7th International Computer Programming Education Conference

ICPEC 2026, June 11–12, 2026, University of Minho, Guimarães, Portugal

Edited by

Filipe Portela

Luís Matos

Tiago Guimarães

Editors

Filipe Portela
Algoritmi Center, University of Minho, Guimarães, Portugal
cfp@dsi.uminho.pt

Luís Matos
Algoritmi Centre, University of Minho, Guimarães, Portugal
EPMQ, CCG ZGDV Institute, Guimarães, Portugal
luis.matos@dsi.uminho.pt

Tiago Guimarães
Algoritmi Center, University of Minho, Guimarães, Portugal
tsg@dsi.uminho.pt

ACM Classification 2012
Applied computing → Education; Applied computing → Interactive learning environments; Social and professional topics → Computing education; Computing methodologies → Artificial intelligence

ISBN 978-3-95977-443-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-443-7.

Publication date
July, 2026

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists all publications of this volume in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/OASIcs.ICPEC.2026.0

ISBN 978-3-95977-443-7 **ISSN 1868-8969** **https://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

OASIcs is a series of high-quality conference proceedings across all fields in informatics. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/oasics

The editors wish to express their special thanks to Marcia Veloso for her outstanding work; to the two Communities of Practice at the University of Minho: Practice in Teaching (CoPEP) and Pedagogical Innovation with Gamification and Artificial Intelligence, for their collaboration; and to the Information System Department EPIC consortium for their support.

Contents

Papers

7th International Computer Programming Education Conference (ICPEC 2026).
Editors: Filipe Portela, Luís Matos, and Tiago Guimarães
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The 7th **International Computer Programming Education Conference** (ICPEC 2026) was held on June 11 and 12 at the Information Systems Department, **University of Minho**'s School of Engineering in Guimarães, Portugal. Ever since it started, **ICPEC** has been a go-to spot for teachers, researchers, and experts passionate about improving programming education. This seventh edition kept that tradition alive, bringing together a global community to chat about new ways of teaching, the latest tech, and proven methods for teaching code in today's connected world. This year's theme, "**The Next Chapter of LeArnIng**" really captured the big shifts happening in education right now.

The rapid emergence of Generative Artificial Intelligence, immersive technologies, and intelligent educational systems is transforming how students learn and how educators teach. These developments raise important questions regarding assessment, academic integrity, critical thinking, and the evolving role of educators in an increasingly AI-driven educational landscape.

To foster discussion around these challenges, the conference hosted a **roundtable** discussion in collaboration with the EPIC Project (Pedagogical Excellence and Innovation in Co-creation), bringing together experts from academia and industry to reflect on the future of learning and programming education. Complementing this discussion, the conference featured the hands-on **workshop** *"From Prompt to Practice: Integrating Generative AI into Higher Education"*, which enabled participants to explore contemporary AI tools and critically examine both their potential benefits and the challenges of adopting them in higher education.

The scientific programme featured seventeen peer-reviewed papers, together with a poster and demonstration session. The accepted contributions were organised into three thematic tracks: Teaching Strategies and Pedagogical Approaches, Emerging Technologies and Innovation, and Engagement, Accessibility and Immersive Learning.

The **Teaching Strategies and Pedagogical Approaches** track explored innovative methods to improve programming education, including Montessori-inspired approaches to introducing programming concepts, gamified learning experiences, competitive programming, competency-based assessment, personalised learning strategies, alternative grading models, and frameworks for studying and evaluating programming education practices. Contributions also examined the role of digital educational resources and the challenges and opportunities created by Artificial Intelligence in contemporary programming education.

The **Emerging Technologies and Innovation** track focused on intelligent educational systems and novel technological solutions to support teaching and learning. Topics included program comprehension assessment, Generative Artificial Intelligence for intelligent tutoring systems, learning analytics and early intervention platforms for identifying students at risk of academic failure, and collaborative platforms that foster interaction and engagement among programming students.

