the underlying principles appear transferable to similar educational settings with appropriate adaptation to local institutional realities and resources. The findings, although preliminary due to the limited sample size, contribute to the growing body of the literature on active methodologies in STEM education generally and VET contexts specifically.

Author Contributions: Conceptualisation, B.D.-L. and D.M.-S.; methodology, B.D.-L. and D.M.-S.; software, B.D.-L.; validation, B.D.-L. and D.M.-S.; formal analysis, B.D.-L. and D.M.-S.; investigation, B.D.-L.; resources, B.D.-L.; writing—original draft preparation, B.D.-L. and D.M.-S.; writing—review and editing, B.D.-L. and D.M.-S.; visualisation, B.D.-L. and D.M.-S.; supervision, D.M.-S.; project administration, D.M.-S.; funding acquisition, D.M.-S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part under project PID2023-146540OB-C41, funded by MCIN/AEI/10.13039/501100011033. Part of this work was also supported by Innovation Group "IEData" GID2016-6.

Informed Consent Statement: The authors declare that no human experimentation has been carried out in this work. The results of the present research are the result of the normal performance of the profession of the first author, a vocational education teacher. This work performance followed the protocols and regulations of the educational centre where the results were obtained. The participants involved in the analysis of the work were all adults and they gave their informed consent to the anonymous use of their academic results. No personal data were stored.

Data Availability Statement: The original contributions presented in this study are included in the article. Further information or inquiries associated with this study are available upon reasonable request to the first author.

Acknowledgments: The authors acknowledge the use of artificial intelligence tools for language editing, grammar checking, and style enhancement. All scientific content, analysis, and conclusions remain solely the responsibility of the authors.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript and are listed in alphabetical order:

BFS	Breadth First Search
CL	Collaborative Learning

CSE	Compulsory Secondary Education
CT	Computational Thinking
DFD	Data Flow Diagram
DFS	Depth First Search
DT	Design Thinking
ERM	Entity-Relationship Model
FP	Flipped Classroom
ML	Markup Language
OOP	Object-Oriented Programming
PBL	Project-Based Learning
PR	Programming
STEM	Science, Technology, Engineering, and Mathematics
VET	Vocational Education and Training
WAD	Web Application Design

Appendix A. Tool Design and Implementation

Appendix A.1. Data Flow and Functioning

To briefly show the design carried out, a set of Data Flow Diagrams (DFDs) is shown, which describes the main processes and actions that were taken into account before the implementation of the software. In Figure A1 the flow diagram with the main processes can be seen. These processes start with the configuration of the application. Then the algorithms must be applied and the thief's routes are analysed. In parallel, the police strategy is also defined. Finally, the game is run after having defined the conclusion conditions, and the solutions are evaluated.

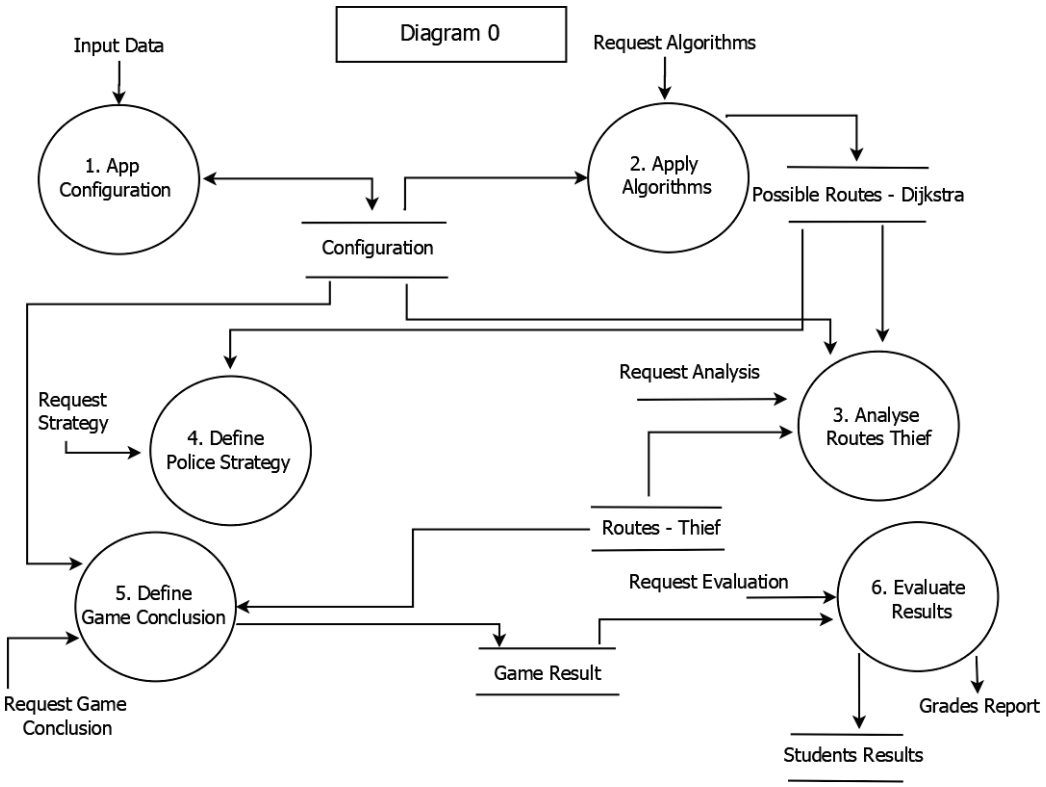


Figure A1. Main processes implemented in the tool and their inputs, outputs and relations.

Some of these processes can be seen in more detail in Figures A2–A4. In Figure A2 the computation of the thief's routes is shown, where the obstacles are analysed, the Dijkstra paths are computed and the possible thief's routes defined. In Figure A3 the game conclusion is established by checking the thief's state, the conditions at which the thief

https://doi.org/10.3390/computers15020082

reaches the final node and the determination of the winner, thief or police. Finally, in Figure A4 the evaluation of the solutions proposed is described, where the solutions are compared and evaluated, and a grade report is generated.

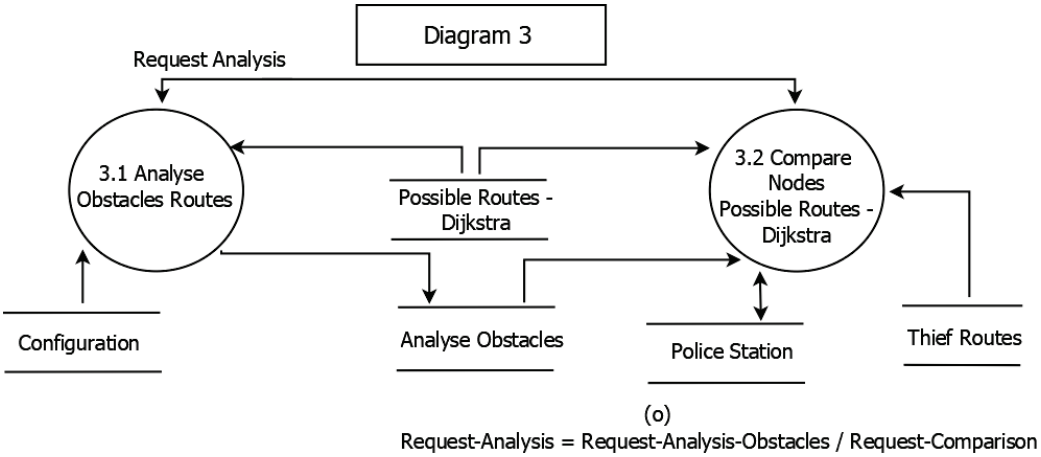


Figure A2. Inputs, outputs and actions to analyse the thief's routes.

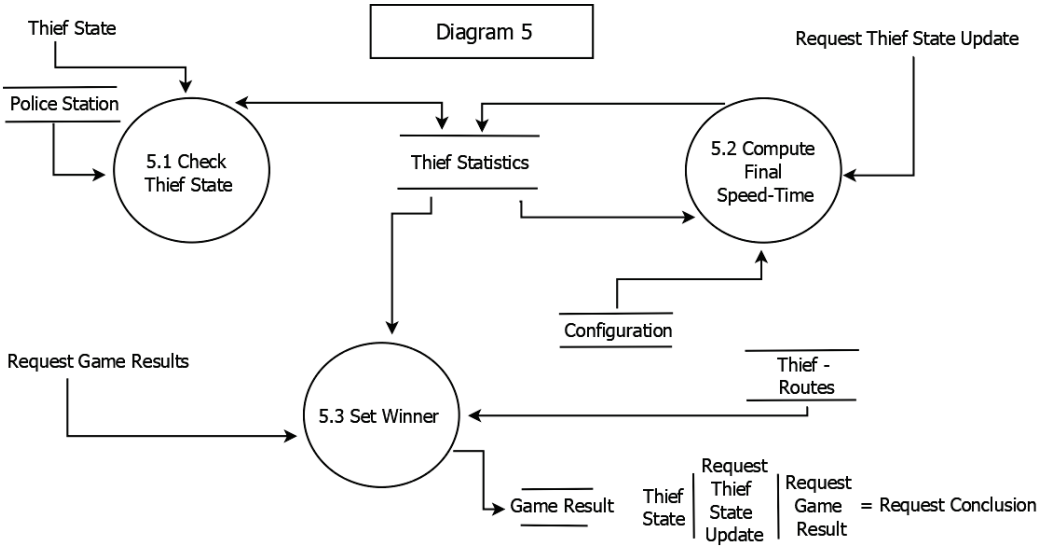


Figure A3. Inputs, outputs and actions to conclude the game, set the winner and give the results.

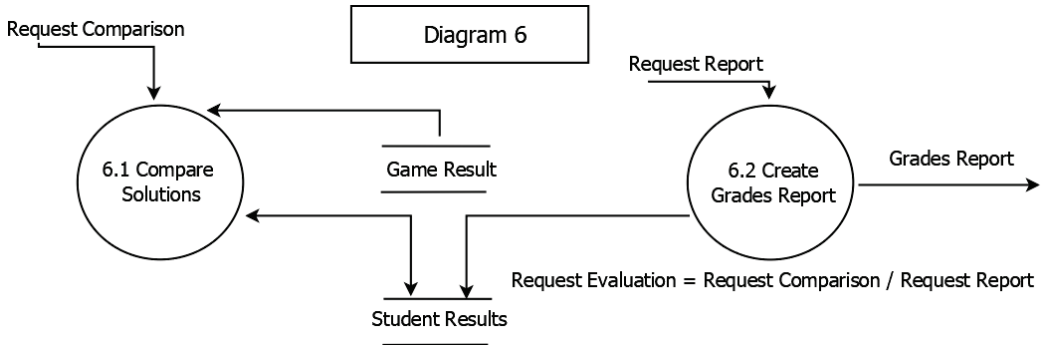


Figure A4. Inputs, outputs and actions to compare and evaluate the solutions.

Appendix A.2. Interface

The interface structure follows a simple pattern, allowing all the data and actions needed during the game to be displayed on a single screen. The main sections, displayed in Figure A5, are: (1) title, (2) initial configuration, (3) configuration actions, (4) step-by-step execution, (5) complete execution, (6) solution evaluation, (7) graph used, and (8) results.

https://doi.org/10.3390/computers15020082

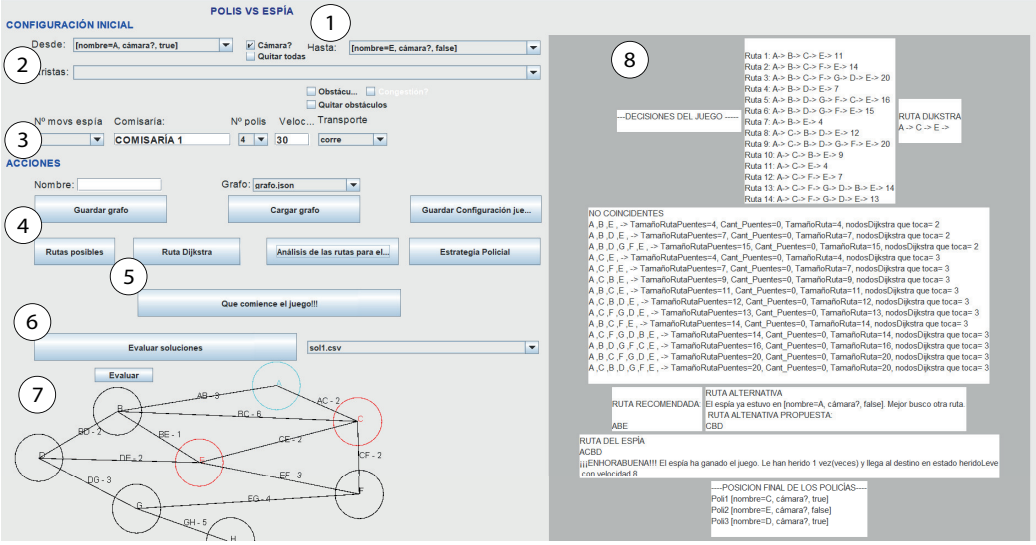


Figure A5. User interface of the custom-developed software tool.

The design has been made to ensure that the application is flexible and easy to maintain or update. Several actions are performed before starting the game (left side of Figure A5):

- Define the initial configuration: When entering the application for the first time, it must be defined where the thief starts the route and where he/she is headed; the means of transport that determines the speed; specifying the characteristics of the streets; recording the name of the police station and the number of police officers with whom it will operate, as well as the nodes where cameras will be located, if any, to inform about the thief's position if he/she passes over any of these nodes.
- Save and load game configuration data: The established configuration is saved in a configuration file, which will allow the data to be loaded at any time.
- Save and load the graph: Similarly, the main data of the graph is saved in a .json file, which can be loaded later.

An objective of the application is to be interactive, providing information about each action that is carried out. Therefore, on the right side of Figure A5, the application keeps the user updated on what is happening, mainly at the step-by-step execution level. Among the main important outputs, we can find:

- Possible routes: All possible routes from the origin to the destination are shown. These are determined using the DFS algorithm.
- Dijkstra route: This is determined by applying Dijkstra's algorithm and obtaining a matrix with the shortest distance from one node to the rest.
- Analysis of routes for the thief: The number of Dijkstra nodes (the nodes belonging to the shortest route) that cross each possible path is analysed, as well as the obstacles presented in each route. In this case, the obstacles are weighted and the cost of the route is recalculated.
- Recommended and alternative routes: The recommended route is calculated, taking into account whether there are obstacles, and, if there are any, a specific coefficient is applied and the size of the route is recalculated. It may happen that the next node to which the thief must go has already been visited. In this case, an alternative route is offered, which the system recommends to avoid loops in the route or unnecessary encounters with a police officer.
- The thief's final route: Describes the nodes of the route, as well as the speed at which the thief arrives at the destination. A message is also displayed with the winner of the game.

https://doi.org/10.3390/computers15020082

- Location of the police officers: Shows the final position of the police officers at the end of the chase.
- Reports of the correct solutions for each student: Summarises the process used by the students to arrive at the solution, according to the initial conditions of the game for each one.

References

1. Staniškis, J.K.; Katiliūtė, E. Complex evaluation of sustainability in engineering education: Case & analysis. *J. Clean. Prod.* **2016**, *120*, 13–20. [CrossRef]
2. UNESCO. *Using ICTs and Blended Learning in Transforming Technical and Vocational Education and Training*; Commonwealth of Learning: Burnaby, BC, Canada, 2017.
3. Díaz-Lauzurica, B.; Moreno-Salinas, D. Applying Design Thinking to Enhance Programming Education in Vocational and Compulsory Secondary Schools. *Appl. Sci.* **2023**, *13*, 12792. [CrossRef]
4. Chondrogiannis, E.; Symeonaki, E.; Papachristos, D.; Loukatos, D.; Arvanitis, K.G. Computational Thinking and STEM in Agriculture Vocational Training: A Case Study in a Greek Vocational Education Institution. *Eur. J. Investig. Health Psychol. Educ.* **2021**, *11*, 230–250. [CrossRef] [PubMed]
5. Cerda-Navarro, A.; Salva-Mut, F.; Comas Forgas, R. A typology of students in intermediate vocational education and training programmes based on student engagement factors, sociodemographic characteristics and intentions of dropping out. *Eur. J. Educ.* **2019**, *54*, 635–650. [CrossRef]
6. Cerdà-Navarro, A.; Quintana-Murci, E.; Salvà-Mut, F. Reasons for dropping out of intermediate vocational education and training in Spain: The influence of sociodemographic characteristics and academic background. *J. Vocat. Educ. Train.* **2022**, *76*, 619–643. [CrossRef]
7. Zhao, Y.; Ko, Y.O.J. Design thinking as an innovative pedagogy and innovation as a learning process in leading vocational education change in China: Lessons from two cases. In Proceedings of the 31st International Congress of School Effectiveness and School Improvement (ICSEI): Deepening School Change for Scaling: Principles, Pathways and Partnerships, Singapore, 8–12 January 2018.
8. Woraphiphat, I.; Roopsuwankun, P. The impact of online design thinking-based learning on entrepreneurial intention: The case of vocational college. *J. Innov. Entrep.* **2023**, *12*, 2192–5372. [CrossRef]
9. Krüger, M. Design Thinking for German Vocational Schools? Discovering of an Innovative Approach by Testing in Teacher Education. *Open Educ. Stud.* **2019**, *1*, 209–219. [CrossRef]
10. Yadav, A.; Good, J.; Voogt, J.; Fisser, P. Computational Thinking as an Emerging Competence Domain. In *Competence-Based Vocational and Professional Education: Bridging the Worlds of Work and Education*; Mulder, M., Ed.; Springer International Publishing: Cham, Switzerland, 2017; pp. 1051–1067. [CrossRef]
11. Su, J.; Yang, W. A systematic review of integrating computational thinking in early childhood education. *Comput. Educ. Open* **2023**, *4*, 100122. [CrossRef]
12. Lv, L.; Zhong, B.; Liu, X. A literature review on the empirical studies of the integration of mathematics and computational thinking. *Educ. Inf. Technol.* **2023**, *28*, 8171–8193. [CrossRef]
13. Díaz-Lauzurica, B.; Moreno-Salinas, D. Computational Thinking and Robotics: A Teaching Experience in Compulsory Secondary Education with Students with High Degree of Apathy and Demotivation. *Sustainability* **2019**, *11*, 5109. [CrossRef]
14. Díaz-Lauzurica, B.; Moreno-Salinas, D. Active Learning Methodologies for Increasing the Interest and Engagement in Computer Science Subjects in Vocational Education and Training. *Educ. Sci.* **2025**, *15*, 1017. [CrossRef]
15. Samah, L.A.; Ismail, A.; Hasan, M.K. The Effectiveness of Gamification for Students' Engagement in Technical and Vocational Education and Training. *Int. J. Adv. Comput. Sci. Appl.* **2022**, *13*, 173–180. [CrossRef]
16. Jayalath, J.; Esichaikul, V. Gamification to Enhance Motivation and Engagement in Blended eLearning for Technical and Vocational Education and Training. *Technol. Knowl. Learn.* **2022**, *27*, 91–118. [CrossRef]
17. Lihui Sun, Z.G.; Hu, L. Educational games promote the development of students' computational thinking: A meta-analytic review. *Interact. Learn. Environ.* **2023**, *31*, 3476–3490. [CrossRef]
18. Huang, W.; Li, X.; Shang, J. Gamified Project-Based Learning: A Systematic Review of the Research Landscape. *Sustainability* **2023**, *15*, 940. [CrossRef]
19. Dahalan, F.; Alias, N.; Shaharom, M. Gamification and Game Based Learning for Vocational Education and Training: A Systematic Literature Review. *Educ. Inf. Technol.* **2024**, *29*, 1279–1317. [CrossRef]
20. Papadakis, S.; Orfanakis, V. The Combined Use of Lego Mindstorms NXT and App Inventor for Teaching Novice Programmers. In *Proceedings of the Educational Robotics in the Makers Era*; Alimisis, D., Moro, M., Menegatti, E., Eds.; Springer: Cham, Switzerland, 2017; pp. 193–204.

21. Arís, N.; Orcos, L. Educational Robotics in the Stage of Secondary Education: Empirical Study on Motivation and STEM Skills. *Educ. Sci.* **2019**, *9*, 73. [CrossRef]
22. Tabet, N.; Gedawy, H.; Alshikhabobakr, H.; Razak, S. From Alice to Python. Introducing Text-Based Programming in Middle Schools. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '16, New York, NY, USA, 11–13 July 2016; pp. 124–129. [CrossRef]
23. Dwyer, H.; Hill, C.; Hansen, A.; Iveland, A.; Franklin, D.; Harlow, D. Fourth Grade Students Reading Block-Based Programs: Predictions, Visual Cues, and Affordances. In Proceedings of the Eleventh Annual International Conference on International Computing Education Research, ICER '15, New York, NY, USA, 9–13 July 2015; pp. 111–119. [CrossRef]
24. Papadakis, S.; Kalogiannakis, M.; Orfanakis, V.; Zaranis, N. The Appropriateness of Scratch and App Inventor as Educational Environments for Teaching Introductory Programming in Primary and Secondary Education. *Int. J. Web-Based Learn. Teach. Technol. (IJWLTT)* **2017**, *12*, 58–77. [CrossRef]
25. Papadakis, S.; Orfanakis, V. Comparing novice programing environments for use in secondary education: App Inventor for Android vs. Alice. *Int. J. Technol. Enhanc. Learn.* **2017**, *10*, 44–72. [CrossRef]
26. de Julián Posada, E.C.; García, M.S.Á. Análisis del uso de plataformas de aprendiza virtual en el proceso de aprendizaje en programación en C (Analysis of the use of virtual learning platforms in the C programming learning process). *Pist. Educ.* **2018**, *40*, 2123–2137.
27. Chen, C.; Haduong, P.; Brennan, K.; Sonnert, G.; Sadler, P. The effects of first programming language on college students' computing attitude and achievement: A comparison of graphical and textual languages. *Comput. Sci. Educ.* **2019**, *29*, 23–48. [CrossRef]
28. Yildiz Durak, H. The effects of using different tools in programming teaching of secondary school students on engagement, computational thinking and reflective thinking skills for problem solving. *Technol. Knowl. Learn.* **2020**, *25*, 179–195. [CrossRef]
29. Weintrop, D.; Wilensky, U. Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Trans. Comput. Educ.* **2017**, *18*, 3. [CrossRef]
30. Xu, Z.; Ritzhaupt, A.D.; Tian, F.; and, K.U. Block-based versus text-based programming environments on novice student learning outcomes: A meta-analysis study. *Comput. Sci. Educ.* **2019**, *29*, 177–204. [CrossRef]
31. Polman, J.L. The zone of proximal identity development in apprenticeship1 learning La zona de desarrollo próximo de la identidad en entornos de aprendizaje de oficios. *Rev. Educ.* **2010**, *353*, 129–155.
32. Bruner, J.S. *The Process of Education*; Harvard University Press: Cambridge, MA, USA, 2009.
33. Guan, L.; Yuan, N.; Xie, Y.; Feng, J.; Sun, B.; Li, L. A study on the application of flipped classroom teaching in higher vocational education. In Proceedings of the 2015 10th International Conference on Computer Science & Education (ICCSE), Cambridge, UK, 22–24 July 2015; pp. 819–823. [CrossRef]
34. Villalba, M.T.; Castilla, G.; Redondo, S. Factors with Influence on the Adoption of the Flipped Classroom Model in Technical and Vocational Education. *J. Inf. Technol. Educ. Res.* **2018**, *17*, 441–469. [CrossRef]
35. Lage, M.J.; Platt, G.J.; Treglia, M. Inverting the classroom: A gateway to creating an inclusive learning environment. *J. Econ. Educ.* **2000**, *31*, 30–43. [CrossRef]
36. Blumenfeld, P.C.; Soloway, E.; Marx, R.W.; Krajcik, J.S.; Guzdial, M.; Palincsar, A. Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning. *Educ. Psychol.* **1991**, *26*, 369–398. [CrossRef]
37. Thomas, J. *A Review of Research on Project-Based Learning*; The Autodesk: San Rafael, CA, USA, 2000.
38. Megayanti, T.; Busono, T.; Maknun, J. Project-based learning efficacy in vocational education: Literature review. In Proceedings of the IOP Conference Series: Materials Science and Engineering, Chennai, India, 16–17 September 2020; IOP Publishing Ltd.: Bristol, UK, 2020; Volume 830. [CrossRef]
39. Mills, J.E.; Treagust, D.F. Engineering education: Is problem-based or project-based learning the answer? *Australas. J. Eng. Educ.* **2003**, 1–16.
40. Jalinus, N.; Nabawi, R.A.; Mardin, A. The Seven Steps of Project Based Learning Model to Enhance Productive Competences of Vocational Students. In Proceedings of the International Conference on Technology and Vocational Teachers (ICTVT 2017), Yogyakarta, Indonesia, 28 September 2017; Atlantis Press: Dordrecht, The Netherlands, 2017; pp. 251–256. [CrossRef]
41. Chiang, C.L.; Lee, H. The Effect of Project-Based Learning on Learning Motivation and Problem-Solving Ability of Vocational High School Students. *Int. J. Inf. Educ. Technol.* **2016**, *6*, 709–712. [CrossRef]
42. Martín de Soto Domínguez, J. Formación Profesional Básica: La etapa educativa al servicio de la equidad. *J. Parents Teach.* **2021**, *385*, 26–31. [CrossRef]
43. Cross, N. *Design Thinking: Understanding How Designers Think and Work*; Bloomsbury Publishing: London, UK, 2023.
44. Guaman-Quintanilla, S.; Everaert, P.; Chiluiza, K.; Valcke, M. Impact of design thinking in higher education: A multi-actor perspective on problem solving and creativity. *Int. J. Technol. Des. Educ.* **2023**, *33*, 217–240. [CrossRef]
45. Calavia, M.B.; Blanco, T.; Casas, R.; Dieste, B. Making design thinking for education sustainable: Training preservice teachers to address practice challenges. *Think. Ski. Creat.* **2023**, *47*, 101199. [CrossRef]

46. Latorre-Cosculluela, C.; Vázquez-Toledo, S.; Rodríguez-Martínez, A.; Liesa-Orús, M. Design Thinking: Creatividad y pensamiento crítico en la universidad. *Rev. Electrónica De Investig. Educ.* **2020**, *22*, e28. [CrossRef]
47. Panke, S. Design Thinking in Education: Perspectives, Opportunities and Challenges. *Open Educ. Stud.* **2019**, *1*, 281–306. [CrossRef]
48. Johnson, D.W.; Johnson, R.T. *Learning Together and Alone: Cooperative, Competitive, and Individualistic Learning*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1987.
49. Kholifah, N.; Sudira, P.; Rachmadtullah, R.; Nurtanto, M.; Suyitno, S. The effectiveness of using blended learning models against vocational education student learning motivation. *Int. J. Adv. Trends Comput. Sci. Eng.* **2020**, *9*, 7964–7968. [CrossRef]
50. Bacca, J.; Baldiris, S.; Fabregat, R.; Kinshuk; Graf, S. Mobile Augmented Reality in Vocational Education and Training. *Procedia Comput. Sci.* **2015**, *75*, 49–58. [CrossRef]
51. Bacca Acosta, J.L.; Baldiris Navarro, S.M.; Fabregat Gesa, R.; Kinshuk, K. Framework for designing motivational augmented reality applications in vocational education and training. *Australas. J. Educ. Technol.* **2019**, *35*, 16. [CrossRef]
52. Angraini, L.M.; Yolanda, F.; Muhammad, I. Augmented reality: The improvement of computational thinking based on students' initial mathematical ability. *Int. J. Instr.* **2023**, *16*, 1033–1054. [CrossRef]
53. Brown, A.L.; Campione, J.C. *Psychological Theory and the Design of Innovative Learning Environments: On Procedures, Principles, and Systems*; Lawrence Erlbaum Associates, Inc.: Hillsdale, NJ, USA, 1996.
54. López, H.L.L.; Rodríguez, L.G.Q.; Mora, E.A.H.; Valenzuela, A.C.C. Paradigma de la programación orientada a objectos (POO). *Rev. Digit. De Tecnol. Inform. Y Sist.* **2024**, *8*, 147–153.
55. Weintrop, D.; Wilensky, U. Bringing blocks-based programming into high school computer science classrooms. In Proceedings of the Annual Meeting of the American Educational Research Association (AERA), Washington, DC, USA, 8–12 April 2016.
56. Perez, R.; Galan, A.; Quintanal, J. *Research Methods and Designs in Education*; Universidad Nacional de Educación a Distancia (UNED): Madrid, Spain, 2012.
57. Bhattacharya, K. *Fundamentals of Qualitative Research: A Practical Guide*; Routledge: Abingdon, UK, 2017.
58. Selltiz, C.; Jahoda, M.; Deutsch, M.; Cook, S. *Research Methods in Social Relations*; Holt, Rinehart and Winston: New York, NY, USA, 1959.
59. Yazan, B. Three approaches to case study methods in education: Yin, Merriam, and Stake. *Qual. Rep.* **2015**, *20*, 134–152. [CrossRef]
60. Mertens, D. *Research and Evaluation in Education and Psychology: Integrating Diversity with Quantitative, Qualitative, and Mixed Methods*; SAGE Publications: London, UK, 2010.
61. Savin-Baden, M.; Major, C. *Qualitative Research: The Essential Guide to Theory and Practice*; Routledge: Abingdon, UK, 2013.
62. Weitzman, E.; Miles, M. *Computer Programs for Qualitative Data Analysis*; SAGE Publications: London, UK, 1995.
63. Silver, C.; Lewins, A. U*sing Software in Qualitative Research: A Step-by-Step Guide*; SAGE Publications: London, UK, 2014. [CrossRef]
64. Sampieri, H. *Methodology of the Research*; McGrawHill: New York, NY, USA, 2014.
65. Rizzo, M.L. *Statistical Computing with R*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2019.
66. Hollander, M.; Wolfe, D.A.; Chicken, E. *Nonparametric Statistical Methods*; John Wiley & Sons: Hoboken, NJ, USA, 2013.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

computers

MDPI

Article

Measuring Undergraduates' Motivation Levels When Learning to Program in Virtual Worlds

Juan Gabriel López Solórzano [1], Christian Jonathan Ángel Rueda [2] and Osslan Osiris Vergara Villegas [3,*]

[1] Facultad de Contaduría Pública y Administración, Universidad Autónoma de Nuevo León, San Nicolás de los Garza 66455, Mexico; juan.lopezsr@uanl.edu.mx
[2] Research, Technological Development and Postgraduate, Universidad Politécnica de Santa Rosa Jáuregui (UPSRJ), Santiago de Querétaro 76220, Mexico; cangel@upsrj.edu.mx
[3] Industrial and Manufacturing Engineering Department, Universidad Autónoma de Ciudad Juárez, Ciudad Juárez 32310, Mexico
* Correspondence: overgara@uacj.mx

Abstract: Teaching/learning programming is complex, and conventional classes often fail to arouse students' motivation in this discipline. Therefore, teachers should look for alternative methods for teaching programming. Information and communication technologies (ICTs) can be a valuable alternative, especially virtual worlds. This study measures the students' motivation level when using virtual worlds to learn introductory programming skills. Moreover, a comparison is conducted regarding their motivation levels when students learn in a traditional teaching setting. In this study, first-semester university students participated in a pedagogical experiment regarding the learning of the programming subject employing virtual worlds. A pre-test-post-test design was carried out. In the pre-test, 102 students participated, and the motivation level when a professor taught in a traditional modality was measured. Then, a post-test was applied to 60 students learning in virtual worlds. With this research, we have found that the activity conducted with virtual worlds presents higher motivation levels than traditional learning with the teacher. Moreover, regarding gender, women present higher confidence than men. We recommend that teachers try this innovation with their students based on our findings. However, teachers must design a didactic model to integrate virtual worlds into daily teaching activities.

Keywords: motivation;virtual worlds; programming; teaching–learning process; Minecraft education edition

1. Introduction

Students generally enroll onto the Introduction to Programming (IP) course in the first college semester because it is imperative in a curriculum of technology-related careers. For undergraduates, IP represents a fundamental pillar throughout their careers since what is learned will be applied in later subjects such as advanced programming. Programming is a complex subject for many students, with high failure and dropout rates. In Mexico, there are high failure rates of more than 70% in most universities. Moreover, the students who pass frequently do not obtain high grades, which coincides with what Groher et al. [1] explained.

IP students face several difficulties that cause their low performance. The work by Qian and Lehman [2] classifies the difficulties into three categories: (i) syntactic, (ii) conceptual, and (iii) strategic knowledge. Syntactic knowledge refers to the programming language rules, including poorly balanced parentheses, forgetting the semicolon at the end of an instruction, or using undeclared or uninitialized variables. Conceptual knowledge is associated with understanding the fundamental constructs of programming and how the computer works as a means to program. Examples of difficulties are not understanding variables, assignments, conditions, cycles, and sequential execution of code. Finally,

strategic knowledge is related to applying syntactic and conceptual knowledge in problem solving. Difficulties in this category include the student not being clear about the appropriate cycle for a particular problem or not knowing how to read or debug the code. Another fundamental aspect that must be considered is motivation, because there is a close relationship between motivation and academic performance [3,4]. Specifically, demotivation is associated with the poor performance of students in educational settings [5].

Researchers and teachers continuously search for new methods to motivate and awaken interest in programming. For example, thanks to the advancement of Information and Communication Technologies (ICTs), 3D environments are frequently used in educational settings. Three-dimensional environments are interactive and immersive and can promote student motivation in teaching–learning processes because they use narrative ludic immersion [6]. Immersive digital environments include Augmented Reality (AR), Virtual Reality (VR), and Virtual Worlds (VWs). Due to the complexity of these three technologies, this research uses the Three-Dimensional Immersive Digital Environments (TDIDE) macro-concept proposed by [7] as a theoretical foundation. Particularly, the TDIDE employed in this research is the virtual world because it represents an educational innovation that, when properly implemented by the teacher, can significantly increase the motivation and engagement of students in the learning process [8–10]. Moreover, game elements can be used when designing VWs. The study of Pellas et al. used the "Learning Mechanics-Game Mechanics" model [11] to identify the most used game mechanics in the game prototypes designed in VWs. The game mechanics allow students to keep attention, participation, and engagement. The most used game mechanics were story, realism, and roleplay [12]. On the other hand, Pellas and Mistakidis identified many studies in which the game system, the narrative, and aesthetics were applied [13].

Currently, intending to make programming more accessible and help students overcome programming challenges, teachers and researchers have been working on two main aspects: (i) pedagogical approaches and (ii) tools. Pedagogical approaches include flipped classrooms, collaborative learning, feedback, game-based learning, metacognition, and problem-solving. The teachers can combine the approaches without losing their individual effectiveness [14]. On the other hand, tools include visual, automated assessment, programming environments, support tools, microworlds, and plagiarism detection [15,16]. Modern visual programming tools are widely used, and Scratch is especially effective [14,17]. It is important to note that developing tools requires effort and time [18], and many of them are not used outside the institution where they were developed [15]. This coincides with what was found in a study on three-dimensional immersive digital environments applied to teaching programming [19]. Therefore, to facilitate the incorporation of technological tools in the teaching–learning process, the proposal is not to develop tools but instead use the existing virtual worlds that are ready to be used.

In the literature, there are records since 24 years ago about the use of VWs for teaching programming [20,21]. The main reference in educational environments is "Second Life" [22], which is mainly used by people averaging 48 years of age [23]. Second life would be the benchmark for first-generation VWs that failed to maintain their popularity [24]. Consequently, an interesting research area is generated to learn more about the new generation of VWs, such as Minecraft and Roblox, that currently enjoy popularity and can also be used in learning environments [25]. Also, it is essential to understand that today's students grew up using technologies such as the Internet, video games, and computers [26].

As far as we know, there are no clues in the literature about works related to teaching/learning with virtual worlds of functions in programming courses. Therefore, the novelty of this work lies in the use of MEE and its corresponding elements to design a didactic activity for learning functions in an introductory programming course.

This study aims to measure the motivation level of students who used virtual worlds for learning functions in IP and compare it with the motivation level regarding traditional classroom learning. The main contributions of this paper are as follows:

- We explain the virtual world's design and the activities to help students learn functions in Minecraft Education Edition (MEE).
- We analyze and assess the motivation level of Mexican undergraduates using the virtual world MEE.
- We present evidence that a didactic model based on virtual worlds for learning programming is suitable for increasing students' motivation.

The rest of the paper is organized as follows. Section 2 discusses related studies on using virtual worlds, particularly MEE, for learning IP. Section 3 explains the research design, the phases that comprise the practice of MEE, and the instruments applied. The results are shown in Section 4. Finally, Section 5 presents the conclusions.

2. Literature Review

Many research studies have been published regarding teaching/learning programming difficulties in educational settings (on-site and online) [14,27,28]. Particularly, we analyzed the papers related to teaching/learning IP in virtual worlds and papers regarding the assessment of students' motivation.

Esteves et al. used Second Life for teaching–learning programming following an action research approach. The students were asked to develop a project within the VW using Linden Scripting Language (LSL). The main outcome of the research is a practical framework consisting of the following elements: (i) communication, (ii) project, (iii) methodology, (iv) classroom, and (v) lessons. The author's recommendations when using VWs include using private and public communication channels for explanations, visual projects, and project-based learning. The results showed that students were enthusiastic and motivated when using LSL [20].

Rico et al. developed a platform called V-LeaF to teach the IP course. The proposed approach used OpenSim as a virtual world engine. Secondary-level students and teachers used LSL as the programming language. As general findings, the study reports that teachers and students indicate that using VWs is a satisfactory experience, as it allows collaboration and cooperation [29].

Hulsey et al. presented Curiosity Grid, a virtual world supported by OpenSim. The VW was used to introduce programming concepts to high school girls in a summer course. The students solved ten design, construction, and scripting challenges using LSL. Pre-camp and post-camp surveys were administered to evaluate the effectiveness of the proposal. The results showed a positive increase in the students' attitudes towards programming [30].

Kao presented JavaStrike, a Java language runtime environment developed in Unity. JavaStrike supports the visualization, execution, and debugging of Java language code. The authors also presented CodeBreakers, a game incorporating JavaStrike as a code execution engine. CodeBreakers can be used to teach programming at basic, intermediate, and advanced levels. Twenty-seven participants evaluated the platform positively in terms of competence, autonomy, immersion, and controls. The participants expressed a need to incorporate more instructions or support for beginner Java programmers [31].

Kutay and Oner used Minecraft to teach programming to 20 high school students. The design corresponds to a pre-test and post-test to investigate the changes in the knowledge of computational concepts. For the pre-test, the knowledge was assessed using Scratch, and the post-test using Minecraft. The results indicate that students performed better when tested on computational concepts after using Minecraft [32].

Debugging and testing were the practices that students employed the most in Minecraft activities because they are excellent visual feedback mechanisms. The student can see the results of their code in real-time within the virtual world. However, in the study conducted with high school students, no significant differences regarding motivation were obtained when using Minecraft [33].

A teaching method to facilitate and motivate learning introductory programming using Minecraft was proposed in [34]. The proposal comprises three abstraction levels: (i) designing with building blocks, (ii) building with code blocks, and (iii) general pro-

gramming and computational thinking. The study involved 16 university students with little or no programming knowledge. The results indicate that the students consider the proposed abstraction levels adequate and facilitate our understanding of programming concepts. Unfortunately, students mentioned that the Minecraft coordinate system is complicated.

In Bile's study, MEE was used for a programming course involving 189 students aged 8–10. The didactic method allowed the students to concretize their knowledge through practical experience. The positive results show that learning through MEE promotes the fast and solid learning of complex concepts [35].

Sajjanhar and Faulkner used the "Second Life" virtual world to teach programming using a constructivist approach. The study involved 12 undergraduate and master's students. Students used Scratch for Second Life to perform activities on variables, cycles, and conditional structures. The findings indicated that the proposed approach effectively teaches basic programming concepts [36].

The research by [37] conducted a Minecraft Hour of Code study involving 63 students aged 10–12. Hour of Code consists of tutorials designed to introduce basic programming skills. The results indicate that Hour of Code helped students understand the concept of programming and fostered a positive disposition towards programming.

Alsaadi et al. used an educational tool called "Creative Programmer", a Minecraft "mod" for teaching basic programming concepts. The tool was used by seven children aged 7–15 years. Three activities were performed: defining variable data types, writing commands to create an algorithm, and understanding the concept of repetition (cycles). The results revealed that the children improved their performance regardless of whether they had previous programming knowledge [38].

Teaching programming with VW encompasses two paradigms: (i) the declarative and (ii) the imperative. The declarative includes logic programming, and the imperative includes procedural programming. Vosinakis et al. presented MeLoISE for teaching–learning programming in Prolog. The authors conducted two studies to evaluate performance, collaboration, and user experience. The results revealed satisfactory student performance, positive collaboration, and good user experience [39].

2.1. Assessing Motivation in Introduction to Programming

Motivation is a complex process and represents a state of energy that manifests itself in goal-directed behaviors [40]. The attention, relevance, confidence, and satisfaction (ARCS) model was chosen because it has been used worldwide and at different educational levels [4], and also because it is based on the expectancy–value theory, which is one of the contemporary motivation theories to learn [41]. The expectancy–value theory explains that people are motivated to perform an activity if two conditions are met: (i) If they perceive that the activity has utility or personal value and (ii) If they perceive themselves capable of performing the activity successfully.

According to Keller, for a person to be motivated and stay motivated, four conditions must be met: (i) attention, (ii) relevance, (iii) confidence, and (iv) satisfaction [42].

- Attention. It is the first condition for motivation and also for learning. The main challenge is to catch the student's attention and keep it.
- Relevance. It is related to whether the course is relevant to the students and meets their needs.
- Confidence. The expectation of success in a particular task is important, as it influences the student's persistence and achievement.
- Satisfaction. When any activity is performed successfully, it is necessary for the person to feel good about it.

The study by Fang et al. analyzed the applications of the ARCS model. A total of 55 studies were part of their analysis. The most important findings indicate that ARCS is most used at the higher level. Most studies (n = 47) applied learner-centered environments, including AR, VR, and VWs. The most commonly used pedagogical approach was

multimedia learning [3]. According to Garzón et al., in their classification of pedagogical approaches, multimedia learning is defined as learning that allows students to construct their knowledge when they have access to other media besides the text. Multimedia learning was also used for the present study since MEE is a constructivist educational tool that provides multiple media such as text, image, sound, 3D models, chat, and NPCs. In addition, multimedia learning includes game-based learning [43].

Some examples of using the ARCS model in teaching programming are the following.

In the study by Bakar et al., Turtle Graphics was employed to maintain high motivation levels when teaching the basics of Java programming. A total of 167 higher-level students participated in the study. The ARCS model was used to assess the students' motivation. The results revealed that the students demonstrated high motivation in all four motivational aspects.

The ARCS model was also employed to assess motivation when using teaching methodologies. Choi et al. present the effects of using the flipped classroom in a block-based programming course. The results revealed that the experimental group that participated in the flipped classroom obtained higher motivation levels than students in the control group that employed traditional teaching techniques [44].

2.2. Pedagogical Foundations of Virtual Worlds

According to Girvan and Savage, VWs have several features such as (i) client–server environment; (ii) construction tools; (iii) programming tools; (iv) communication tools; (v) 3D environment; (vi) avatars, and (vii) multiple users. Teachers and researchers can combine the features in various ways and apply them to the educational environment. However, it is important to ground learning experiences developed in VWs pedagogically. Currently, immersive experiences using VWs are usually based on constructivism (or some of its variants, e.g., social constructivism [36], communal constructivism [45], or constructionism [46,47]). The present research is grounded in constructionism, whose fundamental principle is the creation of meaningful and shareable objects that, during the process, students actively explore, test, and improve their understanding [48,49].

Students learn through active participation, collaboration, problem-solving, and exploration, i.e., "learning by doing". Due to the generally constructionist approach of VWs, the common practices are (i) problem-based learning, (ii) collaborative learning, and (iii) game-based learning [50,51]. VWs, compared to traditional 2D platforms, include technological capabilities such as a sense of presence, different types of communication, avatars, expressiveness, and real-time simulation [52]. The capabilities allow students to experience-based learning [50].

2.3. Gender in E-Learning and VW

Studies regarding the gender variable indicate no significant differences in learning outcomes concerning e-learning [53–55]. However, in the VWs, the opposite happens. Women have more social activity, use learning opportunities and resources, participate in construction activities, and appreciate the entertainment value of VWs [56]. Therefore, it is advisable to study gender. According to Lin et al., learners' gender should be considered in the design of VWs since women use and perceive the systems differently than men [57]. For example, in the work of Korlat et al., women value e-learning, the support of their teachers, and the intrinsic value and engagement in their learning more than men [58].

3. Materials and Methods

After analyzing the research problem to be solved, the following two research questions were posed.

- RQ1. Could IP learners using MEE achieve higher motivation levels than traditional learning?
- RQ2. Could men obtain higher motivation levels than women when learning IP using MEE?

The approach of this study is quantitative with an exploratory and descriptive scope. The purpose is to determine whether there is a significant difference in the motivation levels of students who use virtual worlds compared to traditional learning.

ARCS model was chosen because it has been used worldwide and at different educational levels [4], and also because it is based on the expectancy–value theory, which is one of the contemporary motivation theories to learn [41].

3.1. Instruments

Keller developed the Instructional Materials Motivation Survey (IMMS) based on the ARCS model to measure motivation [59]. IMMS is composed of 36 items on a 5-point Likert scale. IMMS can be used to calculate the overall motivation level or for each ARCS model subscale. There is evidence in the recent literature that different adaptations of IMMS have been used [3]. For example, Loorbach et al. evaluated the instrument's theoretical and statistical aspects. Using structural equation modeling, they showed that the 36 items of the original IMMS could be reduced to 12. Loorbach et al. called it the Reduced Instructional Materials Motivation Survey (RIMMS) [60]. RIMMS is based on the ARCS model, which includes attention, relevance, confidence, and satisfaction. The instrument contains 12 items and uses a Likert-type scale that ranges from 1 (not true) to 5 (very true). RIMMS is valid and reliable since it reflects the conditions and principles of the ARCS model [60] and has been applied in teaching using augmented reality [61]. Therefore, the present study used RIMMS to measure student motivation.

RIMMS was used in the pre-test to measure motivation in the traditional modality that corresponds to face-to-face classes taught by the teacher. Additionally, RIMMS was employed in the post-test when students experimented with virtual worlds.

The proposed method comprises five stages: (i) analysis of virtual worlds; (ii) thematic content; (iii) virtual world design; (iv) materials design; and (v) activity implementation.

3.2. Analysis of Virtual Worlds

In the first stage, we analyzed the technological tools for teaching programming. The literature has many virtual worlds with different characteristics and computational requirements. However, due to the costs, we decided to analyze those that do not require a gamer-type computer, that is, those that do not need a dedicated Graphics Processing Unit (GPU) to conduct fluid execution.

Therefore, the virtual worlds considered were Roblox, Minecraft Education Edition (MEE), Mozilla Hubs, OpenSim, and Decentraland. After the analysis, MEE was selected because it is a VW designed for educational purposes and based on constructionist theory [62]. MEE integrates Scratch and supports programming languages, such as Javascript and Python, which are widely used in software development. Minecraft is similar to microworld because it is an incubator of knowledge [48]. Novel programmers can use Minecraft to generate, experiment, debug, collaborate, and share the products created through code. MEE offers the possibility of concretizing abstract concepts and facilitating understanding by incorporating visual elements [63]. Minecraft is special because it is part of the most recent 3D interactive platforms, such as Roblox and Fortnite, and has become the model for the next development of the Metaverse [64]. Minecraft, unlike Roblox and Fortnite, has a version for educational environments. In addition, MEE is suitable due to its accessibility and availability compared to VR, which needs specialized equipment that cannot be easily acquired [65].

3.3. Thematic Content

When students learn to program, they face several challenges, including creating conditions, loops, and functions. However, when the conditions and cycles are mastered, learning to break a program into more manageable pieces called functions is essential. Therefore, the thematic content chosen for the activity of this paper corresponds to the topic of making "functions" in Python. In introductory programming, students must understand

functions and know how to use or create them. Functions are self-contained modules of code that accomplish a specific task and can be reused in any part of a program.

In MEE, the students can use predefined functions to manipulate the virtual world, and more importantly, new functions can be created. The activity requested from students is similar to the study conducted by [66], where the Logo language was used to control a turtle and draw different figures. In MEE, an "agent" helps students perform construction tasks within the world using programming instructions. An example of a MEE agent (robot) is shown in Figure 1.



Figure 1. Example of an agent in MEE.

Some of the functions that can be performed by the agent and used in practice with MEE are summarized in Table 1.

Table 1. Functions to control the agent.

Function	Description
agent.move(FORWARD,1)	Moves the agent one block forward
agent.move(BACK,1)	Moves the agent one block backward
agent.move(LEFT,1)	Moves the agent one block to the left
agent.move(RIGHT,1)	Moves the agent one block to the right
agent.move(UP,1)	Moves the agent one block up
agent.move(DOWN,1)	Moves the agent one block down
agent.turn_left()	Turn the agent to the left
agent.turn_right()	Turn the agent to the right

3.4. Virtual World Design

Virtual worlds in higher education have mainly been applied for virtual classrooms, discussions, simulations, games, and field trips [6,67]. We selected the virtual classroom's didactic strategy for the virtual world's design. The strategy consists of replicating or simulating a classroom or space where the teaching–learning occurs. Elements incorporated in MEE, such as blackboards, boards, and agents, were used to present information to the students, as shown in Figure 2. Although MEE provides the teacher with many pre-designed lessons created by other teachers or Minecraft administrators, many of these activities target students aged 8–10 and 11–13 [68]. Therefore, the virtual world was designed with university-level students in mind and uses textual programming, commonly employed in programming courses.

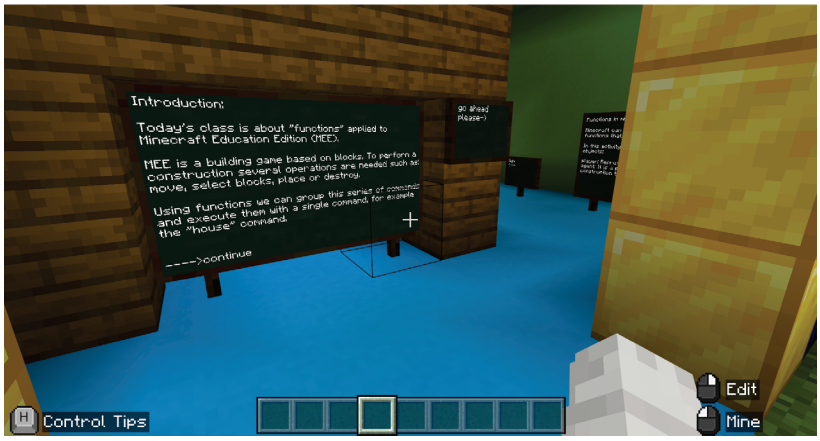


Figure 2. Elements used in MEE for learning functions.

The overall design of the virtual world can be seen in Figure 3. The design consists of a main structure with an entrance, non-player characters (NPCs), and whiteboards with directions that students must follow during the activity.



Figure 3. View of the space built in MEE.

In the virtual world, the student can make changes using commands in Python language. Moreover, the student can visualize the command execution or programming activities in real-time. Figure 4 shows the MEE command editor with an example of Python code.



Figure 4. The code editor in MEE.

Generally, the experience consisted of presenting the student with information regarding the use of functions within MEE. Students must understand that functions are used to allow the execution of tasks within the virtual world. During the learning process, the student was asked to perform a series of programming activities described in Table 2.

Table 2. Activities requested from students in MEE.

Activity	Description
A1. Use Microsoft Make Code	Write code that will be executed within the virtual world to move the "agent" to a specific location.
A2. Create a function to be executed in the chat	Use the commands in Table 1 to create a function to move the agent in the virtual world. The function's execution should be done on demand when the function name is typed in the chat.
A3. Final activity	Define a function for the agent to follow a route marked in yellow. The start and end of the route are marked in red and green, respectively, as observed in Figure 5.
A4. Bonus activity	Place blocks on the marked area (see Figure 6), with a minimum height of one block. The student was asked to use conditional structures and cycles. Additionally, the design of a function that allows the agent to place a block as it moves through the virtual world was requested.



Figure 5. The final activity in MEE.



Figure 6. The bonus activity in MEE.

3.5. Material Design

In this stage, PowerPoint presentations and PDF files were generated with information on how the student could move around in MEE using a desktop computer or mobile device. The documents also provided information on MEE concepts and some commands used within the virtual world. An example of this information is shown in Figure 7.

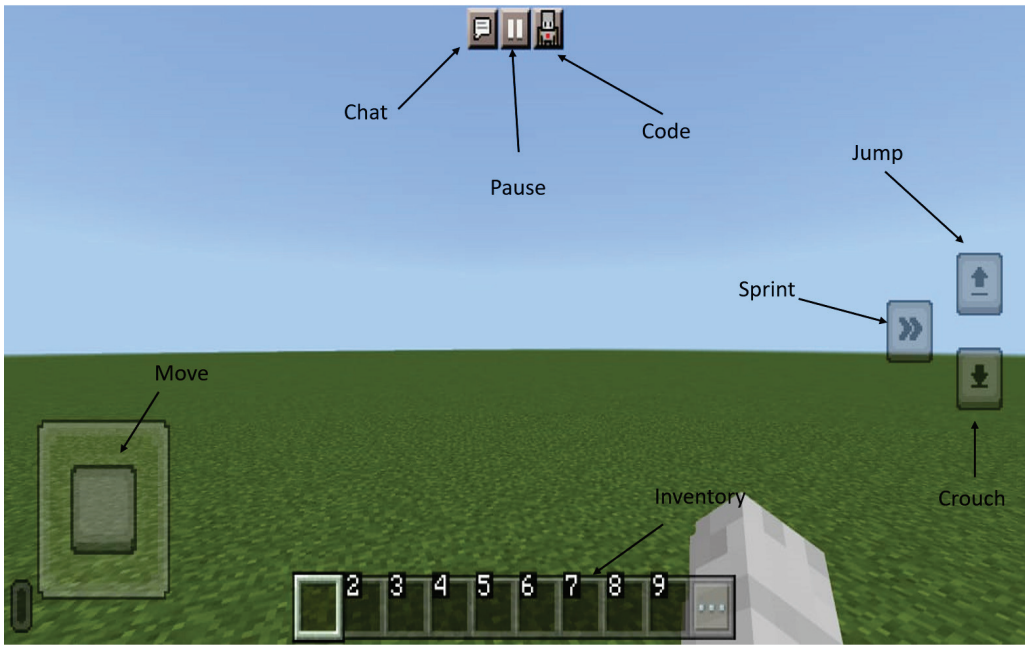


Figure 7. MEE controls for mobile devices. The inventory uses a number for each slot.

Design of Didactic Materials in MEE

Figure 8 shows the phases for designing the didactic material used in the VW. Each phase is explained below.

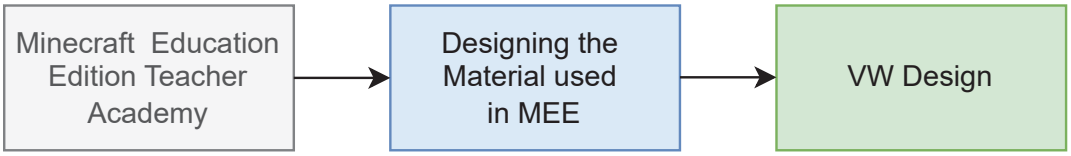


Figure 8. Phases for designing the didactic material used in MEE.