The **Engagement, Accessibility and Immersive Learning** track highlighted innovative approaches to learner motivation, participation, and experiential learning. Contributions addressed serious game-based cybersecurity training, the integration of Generative Artificial Intelligence and gamification into learning management systems, and the design and validation of mixed-reality environments for professional training.

7th International Computer Programming Education Conference (ICPEC 2026).
Editors: Filipe Portela, Luís Matos, and Tiago Guimarães

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The poster and demonstration session complemented the main programme by providing opportunities for interactive discussion of emerging educational platforms and innovative classroom support tools, from the perspectives of students and researchers.

Across all contributions, recurring themes included Artificial Intelligence in education, gamification, adaptive learning, learning analytics, immersive technologies, competency-based assessment, and innovative pedagogical strategies for developing programming knowledge and computational thinking skills. Together, these works reflect the growing diversity and maturity of research in computer programming education and demonstrate the community's commitment to improving teaching and learning in an increasingly technology-driven world.

A high-quality event like this does not happen without a lot of hard work. We want to say a huge thank you to all the authors, our dedicated reviewers on the Program Committee, the session chairs, our speakers, and everyone who attended and joined the conversation. We also really appreciate the local team at the University of Minho for their on-the-ground help. Finally, thanks to OASIcs for their help in publishing these proceedings and ensuring this research reaches people all over the world.

As technology continues to change education, keeping our teaching human-centred and ethical is more important than ever. ICPEC remains dedicated to helping educators collaborate and innovate, ensuring students don't just learn to write code but also to solve tough problems and shape the future of technology responsibly.

Guimarães, June 2026

Filipe Portela, Luís Matos and Tiago Guimarães

Program Committee

Afonso Caniço (ISCTE – University Institute of Lisbon, Portugal)

Alexander Paar (Duale Hochschule Sachsen, Germany)

Ana Francisca Monteiro (University of Minho, Portugal)

Anabela Gomes (Polytechnic Institute of Coimbra and CISUC, Portugal)

André Santos (ISCTE – University Institute of Lisbon, Portugal)

Ángel Sánchez (Rey Juan Carlos University, Spain)

Bárbara Cleto (uniMAD/ESMAD, Polytechnic of Porto, Portugal)

Cristiana Araújo (University of Minho, Portugal)

Fabrizio Messina (University of Catania, Italy)

Jakub Swacha (University of Szczecin, Poland)

Jorge Mendonça (ISEP, Polytechnic of Porto, Portugal)

José Carlos Paiva (Polytechnic of Porto, Portugal)

José Cerqueira (University of Vigo, Spain)

Karolina Baras (University of Madeira, Portugal)

Leonel Morgado (Universidade Aberta, Portugal)

Luís Alves (Polytechnic Institute of Bragança, Portugal)

Luís Magalhães (University of Minho, Portugal)

Maria Sejnova (Ketryx, Czech Republic)

Maria Varanda Pereira (Polytechnic Institute of Bragança, Portugal)

Mariia Vetluzhskikh (University of Maryland, USA)

Marílio Cardoso (ISEP, Polytechnic of Porto, Portugal)

Martinha Piteira (Polytechnic Institute of Setúbal, Portugal)

Micaela Esteves (Polytechnic Institute of Leiria, Portugal)

Míriam Antón-Rodríguez (University of Valladolid, Spain)

Patrícia Leite (Polytechnic Institute of Cávado and Ave, Portugal)

Paula Tavares (ISEP, Polytechnic of Porto, Portugal)

Pedro Vasconcelos (University of Porto, Portugal)

Ricardo Queirós (ESMAD, Polytechnic of Porto, Portugal)

Ruth Lamprecht (Mount St. Mary's University, USA)

Sergio Ilarri (University of Zaragoza, Spain)

Serhii Prykhodchenko (Dnipro University of Technology, Ukraine)