- Phase 1—Minecraft Education Edition Teacher Academy: A face-to-face and online Minecraft Education Edition Teacher Academy certification was conducted. The certification comprises three sections: (i) beginner, (ii) intermediate, and (iii) advanced. In the beginner section, the designer starts playing in MEE, placing blocks, and managing the materials inventory. In the intermediate, the classroom mode is used. Topics include setting up the classroom and managing NPCs, whiteboards, and posters. Finally, the advanced section focuses on programming with Microsoft Make Code.
- Phase 2—Designing the Material used in MEE: All the information concerning functions and the description of each activity to be performed were stored in a plain text file for later use in the MEE. The information was copied and pasted into the VW in the design phase.
- Phase 3—VW Design: The phase started with a paper design of the general structure of the VW. Subsequently, construction proceeded using the following resources provided by MEE:
    - Board (3 × 3 blocks). Used to present information on functions and instructions.
    - Poster (2 × 1 blocks). Used to provide information on activities.
    - Slate (1 × 1 block). Used for signaling within the VW.
    - NPC. Used for presenting information interactively.
    - Agent. A programmable element that students use to execute commands within the VW.

- Miscellaneous blocks. Blocks such as concrete, glass, and lights were used to construct the building.

3.6. Activity Implementation

Figure 9 shows the activities conducted during the study. The workflow is explained in the following subsubsections.

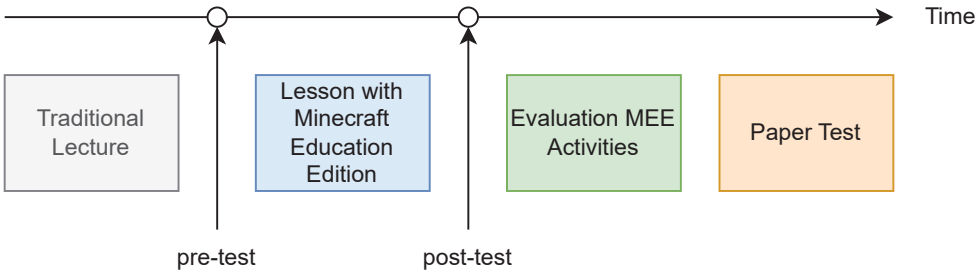


Figure 9. Workflow of the study.

3.6.1. Traditional Lecture

Before using the virtual world in MEE, the teacher taught students the topic of programming functions in the traditional modality.

The traditional approach is teacher-centered, dominated by lectures, in-class exercise resolution, and homework [69,70]. Topics covered included variables, operators, control structures, and functions. After the face-to-face classes, students were asked to answer the instrument to evaluate their motivation level (pre-test). It is essential to mention that, when students answered the survey, they did not know that the activity would be performed by employing MEE in the future. This was decided to avoid bias.

3.6.2. Lesson with Minecraft Education Edition

Before starting the practice, the students were asked to install MEE on their computer equipment or mobile devices because the institution does not have the infrastructure to carry out the activity. Some students could not conduct the activity due to several inconveniences, such as (i) their equipment did not meet the minimum requirements for installing the tool, and (ii) in some devices, the code execution (essential for the activity with MEE) did not work. After the installation, to familiarize the students with the game, they were instructed to log in to MEE with their institutional account and play in the virtual world called "Movement". The movement virtual world is located in the Templates section of MEE and was designed to be experienced using a keyboard or touch device.

The immersive activity was conducted online. Therefore, a session in Microsoft Teams was scheduled to conduct the study. During the study, one professor was connected to respond to the students' queries. Microsoft Teams Chat was used to send evidence of the activities performed.

During the session, the teacher shared the file Activity_MEE_Intro_Prog.mcworld with the group and provided the necessary indications to import the virtual world. Once the world was correctly imported, the students could perform activities related to the "functions" topic.

At the end of the MEE activity, the students sent evidence of the activities through a private Microsoft Teams chat. In addition, students were asked to fill out the survey to measure their motivation level using MEE (post-test).

3.6.3. MEE Activities Evaluation and Paper Test

After the activity with MEE, the teacher graded the activities performed by the students. Finally, a paper test was administered.

4. Results

Two groups of the Introduction to Programming course of the August–December 2022 semester participated in the study. One hundred and two students participated in the traditional class, and only sixty students participated in the activity applying the virtual world. Forty-two students did not carry out the activity with MEE because they did not have the necessary computer equipment to install the tool or decided not to participate. The sample is non-probabilistic because it is convenient since the participants belong to assigned class groups. The paired sample of the experiment is composed of 60 students.

4.1. Results with RIMMS

The results of the RIMMS instrument were analyzed for the materials offered by the teacher and with MEE. The following subsections discuss the results for each dimension of the ARCS model.

4.1.1. Results for Attention

Figure 10 shows that many students agree that the content quality, the information organization, and the interaction used in MEE helped them maintain their attention. It was observed that students were more participative during the activity with MEE. Moreover, some students share evidence screenshots of the exercises they performed in the virtual world.

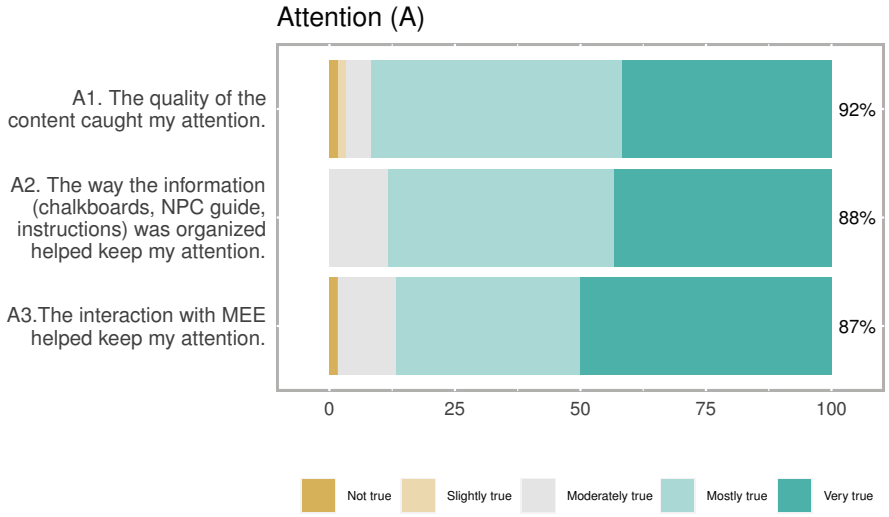


Figure 10. Attention in MEE.

The RIMMS results for the teacher's material are shown in Figure 11. It is essential to highlight that many students agree that the variety of readings, exercises, and illustrations employed by the teacher helped maintain their attention in class. During the lectures, the teacher indicates that the students paid attention to what was explained, but when asked to solve an exercise, it becomes evident that attention is not synonymous with learning or understanding the topic.

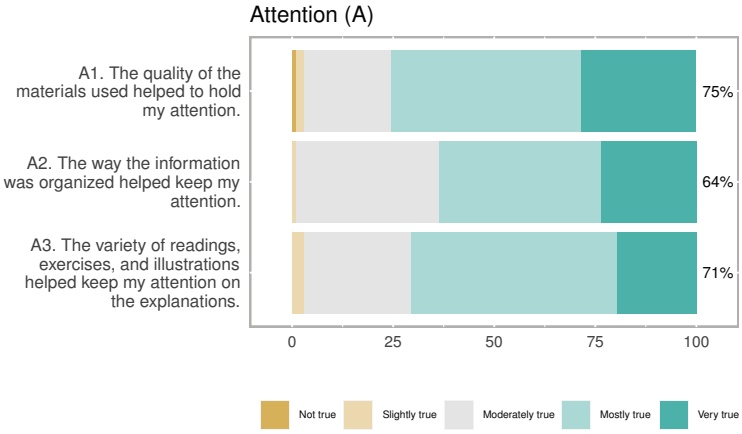


Figure 11. Attention teacher's material.

4.1.2. Results for Relevance

Using MEE in teaching programming promotes students' attention, indicating that the content shown in MEE is relevant (see Figure 12).

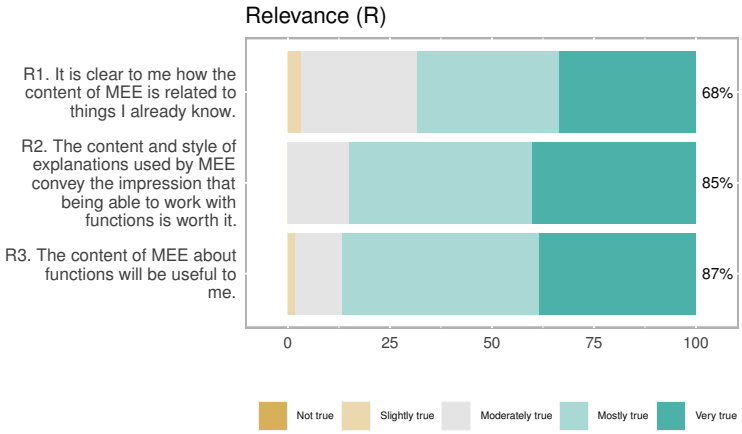


Figure 12. Relevance in MEE.

The results on the relevance of the lectures given by the professor are shown in Figure 13. Students agree that it would be relevant to go deeper into the topic and that employing the contents provided by the professor would be useful. In contrast, students disagree or are undecided that the teacher's material is related to topics they already knew.

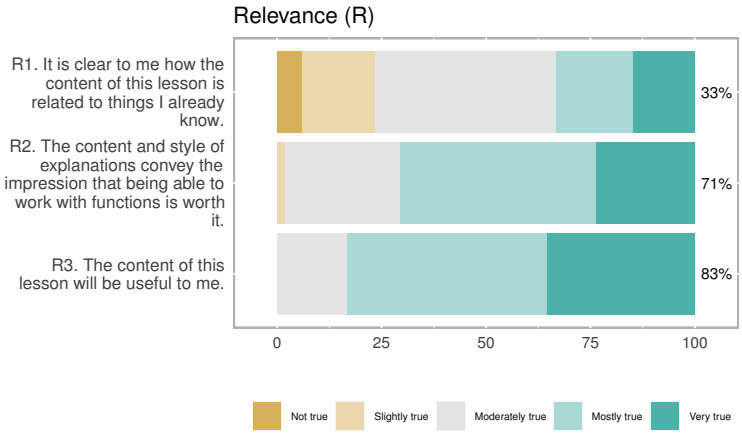


Figure 13. Relevance of teacher's material.

4.1.3. Results for Confidence

Figure 14 shows that students agree that they could learn the content presented in MEE and that this is due to the clarity of the concept explanations. On the other hand, students do not agree that they could pass a test on the topic.

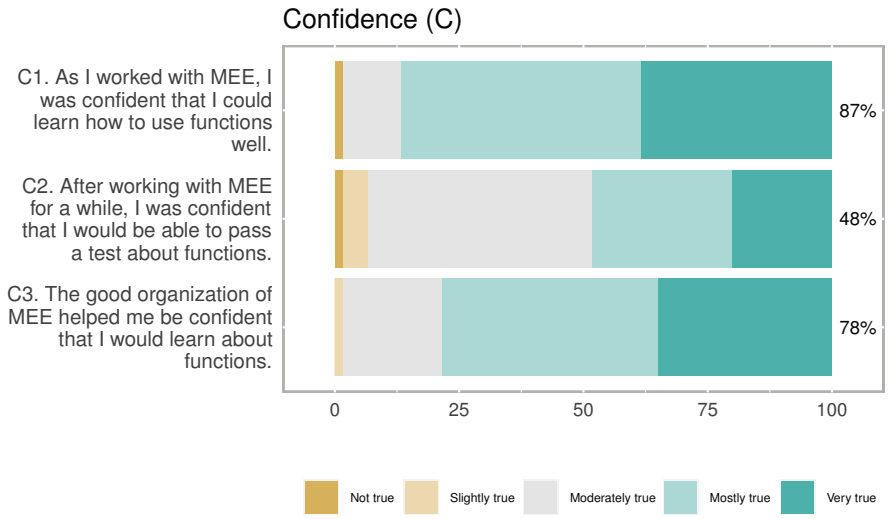


Figure 14. Confidence in MEE.

The results corresponding to the teacher's material can be seen in Figure 15. It is important to highlight that it does not matter whether the teacher's material is used; most students do not agree or are undecided about whether they will pass an exam on the topic. However, students are also confident that the clarity of the explanations and work in class would be conducive to learning and the correct application of the topics raised by the teacher.

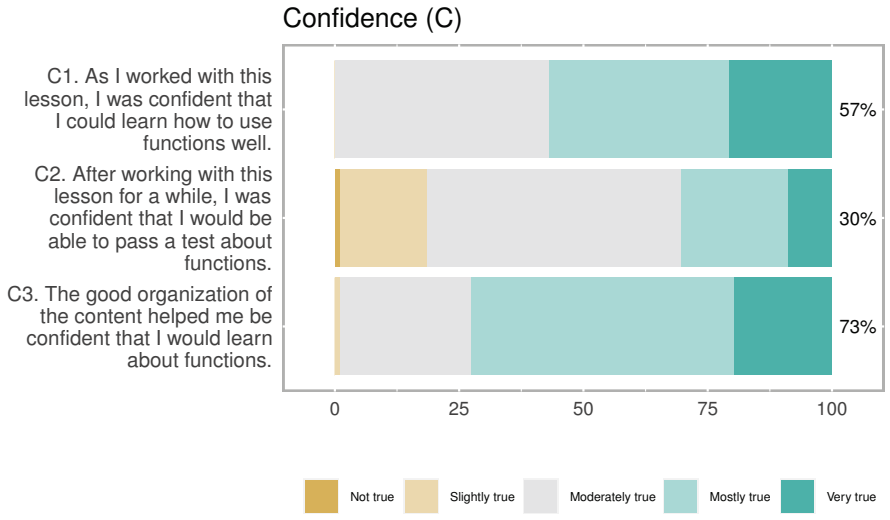


Figure 15. Confidence in the teacher's material.

4.1.4. Results for Satisfaction

Finally, the Satisfaction dimension is presented. Most students experimented with high satisfaction rates using MEE (Figure 16) and the material offered by the teacher (Figure 17).

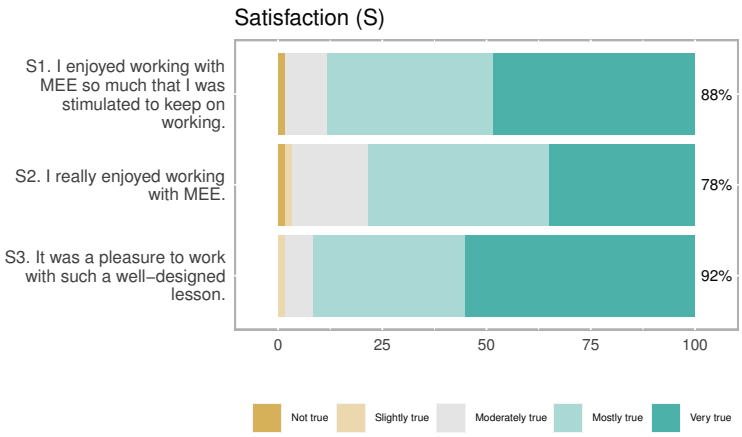


Figure 16. Satisfaction in MEE.

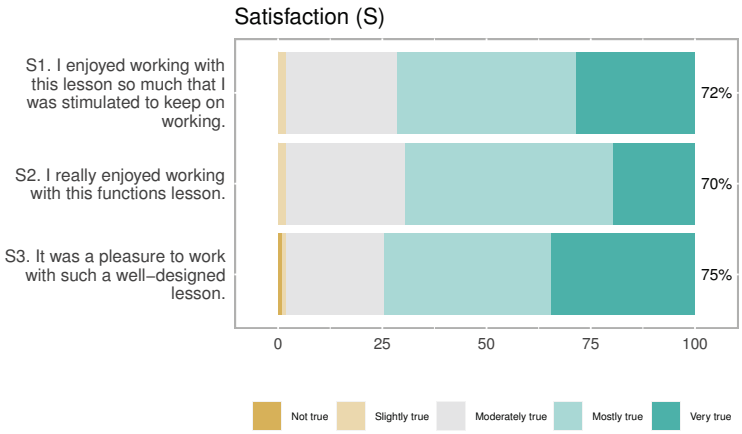


Figure 17. Satisfaction with the teacher's material.

The results were obtained by considering only a paired sample of 60 students, corresponding to the students participating in the MEE activity and the teacher's classes. From the total sample, 18 are women, and 42 are men. Women were observed to have higher motivation levels than men when using MEE. Figure 18 shows that all the factors (attention, relevance, confidence, and satisfaction) are higher for women.

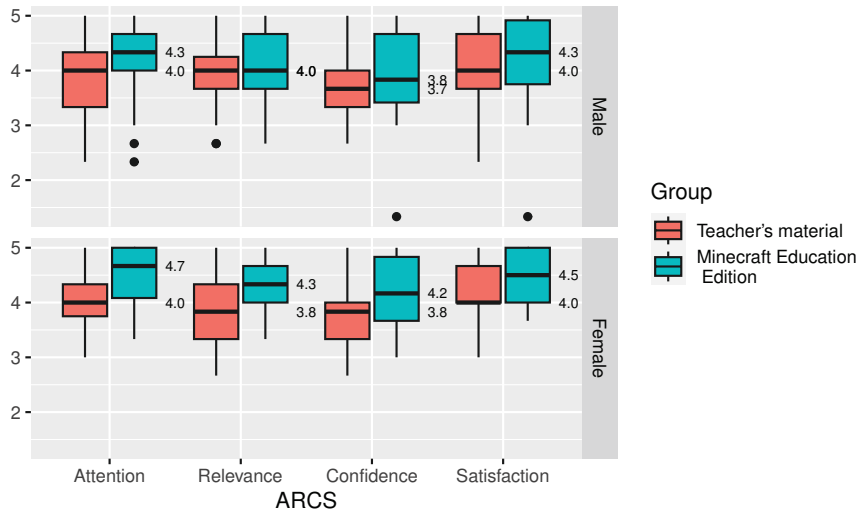


Figure 18. ARCS comparison by gender. The circles correspond to outliers.

Finally, considering the complete sample (n = 60), the results obtained in each factor show an increase in student motivation when using MEE, as seen in Figure 19. The figure shows the pre-test corresponding to the material or class taught by the teacher in their traditional lecture and the post-test in which the virtual world MEE was used.

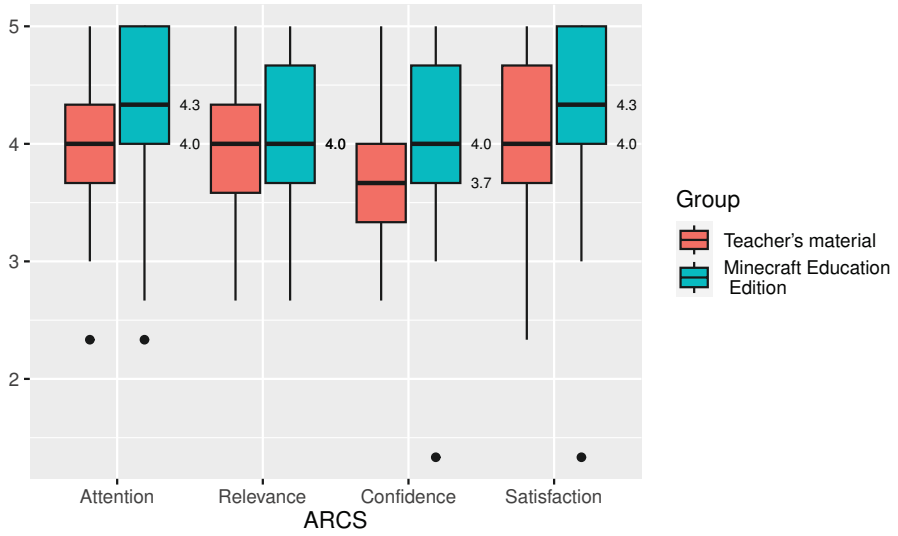


Figure 19. ARCS pre-test and post-test results. The circles correspond to outliers.

In addition to the results obtained with the RIMMS survey, it was possible to verify that the students were very committed and understood the material since, among the 60 students, 73% (n = 44) sent evidence of the exercises carried out within MEE. In contrast, in the traditional modality, when they were asked through an exam to solve a problem using functions with the Python language, only 22% of the students solved it correctly, and 78% (n = 47) did not do it correctly or did not even try to solve it, leaving the answer blank. On the other hand, some students commented that the activity was very fun. Some of the messages sent by the students were the following:

- "The activity was fun. I easily realized the last activity, called bonus. I will send the results as soon as possible."
- "Professor, I was able to do the activity. The instructions say to use an if, for, or while. I only used an if and a while. I could not modify more things in the game because I can not use other blocks, but it works."

4.2. Evaluation in MEE and Test

The students' activities performed in MEE were sent to the teacher for evaluation. In addition, the paper-based test was reviewed. The test contained an exercise about functions. The results are shown in Table 3.

Table 3. Results of MEE and test.

Evaluation	M	SD	Pass	Fail
MEE evaluation	59.33	36.81	43	17
Paper test	31.08	36.40	13	47

As can be observed in Table 3, the mean for MEE was 59.33. The low mean is because the 17 failed students were graded with zero since they did not send their evidence to the professor for evaluation.

4.3. T-Student Test Results

The normality test indicated that the data were normally distributed. Therefore, the paired t-test with a 5% level of significance was conducted. The aim was to determine whether a significant difference exists between the motivation levels in traditional modality and employing MEE. On average, students who participated in MEE had a higher motivation level (M = 4.18, SE = 0.080) than those who participated in the traditional modality (M = 3.91, SE = 0.071). Therefore, the difference −0.27, 95% CI [−0.44,−0.09] is significant, t(59) = −3.09, p-value = 0.003.

Table 4 summarizes the results for each ARCS dimension. As can be observed, all show significant differences except satisfaction, which shows a p-value greater than 0.05.

Table 4. T-student results for the ARCS four dimensions.

Variable	t	p-Value	CI	Mean Difference
Attention	−3.39	0.001	[−0.53, −0.14]	−0.33
Relevance	−3.11	0.003	[−0.44, −0.10]	−0.27
Confidence	−2.56	0.013	[−0.48, −0.06]	−0.27
Satisfaction	−1.96	0.054	[−0.43, 0.004]	−0.21

4.4. Discussion

Our research shows that, using VW, such as MEE, increases students' motivation compared to the teacher's material in obtaining the answer to RQ1. However, because the proposed research design is not longitudinal, it does not allow us to rule out the novelty effect of using MEE. However, it is important to mention that 78% of the students had played video games in the paired sample. This could counteract the novelty effect caused by immersive technologies such as MEE. The novelty effect may appear when students are introduced to the new technology and even more so when 60% of the sample has not used video games in educational settings. Another factor that should be considered is that, in general, students respond positively to using ICTs in the learning process [71]. In addition, it should be considered that the age of the students in the present study affects their sense of presence, interaction, and satisfaction in VWs [72].

The results presented in this paper agree with the findings of Wang et al. since their serious VR game satisfies the elements of the ARCS model, as all items had means higher than four [73].

In the traditional modality, it is alarming that 78% of students did not even try to solve a written programming problem. However, with MEE, 73% were interested in programming activities; this may be due to how fun these types of environments are for students and that they allow them to understand the IP material better. Therefore, teaching IP with virtual worlds is important because of the tool and the type of problems that students can perform. In addition, it is advantageous to change classic problems in programming teaching, such as "calculating the average of n grades", to more visual and interesting problems for students, such as building objects or moving their "agent" in the virtual world. Moreover, when using MEE in IP, teachers have a ready environment where students can create artifacts. MEE provides immediate visual feedback, and the student can modify the code to obtain the desired results. Through trial and error, the student can learn to debug the code [70]. Solomon and Papert used a similar approach with their Turtle Graphics using Logo [66].

MEE is a tool that captures students' attention, and the content shown or the things that can be created make it a relevant virtual world to deepen and increase interest in programming [74]. Regarding satisfaction, students indicate that working with MEE is very satisfactory. When it was proposed to use MEE, many students were surprised and took the proposal with great pleasure, which aligns with the comments mentioned in [75].

It is important to highlight that confidence about the test presented the lowest levels in using MEE and the professor's material. This agrees with [76], as exams generally produce moderate anxiety levels in students.

Regarding the results obtained concerning gender, high levels of motivation were observed in women, which coincides with what is indicated by [77,78] and answers RQ2. Women like games more when they are used for educational purposes. It is important to highlight that women express having higher levels of confidence than men, contrary to what was found in [79], where the authors indicate that men have greater self-efficiency (confidence) than women. Therefore, virtual worlds could be an ideal alternative for women to improve their IP performance.

In summary, based on the motivation results and students' performance, we demonstrated that teaching functions in virtual worlds could mitigate the high failure rates in programming in the first semesters of higher education. Therefore, 72% (n = 43) of the students had satisfactory results in MEE versus only 22% (n = 13) who passed the paper-based exam. The virtual worlds offer professors an innovative way to motivate and involve the students in a learning environment [37]. Furthermore, although there may be different student-centered environments that foster motivation, such as Scratch [80] and App Inventor [81], an evolution to immersive technologies is needed. MEE as an educational game and new metaverse tool [64] will undoubtedly be part of the future trends in computer programming education.

5. Conclusions

It would be important to highlight the need for a didactic model for using virtual worlds in a teaching programming, allowing teachers to incorporate this innovation in classrooms. The proposed teaching model could help teachers face possible problems or difficulties when using virtual worlds in the PI teaching–learning process. According to [19], few studies indicate virtual model implementation problems. However, issues are always present. For example, in this study, a technical problem arose that prevented the execution of code within the MEE on some of the students' computers. The issue was resolved with an MEE update. In addition, infrastructure problems related to the computing equipment necessary for using the tool could be addressed with a multiplatform virtual world. The tool can be used on a tablet, desktop, mobile phone, Chromebook, browser, or video game console. In the case of the present study, many students did not participate in the activity because their equipment did not meet the minimum MEE requirements.

The present study deals with the following limitations:

- The sample was not random, as class groups assigned to the teacher were taken.
- The study was conducted online, not in the institution's computer lab.
- The activity was not performed with MEE in collaborative mode. Each student individually worked on their activities.
- Students used personal computers, and participation was affected as many did not have computers that could properly run MEE.
- From the teacher's point of view, applying the VMs requires additional time and effort compared to traditional teaching.

In future research, it would be important to determine the effects of virtual worlds in other more advanced programming courses, higher-level subjects, and long-term studies.

Author Contributions: Conceptualization, J.G.L.S., C.J.A.R. and O.O.V.V.; methodology, J.G.L.S., C.J.A.R. and O.O.V.V.; software, J.G.L.S., C.J.A.R. and O.O.V.V.; validation, J.G.L.S., C.J.A.R. and O.O.V.V.; formal analysis, J.G.L.S., C.J.A.R. and O.O.V.V.; investigation, J.G.L.S., C.J.A.R. and O.O.V.V.; resources, J.G.L.S., C.J.A.R. and O.O.V.V.; data curation, J.G.L.S., C.J.A.R. and O.O.V.V.; writing—original draft preparation, J.G.L.S., C.J.A.R. and O.O.V.V.; writing—review and editing, J.G.L.S., C.J.A.R. and O.O.V.V.; visualization, J.G.L.S., C.J.A.R. and O.O.V.V.; supervision, J.G.L.S., C.J.A.R. and O.O.V.V.; project administration, J.G.L.S., C.J.A.R. and O.O.V.V. All authors have read and agreed to the published version of the manuscript.

Funding: The authors would like to thank Universidad Autónoma de Nuevo León.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

Acknowledgments: Juan Gabriel López Solórzano would like to thank Consejo Nacional de Humanidades Ciencias y Tecnologías (CONAHCYT) for the doctoral scholarship.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AR	Augmented reality
ARCS	The attention, relevance, confidence, and satisfaction model
GPU	Graphics processing unit
ICTs	Information and communication technologies
IMMS	Instructional Materials Motivation Survey
IP	Introductory programming
LSL	Linden scripting language
MEE	Minecraft education edition
NPC	Non-player characters
RIMMS	Reduced Instructional Materials Motivation Survey
TDIDE	Three-dimensional immersive digital environments
VR	Virtual reality
VW	Virtual worlds

References

1. Groher, I.; Vierhauser, M.; Sabitzer, B.; Kuka, L.; Hofer, A.; Muster, D. Exploring diversity in introductory programming classes: An experience report. In Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training, Pittsburgh, PA, USA, 22–24 May 2022; pp. 102–112. [CrossRef]
2. Qian, Y.; Lehman, J. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* **2017**, *18*, 1–24. [CrossRef]
3. Fang, X.; Ng, D.T.K.; Leung, J.K.L.; Xu, H. The applications of the ARCS model in instructional design, theoretical framework, and measurement tool: A systematic review of empirical studies. *Interact. Learn. Environ.* **2023**, 1–28. [CrossRef]
4. Li, K.; Keller, J.M. Use of the ARCS model in education: A literature review. *Comput. Educ.* **2018**, *122*, 54–62. [CrossRef]
5. Howard, J.L.; Bureau, J.S.; Guay, F.; Chong, J.X.Y.; Ryan, R.M. Student Motivation and Associated Outcomes: A Meta-Analysis From Self-Determination Theory. *Perspect. Psychol. Sci.* **2021**, *16*, 1300–1323. [CrossRef] [PubMed]
6. Angel Rueda, C.J.; Valdés Godínes, J.C.; Douglas Rudman, P. Categorizing the educational affordances of 3-dimensional immersive digital environments. *J. Inf. Technol. Educ. Innov. Pract.* **2018**, *17*, 83–112. [CrossRef] [PubMed]
7. Angel Rueda, C.J. Diseño de un Modelo didáctico para Introducir al Maestro en el uso de los Mundos Virtuales con Fines Educativos. Ph.D. Thesis, Universidad Autónoma de Querétaro, Santiago de Querétaro, Mexico, 2018.
8. Ghanbarzadeh, R.; Ghapanchi, A.H. Investigating various application areas of three-dimensional virtual worlds for higher education. *Br. J. Educ. Technol.* **2018**, *49*, 370–384. [CrossRef]
9. Huang, Y.C.; Backman, S.J.; Backman, K.F.; McGuire, F.A.; Moore, D. An investigation of motivation and experience in virtual learning environments: A self-determination theory. *Educ. Inf. Technol.* **2019**, *24*, 591–611. [CrossRef]
10. Shonfeld, M.; Greenstein, Y. Factors promoting the use of virtual worlds in educational settings. *Br. J. Educ. Technol.* **2021**, *52*, 214–234. [CrossRef]
11. Arnab, S.; Lim, T.; Carvalho, M.B.; Bellotti, F.; De Freitas, S.; Louchart, S.; Suttie, N.; Berta, R.; De Gloria, A. Mapping learning and game mechanics for serious games analysis. *Br. J. Educ. Technol.* **2015**, *46*, 391–411. [CrossRef]
12. Pellas, N.; Mystakidis, S.; Christopoulos, A. A Systematic Literature Review on the User Experience Design for Game-Based Interventions via 3D Virtual Worlds in K-12 Education. *Multimodal Technol. Interact.* **2021**, *5*, 28. [CrossRef]
13. Pellas, N.; Mystakidis, S. A Systematic Review of Research about Game-based Learning in Virtual Worlds. *J. Univ. Comput. Sci.* **2020**, *26*, 1017–1042. [CrossRef]
14. Scherer, R.; Siddiq, F.; Sánchez Viveros, B. A meta-analysis of teaching and learning computer programming: Effective instructional approaches and conditions. *Comput. Hum. Behav.* **2020**, *109*, 106349. [CrossRef]
15. Pears, A.; Seidman, S.; Malmi, L.; Mannila, L.; Adams, E.; Bennedsen, J.; Devlin, M.; Paterson, J. A survey of literature on the teaching of introductory programming. In Proceedings of the Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education, Dundee, UK, 25–27 June 2007; pp. 204–223. [CrossRef]

16. Omer, U.; Farooq, M.S.; Abid, A. Introductory programming course: Review and future implications. *PeerJ Comput. Sci.* **2021**, *7*, e647. [CrossRef]
17. Noone, M.; Mooney, A. Visual and textual programming languages: A systematic review of the literature. *J. Comput. Educ.* **2018**, *5*, 149–174. [CrossRef]
18. Kanika; Chakraverty, S.; Chakraborty, P. Tools and Techniques for Teaching Computer Programming: A Review. *J. Educ. Technol. Syst.* **2020**, *49*, 170–198. [CrossRef]
19. López Solórzano, J.G.; Ángel Rueda, C.J. Revisión sistemática de los entornos digitales inmersivos tridimensionales en la enseñanza de la programación. *Rev. De Educ. A Distancia (RED)* **2023**, *23*, 1–24. [CrossRef]
20. Esteves, M.; Fonseca, B.; Morgado, L.; Martins, P. Improving teaching and learning of computer programming through the use of the Second Life virtual world. *Br. J. Educ. Technol.* **2011**, *42*, 624–637. [CrossRef]
21. Cooper, S.; Dann, W.; Pausch, R. Alice: A 3-D tool for introductory programming concepts. *J. Comput. Sci. Coll.* **2000**, *15*, 107–116.
22. Reisoğlu, I.; Topu, B.; Yılmaz, R.; Karakuş Yılmaz, T.; Göktaş, Y. 3D virtual learning environments in education: A meta-review. *Asia Pac. Educ. Rev.* **2017**, *18*, 81–100. [CrossRef]
23. Au, W.J. *Making a Metaverse That Matters: From Snow Crash & Second Life to a Virtual World Worth Fighting for*; John Wiley and Sons: Hoboken, NJ, USA, 2023.
24. Gregory, S.; Scutter, S.; Jacka, L.; McDonald, M.; Farley, H.; Newman, C. Barriers and Enablers to the Use of Virtual Worlds in Higher Education: An Exploration of Educator Perceptions, Attitudes and Experiences. *J. Educ. Technol. Soc.* **2015**, *18*, 3–12.
25. Büşra, A.Ç.; Ediz, Ç. Top concerns of user experiences in Metaverse games: A text-mining based approach. *Entertain. Comput.* **2023**, *46*, 100576. [CrossRef]
26. Prensky, M. Digital Natives, Digital Immigrants. *Horizon* **2001**, *9*, 6.
27. Robins, A.; Rountree, J.; Rountree, N. Learning and Teaching Programming: A Review and Discussion. *Comput. Sci. Educ.* **2010**, *13*, 137–172. [CrossRef]
28. Shokaliuk, S.; Bohunenko, Y.; Lovianova, I.; Shyshkina, M. Technologies of distance learning for programming basics on the principles of integrated development of key competences. *CTE Workshop Proc.* **2020**, *7*, 548–562.
29. Rico, M.; Oz, G.M.M.; Alaman, X.; Camacho, D.; Pulido, E. A Programming Experience of High School Students in a Virtual World Platform. *Int. J. Eng. Educ.* **2011**, *27*, 52.
30. Hulsey, C.; Pence, T.B.; Hodges, L.F. Camp CyberGirls: Using a virtual world to introduce computing concepts to middle school girls. In Proceedings of the 45th ACM Technical Symposium on Computer Science Education, Denver, CO, USA, 6–9 March 2013; ACM: New York, NY, USA, 2014; pp. 331–336. [CrossRef]
31. Kao, D. JavaStrike: A Java programming engine embedded in virtual worlds. In Proceedings of the 14th International Conference on the Foundations of Digital Games, San Luis Obispo, California, 26–30 August 2019; ACM: New York, NY, USA, 2019; pp. 1–5. [CrossRef]
32. Kutay, E.; Oner, D. Coding with Minecraft: The Development of Middle School Students' Computational Thinking. *ACM Trans. Comput. Educ.* **2022**, *22*, 1–19. [CrossRef]
33. Aktaş Kumral, H.; Çam, E. The Effect of Coding Education with Minecraft EDU on Students' Problem Solving Skills and Their Attitudes towards Coding. *Uluslararası Türk Eğitim Bilim. Derg.* **2023**, *2023*, 1–40. [CrossRef]
34. Mørch, A.I.; Andersen, R. Programming with Minecraft Bedrock Up: Modeling, Coding, and Computational Concepts. In *End-User Development*; Spano, L.D., Schmidt, A., Santoro, C., Stumpf, S., Eds.; Lecture Notes in Computer Science; Springer Nature: Cham, Switzerland, 2023; Volume 13917, pp. 230–240. [CrossRef]
35. Bile, A. Development of intellectual and scientific abilities through game-programming in Minecraft. *Educ. Inf. Technol.* **2022**, *27*, 7241–7256. [CrossRef]
36. Sajjanhar, A.; Faulkner, J. Second life as a learning environment for computer programming. *Educ. Inf. Technol.* **2019**, *24*, 2403–2428. [CrossRef]
37. Demirkiran, M.; Tansu, F. An investigation on primary school students' dispositions towards programming with game-based learning. *Educ. Inf. Technol.* **2021**, *26*, 3871–3892. [CrossRef]
38. Alsaadi, B.; Alsaadi, B.; Alghamdi, A.; Alfhaid, M.; Almuallim, N.; Meccawy, M. Learning While Playing: Introducing Programming Concepts to Children in Minecraft. *Int. J. Online Biomed. Eng. (iJOE)* **2022**, *18*, 4–24. [CrossRef]
39. Vosinakis, S.; Anastassakis, G.; Koutsabasis, P. Teaching and learning logic programming in virtual worlds using interactive microworld representations. *Br. J. Educ. Technol.* **2018**, *49*, 30–44. [CrossRef]
40. Schunk, D.H.; Meece, J.L.; Pintrich, P.R. *Motivation in Education: Theory, Research, and Applications*, 4th ed.; Pearson New International Edition; Always Learning; Pearson: London, UK, 2014.
41. Cook, D.A.; Artino, A.R. Motivation to learn: An overview of contemporary theories. *Med. Educ.* **2016**, *50*, 997–1014. [CrossRef]
42. Keller, J.M. Development and use of the ARCS model of instructional design. *J. Instr. Dev.* **1987**, *10*, 2–10. [CrossRef]
43. Garzón, J.; Kinshuk; Baldiris, S.; Gutiérrez, J.; Pavón, J. How do pedagogical approaches affect the impact of augmented reality on education? A meta-analysis and research synthesis. *Educ. Res. Rev.* **2020**, *31*, 100334. [CrossRef]
44. Choi, W.C.; Lei, H.; Mendes, A.J. Motivating the New Generation: Using Flipped Classroom and ARCS Model to Enhance Block-Based Programming Education. In Proceedings of the 2023 IEEE Frontiers in Education Conference (FIE), College Station, TX, USA, 18–21 October 2023; pp. 1–8. [CrossRef]

45. Girvan, C.; Savage, T. Identifying an appropriate pedagogy for virtual worlds: A Communal Constructivism case study. *Comput. Educ.* **2010**, *55*, 342–349. [CrossRef]
46. Girvan, C.; Savage, T. Virtual worlds: A new environment for constructionist learning. *Comput. Hum. Behav.* **2019**, *99*, 396–414. [CrossRef]
47. Girvan, C.; Tangney, B.; Savage, T. SLurtles: Supporting constructionist learning in Second Life. *Comput. Educ.* **2013**, *61*, 115–132. [CrossRef]
48. Papert, S.A. *Mindstorms*; Hachette: London, UK, 2020; OCLC: 1239803576.
49. Papert, S. A Critique of Technocentrism in Thinking about the School of the Future. In *Children in the Information Age*; Elsevier: Berlin/Heidelberg, Germany, 1988; pp. 3–18. [CrossRef]
50. Duncan, I.; Miller, A.; Jiang, S. A taxonomy of virtual worlds usage in education. *Br. J. Educ. Technol.* **2012**, *43*, 949–964. [CrossRef]
51. Mystakidis, S.; Berki, E.; Valtanen, J.P. Deep and Meaningful E-Learning with Social Virtual Reality Environments in Higher Education: A Systematic Literature Review. *Appl. Sci.* **2021**, *11*, 2412. [CrossRef]
52. Pellas, N.; Kazanidis, I.; Konstantinou, N.; Georgiou, G. Exploring the educational potential of three-dimensional multi-user virtual worlds for STEM education: A mixed-method systematic literature review. *Educ. Inf. Technol.* **2017**, *22*, 2235–2279. [CrossRef]
53. Yu, Z.; Deng, X. A Meta-Analysis of Gender Differences in e-Learners' Self-Efficacy, Satisfaction, Motivation, Attitude, and Performance across the World. *Front. Psychol.* **2022**, *13*, 897327. [CrossRef] [PubMed]
54. Yu, Z. The effects of gender, educational level, and personality on online learning outcomes during the COVID-19 pandemic. *Int. J. Educ. Technol. High. Educ. Vol.* **2021**, *18*, 14. [CrossRef] [PubMed]
55. Chen, S.; Yang, S.J.; Hsiao, C. Exploring student perceptions, learning outcome and gender differences in a flipped mathematics course. *Br. J. Educ. Technol.* **2016**, *47*, 1096–1112. [CrossRef]
56. Choi, G.; Chung, H.; Kim, Y. Are Stereotypes Relative to Gender Usage Applicable to Virtual Worlds? *Int. J.-Hum.-Comput. Interact.* **2012**, *28*, 399–405. [CrossRef]
57. Lin, M.; Tutwiler, M.S.; Chang, C. Gender bias in virtual learning environments: An exploratory study. *Br. J. Educ. Technol.* **2012**, *43*, E59–E63. [CrossRef]
58. Korlat, S.; Kollmayer, M.; Holzer, J.; Lüftenegger, M.; Pelikan, E.R.; Schober, B.; Spiel, C. Gender Differences in Digital Learning during COVID-19: Competence Beliefs, Intrinsic Value, Learning Engagement, and Perceived Teacher Support. **2021**, *12*, 637776. [CrossRef] [PubMed]
59. Keller, J.M. *Motivational Design for Learning and Performance: The ARCS Model Approach*; Springer: New York, NY, USA, 2010. [CrossRef]
60. Loorbach, N.; Peters, O.; Karreman, J.; Steehouder, M. Validation of the Instructional Materials Motivation Survey (IMMS) in a self-directed instructional setting aimed at working with technology. *Br. J. Educ. Technol.* **2015**, *46*, 204–218. [CrossRef]
61. Hernández Moreno, L.A.; López Solórzano, J.G.; Tovar Morales, M.T.; Vergara Villegas, O.O.; Cruz Sánchez, V.G. Effects of using mobile augmented reality for simple interest computation in a financial mathematics course. *PeerJ Comput. Sci.* **2021**, *7*, e618. [CrossRef]
62. Cipollone, M.; Schifter, C.C.; Moffat, R.A. Minecraft as a Creative Tool: A Case Study. *Int. J. Game-Based Learn.* **2014**, *4*, 1–14. [CrossRef]
63. Nkadimeng, M.; Ankiewicz, P. The Affordances of Minecraft Education as a Game-Based Learning Tool for Atomic Structure in Junior High School Science Education. *J. Sci. Educ. Technol.* **2022**, *31*, 605–620. [CrossRef]
64. Chua, H.W.; Yu, Z. A systematic literature review of the acceptability of the use of Metaverse in education over 16 years. *J. Comput. Educ.* **2023**, *11*, 615–665. [CrossRef]
65. Hamad, A.; Jia, B. How Virtual Reality Technology Has Changed Our Lives: An Overview of the Current and Potential Applications and Limitations. *Int. J. Environ. Res. Public Health* **2022**, *19*, 11278. [CrossRef] [PubMed]
66. Solomon, C.J.; Papert, S. A case study of a young child doing turtle graphics in LOGO. In Proceedings of the National Computer Conference and Exposition on—AFIPS '76, New York, NY, USA, 7–10 June 1976; p. 1049. [CrossRef]
67. Ghanbarzadeh, R.; Ghapanchi, A.H. Uncovering educational outcomes deriving from students' acceptance and involvement with 3D virtual worlds. *Educ. Inf. Technol.* **2021**, *26*, 311–337. [CrossRef]
68. Bar-El, D.; E. Ringland, K. Crafting Game-Based Learning: An Analysis of Lessons for Minecraft Education Edition. In Proceedings of the International Conference on the Foundations of Digital Games, Bugibba, Malta, 15–18 September 2020; pp. 1–4. [CrossRef]
69. Hopcan, S.; Polat, E.; Albayrak, E. Whether to flip Extreme Apprenticeship: Which is more effective in programming instruction? *Educ. Inf. Technol.* **2022**, *27*, 10731–10756. [CrossRef] [PubMed]
70. Tsai, C.Y. Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. *Comput. Hum. Behav.* **2019**, *95*, 224–232. [CrossRef]
71. Kozlova, D.; Pikhart, M. The Use of ICT in Higher Education from the Perspective of the University Students. *Procedia Comput. Sci.* **2021**, *192*, 2309–2317. [CrossRef]
72. Rahman, M.H.A.; Phon, D.N.E.; Utama, N.I.; Halim, N.D.A.; Kasim, S. Sense of Presence and Learning Satisfaction among Students of Different Age Groups in a 3-D Virtual World. *Int. J. Adv. Sci. Eng. Inf. Technol.* **2019**, *9*, 711–716. [CrossRef]

73. Wang, S.T.; Liu, L.M.; Wang, S.M. The Design and Evaluate of Virtual Reality Immersive Learning—The Case of Serious Game "Calcium Looping for Carbon Capture". In Proceedings of the 2018 International Conference on System Science and Engineering (ICSSE), New Taipei City, Taiwan, 15 March 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–4. [CrossRef]
74. Zorn, C.; Wingrave, C.A.; Charbonneau, E.; Laviola, J.J. Exploring Minecraft as a conduit for increasing interest in programming. In Proceedings of the International Conference on Foundations of Digital Games, Chania, Crete, Greece, 14–17 May 2013.
75. AlJanah, S.; Teh, P.S.; Tay, J.Y.; Aiyenitaju, O.; Nawaz, R. Minecraft as a Tool to Enhance Engagement in Higher Education. In *Research and Innovation Forum 2022*; Visvizi, A., Troisi, O., Grimaldi, M., Eds.; Springer Proceedings in Complexity; Springer International Publishing: Cham, Switzerland, 2023; pp. 465–476. [CrossRef]
76. Deloatch, R.; Bailey, B.P.; Kirlik, A. Measuring Effects of Modality on Perceived Test Anxiety for Computer Programming Exams. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education, Memphis, TN, USA, 2–5 March 2016; pp. 291–296. [CrossRef]
77. Gómez-Gonzalvo, F.; Molina, P.; Devís-Devís, J. Which are the patterns of video game use in Spanish school adolescents? Gender as a key factor. *Entertain. Comput.* **2020**, *34*, 100366. [CrossRef]
78. Joiner, R.; Iacovides, J.; Owen, M.; Gavin, C.; Clibbery, S.; Darling, J.; Drew, B. Digital Games, Gender and Learning in Engineering: Do Females Benefit as Much as Males? *J. Sci. Educ. Technol.* **2011**, *20*, 178–185. [CrossRef]
79. Tellhed, U.; Björklund, F.; Kallio Strand, K. Sure I can code (but do I want to?). Why boys' and girls' programming beliefs differ and the effects of mandatory programming education. *Comput. Hum. Behav.* **2022**, *135*, 107370. [CrossRef]
80. Erol, O.; Kurt, A.A. The effects of teaching programming with scratch on pre-service information technology teachers' motivation and achievement. *Comput. Hum. Behav.* **2017**, *77*, 11–18. [CrossRef]
81. Pérez-Jorge, D.; Martínez-Murciano, M.C. Gamification with Scratch or App Inventor in Higher Education: A Systematic Review. *Future Internet* **2022**, *14*, 374. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

computers

MDPI

Article

Searching Questions and Learning Problems in Large Problem Banks: Constructing Tests and Assignments on the Fly

Oleg Sychev

Software Engineering Department, Volgograd State Technical University, Lenin Ave., 28, 400005 Volgograd, Russia; oasychev@gmail.com

Abstract: Modern advances in creating shared banks of learning problems and automatic question and problem generation have led to the creation of large question banks in which human teachers cannot view every question. These questions are classified according to the knowledge necessary to solve them and the question difficulties. Constructing tests and assignments on the fly at the teacher's request eliminates the possibility of cheating by sharing solutions because each student receives a unique set of questions. However, the random generation of predictable and effective assignments from a set of problems is a non-trivial task. In this article, an algorithm for generating assignments based on teachers' requests for their content is proposed. The algorithm is evaluated on a bank of expression-evaluation questions containing more than 5000 questions. The evaluation shows that the proposed algorithm can guarantee the minimum expected number of target concepts (rules) in an exercise with any settings. The available bank and exercise difficulty chiefly determine the difficulty of the found questions. It almost does not depend on the number of target concepts per item in the exercise: teaching more rules is achieved by rotating them among the exercise items on lower difficulty settings. An ablation study show that all the principal components of the algorithm contribute to its performance. The proposed algorithm can be used to reliably generate individual exercises from large, automatically generated question banks according to teachers' requests, which is important in massive open online courses.

Keywords: exercise generation; question search; knowledge graphs; test-sheet generation; assignment generation; question banks

1. Introduction

An increasing number of educational institutions working on different levels of education have adopted e-learning and online learning technologies [1]. That trend was heavily boosted by lockdowns during the COVID-19 pandemic when many institutions were forced to rapidly employ e-learning technologies [2,3]. The transition to online learning is not always smooth: when educational institutions are not ready for it, it often causes problems and disadvantages [4]. One of the problems during the transition to online learning is performing formative and summative assessments online, which is shown to be beneficial for students but puts additional workload on the teaching staff [5,6]. This requires creating and maintaining electronic banks of questions and learning problems. Online learning allows and sometimes requires bigger problem banks than regular classroom learning because of the following reasons:

- When tests and assignments are graded automatically, online learners can do a lot more exercises than they can do in the classroom, especially if the learning system provides feedback;
- Online learning allows greater variety in the exercises performed; they can be chosen either by students or using recommender systems to create personal learning trajectories, but the problem bank should provide enough variety so that all kinds of students have exercises suited for them;

- Performing assessments without supervision from the teaching staff is more vulnerable to cheating, especially with closed-answer or short-answer questions where the number of possible correct answers is limited, so the institutions need more variety in the learning tasks students must perform.

Developing questions and learning problems or gathering them from existing sources takes significant time. Teachers are often overworked and need more time to spare on the routine job of authoring many similar exercises. Developing and adopting Open Educational Resources can help with that, but it has its own problems and concerns [7].

One of the ways to counter this problem is by using artificial intelligence techniques to generate questions and learning problems automatically. Automatic generation saves effort on creating question and problem banks and increases the size and consistency of these banks [8]. Usage of automatically generated problem and question banks provides the following benefits in the learning process [9]:

- Preventing some types of cheating by supplying each learner with a unique set of questions and learning problems;
- Training until mastery because the learner is not limited by the size of the problem bank, which is especially important when these problem banks are used in conjunction with intelligent tutoring systems that can provide automatic feedback on errors and hints;
- Better support adaptive learning in intelligent tutoring systems (ITS) because the bank contains a lot of various learning problems so that the adaptive system can find a learning problem for every educational situation.

However, these benefits come with a cost that impedes the adoption of generated questions in the learning process [10]. While using an existing bank requires far less effort than creating a teacher's own bank, teachers' trust in question and problem banks received from external sources is significantly lower, especially when the bank is so large that the teacher cannot read, solve, and/or evaluate all the learning problems or even a significant portion of them. Teachers are naturally hesitant to use questions and learning problems with which they are not acquainted. Also, large problem banks, often created from different sources, raise the issue of problem homogeneity and consistent evaluation of their difficulties. It is also important to avoid learning problems containing topics the students have not yet learned. Even the concepts (or topics) that do not affect the answer to the question (problem solution) should be taken into account because they can confuse some students. Thus, a good filtering system is a must to deal with any large problem bank, especially when solving the problems is required to complete the course. All these factors restrict the widespread usage of automatically generated banks of learning problems.

Question (problem) banks are used to construct specific exercises (tests, assessments). When the question banks were small, teachers combined questions (or problems) into an exercise that all the students had to pass. When the number of questions increased, the typical organization of a random test involved the teachers grouping similar questions into categories and constructing tests by specifying the number of randomly selected questions from each category. This allowed more diversity in the questions students received, which lessened cheating and made attempting one test several times non-trivial but left the test plan under the teacher's control as categories usually contained similar questions (learning problems). However, that method requires a teacher-defined classification of questions in complex categories according to their purpose, which is not always available for automatically generated questions and learning problems. The same requirements are used in methods of automatic test generation (e.g., [11]).

The problem of automatic generation and the usage of large banks of questions and learning problems is especially important in the field of computer programming education. Learning to program requires learning a significant number of new concepts and mental manipulation of abstract objects. This requires a lot of training by solving well-defined tasks, which creates good conditions for developing and using automatic tutors with explanatory feedback and formal, automatically graded exams [12]. A high failure rate in

introductory programming courses [13] also signals that students need additional support in learning the material.

However, training until mastery by using automatic tutors and the frequent usage of exam tests requires big banks of questions and learning problems so that students do not get the same questions several times, which requires developing techniques of generating questions and generating balanced test sheets from them. This study is aimed at solving the second problem, which has received significantly less attention than the actively developing field of question generation.

Developing automatic question and learning-problem generation methods creates problems for classic methods of building tests and exercises from questions and learning problems. When the question bank becomes too big to be observable by a human teacher, we need to develop new question search and retrieval methods according to the teachers' needs. This is a significant and complex problem because in many subject domains, learning problem features depend on the problem structure, not just the types of its elements. For example, in introductory programming courses, when teaching control-flow statements to assess the problem topic, it is often enough to take into account the statements that are present in the problem formulation (even though they can be in dead code that is never reached during the problem solving). However, the same principles do not work for teaching expressions: rules of operator precedence and associativity influence the answer under certain circumstances. For example, to teach (or verify knowledge of) associativity, you need an expression where two operators with the same precedence stand in a row without parentheses affecting their evaluation. That does not allow constraining learning-problem generation simply by constraining the types of objects (e.g., statements or operators) participating in the formal problem formulation, so we cannot reliably predict classes of generated questions without a complex analysis of the problem formulation.

This study aims at developing an algorithm for constructing exercises from questions and learning problems stored in large problem banks when the problems are labeled according to the knowledge necessary to solve them and difficulty metrics in the absence of significant statistics of the usage of these learning problems because they can be specifically generated to avoid giving the same problem to different students.

The contribution of this study includes:

- A problem formulation for learning problem (question) retrieval from large banks of automatically generated questions, including the required formalization of learning problem metadata, problem request, and retrieval goals that does not require manually assigning slots for questions and statistical data on question usage;
- An algorithm for question search taking into account the subject-domain concepts and rules required to solve them and question difficulties;
- An evaluation of the proposed algorithm on a knowledge bank consisting of thousands of automatically generated questions on the order of expression evaluation.

The rest of the article is organized as follows. Section 2 discusses related work. Section 3 describes the proposed algorithm. Section 4 describes the experimental evaluation of that algorithm on a large problem bank for a particular topic. Section 5 discusses the study's findings. Section 6 contains conclusions and further work. The line between complex questions and easy learning problems are not clearly established, so in the rest of the article, the term "question" is used to describe both questions and learning problems except in places where the distinction between them is important.

2. Related Work

There are two kinds of related work to review. Methods of question and learning problem generation are essential because they determine what kinds of information we can get about the generated questions. Methods of selecting and recommending questions are used to construct exercises from banks.

2.1. Methods of Question and Learning-Problem Generation

Different methods of generating questions and learning problems give different information about their results. It determines the possible usage of the resulting questions and the methods of their retrieval for constructing tests and exercises.