7th International Computer Programming Education Conference (ICPEC 2026).
Editors: Filipe Portela, Luís Matos, and Tiago Guimarães
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Spyros Panagiotakis (Hellenic Mediterranean University, Greece)

Teresa Terroso (ESMAD, Polytechnic of Porto, Portugal)

Tiago Gasiba (Siemens AG, Germany)

Tomas Blažauskas (Kaunas University of Technology, Lithuania)

Tomáš Kormaník (Technical University of Košice, Slovakia)

Yannik Bauer (INESC TEC, Portugal)

List of Authors

Sathwik Amburi (16)
Siemens AG, Munich, Germany

Kelly Androutsopoulos (13)
Department of Computer Science, Middlesex University, London, UK

Afonso B. Caniço (1)
ISTAR, Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Dmytro Babets (15)
Department of Applied Mathematics, Dnipro University of Technology, Ukraine

Alice Dutra Balbé (3)
Communication and Society Research Centre, CECS, University of Minho, Braga, Portugal

Ana Paula Silva Paulina Bezerra (12)
Graduate Program in Computer Science, State University of Rio Grande do Norte (UERN) and Federal Rural University of the Semi-Arid Region (UFERSA), Brazil

Clodis Boscarioli (6)
Graduate Program in Computer Science (PPGComp), Western Paraná State University (Unioeste), Cascavel, Brazil

Lenka Bubeňková (9, 10)
Department of Computers and Informatics, Technical University of Košice, Slovakia

Alvaro Costa Neto (3)
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal

Krzysztof Cyran (15)
merQlab, Department of Computer Graphics, Vision and Digital Systems, Silesian University of Technology, Gliwice, Poland

Frederico d'Abreu (16)
Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Raquel Simões de Almeida (17)
CIR, ESS, Polytechnic of Porto, Portugal

David Dorvekinger (13)
Department of Computer Science, Middlesex University, London, UK

Marlene Engels (5)
University of the Bundeswehr Munich, Germany

Tiago Espinha Gasiba (16)
Siemens AG, Munich, Germany

Miguel Ferreira (2)
DCC - FCUP, Porto, Portugal

Max Fritsche (7)
DHSN Duale Hochschule Sachsen, Staatliche Studienakademie Leipzig, Germany; Security Robotics D&S GmbH, Leipzig, Germany

Michael Heeney (13)
Department of Computer Science, Middlesex University, London, UK

Pedro Rangel Henriques (3)
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal

Marek Horváth (9, 10)
Department of Computers and Informatics, Technical University of Košice, Slovakia

José Paulo Leal (2)
CRACS – INESC TEC, Portugal; DCC – FCUP, Porto, Portugal

Andrii A. Martynenko (15)
Department of Computer systems' software, Dnipro University of Technology, Ukraine

Abdallah Mohamed (8)
University of British Columbia Okanagan, Canada

Ceres Morais (14)
Graduate Program in Computer Science (UERN/UFERSA), State University of Rio Grande do Norte (UERN), Federal University of the Semi-Arid Region (UFERSA), Brazil

Ceres Germanna Braga Morais (12)
Department of Informatics, State University of Rio Grande do Norte (UERN), Brazil; Graduate Program in Computer Science, State University of Rio Grande do Norte (UERN) and Federal Rural University of the Semi-Arid Region (UFERSA), Brazil

Dariusz Myszor (15)
Department of Algorithmics and Software, Faculty of Automatic Control, Electronics and Computer Science, Silesian University of Technology, Gliwice, Poland

7th International Computer Programming Education Conference (ICPEC 2026).
Editors: Filipe Portela, Luís Matos, and Tiago Guimarães
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

António Osório (14)
Research Centre on Education (CIEd), Institute of Education, University of Minho, Braga, Portugal

Alexander Paar (7)
DHSN Duale Hochschule Sachsen, Staatliche Studienakademie Leipzig, Germany

José Carlos Paiva (17)
Pedagogical Innovation Center, Polytechnic of Porto, Portugal

Marcin Paszkuta (15)
merQlab, Department of Computer Graphics, Vision and Digital Systems, Silesian University of Technology, Gliwice, Poland

Letícia Mara Peres (6)
Graduate Program in Informatics (PPGInf), Federal University of Paraná (UFPR), Curitiba, Brazil

Emília Pietriková (9, 10)
Department of Computers and Informatics, Technical University of Košice, Slovakia

Mário Pinto (17)
ID+, ESMAD, Polytechnic of Porto, Portugal

Maria Pinto-Albuquerque (16)
ISTAR, Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Filipe Portela (11)
Algoritmi Centre, University of Minho, Guimarães, Portugal

Oksana Yu. Prykhodchenko (15)
Department of Economics and Economic Cybernetics, Dnipro University of Technology, Ukraine

Serhii D. Prykhodchenko (15)
Department of Computer systems' software, Dnipro University of Technology, Ukraine

Ricardo Queirós (4, 17)
CRACS, INESC TEC & ESMAD, Polytechnic of Porto, Portugal

Gabryella Rodrigues (14)
Federal Institute of Pará (IFPA), Brazil

Gabryella Rocha Rodrigues (12)
Federal Institute of Pará (IFPA), Brazil

João Rodrigues (3)
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal

Steffi Rudel (5)
University of the Bundeswehr Munich, Germany

André L. Santos (1)
ISTAR, Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Necmi Kaan Sapoglu (8)
University of British Columbia Okanagan, Canada

Przemysław Skurowski (15)
merQlab, Department of Computer Graphics, Vision and Digital Systems, Silesian University of Technology, Gliwice, Poland

Odair Moreira de Souza (6)
Federal Institute of Education, Science and Technology of Paraná (IFPR), Cascavel, Brazil; Graduate Program in Informatics (PPGInf), Federal University of Paraná (UFPR), Curitiba, Brazil

Teresa Terroso (17)
ID+, ESMAD, Polytechnic of Porto, Portugal

Ľubomír Urbančík (10)
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia

Michaela Ďurkovičová (9)
Department of Computers and Informatics, Technical University of Košice, Slovakia

Summative Assessment of Program Comprehension in CS1 with Questions About Learners' Code

Afonso B. Caniço
ISTAR, Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

André L. Santos
ISTAR, Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Abstract

Questions about Learners' Code (QLCs) assess to what extent learners understand their own code through personalized questions that target previously written code. We conducted an experiment involving 18 students of an Introductory Programming (CS1) course taught in Java, comparing the scores of their midterm test based on code-writing questions with the scores of a multiple-choice test based on QLCs. The questions targeted student code submitted throughout the semester in an automated assessment system. Regression analysis shows a strong association between the scores of both tests, suggesting comparable discriminatory power. Participants generally reacted positively and recognised the benefits of this kind of assessment. Our results suggest that QLC tests may serve as a valid complement for CS1 summative assessment targeting program comprehension.

2012 ACM Subject Classification Social and professional topics → Software engineering education; Social and professional topics → Student assessment

Keywords and phrases introductory programming, CS1, programming education, student assessment, program comprehension, questions about learners' code

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.1

Supplementary Material *Software (Source Code)*: `https://github.com/ambco-iscte/jask` [1]

Funding This research was partially supported by national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT), ISTAR-Iscte Pluriannual Projects UID/04466/2025 (`https://doi.org/10.54499/UID/04466/2025`), and an ISCTE-IUL scholarship.

Acknowledgements We thank the anonymous participants for taking part in the study.

1 Introduction

Traditionally, introductory programming students are tasked with assignments focusing on writing programs that produce specific outputs. However, students hold misconceptions [26], and struggle to explain their own code even when they have successfully completed an assignment [13], indicating that completion does not necessarily imply comprehension.