A significant part of the methods of question generation is concentrated on processing natural-language texts and generating questions about the content of that text [8,14–17]. Some researchers add searching texts by keywords to leverage this to generate sets of questions for the given keyword [18]; this approach has obvious drawbacks because the teacher cannot control the content of these texts. Some recent works on Visual Question Generation try to use an image as input for question generation instead of text, using methods of solving the problem of image understanding and summarization [19–21]. These questions are aimed at solving the reading comprehension problem, i.e., verifying that the learner has read and understood the given text or image. Thus, these questions are not linked to studied topics (except the keyword-based generation [18]) but to particular learning materials (textbook fragments, lecture notes, etc.). They can be used to generate quizzes placed right after the learning materials and combined to generate summative quizzes by specifying the number of questions about each material in the topic or course. Chen et al. created a sophisticated dataset of teacher- and learner-created questions, with a significant percentage of questions aimed at high-level thinking skills, and found that modern methods performed poorly compared to humans exercising skills on medium and high levels of Bloom's taxonomy (comprehension, application, analysis) [22]. The chief problem of questions generated by automatic methods, as it was noted by Kurdi et al. [8], is that they have low cognitive levels and simply ask for the facts mentioned in the text (image); they do not verify understanding of these facts, the ability to combine them with other facts (from the same text or other texts) and use them in practical situations.

Ontologies (and similar representations of structured knowledge) are also a popular approach to generate questions, which has been used to generate questions to teach medicine [23], programming [24], and general knowledge stored in commonsense knowledge graphs [25]. These questions are generated using the graph representation of the required knowledge. Therefore, they can be linked not to a particular textbook fragment but to the subject-domain concepts they are about. This allows the generation of topic-based question banks and requires efficient mechanisms of question retrieval according to the teacher's plans. The difficulty of these questions varies significantly.

More complex tasks are solved in the field of learning problem generation. The chief approaches to generating learning problems are template-based generation [26,27], random generation within subject-domain constraints [28,29], and problem mining [30].

Template-based generation practically means that the learning problems are planned by teachers as templates. A set of problems is created from the same template by changing secondary parameters, which usually affect the correct answer but do not affect the knowledge required to solve problems [31]. Examples of systems using template-based generation are some of Problet tutors [32] (which generate programs and results of their executions from teacher-written templates in pseudo-BNF notation) and QuizJET [33] (a tutoring system aimed at teaching the Java programming language and object-oriented programming which contains about 100 manually created problem templates). This approach can also be used outside of teaching programming, e.g., for generating natural deduction problems [34].

The used templates (and learning problems generated from them can be manually classified; they can use regular methods of exercise generation by specifying the required number of problems of each type. The downside of that method is its low scalability and significant intellectual work from the author of the exercise, who still has to invent every kind of exercise and code it according to the learning problem template. The generation algorithm only multiplies existing learning problems without varying their structure.

The method of random generation within subject-domain constraints has entirely different features. It can generate genuinely new problems (i.e., problems whose structure and

solutions are not planned by a human author), produce problems with different difficulties, and require knowledge within a particular topic while maintaining the correctness of the generated problems using constraints defined by the subject domain. The most popular formalism in this area is Answer Set Programming (ASP), which can be used both to infer solutions to problems and generate problems that can be solved by the same program [35]. It is a widely used approach to procedural generation for designing computer games [36,37]. O'Rourke et al. used ASP to generate algebra problems [29]. However, this approach is limited to learning problems in relatively simple domains where all problem elements can be combined in almost any way (e.g., arithmetic operators in algebraic expressions). When the domain becomes more complex (e.g., algebraic expressions which are homogeneous by data types with expressions in programming languages that are heterogeneous, with different operators applicable to different data types), random generation can only guarantee the basic correctness of the generated problems, but it cannot guarantee their readability and sensibility. For example, Martin and Mitrovich used this approach to generate learning problems for a constraint-based tutor for the SQL language; the generated problems were not instantly applicable and required clarification and validation by a human [28]. Kumar argues that in programming education, randomly generated problems are only helpful for syntax-oriented exercises [38].

Problem mining also relies on subject-domain constraints to solve and classify generated problems [30], but instead of random generation, it uses real data created and verified by humans (e.g., program code, banks of parsed sentences) or digital twins of real-world objects. This allows the large-scale generation of complex, naturally looking learning problems classified by their difficulties and required knowledge in the form of domain concepts and rules required to solve them or possible semantic errors in their solutions [39]. These learning problems often can be used without human verification, which allows scaling generation to problem banks containing many thousands of problems on the topic. It is a promising approach in the fields when the necessary data can be obtained, but it requires developing complex subject-domain models to solve and classify the generated problems [12,34].

Both randomly generated and mined learning problems in complex domains cannot be automatically assigned categories for exercise creation and require special methods of learning problem search according to the particular learning situation. These learning problems can be labeled by the concepts used in the problem definition and domain rules and laws whose knowledge is necessary to solve them. However, the number of combinations of these labels is high, so most of the time, these labels cannot be used directly in exercise construction. For example, the automatically generated problem bank of more than 20,000 learning problems described in [39] contains 613 different kinds of problems (taking into account the concepts and rules whose knowledge is required to solve the problem and the problem difficulty). In some domains, it can be even more difficult: for example, when studying expressions, if we consider knowing the precedence and associativity of each operator a separate kind of knowledge, we obtain more than 840 kinds of learning problems. This makes it impossible for a human teacher to look through all the categories of learning problems in the problem bank and select the categories that should be used in their assignments. Thus, we need to develop algorithms of learning problem search and retrieval from large banks with a reasonable amount of information provided by the teacher.

2.2. Methods of Selecting and Recommending Questions and Learning Problems

One of the most established models of estimating the difficulty of questions and learning problems is Item Response Theory (IRT) [40]. This model is actively used in intelligent tutoring systems for modeling the difficulty of exercise items for students [41]. However, this model relies on gathering statistical data for each item (i.e., a question) in the bank and so cannot be used in large knowledge banks that individualize learning by avoiding using the same question for different learners.

Probabilistic models like Bayesian Knowledge Tracing (BKT) are often used to estimate the students' mastery and choose the next learning problem in exercises. It considers the probabilities of learning (transitioning to the mastery state), guessing (answering correctly without knowledge), and slipping (answering incorrectly while knowing) depending on the skill [42,43]. This theoretically can be used in large generated question banks because students receive different questions but learn the same skills; however, it is impossible to use it in learning-problem banks because one learning problem often requires several skills to solve. Attempts to improve the original BKT model include individualization (i.e., modeling the rate of learning or initial knowledge of each student) [44,45] and item difficulty (i.e., the difficulty of a particular learning problem) [46,47]. While they improve the performance of the original model, they also require significantly more data to train. Pardos et al. found that the model that takes into account item difficulty (KT-IDEM) showed better results over the original BKT model only when the number of solving data per problem exceeded six. This will not work for large, automatically generated problem banks aimed at providing unique learning problems for each student. Deep Knowledge Tracing, a development of Bayesian Knowledge Tracing using Recurrent Neural Networks, also relies on data from previous attempts of existing questions to teach the neural network [48].

Another field of research concerning selecting questions to solve for a student is exercise recommendation systems. That field has been actively researched in the last years [49–51]. Most of these systems use hybrid approaches and rely either on the data concerning item completion by students or student proficiency data. However, their evaluation is often based on prediction accuracy instead of pedagogical effect [52]. This makes them unsuitable for use in the learning process where ideally, each learning item is used only once. For now, the attempts to predict difficulty and necessary knowledge of learning problems and digital course sessions are concentrated on labeling them by difficulty [53] and/or used concepts [54], which does not solve the problem of retrieval of appropriate exercises from a labeled bank.

The problem of generating tests and assignments with necessary characteristics has been studied for many years. Hwang et al. proposed an algorithm of generating test sheets from big question banks [55]. However, their algorithm was aimed at generating a single test sheet with the optimized discrimination coefficient relevant to listed concepts; they did not take into account the problem of generating multiple similar test sheets for different students to avoid cheating. They also relied on question parameters that were measured statistically during their usage and so were poorly suited to large banks of sparsely used questions where the relevant information is not available. In [56], they improved their algorithm to generate a set of test sheets with the requested characteristics, but they used a hard constraint "no question should be identical in two tests", which is impractical when generating a large number of tests (e.g., for each student). They only generated at most four parallel test sheets with the same specification.

Paul proposed generating a set of solution, but the goal of that algorithm was to achieve satisfactory performance regarding several objectives [57], not variable tests for different learners. Sud et al. considered the problem of generating alternative assessments to avoid cheating by sharing information among students [11]. They managed to reduce the standard deviation of questions' difficulty to about half of its regular values. However, their problem formulation required a pre-classification of questions into different slots, which required manual work and could not be done in diverse automatically generated question banks.

Thus, the problem of generating a large number of test sheets from diverse question banks without previous manual question classification and gathering statistics on question usage is not covered in previous works. This study is aimed at solving that problem.

3. Proposed Algorithm

To discuss the algorithm of question retrieval, we must first consider the structure of question metadata available in the question bank and question request to develop a formal problem statement. Then, the algorithm for satisfying that request is proposed.

3.1. Question Metadata and Requests

In order to find questions satisfying tutors' needs (both for human and automatic tutors), we need diverse information about questions and learning problems contained in the bank. Let us consider a question bank Q consisting of questions $q_i \in Q$. We have sets of studied concepts C and rules (domain laws) R for the given subject domain. Rules define the application of concepts in particular situations. In practice, it is often useful to consider possible semantic errors defined as constraints as domain rules because it allows a more precise covering of the relevant knowledge than using rules that govern answering questions and problem solving, i.e., one rule that determines if a particular solution step or answer is possible can be violated in several different ways [12]. We also need to estimate question difficulty, which is based not only on the concepts and rules whose knowledge are necessary to answer the given question but also on the structure of the question definition and the number of objects (concept individuals) in it, i.e., solving an algorithm with more statements, statement nesting, or more trace steps is more difficult than solving a simpler algorithm using the same statements (for example, in [39], it was shown that for code-tracing problems, the optimal difficulty function included the number of used concepts, number of trace steps and cyclomatic complexity of the algorithm.

Thus, the metadata for question q_i can be defined as (1)

$$q_i =< C_i, R_i, d_i >, \tag{1}$$

where $C_i \subset C$ is the set of concepts whose knowledge is necessary to answer the question, $R_i \subset R$ is the set of domain rules that are used to find the correct answer to the question, and d_i is the estimated difficulty of the question.

The non-adaptive exercise consists of groups of questions with defined requests (i.e., one request can specify a series of questions). For each group of questions, the exercise author should divide subject-domain concepts C and rules R into three categories:

- `target` concepts C^t and rules R^t define the content that the teacher intends to teach (or assess learning of) in this exercise (question);
- `denied` concepts C^d and rules R^d must not appear in the exercise (or a particular part of it) because students are not expected to know them;
- `allowed` concepts C^a and rules R^a can appear in the exercise but are unnecessary (e.g., previous knowledge).

Every concept and rule must belong to one of these categories, i.e., for all groups of questions in the exercise, $C^t \cup C^d \cup C^a = C$ and $R^t \cup R^d \cup R^a = C$. This allows flexibility in selecting relevant questions because the concepts and rules that do not belong to the goal of this question (i.e., non-target concepts and rules) can be divided into those that must be excluded and those that can appear in the exercise, thus broadening the range of possible questions that can match the request. They key difference of a question request from a regularly used question category is that the system does not have pre-made categories fitting teachers' requests but chooses the questions from the bank according to their features.

In real-world tutoring situations, concepts and especially rules that are shown in the tutor's interface and assigned to the target, denied, and allowed categories must not match exactly the concepts and especially the rules that are stored in question metadata. For example, when teaching the order of evaluation of programming-language expressions, the situations of taking into account the precedence of the operator to the left and the precedence of the operator to the right are different rules in the formal model, but nobody teaches them separately—for teachers and students, they are represented as one rule. Similarly, in many expression-related educational situations, it is not necessary to discern

arithmetic operators because they are governed by the same rules and are taught together. Sometimes, human problems of perception and understanding affect the formal model, e.g., while handling two operators with right associativity in a row can be modeled as one rule, for the human perception, it is different for unary (e.g., !!a) and binary ($a = b = 0$) operators and should be taught and verified separately because visually, these situations are too different.

Concepts can be connected by links like "requisite" and "parent–child". For example, when teaching control-flow statements, it is impossible to create a question with the "else" concept without the "if" concept, so "if" is considered "requisite" for "else". This allows one to restrict the user interface to avoid requests that are impossible to satisfy (e.g., with target "else" and denied "if"). "Parent–child" relationships allow exercise authors to select several concepts (e.g., all kinds of loop statements) in one click.

Question request can be formalized as (2)

$$qr_j =< Nq_j, C_j^t, C_j^d, R_j^t, R_j^d, d_j >, \quad (2)$$

where Nq is the desired number of questions, C^t, C^d, R^t, R^d are sets of target and allowed concepts and rules, and d is the approximate desired difficulty.

The task of question retrieval can be formulated as follows. Given a question request qr_j, find a set of questions $Q_j \subset Q$ so that:

1. $|Q_j| = Nq_j$ (obtain the required number of questions);
2. $\forall q_{ij} \in Q_j \exists C_{ij}, C_{ij} \in C_j^t \wedge C_{ij} \in C_i$ (each question contains at least one target concept);
3. $\forall q_{ij} \in Q_j \exists R_{ij}, R_{ij} \in R_j^t \wedge R_{ij} \in R_i$ (each question contains at least one target rule);
4. $\forall q_{ij} \in Q_j \nexists C_{ij}, C_{ij} \in C_j^d \wedge C_{ij} \in C_i$ (there are no denied concepts in the found questions);
5. $\forall q_{ij} \in Q_j \nexists R_{ij}, R_{ij} \in R_j^d \wedge R_{ij} \in R_i$ (there are no denied rules in the found questions);
6. The distributions of elements of C^t and R^t across the found questions are as close to a uniform distribution as possible (spend equal effort on all target concepts and rules);
7. The difference $sum(d_i)/Nq - d$ is minimized (the questions are as close to the desired difficulty as possible).

3.2. Algorithm for the Retrieval of Relevant Questions

While filtering out questions with denied concepts is not hard, coming close to an even distribution of target concepts and rules is a challenging goal. First, the question banks that are generated automatically without using human-written templates can have an uneven distribution of different kinds of questions: some concepts and rules can be present several times and more often than others [9]. While this reflects real-world situations where some subject-domain rules are used more often than others, it is bad for learning because learning should prepare students for every situation they can encounter when solving problems in their subject domain. Also, the connection between rules and concepts is not even: one rule can involve many concepts while the other can involve only a few. For example, many operators are left-associative, but in most of the programming languages, there are few right-associative operators, so if we are teaching associativity, we cannot have even coverage of left and right associativity (rules) and different operator groups (concepts).

Using a random selection of appropriate questions is not good for achieving an even distribution of target concepts and rules because of their possible uneven distribution in the question bank. Also, with a small sample size (many learning exercises consist of a small number of questions), a random selection will often produce significant deviations from the uniform distribution. To better achieve an even representation of different learning targets in the exercise, we propose to adjust the selection of each question by rotating targets (i.e., prioritizing the targets that were used less in the previous questions of an exercise attempt and marking the targets that have already been used often as undesirable).

Therefore, for generating each question, we must determine the specific question request (SQR) based on the original questions request. The main goal of the specific

question request is to find the least used target concepts or rules and make them targets for the next question. Algorithm 1 shows how a specific question request *sqr* is found based on the data from the general question request for this exercise part *qr* and the data about previous questions in exercise *exc*.

Algorithm 1 The algorithm for determining specific question request *sqr* based on the question request *qr* and exercise data *exc*.

function SPECQR(qr, exc)
  $sqr_{Nq} \leftarrow 1$ ▷ Specific request is for the next question.
  $sqr_{C^d} \leftarrow qr_{C^d}$ ▷ Denied concepts and rules remain the same.
  $sqr_{R^d} \leftarrow qr_{R^d}$
  $sqr_d \leftarrow qr_d \times (1 + \text{RAND}(-0.2, 0.2))$ ▷ Change the question difficulty slightly to broaden the range of available questions.
  for all $c_i^t \in qr_{C^t}$ **do**
    $cusage[c_i^t] \leftarrow |UC_i|, where\, UC_i = \{exc_q | c_i^t \in exc_{q_C}\}$ ▷ ex_q is a previous question used in the attempt.
  end for
  $sqr_{C^t} \leftarrow \{C_i^t | C_i^t \in qr_{C_t} \wedge cusage[c_i^t] = \text{MIN}(cusage)\}$ ▷ Make the least-used concepts targets for the next question.
  $sqr_{C^a} \leftarrow qr_{C^a} \cup (qr_{C^t} \cap sqr_{C^t})$ ▷ All other target concepts become allowed for this question.
  for all $r_i^t \in qr_{R^t}$ **do** ▷ The same applies to building sets of target and allowed rules.
    $rusage[r_i^t] \leftarrow |UR_i|, where\, UR_i = \{exc_q | r_i^t \in exc_{q_R}\}$
  end for
  $sqr_{R^t} \leftarrow \{R_i^t | R_i^t \in qr_{R_t} \wedge rusage[r_i^t] = \text{MIN}(rusage)\}$
  $sqr_{R^a} \leftarrow qr_{R^a} \cup (qr_{R^t} \cap sqr_{R^t})$
  return *sqr*
end function

The generated specific question request is then passed to the function that searches relevant questions in the question bank. The algorithm of question search according to a specific question request consists of the following steps:

1. Prepare data for question search.
   (a) Normalize the request's difficulty from the range $[0, 1]$ to the actual range of question difficulties in the question bank.
   (b) Create sets of undesirable concepts and rules: the concepts and rules which were present in the previous N_{prev} questions (with N_{prev} being a search hyper-parameter).
2. Execute a search request in the question bank. The request searches for up to N_{pool} best questions (according to the number of target concepts and rules per question), taking into account the following restrictions:
   (a) Question difficulty must be in range of $\pm Diff_{dev}$% relative to the request difficulty.
   (b) No questions with at least one denied concept or rule are allowed.
   (c) Questions are sorted according to the total number of target concepts and rules in them (descending order), then for the number of their usage in the current exercise (ascending order), so that the most used questions come last.
3. Rank the found questions by three ranking measures:
   (a) The absolute value of the difference between the question difficulty and the requested difficulty (the smaller the better).
   (b) Question "suitability": every target concept and rule adds 3 points of suitability while every undesirable concept and rule subtracts 1 point of suitability (the bigger, the better).

(c) Question diversity: the number of different possible semantic errors while answering the question (the bigger, the better).

4. Calculate the final question rank as a weighted sum of the three ranks discussed above.
5. Return the desired number of best questions according to the final question rank.

That algorithm penalizes repeating the same concepts and rules in subsequent questions to allow the equal representation of all the target concepts and rules in the generated set. It allows a varied range of question difficulties (controlled by hyperparameter $Diff_{dev}$) and more variety in the found questions because of pooling the best-fitting questions and random selection among them.

4. Evaluation

We used a bank of questions generated from program code (see [9] for the description of the generation process) to evaluate the efficiency of the proposed algorithm. The bank used the widely accepted RDF data storage format and had a high variability in the questions because they were generated from randomly selected human-written production program code, not specifically designed for learning purposes. As it can be seen in [9], different concepts and rules (subject-domain laws) were not evenly represented in the bank, so it was well-suited to measure the ability of the proposed algorithm to deal with large, uneven banks of questions.

The bank contained questions about the order of expression evaluation and consisted of 5648 questions with difficulties ranging from 0.54 to 1.0, 11 concept groups, and seven rules.

4.1. Criteria of Evaluation

To evaluate the quality of the generated exercises, first of all, it was necessary to estimate if all the target concepts were present enough times, i.e., if an exercise with N_q questions had N_t target rules (concepts), each target had to appear at least N_q/N_t times. It usually appeared more times because most of the questions contained more than one concept, but N_q/N_t was the required minimum that, ideally, should not be violated in any generated exercise instance. To verify that, we calculated the relative frequency of target concepts (rules) RF_i, which was equal to the number of questions with the i-th target in the attempt divided by the minimum expected number N_q/N_t; see (3).

$$RF_i = |Q_i| * N_t/N_q, \tag{3}$$

where Q_i is the set of the questions with the i-th concept in the generated attempt.

In order to study how well the proposed algorithm adapted to different circumstances, it was decided to study the average number of used rules in questions depending on the number of target rules in the exercise: ideally, both the number of rules per question and question difficulty should remain the same because they should be affected by the exercise difficulty, not by the number of target rules, which should be distributed among different questions in the exercise on lower difficulties. In contrast, average question difficulty, number of rules per question, and number of solution steps per question ideally should monotonically increase with the exercise difficulty, which can be negatively affected if the bank is heavily unbalanced.

4.2. Evaluation of the Algorithm as a Whole

The question bank contained questions asking the student to determine the order of evaluation of the given programming-language expression. The main source of the information in that case was not just concepts but used subject-domain rules, because the rules necessary to answer the questions are determined by the way the operators are placed within the expression, i.e., questions with the same concepts can require knowledge of different rules to solve. Tables 1 and 2 show the distribution of questions with different

concepts and rules in the bank. The bank was diverse, with the number of questions requiring knowing certain rules ranging from 5317 questions for operator precedence to 29 questions for right associativity for binary operators.

Table 1. Number of questions featuring different concepts in the expression bank.

Concept	Number of Questions
Logical operators	4073
Pointer access operators	3749
Object access operators	3682
Comparison operators	3354
Arithmetic operators	1651
Array access operator	1492
Assignment operator	1250
Comma operator	1019
Bitwise operators	987
Increment and decrement operators	79
Assignments with arithmetic operators	67
Total	5648

Table 2. Number of questions requiring knowledge of different rules in the expression bank.

Rule	Number of Questions
Operator precedence	5317
Binary operator with left associativity	3265
Operators with a strict order of evaluating operands	2807
Operators enclosing other operators	1601
Unary operator with right associativity	287
Unary operator with left associativity	30
Binary operator with right associativity	29
Total	5648

Different rule sets were chosen as targets for that knowledge bank, as shown in Table A1. Five rules most frequently learned were chosen, and 45 different kinds of exercises were generated (unary-operator associativity can be understood as "the operator closest to the operand is executed first"; it poses much fewer problems than binary-operator associativity). Each exercise consisted of 10 questions and was generated 20 times. All concepts were allowed to avoid getting into a situation where the system could not find questions for some rules because all the relevant operators belonged to denied concepts.

During this experiment, 9000 questions were retrieved from the bank to create exercises. Among them, there were 3362 unique questions. The mean of the number of times a question was retrieved was 2.68. While the maximum number of retrievals of the same question was 98, 75% of questions were retrieved two or fewer times, and more than 50% of questions were retrieved only once. The most often used question (98 times) had a combination of enclosing operators and binary operator sequences with both left and right associativity affecting the answer, which is a very rare combination.

Figure 1 shows the distribution of relative frequencies of different target rules in the generated exercises. It can be seen that this parameter never fell below the required minimum of one, even for the rules concerning binary operators with the right associativity and strict order of operands that had very few questions. Figure 2 shows the distributions of the number of subject-domain rules in questions retrieved for exercises with the same difficulty but different numbers of target concepts. The average was 2–3 rules per question; uniform exercises (only one target rule) received slightly easier questions because the proposed algorithm did not try to satisfy complex requests at first, while exercises with big numbers of target concepts ($N = 5$) generated slightly more complex questions. Figure 3

shows that the average question difficulty increased slightly when the number of target subject-domain rules increased. The proposed algorithm performed relatively well even with a small question bank, allowing the teacher to set question difficulty separately from the number of target rules.

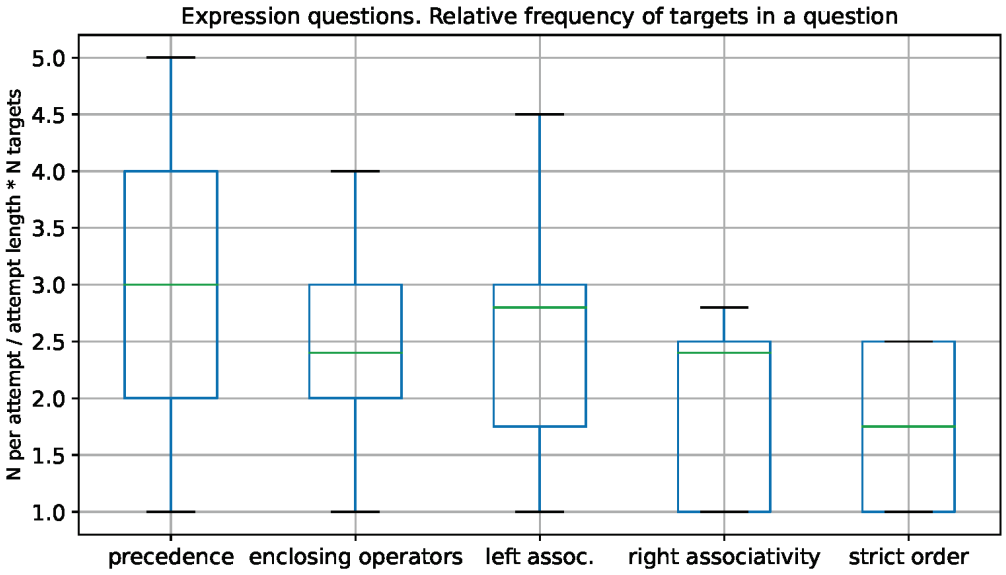


Figure 1. Relative frequencies of different rules in the exercises generated using the expressions bank.

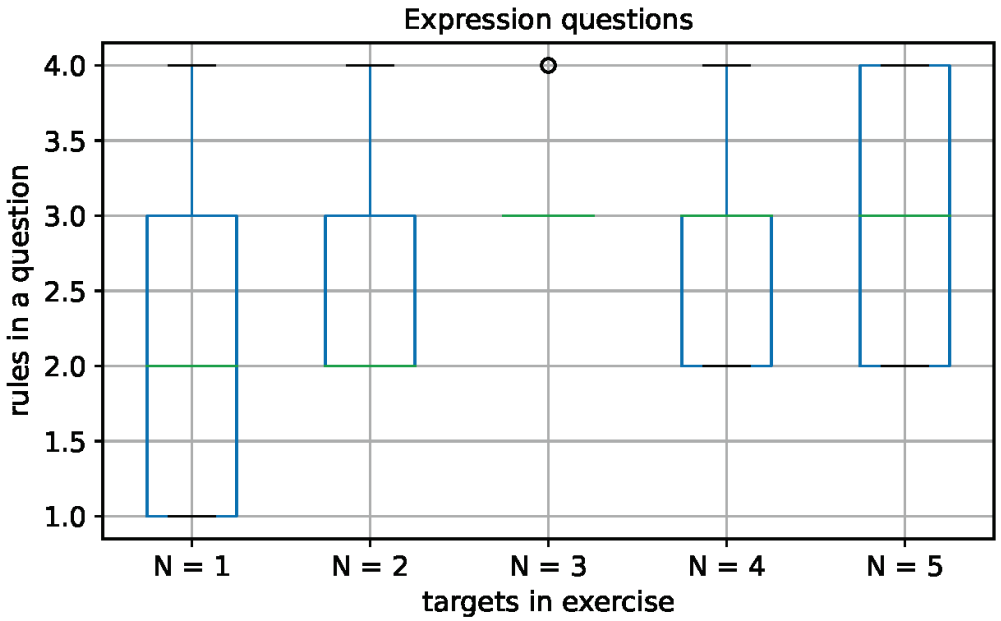


Figure 2. Distribution of the number of subject-domain rules per questions for exercises with different numbers of target rules and the same difficulty. The X axis is the number of target rules in the exercise; the Y axis shows the number of rules in questions.

Figures 4–6 show the average question difficulties, numbers of rules per question, and numbers of solution steps per question depending on the desired difficulty of exercises. Table 3 shows the target rules in different rule groups. The algorithm did not behave ideally for groups #1, #5, and #2, giving a slight decrease in the question difficulty, number of rules per question, and solution steps when the exercise difficulty increased from 0.1 to 0.3. This can be caused by the relatively small size of the question bank and requires further improvement of the algorithm. However, in most situations, the difficulty of included questions increased (or remained the same) when the requested exercise difficulty increased.