Given that a learner's ability to produce working code does not imply full understanding of the underlying concepts, student efficacy is affected, as programming students tend to perform better when they work towards mastering assignments' underlying concepts [33]. Code-reading skills are an important prerequisite for problem-solving [19].

Students may overlook the importance of not only producing a working program, but also understanding its underlying concepts [16]. Moreover, the emergence of Large Language Models (LLMs) – capable of generating working solutions to most introductory programming problems [6, 7, 29] – introduces a new paradigm for students, who may overrely on LLMs for completing assignments. When doing so, students effectively bypass the struggle involved in the solving programming problems. Prather et al. [25] found that unconstrained use of LLMs

may be counter-productive to lower-performing students, creating an illusion of competence. We believe that educational tools to foster and assess deep understanding of programs may play a role in counteracting misleading perceptions of self-efficacy in learners.

Computing courses usually employ a mixture of theoretical and practical elements, namely practical assignments accompanied by written examinations [3]. When these include Multiple-Choice Questions (MCQs), concerns include the possibility of plagiarism or academic dishonesty [3], and the need for automated assessment [24], even more so in large-scale courses.

Questions about Learners' Code (QLCs) [14] were proposed as a form of assessing program comprehension skills. In this approach, questions (and possibly multiple-choice options for it) are formulated against a learner's own code structure and behaviour. Several contributions have been made in automating this process for different languages used in introductory programming courses (e.g., Java [28], Python [15], JavaScript [12]).

We explore whether QLC tests are a valid addition to a programming course's assessment process. We hypothesise that these tests can work alongside code-writing (i.e., writing functions to perform specific tasks) tests to facilitate the summative assessment of students' program comprehension skills. Further, since QLCs are personalised by each student's code, the possibility of cheating (e.g., copying from peers) is reduced because it is unlikely that the exact same QLC will be generated for different students. While prior work has explored QLCs in a formative context (e.g., [12, 14, 15]), we aim to assess whether QLC tests can serve as an augmentation to the summative assessment practices of CS1 courses. To this effect, we investigate the relation between students' performance in a code-writing pen-and-paper test, and a personalised test composed of QLCs generated from the students' own code submitted during the first 6 weeks of a CS1 course taught in Java. We address the following research questions:

RQ 1. How do QLC test scores relate to those of code-writing tests?
RQ 2. What are students' perceptions on summative assessment using QLCs?

We conducted a study comparing student performance on a written midterm test and a QLC test among 18 first-time enrollees in the Introductory Programming (CS1) course at our institution. The results revealed a statistically significant relation between the two sets of scores, suggesting that QLC-based assessment scores align with those of code-writing assessment. Since both types of assessment measure different, yet related, skills (code-writing vs. code comprehension ability), employing both could offer a more thorough assessment practice in introductory programming. The strong relation between the scores of both assessment types suggests comparable discriminatory power. A post-test survey indicated that students held a positive perception of the QLC test, recognising its usefulness as an assessment and self-learning tool.

2 Related Work

Traditionally, programming courses focus on teaching a programming language through a writing-focused approach, that is, students are encouraged to learn the constructs of a programming language through writing programs. However, completing assignments has been shown to not necessarily contribute to students' comprehension of underlying concepts [10, 20], and students may struggle to explain their own code even when they were successful in constructing working programs [13]. Successfully completing assignments without developing an understanding of the underlying concepts is defined by Kapur [9] as "unproductive success", for which there exists extensive research in the context of introductory

programming (e.g., [10, 11, 13, 27]). Comprehension-first teaching methods, where code reading and tracing activities precede code writing, have been demonstrated to be more effective in fostering deep understanding (e.g., Xie et al. [32]). Several studies have observed the relation between code reading and writing ability (e.g., [17, 18, 31]). Nurollahian et al. find that code comprehension ability is predictive of code-writing success [23].