The examples of exercises for low, medium, and high difficulty are shown in Appendix B. It can be seen that the low difficulty setting made the algorithm choose the simplest expressions still containing target concepts; medium difficulty settings produced more complex questions and the high-difficulty exercise contained big expressions with a significant number of distracting parentheses and operators.

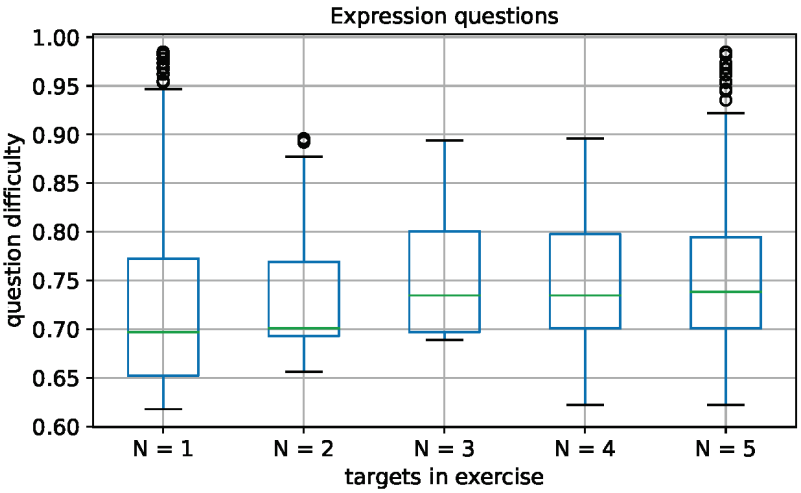


Figure 3. Distribution of the average question difficulty for exercises with different numbers of target rules and the same difficulty. The X axis is the number of target rules in the exercise; the Y axis shows the average difficulties of questions in generated exercises.

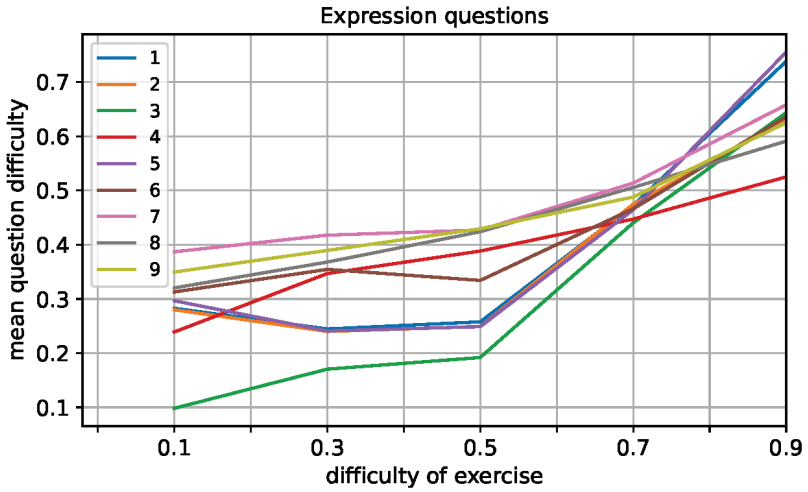


Figure 4. Average question difficulty (Y axis) by exercise difficulty (X axis).

Table 3. Groups of target subject-domain rules for exercise generation with varied difficulty (exercise questions).

Rule Group #	Target Concepts
1	prec
2	encl-ops
3	left-assoc
4	right-assoc
5	strict-order
6	encl-ops prec
7	encl-ops prec left-assoc
8	encl-ops prec left-assoc rightassoc
9	encl-ops prec left-assoc right-assoc strict-order

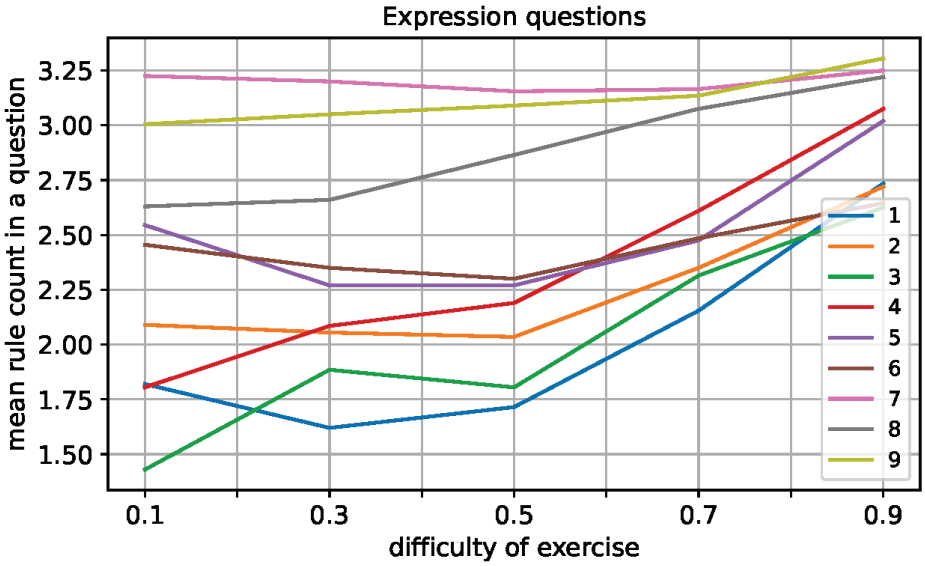


Figure 5. Average number of subject-domain rules per learning question (Y axis) by exercise difficulty (X axis).

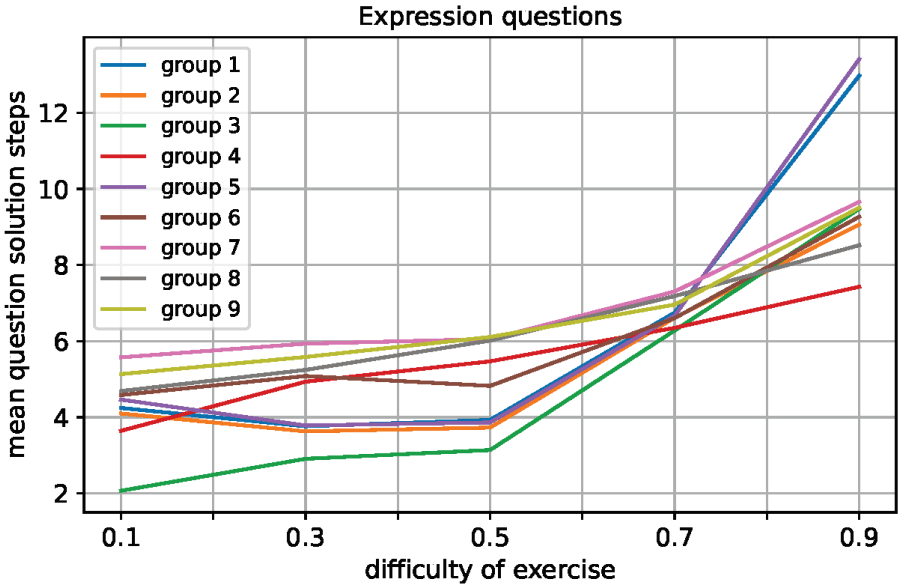


Figure 6. Average number of solution steps per question (Y axis) by exercise difficulty (X axis).

4.3. Ablation Study

In order to verify that all the elements of the presented algorithm were necessary, an ablation study was conducted by turning off essential parts of the algorithm. Three essential components of the algorithm were identified:

1. QR_ADJ—preprocessing of question requests to balance used rules (concepts);
2. SORT_BY_USAGE—taking into account how often a question was used in that exercise before;
3. POST_FILTER—sorting of the found questions by undesirable concepts (rules) and difference to the target question difficulty.

We performed mass exercise generation for the baseline simple random choice among the suitable questions (i.e., the questions of required difficulty without denied concepts and with at least one target concept) which was labeled BASE, the proposed algorithm with all key components turned off (ALL_OFF), the proposed algorithm with one of the components

turned off, and the full version of the proposed algorithm (ALL_ON). Each time, exercises using 9000 questions were generated with different settings. Table 4 shows the number of unique questions in the generated exercises and the statistics on question usage. Figure 7 shows box plots for the relative frequencies of rule usage in different settings.

Table 4. Ablation study of the proposed algorithm: question usage.

Parameter	BASE	ALL_OFF	QR_ADJ Off	SORT_BY_USAGE Off	POST_FILTER Off	ALL_ON
Number of used questions	1530	2964	778	2980	835	3362
Mean number of times a question was used	5.9	3.0	11.6	3.0	10.8	2.7
Maximum number of times a question was used	61	81	142	127	128	98
Maximum number of times a question was used for 50% of questions	4	1	5	1	4	1
Maximum number of times a question was used for 75% of questions	6	3	14	3	11	2

As it can be seen, the naive random question selection (BASE) produced exercises with a relatively low variety of questions (1530 unique questions per 9000 questions in the generated exercises; the median number of times a question was used was 4). Its box plots showed the minimum relative frequency of four rules as zero (which means that there were exercises where these rules were target, but no question on them was included in the attempt, which is unacceptable in practice). For two of them, the lower quartile was zero, and for the "right-associativity" rule even the median was zero (i.e., more than half of exercises targeting the "right-associativity" rule did not have relevant questions).

Introducing the new algorithm significantly increased the number of used questions (lowering question reuse), but the lower quartile for the "right-associativity" rule remained zero. The QR_ADJ component of the algorithm played a chief part in satisfying the condition "every target rule must appear at least once in each exercise", as turning it off brought the "right-associativity" rule to zero. Turning off either QR_ADJ or POST_FILTER resulted in a sharp increase in question re-usage, as the algorithm tried to fulfill the requirements using the same best-fitting questions, which significantly increased the risk of cheating by sharing answers. SORT_BY_USAGE seemed to be the least important component, but turning it on still increased the number of unique questions used from 2980 to 3362 and decreased the maximum usage of one question by almost 25%. Turning SORT_BY_USAGE on also increased the median of occurrence of the less frequent rules, making learners study them more thoroughly.

Thus, the ablation study showed that each component had its role in the performance of the entire algorithm. The proposed algorithm significantly outperformed the baseline random question selection; the baseline algorithm produced practically unacceptable results with unbalanced question banks, totally omitting target rules with low frequency in more than half of the generated exercises.

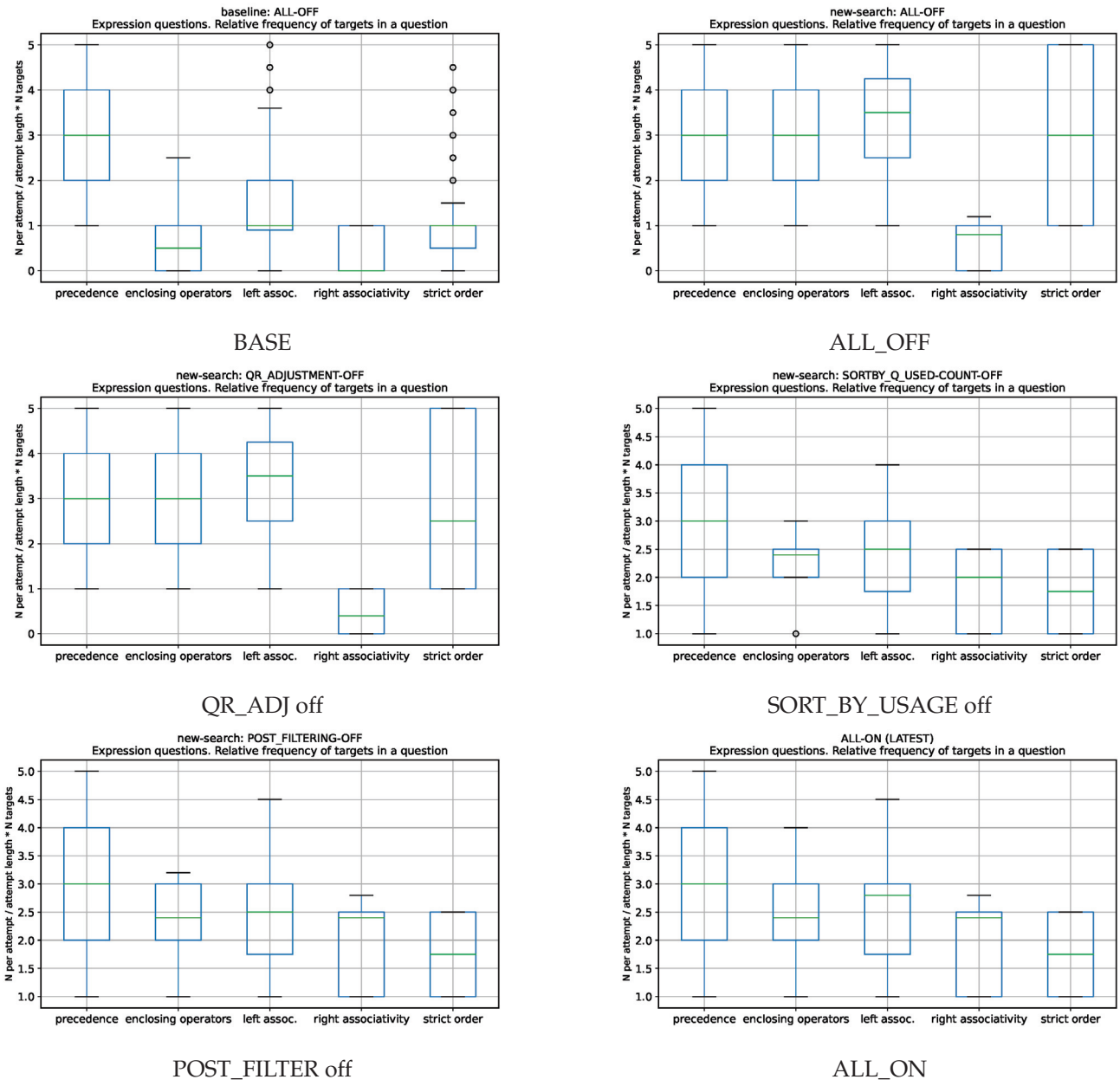


Figure 7. Box plots for relative frequencies of rule usage during the ablation study.

5. Discussion

Large-scale automatic question and learning-problem generation allows us to overcome traditional limitations of human-authored exercises and assignments: it creates a lot of learning items that can be used for unlimited practice until mastery (many questions for one student), avoiding cheating (different questions for different students), and adapting to the needs of a particular student (many questions for different learning situations). Big banks of questions and learning problems that are generated randomly (e.g., [28]) or mined from real-world data (e.g., [39]) can solve that problem because they provide not only vast numbers of generated questions but also significant variability, unlike template-based generation that increases only the number of questions but not the coverage of different topics (e.g., [32]). However, as these big banks come without any classification of questions according to the learning goals and are too big to be observable by a human teacher, they need special algorithms for constructing exercises depending on the teacher's request.

In this study, the teacher's request was formalized in terms of determining the three sets of concepts and subject-domain rules: target (which should be studied and verified), denied (concepts and rules that students are not expected to know at this point), and allowed (known, but not necessary for this exercise) and setting the desired difficulty. The task of creating a balanced exercise based on the request was formalized, and the algorithm for solving it was proposed.

According to the results of the computational experiment, the proposed algorithm guarantees that in an exercise with N_t target concepts and rules consisting of N_q questions, each target concept (rule) appears in questions at least N_q/N_t times, which guarantees the generation of a balanced exercise each time by spreading the learning targets between different questions in the exercise. In practice, the number of target concepts and rules often exceeds the required minimum because questions often concern more than one concept (rule). The number of used rules in retrieved questions depends on the exercise difficulty instead of the number of target concepts in the exercise. It shows that for easier exercises, when it is useful to have fewer concepts per question, the proposed algorithm can rotate target concepts, diversifying questions without increasing their difficulty if the relevant questions are present in the question bank. This shows that the proposed algorithm discerns between the exercise topic (defined by sets of allowed and denied concepts and rules) and exercise difficulty and can find diverse and relevant questions in the given bank.

The algorithm has a good randomization of retrieved questions: even with many exercises with similar settings and generating each exercise 20 times, the mean number of times the same question was used stood below three, and it was mostly achieved by a high usage of a small number of rare questions: 50% of questions were used only once, and 75% of questions were used no more than 2–3 times. This was achieved by taking into account question usage and active usage of questions that were not retrieved yet.

With smaller, less diverse banks, the algorithm sometimes was forced to choose between requested difficulty and target concepts and rules. In that case, the algorithm chose to keep the requested learning topic by deviating from the necessary difficulty. More research may be necessary to make the results fully stable on smaller sets of questions.

The described features of the proposed algorithm allow us to recommend it for the automatic construction of formative (designed for training) and summative (designed for grading) assignments from big, automatically generated question banks. It can also be used for the cold start in adaptive learning systems until enough data for training adaptive algorithms can be collected and for searching diverse questions using requests generated by adaptive systems.

The results of this study improve the results achieved by previous researchers devoted to generating diverse test sheets. The proposed method allows us to replace the hard constraint "no question should be identical in two tests" used by Hwang et al. [56] by the soft constraint "repeat every question as little as possible", which opens the way to a large-scale sheet generation on diverse question banks that cannot support the hard constraint. The algorithm described in this study generated 20 different sheets by re-using rare questions while using more than half of the questions only once. Compared to the results of Sud et al. [11] and Paul [57], the proposed algorithm does not use manual planning of the test sheet and manual marking of the question in the question bank. The proposed algorithm generates tests with flexible composition, which is aimed at a balanced representation of the target concepts, so that the generated test can automatically adapt to changes in difficulty by changing the number of different target concepts per question according to the desired difficulty level. When using the proposed algorithm, the teacher should only specify the requirements for the entire sheet, not the plan of particular question slots as when using previously published algorithms. By only using an automatic classification of questions by the subject-domain concepts and rules whose knowledge is required to answer the questions, the proposed algorithm can be used on far larger question banks consisting of thousands of automatically generated questions that cannot be marked manually.

The limitations of the proposed algorithm are that it does not use the statistics gathered by question usage (e.g., by using the Item Response Theory) unlike, for example, the algorithms proposed in [55,56]. That comes from the algorithm being aimed at large banks of sparsely used questions in which it is impossible to reliably measure item discrimination (and so on) because each item is used very rarely (ideally, only once) which works best for preventing cheating. When such statistics are available, the algorithm should be improved to use it, which can be a good direction of further research.

6. Conclusions

Currently, the methods of generating questions and learning problems are actively researched. The effective usage of the generated questions has received far less attention. This article was intended to close the gap by considering an algorithm of automatic search and retrieval of questions from a large bank and constructing diverse exercises according to teachers' requests.

The article presented an algorithm that can be used to automatically construct complex exercises with desired topics and difficulty. The required metadata for questions in the bank were their estimated difficulty and concepts and subject-domain rules whose knowledge was necessary to answer the question. A question request allowed the teacher to specify the desired difficulty of the exercise, target concepts and rules (those that should be learned), and denied concepts and rules (those that students are not expected to know when doing the exercise). The presented algorithm relied on the metadata in the question bank generated by the question generation engine and can be used in different areas of programming: expression evaluation (see Appendix B), control-flow statements (see the example in the Supplementary Materials), variable scope and lifetime, accessing objects stored in memory, and other topics. The algorithm did not depend on the programming language in which the questions were formulated; in practice, we used tag questions with the programming language so that the questions with the appropriate language are selected according to the teacher's request. Question generation itself was not covered by this algorithm; we used the automatic banks of programming questions generated by the methods described in [9,39]. Those methods provide a good variability in generated questions and generate questions containing natural-looking expressions and algorithms.

The proposed algorithm was based on generating specific question requests for each question in the exercise, taking into account the previously found question, to solve the problem of automatic exercise generation. The algorithm actively used the principle of parameter rotation (keeping track of parameter usage and searching for the less used values) both to rotate target concepts (rules) between the questions (learning problems) in the exercise and to rotate similar questions in different instances of exercises. Rotation helped to achieve more stable results than random selection.

A computational experiment showed that the algorithm satisfied most requirements, including guaranteeing the minimal number of target concepts (rules) in the exercise (so that each question had at least one target and the targets were distributed evenly among them). It did not increase the question difficulty because of the increase in the number of target concepts, increasing the question variety instead. Question difficulty, number of concepts (rules) per question, and solution steps generally increased with exercise difficulty, though the proposed algorithm was not perfect at that. An ablation study showed that all the components of the algorithm improved performance; the most important component for satisfying the strict conditions was the adjustments of question request each time (specific question request generation algorithm).

The algorithm can be used for question retrieval and exercise construction in online learning systems, replacing the traditional "categorization" approach, which is not viable for large question banks with a large number of possible categories. It can also be used in adaptive testing and intelligent tutoring systems because question requests can be generated automatically based on the learner's progress; in that case, the stage of generating specific question requests can be omitted as the request from the adaptive system can

already be treated as a specific request. That algorithm should increase the adoption of large question and problem banks developed during large-scale question and learning problem generation and fill the gap between theory (generating questions) and practice (their usage in the learning process).

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/computers13060144/s1, Data of the attempts generated during the algorithm's evaluation. Three examples of generated exercises for learning control-flow statements with different difficulties. Source code of the described algorithm.

Funding: The reported study has received no funding.

Data Availability Statement: The data presented in this study are available in the Supplementary Materials.

Acknowledgments: I want to express my thanks to Artem Prokudin and especially Mikhail Denisov for their help in producing the question bank that was used for conducting this study.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ASP	Answer Set Programming
BKT	Bayesian Knowledge Tracing
IRT	Item Response Theory
ITS	Intelligent tutoring system
LMS	Learning Management System

Appendix A. Features of Exercises Generated during the Experiment

Table A1. Features of exercises that were generated for expression questions. A—allowed rule, D—denied rule, T—target rule.

#	Difficulty [0...1]	Precedence	Enclosing	Binary Left Assoc.	Binary Right Assoc.	Strict Order
1	0.1	T	A	A	A	A
2	0.3	T	A	A	A	A
3	0.5	T	A	A	A	A
4	0.7	T	A	A	A	A
5	0.9	T	A	A	A	A
6	0.1	A	T	A	A	A
7	0.3	A	T	A	A	A
8	0.5	A	T	A	A	A
9	0.7	A	T	A	A	A
10	0.9	A	T	A	A	A
11	0.1	A	A	T	A	A
12	0.3	A	A	T	A	A
13	0.5	A	A	T	A	A
14	0.7	A	A	T	A	A
15	0.9	A	A	T	A	A
16	0.1	A	A	A	T	A
17	0.3	A	A	A	T	A
18	0.5	A	A	A	T	A
19	0.7	A	A	A	T	A
20	0.9	A	A	A	T	A
21	0.1	A	A	A	A	T
22	0.3	A	A	A	A	T
23	0.5	A	A	A	A	T
24	0.7	A	A	A	A	T
25	0.9	A	A	A	A	T
26	0.1	T	T	A	A	A
27	0.3	T	T	A	A	A
28	0.5	T	T	A	A	A
29	0.7	T	T	A	A	A
30	0.9	T	T	A	A	A
31	0.1	T	T	T	A	A
32	0.3	T	T	T	A	A

Table A1. *Cont.*

#	Difficulty [0…1]	Precedence	Enclosing	Binary Left Assoc.	Binary Right Assoc.	Strict Order
33	0.5	T	T	T	A	A
34	0.7	T	T	T	A	A
35	0.9	T	T	T	A	A
36	0.1	T	T	T	T	A
37	0.3	T	T	T	T	A
38	0.5	T	T	T	T	A
39	0.7	T	T	T	T	A
40	0.9	T	T	T	T	A
41	0.1	T	T	T	T	T
42	0.3	T	T	T	T	T
43	0.5	T	T	T	T	T
44	0.7	T	T	T	T	T
45	0.9	T	T	T	T	T

Appendix B. Examples of Exercises Generated during the Experiment Using Different Difficulty Settings

The following exercises were generated to learn solving the question evaluation order problem with the following target rules: operator precedence, binary left-associative operators, binary right-associative operators, and enclosing operators. The question bank contained questions for those rules with normalized difficulties from 0.53 to 1.0.