Questions about Learners' Code (QLCs) consist of questions regarding constructs, patterns, or behaviour of code written by the learner [14]. QLCs typically have single or multiple-choice answers, which are easily evaluated automatically. QLCs may be generated from a combination of *static* program analysis (e.g., which constructs are employed) and *dynamic* analysis (e.g., code behaviour during execution). Dynamic QLCs usually target code tracing skills (e.g., "which values does this variable take during the program's execution?"), which have been shown by earlier studies (e.g., [4, 17, 18, 19, 31, 32]) to be deeply related to program comprehension and code writing skills.

Lehtinen et al. explored the correlation between students successfully submitting JavaScript code assignments and their understanding of the underlying concepts [13]. They found that finishing an assignment does not imply that the student understands the code they wrote or its underlying concepts, nor does it guarantee the absence of misconceptions the student might have about those concepts. The authors found that a third of students struggled to explain their own code even when they succeeded in finishing the corresponding assignments. A similar study found that students who tend to answer QLCs about their own code incorrectly have a higher tendency to drop out and more difficulty in writing programs [12]. These results might hint, as in the work of Nelson et al. [21], that students who struggle with program reading might also struggle with program writing. Students who incorrectly answered QLCs about their Python code tended to have weaker prerequisite knowledge [15].

Jask is a QLC generation system for Java code, where users submit code snippets from which static and dynamic QLCs are automatically generated [28]. In a preliminary study, their authors found that students tend to correctly answer QLCs addressing static aspects, yet may struggle when answering those related to program dynamics, revealing some difficulty in reasoning about their programs' execution. Further, students tend to react positively to the QLCs, particularly for autonomous learning purposes (e.g., for consolidating terminology).

A recent systematic review by Nurollahian et al. [22] found that most approaches for assessing student code structure focus on finding errors using automated tools, which may not capture skills such as program comprehension. Their study calls for novel assessment methods that capture such skills. QLCs, which target program comprehension, may provide an opportunity to complement existing CS1 assessment practices.

3 Background: Generating Questions about Learners' Code

QLCs are generated from *templates*, which represent questions that have not yet been generated from concrete code constructs. Conceptually, the generation of concrete QLCs may be thought of as using code constructs (e.g., types and identifiers) or behaviour (e.g., number of loop iterations, number of variable assignments) to "fill in the blanks" in a template. This may only occur if the targeted code is applicable to the corresponding type of QLC. For example, a QLC template targeting loop behaviour is only applicable to code which contains at least one loop structure. Figure 1 presents an example visualisation of a QLC template targeting variable tracing. A concrete QLC is generated by using student code to assign names and values to v, u, and $f(\text{args})$.

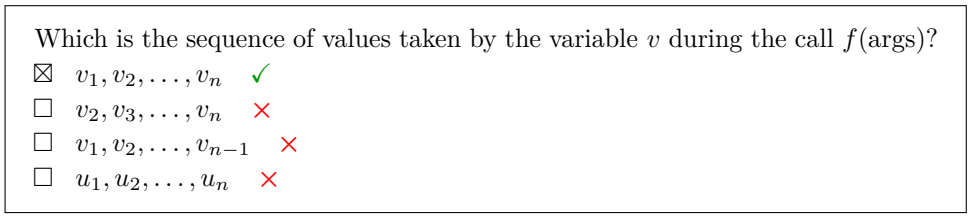


Figure 1 Example template structure for QLCs targeting variable tracing (dynamic).

Typically, QLCs contain one correct option and three distractors generated from student code. Whenever possible, these distractors are based on known introductory programming misconceptions such as boundary errors (e.g., off-by-one, as in the second and third options in Figure 1). If there is not enough data to generate distractors targeting misconceptions, options may be generated from other code constructs (e.g., values from another variable, as in the last option in Figure 1). For assignments of low complexity or those where enough data cannot be collected, broader options may be included (e.g., "none of the above"). Figure 2 presents an example QLC generated from this template, where f is materialised with `countDivisors`, while v and u are bound to its local variables `n` and `d`, respectively. The code used to generate QLCs for the purpose of our study is available as a standalone library[1].