Appendix B.1. Low Difficulty

Determine the order of evaluation of the following expressions:

1. sqlite3Strlen30 (pStr2 -> u . zToken) − 1
2. i = j = 0
3. cachedRegex = rkl_lastCachedRegex = (0)
4. nMinKey = nMaxKey = info . nKey
5. pIdx = pProbe = pProbe -> pNext
6. len = i = sqlite3VdbeSerialTypeLen (serial_type)
7. flag_alternateform = flag_altform2 = flag_zeropad = 0
8. rc = db -> errCode = p -> rc
9. * prNotFound = rMayHaveNull = ++ pParse -> nMem
10. pMem -> zMalloc = pMem -> z = z

Appendix B.2. Medium Difficulty

Determine the order of evaluation of the following expressions:

1. (memcpy (& uiwork , (acc + 6) + 34 − 4 , 4) , uiwork) == 0 && (memcpy (& uiwork , (acc + 6) + 34 − 8 , 4) , uiwork) == 0
2. pTabList -> a [0] . iCursor = iCur = pParse -> nTab ++
3. pPager -> sjfd = (pPtr += (((pVfs -> szOsFile) + 7) & 7))
4. flag_leftjustify = flag_plussign = flag_blanksign = flag_alternateform = flag_altform2 = flag_zeropad = 0
5. pMem -> zMalloc = pMem -> z = z
6. u . ad . pMem = p -> pResultSet = & aMem [pOp -> p1]
7. addrBrk = pLevel -> addrBrk = pLevel -> addrNxt = sqlite3VdbeMakeLabel (v)
8. pPage -> cellOffset = cellOffset = hdr + 12 − 4 * pPage -> leaf
9. mxReadMark = pInfo -> aReadMark [i] = pWal -> hdr . mxFrame
10. iCur = pTabList -> a [0] . iCursor = pParse -> nTab ++

Appendix B.3. High Difficulty

Determine the order of evaluation of the following expressions:

1. (as -> src . alpnid == srcalpnid) && Curl_strcasecompare (as -> src . host , srchost) && (as -> src . port == srcport) && (versions & as -> dst . alpnid)

2. lo == 147483648 && hi == 2 && ((((& result) -> words [(16/4) − 1 − (0)]) & 2147483648) != 0)
3. p -> aOp = aOp = u . by . pProgram -> aOp
4. g_use_ansi | | ! isatty (fileno (fp)) | | (− 1) == (stdo = _get_osfhandle (fileno (fp)))
5. p -> iLimit = iLimit = ++ pParse -> nMem
6. * prNotFound = rMayHaveNull = ++ pParse -> nMem
7. u . cb . ctx . pMem = u . cb . pMem = & aMem [pOp -> p3]
8. ((h -> flags [i » 4] » ((i & 15) « 1)) & 1) | | ! ((h -> keys [i]) == (key))
9. g_use_ansi | | ! isatty (fileno (fp)) | | (− 1) == (stdo = _get_osfhandle (fileno (fp))) | | ! GetConsoleScreenBufferInfo (stdo , & csbi)
10. ! ((((df) -> words [(8/4) − 1 − (0)]) & 2080374784) == 2080374784) && DECCOMBMSD [sourhi » 26]

References

1. Baig, M.I.; Shuib, L.; Yadegaridehkordi, E. E-learning adoption in higher education: A review. *Inf. Dev.* **2022**, *38*, 570–588. [CrossRef]
2. Qiao, P.; Zhu, X.; Guo, Y.; Sun, Y.; Qin, C. The Development and Adoption of Online Learning in Pre- and Post-COVID-19: Combination of Technological System Evolution Theory and Unified Theory of Acceptance and Use of Technology. *J. Risk Financ. Manag.* **2021**, *14*, 162. [CrossRef]
3. Mensah, I.K.; Zeng, G.; Luo, C.; Lu, M.; Xiao, Z.W. Exploring the E-Learning Adoption Intentions of College Students Amidst the COVID-19 Epidemic Outbreak in China. *SAGE Open* **2022**, *12*, 21582440221086629. [CrossRef]
4. Coman, C.; Țîru, L.G.; Meseșan-Schmitz, L.; Stanciu, C.; Bularca, M.C. Online Teaching and Learning in Higher Education during the Coronavirus Pandemic: Students' Perspective. *Sustainability* **2020**, *12*, 10367. [CrossRef]
5. Ali, L.; Dmour, N.A.H.H.A. The Shift to Online Assessment Due to COVID-19: An Empirical Study of University Students, Behaviour and Performance, in the Region of UAE. *Int. J. Inf. Educ. Technol.* **2021**, *11*, 220–228. [CrossRef]
6. Mate, K.; Weidenhofer, J. Considerations and strategies for effective online assessment with a focus on the biomedical sciences. *FASEB BioAdvances* **2021**, *4*, 9–21. [CrossRef] [PubMed]
7. Otto, D. Adoption and Diffusion of Open Educational Resources (OER) in Education: A Meta-Analysis of 25 OER-Projects. *Int. Rev. Res. Open Distrib. Learn.* **2019**, *20*, 122–140. [CrossRef]
8. Kurdi, G.; Leo, J.; Parsia, B.; Sattler, U.; Al-Emari, S. A systematic review of automatic question generation for educational purposes. *Int. J. Artif. Intell. Educ.* **2020**, *30*, 121–204. [CrossRef]
9. Sychev, O.; Penskoy, N.; Prokudin, A. Generating Expression Evaluation Learning Problems from Existing Program Code. In Proceedings of the International Conference on Advanced Learning Technologies, ICALT 2022, Bucharest, Romania, 1–4 July 2022; pp. 183–187. [CrossRef]
10. Wang, X.; Fan, S.; Houghton, J.; Wang, L. Towards Process-Oriented, Modular, and Versatile Question Generation that Meets Educational Needs. In Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Seattle, WA, USA, 10–15 July 2022; pp. 291–302. [CrossRef]
11. Sud, P.; West, M.; Zilles, C. Reducing Difficulty Variance in Randomized Assessments. In Proceedings of the 2019 ASEE Annual Conference & Exposition, Tampa, FL, USA, 16–19 June 2019. [CrossRef]
12. Sychev, O.; Anikin, A.; Penskoy, N.; Denisov, M.; Prokudin, A. CompPrehension-Model-Based Intelligent Tutoring System on Comprehension Level. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; 12677 LNCS; Springer: Berlin/Heidelberg, Germany, 2021; pp. 52–59. [CrossRef]
13. Luxton-Reilly, A.; Simon; Albluwi, I.; Becker, B.A.; Giannakos, M.; Kumar, A.N.; Ott, L.; Paterson, J.; Scott, M.J.; Sheard, J.; et al. Introductory Programming: A Systematic Literature Review. In Proceedings of the Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2018 Companion, New York, NY, USA, 2–4 July 2018; pp. 55–106. [CrossRef]
14. Du, X.; Shao, J.; Cardie, C. Learning to Ask: Neural Question Generation for Reading Comprehension. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vancouver, BC, Canada, 30 July–4 August 2017; pp. 1342–1352. [CrossRef]
15. Subramanian, S.; Wang, T.; Yuan, X.; Zhang, S.; Trischler, A.; Bengio, Y. Neural Models for Key Phrase Extraction and Question Generation. In Proceedings of the Workshop on Machine Reading for Question Answering, Melbourne, Australia, 19 July 2018; pp. 78–88. [CrossRef]
16. Qiu, J.; Xiong, D. Generating Highly Relevant Questions. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, 3–7 November 2019; pp. 5983–5987. [CrossRef]
17. Kim, Y.; Lee, H.; Shin, J.; Jung, K. Improving Neural Question Generation Using Answer Separation. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 6602–6609. [CrossRef]

18. Foucher., S.; Pascual., D.; Richter., O.; Wattenhofer, R. Word2Course: Creating Interactive Courses from as Little as a Keyword. In Proceedings of the 14th International Conference on Computer Supported Education CSEDU, Online, 22–24 April 2022; INSTICC, SciTePress: Setúbal, Portugal, 2022; Volume 1, pp. 105–115. [CrossRef]
19. Chen, F.; Xie, J.; Cai, Y.; Wang, T.; Li, Q. Difficulty-Controllable Visual Question Generation. In *Proceedings of the Web and Big Data: 5th International Joint Conference, APWeb-WAIM 2021, Guangzhou, China, 23–25 August 2021, Proceedings, Part I*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 332–347. [CrossRef]
20. Bi, C.; Wang, S.; Xue, Z.; Chen, S.; Huang, Q. Inferential Visual Question Generation. In Proceedings of the 30th ACM International Conference on Multimedia, MM '22, New York, NY, USA, 10–14 October 2022 ; pp. 4164–4174. [CrossRef]
21. Patil, C.; Patwardhan, M. Visual Question Generation: The State of the Art. *ACM Comput. Surv.* **2020**, *53*, 1–22. [CrossRef]
22. Chen, G.; Yang, J.; Hauff, C.; Houben, G.J. LearningQ: A Large-Scale Dataset for Educational Question Generation. In Proceedings of the International AAAI Conference on Web and Social Media, Washington, DC, USA, 25–28 June 2018; Volume 12. [CrossRef]
23. Leo, J.; Kurdi, G.; Matentzoglu, N.; Parsia, B.; Sattler, U.; Forge, S.; Donato, G.; Dowling, W. Ontology-Based Generation of Medical, Multi-term MCQs. *Int. J. Artif. Intell. Educ.* **2019**, *29*, 145–188. [CrossRef]
24. Kumar, A.N. Solving Code-Tracing Problems and Its Effect on Code-Writing Skills Pertaining to Program Semantics. In Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '15, Vilnius, Lithuania, 4–8 July 2015 ; pp. 314–319. [CrossRef]
25. Rodríguez Rocha, O.; Faron Zucker, C. Automatic Generation of Quizzes from DBpedia According to Educational Standards. In Proceedings of the Companion The Web Conference 2018, Republic and Canton of Geneva, CHE, WWW '18, Lyon, France, 23–27 April 2018; pp. 1035–1041. [CrossRef]
26. Thomas, A.; Stopera, T.; Frank-Bolton, P.; Simha, R. Stochastic Tree-Based Generation of Program-Tracing Practice Questions. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education, SIGCSE '19, Minneapolis, MN, USA, 27 February–2 March 2019; pp. 91–97. [CrossRef]
27. Russell, S. Automatically Generated and Graded Program Tracing Quizzes with Feedback. In Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2, ITiCSE '21, Virtual, 26 June–1 July 2021; p. 652. [CrossRef]
28. Martin, B.; Mitrovic, A. Automatic Problem Generation in Constraint-Based Tutors. In Proceedings of the 6th International Conference on Intelligent Tutoring Systems, ITS '02, Biarritz, France and San Sebastian, Spain, 2–7 June 2002; Springer: Berlin/Heidelberg, Germany, 2002; pp. 388–398.
29. O'Rourke, E.; Butler, E.; Tolentino, A.D.; Popović, Z. Automatic Generation of Problems and Explanations for an Intelligent Algebra Tutor. In Proceedings of the 20th International Conference on Artificial Intelligence in Education, Chicago, IL, USA, 25–29 June 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 383–395. [CrossRef]
30. Sychev, O. From Question Generation to Problem Mining and Classification. In Proceedings of the 2022 International Conference on Advanced Learning Technologies (ICALT), Bucharest, Romania, 1–4 July 2022; pp. 304–305. [CrossRef]
31. Sadigh, D.; Seshia, S.A.; Gupta, M. Automating Exercise Generation: A Step towards Meeting the MOOC Challenge for Embedded Systems. In Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education, WESE '12, Tampere, Finland, 12 October 2012. [CrossRef]
32. Shah, H.; Kumar, A.N. A Tutoring System for Parameter Passing in Programming Languages. *SIGCSE Bull.* **2002**, *34*, 170–174. [CrossRef]
33. Hsiao, I.H.; Brusilovsky, P.; Sosnovsky, S. Web-based Parameterized Questions for Object-Oriented Programming. In Proceedings of the E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2008, Las Vegas, NV, USA, 17 November 2008; Bonk, C.J., Lee, M.M., Reynolds, T., Eds.; Association for the Advancement of Computing in Education (AACE): San Diego, CA, USA, 2008; pp. 3728–3735.
34. Ahmed, U.Z.; Gulwani, S.; Karkare, A. Automatically Generating Problems and Solutions for Natural Deduction. In Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI '13, Beijing, China, 3–9 August 2013; AAAI Press: Washington, DC, USA, 2013; pp. 1968–1975.
35. Polozov, O.; O'Rourke, E.; Smith, A.M.; Zettlemoyer, L.; Gulwani, S.; Popovic, Z. Personalized Mathematical Word Problem Generation. In Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15, Virtual, 13–15 April 2021; AAAI Press: Washington, DC, USA, 2015; pp. 381–388.
36. Whitehead, J. Spatial Layout of Procedural Dungeons Using Linear Constraints and SMT Solvers. In Proceedings of the International Conference on the Foundations of Digital Games, FDG '20, Bugibba, Malta, 15–18 September 2020. [CrossRef]
37. Smith, A.M.; Andersen, E.; Mateas, M.; Popović, Z. A Case Study of Expressively Constrainable Level Design Automation Tools for a Puzzle Game. In Proceedings of the International Conference on the Foundations of Digital Games, FDG '12, Raleigh, NC, USA, 29 May–1 June 2012; pp. 156–163. [CrossRef]
38. Kumar, A.N. Generation of Problems, Answers, Grade, and Feedback—Case Study of a Fully Automated Tutor. *J. Educ. Resour. Comput.* **2005**, *5*, 3–es. [CrossRef]
39. Sychev, O.; Prokudin, A.; Denisov, M. Generation of Code Tracing Problems from Open-Source Code. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2023, Toronto, ON, Canada, 15–18 March 2023; pp. 875–881. [CrossRef]
40. Baker, F.B.; Kim, S.H. *Item Response Theory: Parameter Estimation Techniques*; CRC Press: Boca Raton, FL, USA, 2004.

41. Johns, J.; Mahadevan, S.; Woolf, B. Estimating Student Proficiency Using an Item Response Theory Model. In *Proceedings of the Intelligent Tutoring Systems*; Ikeda, M., Ashley, K.D., Chan, T.W., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 473–480.
42. Corbett, A.T.; Anderson, J.R. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Model. User-Adapt. Interact.* **1995**, *4*, 253–278. [CrossRef]
43. Martori, F.; Cuadros, J.; González-Sabaté, L. Studying the Relationship between BKT Fitting Error and the Skill Difficulty Index. In Proceedings of the Sixth International Conference on Learning Analytics & Knowledge, LAK '16, Edinburgh, UK, 25–29 April 2016; pp. 364–368. [CrossRef]
44. Pardos, Z.A.; Heffernan, N.T. Modeling Individualization in a Bayesian Networks Implementation of Knowledge Tracing. In *Proceedings of the User Modeling, Adaptation, and Personalization*; De Bra, P., Kobsa, A., Chin, D., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 255–266.
45. Yudelson, M.V.; Koedinger, K.R.; Gordon, G.J. Individualized Bayesian Knowledge Tracing Models. In *Proceedings of the Artificial Intelligence in Education*; Lane, H.C., Yacef, K., Mostow, J., Pavlik, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 171–180.
46. Pardos, Z.A.; Heffernan, N.T. KT-IDEM: Introducing Item Difficulty to the Knowledge Tracing Model. In *Proceedings of the User Modeling, Adaption and Personalization*; Konstan, J.A., Conejo, R., Marzo, J.L., Oliver, N., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 243–254.
47. Meng, L.; Zhang, M.; Zhang, W.; Chu, Y. CS-BKT: Introducing item relationship to the Bayesian knowledge tracing model. *Interact. Learn. Environ.* **2021**, *29*, 1393–1403. [CrossRef]
48. Piech, C.; Bassen, J.; Huang, J.; Ganguli, S.; Sahami, M.; Guibas, L.J.; Sohl-Dickstein, J. Deep Knowledge Tracing. In *Proceedings of the Advances in Neural Information Processing Systems*; Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2015; Volume 28.
49. Chavarriaga, O.; Florian-Gaviria, B.; Solarte, O. A Recommender System for Students Based on Social Knowledge and Assessment Data of Competences. In *Proceedings of the Open Learning and Teaching in Educational Communities*; Rensing, C., de Freitas, S., Ley, T.; Muñoz-Merino, P.J., Eds.; Springer: Cham, Switzerland, 2014; pp. 56–69.
50. Wang, S.; Ma, Z.; Ji, H.; Liu, T.; Chen, A.; Zhao, D. Personalized exercise recommendation method based on causal deep learning: Experiments and implications. *STEM Educ.* **2022**, *2*, 157–172. [CrossRef]
51. Troussas, C.; Krouska, A. Path-Based Recommender System for Learning Activities Using Knowledge Graphs. *Information* **2023**, *14*, *9*. [CrossRef]
52. Da Silva, F.L.; Slodkowski, B.K.; da Silva, K.K.A.; Cazella, S.C. A systematic literature review on educational recommender systems for teaching and learning: Research trends, limitations and opportunities. *Educ. Inf. Technol.* **2022**, *28*, 3289–3328. [CrossRef] [PubMed]
53. Hussain, M.; Zhu, W.; Zhang, W.; Abidi, S.M.R.; Ali, S. Using machine learning to predict student difficulties from learning session data. *Artif. Intell. Rev.* **2018**, *52*, 381–407. [CrossRef]
54. Pereira, F.D.; Fonseca, S.C.; Wiktor, S.; Oliveira, D.B.F.; Cristea, A.I.; Benedict, A.; Fallahian, M.; Dorodchi, M.; Carvalho, L.S.G.; Mello, R.F.; et al. Toward Supporting CS1 Instructors and Learners With Fine-Grained Topic Detection in Online Judges. *IEEE Access* **2023**, *11*, 22513–22525. [CrossRef]
55. Hwang, G.J.; Lin, B.M.; Lin, T.L. An effective approach for test-sheet composition with large-scale item banks. *Comput. Educ.* **2006**, *46*, 122–139. [CrossRef]
56. Hwang, G.J.; Chu, H.C.; Yin, P.Y.; Lin, J.Y. An innovative parallel test sheet composition approach to meet multiple assessment criteria for national tests. *Comput. Educ.* **2008**, *51*, 1058–1072. [CrossRef]
57. Paul, D.V. Metaheuristic Algorithms for Designing Optimal Test Blueprint. *Comput. Sist.* **2020**, *24*, 1627–1642. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

computers

MDPI

Article

Bridging the Gap between Project-Oriented and Exercise-Oriented Automatic Assessment Tools

Bruno Pereira Cipriano [1,*], Bernardo Baltazar [1], Nuno Fachada [1,2], Athanasios Vourvopoulos [3] and Pedro Alves [1]

[1] ECATI, Lusófona University, Campo Grande, 376, 1749-024 Lisboa, Portugal; a21704025@alunos.ulht.pt (B.B.); nuno.fachada@ulusofona.pt (N.F.); pedro.alves@ulusofona.pt (P.A.)
[2] COPELABS, Lusófona University, Campo Grande, 376, 1749-024 Lisboa, Portugal
[3] Institute for Systems and Robotics—Lisboa, Instituto Superior Técnico, Universidade de Lisboa, Av. Rovisco Pais, 1049-001 Lisboa, Portugal; athanasios.vourvopoulos@tecnico.ulisboa.pt
* Correspondence: bcipriano@ulusofona.pt

Abstract: In this study, we present the DP Plugin for IntelliJ IDEA, designed to extend the Drop Project (DP) Automatic Assessment Tool (AAT) by making it more suitable for handling small exercises in exercise-based learning environments. Our aim was to address the limitations of DP in supporting small assignments while retaining its strengths in project-based learning. The plugin leverages DP's REST API to streamline the submission process, integrating assignment instructions and feedback directly within the IDE. A student survey conducted during the 2022/23 academic year revealed a positive reception, highlighting benefits such as time efficiency and ease of use. Students also provided valuable feedback, leading to various improvements that have since been integrated into the plugin. Despite these promising results, the study is limited by the relatively small percentage of survey respondents. Our findings suggest that an IDE plugin can significantly improve the usability of project-oriented AATs for small exercises, informing the development of future educational tools suitable for mixed project-based and exercise-based learning environments.

Keywords: automated assessment; computer science education; programming education; hci; IntelliJ IDEA

1. Introduction

Automatic assessment tools (AATs) are software systems that automatically grade programming assignments. These tools are common in computer science and engineering courses, providing recognized benefits for students, as automatic feedback improves their autonomy. Additionally, they are beneficial for educators, as they reduce assessment time and promote grade consistency [1–4]. However, many of these tools are course- or institution-specific and are not easily adaptable to other scenarios due to a lack of documentation or unavailability of the tool's binary or source code, among other issues [5, 6]. Furthermore, existing commercial tools (e.g., repl.it and HackerRank) are usually web-based systems, which do not align well with real-world scenarios where integrated development environments (IDEs), such as NetBeans and IntelliJ IDEA, are prevalent [7,8].

Some courses use an exercise-based approach, requiring students to solve a number of small assignments (e.g., weekly) [9,10], while others use a project-based methodology, where students develop larger projects that can take multiple weeks or even months to complete [10,11]. This duality is also observable in many AATs available in the market or described in the literature. For example, most web-based AATs are better suited for small exercises, since larger projects usually require multiple files. Although some web editors support multiple files, we argue that this is not ideal from a usability and tool perspective—not to mention their inadequacy in preparing students for the job market, as they create artificial environments that do not compare with the software development

workflows commonly found in professional settings. On the other hand, AATs designed for project assessment often require impractical submission legwork for the grading of small exercises.

In turn, several courses follow a mixed approach, using both small exercises and larger projects as teaching and assessment methods [10,12,13]. This is the case for several courses taught by the authors, where automatic assessment is performed with Drop Project (DP), an AAT originally designed to support the development and grading of larger projects. Furthermore, educators using this teaching approach sometimes have to use multiple AATs within the same course [13], requiring both students and teachers to interact with multiple systems.

DP supports submissions via web upload or using a Git repository, requiring the inclusion of an AUTHORS.txt file with the name of the student or students. Additionally, in the web upload method, code and supporting files must be submitted in a ZIP file. Therefore, DP is not ideal from a usability perspective in scenarios involving smaller exercises. For example, one of our courses employs a 'mini-test' in which students are asked to implement several functions in one or—at most—two files. These 'mini-tests' are taken during laboratory classes, and students have up to one hour to solve them. Students submit their solutions through the web upload method, since version control with Git is yet to be introduced. Also, given the pressure of being assessed, students sometimes make small, time-wasting mistakes, potentially resulting in grades that do not reflect their actual level of knowledge. Examples of such mistakes include creating the `AUTHORS.txt` file with an incorrect format or problems creating the ZIP file, which easily lead to precious minutes being wasted.

To streamline this procedure and bridge the gap between a project-oriented AAT and its exercise-oriented utilization, we propose the DP Plugin for the IntelliJ IDEA IDE. This plugin automates submission-related perfunctory tasks, such as creating the required documentation files and compressing and uploading the solution, while displaying assignment instructions and submission reports directly within the IDE. The plugin reduces the cost of context switching, allowing students to focus on the exercises without leaving the IDE. The proposed plugin was developed using an agile methodology, with students directly involved in the coding and testing process. This work aims to address whether it is possible to improve the usability of a project-oriented AAT via an IDE plugin, enhancing its capability in assisting students in solving small exercises.

This paper is organized as follows. In Section 2, we review existing literature, highlighting how several popular AATs manage project- and exercise-based assessments. Section 3 describes the materials used in this work. Specifically, Section 3.1 examines the DP project-oriented AAT and its limitations regarding small exercises, while Section 3.2 introduces the DP Plugin for IntelliJ IDEA, providing technical details and illustrating how it aims to address the aforementioned issues with DP by keeping students engaged within the IDE and minimizing cursory submission tasks. Section 4 describes the research methodology, which consists of a student survey to assess the plugin's impact on students' assignment submission experiences. The results of this survey are presented in Section 5, while Section 6 offers a discussion of these results, considering their implications and potential limitations. Finally, Section 7 summarizes our findings and offers directions for future research.

2. Related Work

Project-based (PB) learning describes a course structure in which students work towards finding solutions to real-world problems [10,14]. These problems are represented as projects that often span the entire course duration. On the other hand, in exercise-based (EB) courses, students solve questions with predetermined answers (i.e., small coding questions) [10]. In EB courses, individual assignments have shorter durations and are typically less creative than those used in PB learning [10]. PB learning is not a new idea, and research has shown that engineering students tend to be more motivated when working with real-life case studies and challenging open-ended questions [15,16], which are more easily

implemented with projects rather than small exercises. A more recent study evaluated PB learning practices in several engineering degrees and noted that while students who participate in PB learning are generally motivated and demonstrate better teamwork and communication skills, they also display a less rigorous understanding of engineering fundamentals [17]. The same study concluded that a mixed approach of traditionally lectured courses with both PB and EB components might be the best way to satisfy industry needs without sacrificing knowledge of engineering fundamentals.

The use of AATs is common in computer science courses for the automatic assessment of PB and/or EB assignments [18], with a variety of available commercial and academic systems. However, many of these are proprietary and/or not packaged in a production-ready fashion, making their adoption costly or impractical. Previous research has shown that some of these AATs were created as part of a thesis or are sufficiently developed to study a specific research question or support the needs of a particular course but are generally not suitable for widespread deployment [3,5]. However, the situation appears to be improving, with some of the more recent developments being open-source [3], which could lead to more widespread deployment of these tools.

Despite these issues, some relevant tools should be considered when adopting an AAT. For instance, repl.it is an online tool where teachers can create assignments that students solve directly in its web-based IDE [19]. HackerRank [20] is a similar platform, originally developed for competitive programming and programming interviews. While such services might be practical for small exercises, they are not ideal for mid- to large-sized, multi-file projects, as they offer environments far removed from common professional development workflows [7]. Another relevant system is Mooshak [21,22], which supports programming contests and allows submissions via the upload of a single file, making it more appropriate for validating small exercises. However, Raccode, Mooshak's companion plugin, allows students to submit a code file directly from the NetBeans IDE [7]. Web-CAT [23] is an AAT where solutions can be uploaded via a browser or submitted through a NetBeans IDE plugin, making it apparently appropriate for handling both small exercises and projects. However, Web-CAT's code is spread over various repositories, and the most recent binary was last updated in November 2008. Although Web-CAT's documentation seems comprehensive, at the time of writing, multiple links on its GitHub-hosted website were broken, making it hard to recommend. A more recent AAT, check50 [24], is a command-line tool that uses a domain-specific language to write tests for student code and does not enforce any specific structure (e.g., mandatory functions). This makes it inappropriate for large projects, which need to be well-structured following software engineering best practices. The Canary Framework (CF) AAT follows a continuous integration approach, mimicking real professional settings [8], and allows for interaction via commits to a Git repository. This approach is more suitable for PB assessments. However, the need to teach Git skills might not be feasible in certain academic contexts, such as introductory-level courses. Additionally, CF is not a single integrated piece of software but a set of six independent tools that can be combined to achieve the intended goals, and the source code for several of these tools does not seem to be available. Finally, DP is an AAT originally developed to support the development of projects [2], offering submissions through ZIP file upload or via a Git repository, making it less than ideal for smaller exercises, although the plugin discussed in this paper aims to address that issue. DP is fully open-source and can be readily deployed using a Docker image, for example [25].

The discussed tools and their main characteristics, including DP with the proposed plugin, are summarized in Table 1.

Table 1. Summary and main characteristics of the discussed AATs, including Drop Project featuring the proposed plugin. "Open?" indicates whether the tool is open-source. "EB?" and "PB?" refer to suitability for exercise-based and project-based approaches, respectively. The term "web-IDE" is used as a shorthand for "online code editors".

Tool	Open?	Installation	EB?	PB?	Beginner-Friendly?
repl.it	No	N/A	Yes	No	Yes (web-IDE)
HackerRank	No	N/A	Yes	No	Yes (web-IDE)
Mooshak	Yes	Pre-compiled JAR file	Yes	No	Yes (submissions via upload)
Web-CAT	Yes	Pre-compiled WAR file (from 2008); source install requires compilation of multiple sub-systems	Yes [a]	Yes	Yes (submissions via upload and IDE plugin)
check50	Yes	Requires installation of multiple tools	Yes	No	Yes
Canary framework	Partially [b]	Requires installation of multiple tools	No	Yes	No (only Git submissions)
VPL	Yes	Must be installed by Moodle's administrator	Yes	No	Yes (submissions via upload and web-IDE)
Drop Project	Yes	WAR file compiled from latest sources/one-click deploy to Heroku (single tool)	Yes [c]	Yes	Yes (submissions via upload and Git)

[a] With EB plugin for NetBeans IDE. [b] Since the code for half the repositories is not available. [c] With the proposed plugin for IntelliJ IDEA.

From our investigation, the AAT that comes closest to being suitable for both small exercises and projects is Web-CAT. However, it does not appear to be easy to install, deploy, and/or use due to a lack of updated binaries and documentation. In summary, all of the AATs discussed so far have one or more types of limitations, which can be summarized as follows:

- They are difficult to install, deploy, and/or use;
- They are inappropriate for beginner students, e.g., by only allowing submissions using Git;
- They are unsuitable for assessing and evaluating multi-file projects (i.e., PB learning);
- They are unsuitable for assessing and evaluating small exercises (i.e., EB learning).

DP suffers from the last of these issues, making it an imperfect choice for small exercises. The proposed DP Plugin for IntelliJ IDEA aims to solve this problem, fostering the use of a single assessment tool in mixed courses (EB and PB), minimizing tool switching, and allowing teachers and students to save time and avoid problems with cursory tasks.

3. Materials

3.1. Drop Project

DP is an open-source AAT designed and implemented to support the evaluation of academic software projects [2]. It is a mature tool, used in multiple courses for over five years. The following list, adapted from [2], highlights a subset of DP's features.

- Supports individual and group submissions;
- JVM and Maven-based, supporting Java and Kotlin out of the box, with easy integration of other JVM-based languages such as Clojure and Scala;
- Features a waiting time between submissions, termed "Cool-off period", which can be set to limit students' trial-and-error attempts [26];
- Optional assignment-specific access-control lists to ensure only authorized students can access certain assignments;
- Supports web upload and Git submissions, allowing teachers to adapt the submission procedure to the students' experience level;
- Teachers can define assignments and tests using all strategies available in JUnit (e.g., unit tests, input/output matching, and so on). Tests can be public (students receive feedback) or private/hidden (students receive no feedback);
- Can run and assess student-defined unit tests, as well as measure the respective coverage;

- Besides performing functional validations, DP can check the quality of students' code using teacher-defined style rules. It can also restrict keywords, like forbidding loops in exercises requiring recursive implementations;
- Teachers can download students' submissions in Maven format and easily import them into an IDE, facilitating the debugging process;
- Students' results can be exported to a CSV file.

These and other functionalities make DP a good option when choosing an AAT for PB learning. However, as mentioned earlier, it is not ideal for supporting students with small exercises due to the legwork required by both submission types.

On a more technical note, DP also provides a REST API that can be used to integrate external tools (https://deisi.ulusofona.pt/drop-project/swagger-ui/, accessed on 26 June 2024). This API provides functionalities such as assignment listing, assignment presentation, and solution submission. The API implements the same security and access control measures as DP itself. For example, the 'List of public assignments' endpoint only returns assignments to which all students have access. To access a private assignment, the student must supply the respective assignment ID, and the API will check if the given student has access to that assignment before returning its contents. If an assignment has a defined "cool-off" period, the API will check if the time for a certain student has passed before allowing that student to perform new submissions.

A video demonstration of DP's workflow from both the teacher's and the student's points of view is available [27].

3.2. DP Plugin for IntelliJ IDEA

The DP Plugin for IntelliJ IDEA presented in this paper leverages DP's REST API, providing an abstraction layer on top of DP's web upload submission method, simplifying parts of this process. Figure 1 presents an overview of how the various components integrate and communicate with each other.

The plugin is integrated into IntelliJ IDEA's main window (Figure 2), allowing students to simultaneously view their code, the assignment instructions (Figure 3), and the assessment results of their latest submission (Figure 4). The plugin uses assignments created for DP [2]. Therefore, there is no additional assignment creation overhead for teachers already familiar with DP, while students can still use DP's standard submission methods if they so desire.

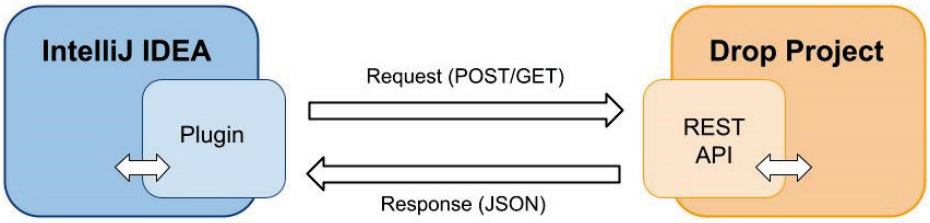


Figure 1. Overview of the solution's data flow. The plugin integrates with IntelliJ IDEA and connects to DP via its REST API.

The plugin also comes with its own security measures. For instance, the feature that updates the assignment list will only make an API request if a specific number of seconds has passed since the last update request.

The plugin was developed during the 2022/23 academic year. In both semesters, development versions of the plugin were deployed and distributed among students during selected classes of specific courses to test and obtain feedback. These efforts yielded a production-ready version of the plugin, which is available for download at IntelliJ's marketplace (https://plugins.jetbrains.com/plugin/21870-drop-project, accessed on 26 June 2024). The tool's source code is also available on GitHub (https://github.com/drop-project-

edu/Drop-Project-for-Intellij-Idea, accessed on 26 June 2024). A video demonstration of the proposed plugin is available [28].

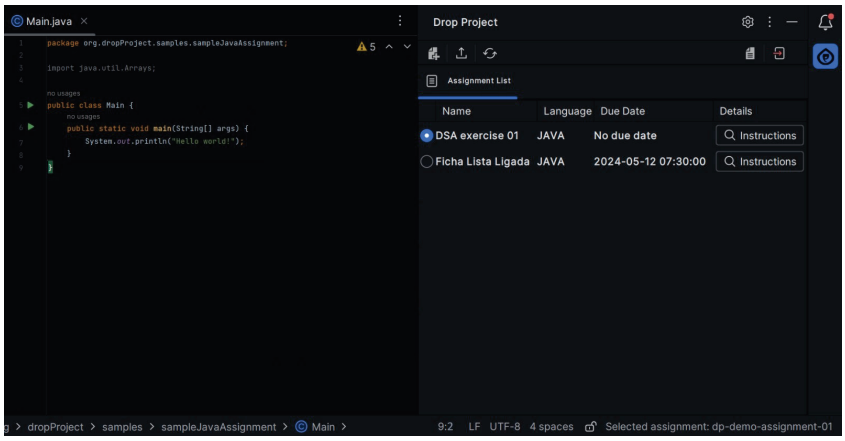


Figure 2. The plugin integrated into IntelliJ IDEA's main window. The plugin lists the assignments available to the student. The 'Instructions' button allows students to access the instructions for each available assignment (see Figure 3).

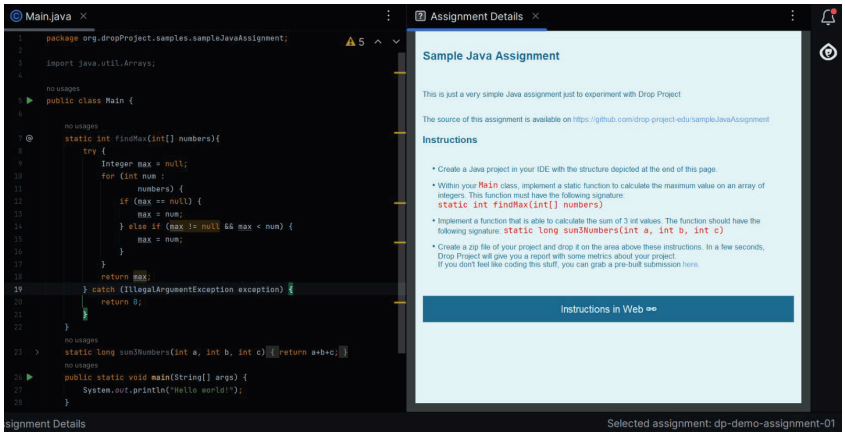


Figure 3. The assignment instructions inform students of the tasks they need to perform. Students can see their code and the assignment instructions in the same window, reducing the need for context switching.

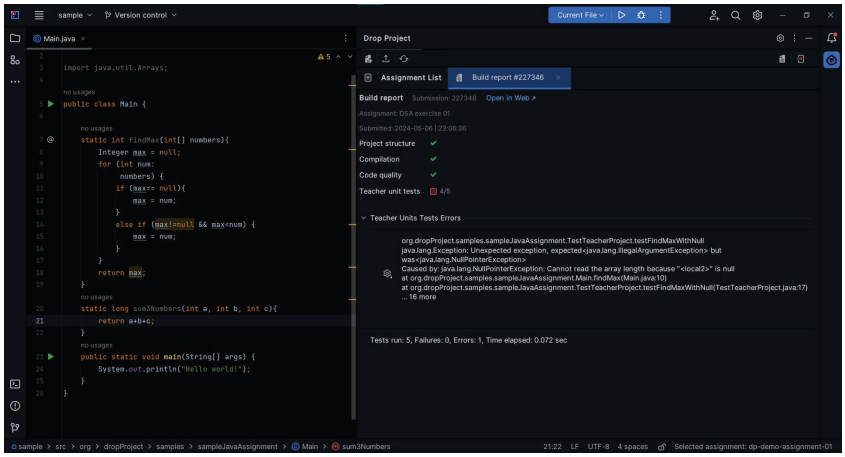


Figure 4. The submission report informs students of their progress and presents them with any unit test failures. Students can see their code and its issues in the same window.

4. Research Methods

A student survey was conducted to evaluate the plugin in the context of a Data Structures and Algorithms course during the second semester of the 2022/23 academic year, i.e., during the final stages of the plugin's development. The survey comprised both qualitative and quantitative questions. The questions aimed to assess students' perceptions regarding various aspects of the plugin, including its impact on reducing submission time, students' confidence in the plugin, the value of having specific reporting functionalities integrated into the IDE, and the psychological impact of using the plugin in an evaluation environment. The research context is further detailed in Section 4.1, while the survey questions are presented in Section 4.2.

4.1. Research Context

The Data Structures and Algorithms course follows a mixed approach, utilizing both PB and EB learning methodologies, with DP used for automatic assessment. All students in this course have interacted with DP, using its web interface, as DP is also used in the previous programming course of the curriculum (i.e., Programming Fundamentals).

This course employs multiple evaluation mechanisms. One of them—weekly assignments—aims to promote student participation and engagement, contributing 5% to the final course grade. The course also includes 'mini-tests', which are one-hour assessments where students implement various related data structures and functions under the supervision of a teacher. Each 'mini-test' has a weight of 6.75% in the final course grade.

4.2. Survey Questions

Students were divided into the following two main groups, with specific questions for each group:

Adopters: Students who identified as regular plugin users;

Non-adopters: Students who reported not having used the plugin.

Students who self-identified as regular plugin users (i.e., adopters) were asked multiple plugin-specific questions, with answers given on a 5-level Likert scale, as shown in Table 2. Considering that students who self-identified as regular plugin users may not have actually used the plugin during the 'mini-test' assessment, questions 7 and 8 in Table 2 were only available for students who did so.

Table 2. Survey questions for students who adopted the plugin. Questions 1–6 were answered by all adopter students, while questions 7–8 were only answered by students who used the plugin during the second 'mini-test'. Questions were answered on a 5-level Likert scale.

ID	Question
1	Rate the plugin's effectiveness in simplifying the DP submission process.
2	Rate the plugin in terms of saving time and/or effort when compared to the traditional method (web upload).
3	Rate the relevance of the automatic creation of the AUTHORS.txt file and the automatic submission of the ZIP file with the '/src' folder and AUTHORS.txt.
4	Rate your confidence in submitting exercises through the plugin.
5	Rate the positive effects on your productivity from the ability to see assignment instructions directly in the IDE.
6	Rate the positive effects on your productivity from the ability to view a submission's build report directly in the IDE.
7	Rate the contribution of using the Plugin in reducing anxiety and/or nervousness during the 'mini-test'.
8	Rate the contribution of using the Plugin in improving your performance during the 'mini-test'.

To gain additional insights into plugin usage (or lack thereof) during the semester, three open-ended questions (OE1–OE3), as shown in Table 3, were posed to specific student groups. First, to understand why some students who self-identified as plugin users (i.e., adopters) did not use the plugin during the 'mini-test', we asked for their reasons (open-ended question OE1, Table 3). Second, all adopters answered a qualitative question to leave any relevant comments (open-ended question OE2, Table 3). Finally, we asked the

students who had not tried the plugin (i.e., non-adopters) for their reasons for not doing so (open-ended question OE3, Table 3).

Table 3. Open-ended questions posed to students in the specified target groups.

ID	Question	Target Student Group
OE1	If you did not use the DP Student Plugin in the 'mini-test': explain why.	Adopters that did not use the plugin during the 'mini-test'.
OE2	Leave any comment that you find relevant.	Adopters.
OE3	If you do not use the plugin, explain why.	Non-adopters.

5. Results

The survey was conducted during the last two weeks of the second semester of the 2022/23 academic year. Out of a total of 154 students, 35 participated in the survey (22.7%). Among these respondents, 20 were identified as adopters of the plugin, while the remaining 15 were non-adopters. Furthermore, of the 20 adopters, half (10 students) used the plugin during the 'mini-test', whereas the other half did not. Figure 5 summarizes this breakdown.

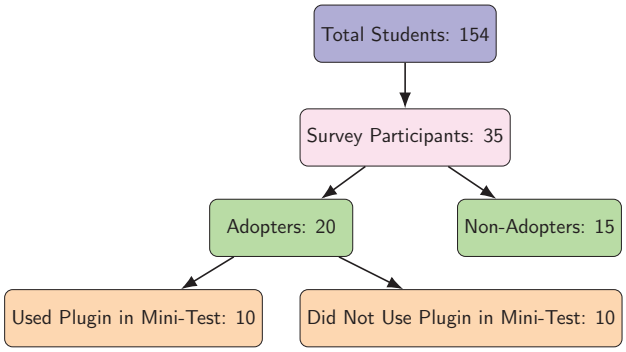


Figure 5. Hierarchical breakdown of student participation and plugin adoption based on the survey responses.

5.1. Impact of the Plugin on Adopters

Table 4 shows the distribution and summary statistics for the replies given by adopters of the plugin. Replies to questions 1–6, which tend toward the highest possible evaluation, indicate that students found the plugin useful. Questions 1–4 have mean scores of at least 4.5, all with modes equal to 5. However, questions 5 and 6—regarding the assignment instructions and submission report displayed within the IDE—had slightly lower scores, with averages of 3.8 (mode 4) and 4.1 (mode 5), respectively. Students do not seem to greatly value having this information within the IDE, particularly the assignment instructions. We believe this is due to the richer formatting of these instructions in the web environment, which displays information such as required function prototypes in a colored format.

Question 7 asked if the plugin helps reduce students' anxiety and/or nervousness during the 'mini-test'. The mean score given by the participants was 3.6, indicating a generally positive response towards the plugin's efficacy. The standard deviation of 1.173 and the histogram distribution denote some variability in the participants' experiences and perceptions. Interestingly, the data were bimodal, with modes at both 3 and 5. This indicates that while a significant number of participants felt the plugin had a moderate effect (rating of 3), there was also a substantial group that found it highly effective (rating of 5) in reducing anxiety and nervousness. As for question 8, which focused on the plugin's effect on the students' performance, the mean response was 3.7, suggesting that many participants felt the plugin had a positive impact. The longer-tailed distribution (standard deviation of 1.159) indicates a moderate spread in the responses, meaning that while many participants agreed with the statement, there was still some variability in opinions. Notably, both options 4 and 5 were selected with the same frequency above all other options. This

indicates that a significant number of participants either agreed or strongly agreed with the statement, highlighting the perceived effectiveness of the plugin during the 'mini-test'.

Table 4. Histogram and summary statistics for the questions posed to plugin adopters. Questions 1–6 were answered by the 20 students who reported using the plugin, while questions 7–8 were only answered by the 10 students who reported using the plugin during the second 'mini-test'. All questions were answered using a 5-level Likert scale. Histograms are normalized to the maximum count value for visualization purposes.

ID	Histogram	Count	Mean	Std. Dev.	Mode
1		20	4.6	0.680	5
2		20	4.5	0.827	5
3		20	4.3	0.978	5
4		20	4.6	0.598	5
5		20	3.8	1.105	4
6		20	4.1	1.160	5
7		10	3.6	1.173	3; 5
8		10	3.7	1.159	4; 5

5.2. Reasons for Not Using the Plugin during the 'Mini-Test'

Nine out of ten adopter students who did not use the plugin during the 'mini-test' replied to question OE1 (Table 3). These replies were analyzed qualitatively to find common reasons for not using the plugin during the assessment.

The predominant topic found in the responses centered around issues arising from the compatibility between the version of IntelliJ IDEA that certain students were using and the plugin itself. This concern was raised by three students, leading them to opt against utilizing the plugin during the assessment. The remaining six students reported different issues that could not be grouped into any specific topic. For example, one student mentioned, *"I didn't think it was necessary and I wanted to be confident that I had sent it correctly and be able to see the results and errors immediately after the page loaded"*. Another student commented, *"I had to log in, and the instructions in the IDE are not as optimized and easy to read"*, while another student indicated, *"I changed computers and forgot to install it"*. Notably, one student commented on the instructions' readability, which is consistent with the lower mean result of the respective qualitative question (i.e., question 5, mean = 3.8).

5.3. Open-Ended Comments about the Plugin

A total of 10 students (out of 20 adopters) answered question OE2 (Table 3). Again, students' responses were analyzed qualitatively to find common topics. The following three recurring topics were found: (1) positive comments about the plugin (e.g., "useful", "simplifies", "saves time") (four students), (2) suggestions for improving the formatting of the assignment instructions (two students), and (3) suggestions to avoid the need to log in again when changing assignments and/or when re-opening the IDE (two students). It should be noted that the suggestion about saving login data between uses has since been implemented.

5.4. Reasons for Not Adopting the Plugin

Thirteen out of 15 students replied to question OE3 (Table 3). A qualitative analysis highlighted the followin two main topics: (1) "I'm not interested in it'/"I don't feel the need" (six students) and (2) "I don't know how to" (four students). An example reply that falls under the first topic is, "I never thought it was necessary, although I give credit and think it is useful", while an example response for the second topic is, "I don't understand very well how to use it. I think a video explanation would be useful".

6. Discussion and Limitations

As the evaluation survey shows, the tool was well received by our students, who identified a number of positive effects, such as time savings. Students also provided interesting suggestions, some of which have already been implemented, such as the plugin remembering login details. Our findings suggest that an IDE plugin does improve the usability of a project-oriented AAT for small exercises, thus informing the design of future AATs suitable for mixed learning environments combining PB and EB assessment. The positive reception of the plugin highlights its potential to enhance the learning experience by streamlining workflows and reducing the time required for administrative tasks (e.g., ZIP file creation), allowing students to focus more on learning the material.

Results also show that there are two main types of non-adopters, namely (1) students who are happy and/or comfortable using DP's web interface to the point that they are not open to trying new interaction formats and (2) students who need some help in order to be able to use the plugin. For the former, we believe they are acting out of habit, since their initial exposure to DP was through the web interface. This will likely change in the future, as students also have the plugin available in the preceding Programming Fundamentals course. Regarding the latter group, who need more help, a video explaining the plugin's installation and usage has since been made available [28].

Moreover, the integration of student suggestions into the plugin development process demonstrates the value of student feedback in creating more effective educational tools. The implementation of simple features such as remembering login details shows how even minor enhancements can improve the overall user experience, potentially leading to a more user-friendly and efficient learning environment. Additionally, student feedback raises awareness of the need for documentation (e.g., demonstration videos) to help learners use these educational tools effectively.

Given these results, we believe the proposed IDE plugin effectively extends DP to be a more practical AAT for small exercises, bridging the gap between its PB roots and its EB usage. As discussed in Section 2, other AATs have proposed IDE plugins. However, Mooshak's IDE plugin—Raccode—simply enhances the existing EB capabilities of Mooshak, itself an EB AAT. Web-CAT, a PB-oriented AAT, does provide an IDE plugin for improving its EB capabilities. However, the issues with Web-CAT—poorly organized code, out-of-date binaries, and missing documentation—make it a difficult choice as a modern and versatile AAT.

Nonetheless, our work presents several limitations. First, while the survey indicates that the majority of students find the plugin useful and believe it improves their workflows, the number of students who answered the survey was relatively small, which presents some threats to validity. Also, since participation was optional, there might have been some self-selection bias, as students who opted to participate in the survey might have been the ones more interested or motivated to discuss the plugin. However, we do not believe this to be the case, since approximately 42.5% of the responses were from students who did not use the plugin.

Furthermore, a student survey may not be sufficient to accurately measure the pedagogical impact of the proposed plugin. To gain a more comprehensive understanding and increase confidence in the plugin's pedagogical value, it would be beneficial to complement the current survey data with other research methods, such as direct classroom observations and analysis of students' academic results over time.

7. Conclusions

In this paper, we proposed the DP Plugin for IntelliJ IDEA, a companion tool for the DP AAT, with the goal of making it more apt to handle small exercises used in EB learning. As the evaluation survey shows, the tool was well received by our students, who identified a number of positive effects, such as time savings. Students also provided interesting suggestions, some of which have already been implemented, such as the plugin remembering login details. Our findings suggest that an IDE plugin does improve the

usability of a project-oriented AAT for small exercises, thus informing the design of future AATs suitable for mixed learning environments combining PB and EB assessment.

Despite the promising feedback on the plugin's effectiveness, our study has some limitations. The small number of survey participants raises concerns about validity, and the optional nature of the survey could lead to self-selection bias. However, the significant number of non-adopters among respondents helps mitigate this concern. Additionally, a student survey alone may not fully capture the plugin's educational impact. Future research should include direct classroom observations and long-term academic performance analysis to provide a more thorough assessment of the plugin's pedagogical value.

We are currently integrating the plugin with OpenAI's GPT [29] to allow students to promptly get help with their assignments without leaving their IDE [30]. The plugin will also collect usage interaction data, with student consent. We plan to harvest these data with the goal of preparing, curating, and publishing a dataset that will allow the computer science education community to study these interactions between students and large language models.

Author Contributions: Conceptualization, B.P.C. and P.A.; funding acquisition, A.V. and P.A.; methodology, B.P.C. and P.A.; software, B.P.C., B.B. and P.A.; validation, B.P.C., N.F. and P.A.; formal analysis, B.P.C., N.F. and P.A.; investigation, B.P.C.; data curation, B.P.C.; writing—original draft preparation, B.P.C. and N.F.; writing—review and editing, B.P.C., N.F. and P.A.; visualization, B.P.C. and N.F.; supervision, N.F., A.V. and P.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Fundação para a Ciência e a Tecnologia (FCT) through the LARSyS—FCT Project (DOI: 10.54499/LA/P/0083/2020, 10.54499/UIDP/50009/2020, and 10.54499/UIDB/50009/2020) and COPELABS—FCT Project (DOI: 10.54499/UIDB/04111/2020). Author Nuno Fachada acknowledges his FCT grant (DOI: 10.54499/CEECINST/00002/2021/CP2788/CT0001).

Informed Consent Statement: Informed consent was obtained from all students involved in the study.

Data Availability Statement: A CSV file with the survey replies is available [31].

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AAT	Automatic Assessment Tool
API	Application Programming Interface
CSV	Comma-separated Values
DP	Drop Project
EB	Exercise-based
IDE	Integrated Development Environment
JVM	Java Virtual Machine
PB	Project-based
REST	Representational State Transfer

References

1. Enström, E.; Kreitz, G.; Niemelä, F.; Söderman, P.; Kann, V. Five years with kattis—Using an automated assessment system in teaching. In Proceedings of the 2011 Frontiers in Education conference (FIE), Rapid City, SD, USA, 12–15 October 2011; p. T3J-1.
2. Cipriano, B.P.; Fachada, N.; Alves, P. Drop Project: An automatic assessment tool for programming assignments. *SoftwareX* **2022**, *18*, 101079. [CrossRef]
3. Paiva, J.C.; Leal, J.P.; Figueira, Á. Automated Assessment in Computer Science Education: A State-of-the-Art review. *ACM Trans. Comput. Educ. (TOCE)* **2022**, *22*, 34. [CrossRef]
4. Luxton-Reilly, A.; Tempero, E.; Arachchilage, N.; Chang, A.; Denny, P.; Fowler, A.; Giacaman, N.; Kontorovich, I.; Lottridge, D.; Manoharan, S.; et al. Automated Assessment: Experiences From the Trenches. In Proceedings of the 25th Australasian Computing Education Conference, Melbourne, VIC, Australia, 30 January–3 February 2023; pp. 1–10.

5. Ihantola, P.; Ahoniemi, T.; Karavirta, V.; Seppälä, O. Review of recent systems for automatic assessment of programming assignments. In Proceedings of the 10th Koli Calling International Conference on Computing Education Research, Koli, Finland, 28–31 October 2010; pp. 86–93.
6. Keuning, H.; Jeuring, J.; Heeren, B. A systematic literature review of automated feedback generation for programming exercises. *ACM Trans. Comput. Educ. (TOCE)* **2018**, *19*, 3. [CrossRef]
7. Silva, A.; Leal, J.P.; Paiva, J.C. Raccode: An Eclipse Plugin for Assessment of Programming Exercises (Short Paper). In Proceedings of the 7th Symposium on Languages, Applications and Technologies (SLATE 2018), Guimaraes, Portugal, 21–22 June 2018; Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
8. Heckman, S.; King, J. Developing Software Engineering Skills using Real Tools for Automated Grading. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education, Baltimore, MD, USA, 21–24 February 2018; pp. 794–799.
9. Offutt, J.; Ammann, P.; Dobolyi, K.; Kauffmann, C.; Lester, J.; Praphamontripong, U.; Rangwala, H.; Setia, S.; Wang, P.; White, L. A novel self-paced model for teaching programming. In Proceedings of the Fourth (2017) ACM Conference on Learning@Scale, Cambridge, MA, USA, 20–21 April 2017; pp. 177–180.
10. Lenfant, R.; Wanner, A.; Hott, J.R.; Pettit, R. Project-Based and Assignment-Based Courses: A Study of Piazza Engagement and Gender in Online Courses. In Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1, Turku, Finland, 7–12 July 2023; pp. 138–144.
11. Lukefahr, H.; Watterson, C.; Roberts, A.; Carnegie, D.; Atkins, M. Project-Based Learning to Improve Core First-Year Engineering Courses. In Proceedings of the 2019 IEEE Frontiers in Education Conference (FIE), Covington, KY, USA, 16–19 October 2019; pp. 1–7.
12. Bridson, K.; Atkinson, J.; Fleming, S.D. Delivering Round-the-Clock Help to Software Engineering Students Using Discord: An Experience Report. In Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1, Providence, RI, USA, 3–5 March 2022; pp. 759–765.
13. Edwards, S.H.; Murali, K.P. CodeWorkout: Short Programming Exercises with Built-in Data Collection. In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, Bologna, Italy, 3–5 July 2017; pp. 188–193.
14. Schwalm, J.; Tylek, K.S. Systemwide Implementation of Project-Based Learning: The Philadelphia Approach. *Afterschool Matters* **2012**, *15*, 1–8.
15. Baillie, C.; Fitzgerald, G. Motivation and attrition in engineering students. *Eur. J. Eng. Educ.* **2000**, *25*, 145–155. [CrossRef]
16. Grotta, A.; Prado, E.P.V. Benefits of the project-based learning to cope with computer programming education: A systematic literature review. In Proceedings of the PBL2019 Immersive Virtual International Conference, Virtual, 12–13 July 2019; pp. 12–13.
17. Mills, J.; Treagust, D. Engineering Education—Is Problem-based Or Project-Based Learning The Answer? *Australas. J. Eng. Educ.* **2003**, *3*, 2–16.
18. Insa, D.; Silva, J. Semi-Automatic Assessment of Unrestrained Java Code: A Library, a DSL, and a Workbench to Assess Exams and Exercises. In Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, Vilnius, Lithuania, 4–8 July 2015; pp. 39–44.
19. Škorić, I.; Orehovački, T.; Ivašić-Kos, M. Exploring the acceptance of the web-based coding tool in an introductory programming course: A pilot study. In Proceedings of the International Conference on Human Interaction and Emerging Technologies, Paris, France, 27–29 August 2020; pp. 42–48.
20. Maguire, P.; Maguire, R.; Kelly, R. Using automatic machine assessment to teach computer programming. *Comput. Sci. Educ.* **2017**, *27*, 197–214. [CrossRef]
21. Leal, J.P.; Silva, F. Mooshak: A Web-based Multi-site Programming Contest System. *Softw. Pract. Exp.* **2003**, *33*, 567–581. [CrossRef]
22. Guerreiro, P.; Georgouli, K. Combating Anonymousness in Populous CS1 and CS2 Courses. *ACM SIGCSE Bull.* **2006**, *38*, 5. [CrossRef]
23. Edwards, S.H.; Perez-Quinones, M.A. Web-CAT: Automatically Grading Programming Assignments. *SIGCSE Bull.* **2008**, *40*, 328. [CrossRef]
24. Sharp, C.; van Assema, J.; Yu, B.; Zidane, K.; Malan, D.J. An Open-Source, API-Based Framework for Assessing the Correctness of Code in CS50. In Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, Trondheim, Norway, 15–19 June 2020; ITiCSE '20, pp. 487–492. [CrossRef]
25. Boettiger, C. An introduction to Docker for reproducible research. *ACM SIGOPS Oper. Syst. Rev.* **2015**, *49*, 71–79. [CrossRef]
26. Baniassad, E.; Zamprogno, L.; Hall, B.; Holmes, R. Stop the (autograder) insanity: Regression penalties to deter autograder overreliance. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, Virtual, 13–20 March 2021; pp. 1062–1068.
27. Cipriano, B.P. How to Use Drop Project to Create Programming Assignments. 2021. Available online: https://www.youtube.com/watch?v=65IwIBuMDlE (accessed on 26 June 2024).
28. Baltazar, B. Drop Project Student Plugin. 2023. Available online: https://www.youtube.com/watch?v=N2YUvU3Lxm4 (accessed on 26 June 2024).
29. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. In Proceedings of the Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, Virtual, 6–12 December 2020; pp. 1877–1901.

30. Cipriano, B.P. Towards the Integration of Large Language Models in an Object-Oriented Programming Course. In Proceedings of the 2024 Innovation and Technology in Computer Science Education, Milan, Italy, 8–10 July 2024. [CrossRef]
31. Cipriano, B.P.; Baltazar, B.; Alves, P.; Fachada, N. Drop Project Student Plugin for IntelliJ IDEA—Evaluation Survey. 2023. Available online: https://zenodo.org/records/8432997 (accessed on 26 June 2024).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

MDPI AG
Grosspeteranlage 5
4052 Basel
Switzerland
Tel.: +41 61 683 77 34

Computers Editorial Office
E-mail: computers@mdpi.com
www.mdpi.com/journal/computers



Disclaimer/Publisher's Note: The title and front matter of this reprint are at the discretion of the Guest Editor. The publisher is not responsible for their content or any associated concerns. The statements, opinions and data contained in all individual articles are solely those of the individual Editor and contributors and not of MDPI. MDPI disclaims responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.





Academic Open
Access Publishing

mdpi.com

ISBN 978-3-7258-9168-9