4 Context

Our approach involves the automated generation of programming tests consisting of QLCs targeting student code previously submitted in assignments. Hence, applying it in a large-scale course requires an Automated Assessment System (AAS), or some form of collecting the students' solutions in a structured format. Figure 3 presents concepts related to AASs (grey container). Although different systems may have variations, we describe general concepts that fit most AAS realisations. A *course* holds several code *assignments*. Each assignment has several *test cases*, which associate *arguments* (inputs) to *expected* results, and define one (or more) reference *solutions* that meet its requirements. *Students* are enrolled in the course and perform *submissions* to each *assignment*. Each submission holds the student solution, which is marked with a *score*.

The remaining elements of Figure 3 relate to our approach. A *QLC test specification* is defined against the course exercises, holding several *items* that address a QLC of a certain type. Each item is associated to one or more assignments, meaning that an individual QLC will target the code that a student submitted to those assignments. A *QLC test* for a specific *student* is an instance of the test specification, comprising *questions* derived from their code. Figure 2 consists of an example of a QLC generated from a student's code.

Since different students have written different solutions to the assignments, the generated QLCs addressing that assignment will be slightly different with respect to: (a) formulation, because they refer to particular identifiers of the student solution; and (b) options, because the answers depend on the code structure and/or behaviour. Additionally, course instructors may define that a question will target a random assignment of a predefined set (e.g., with similar characteristics). This allows for more variability in the set of individual test QLCs.

[1] `https://github.com/ambco-iscte/jask`

Which is the sequence of values taken by the variable **n** during the function call **countDivisors(7)**?

```
static int countDivisors(int a) {
    int n = 1;
    int d = 0;
    while (n <= a) {
        if (a % n == 0)
            d = d + 1;
        n = n + 1;
    }
    return d;
}
```

☐ 2, 3, 4, 5, 6, 7, 8 ✗
☐ 0, 1, 2 ✗
☒ 1, 2, 3, 4, 5, 6, 7, 8 ✓
☐ 1, 2, 3, 4, 5, 6, 7 ✗

Figure 2 A concrete QLC that materialises the template of Figure 1. This example was taken from a participant's test generated for the study (translated to English).

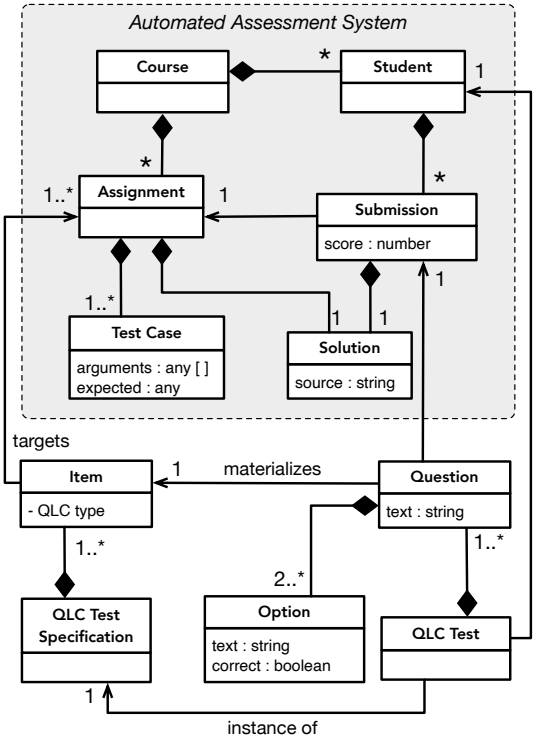


Figure 3 QLC integration within an Automated Assessment System. A test specification consists of a mapping between QLC templates and assignments. A concrete QLC test materialises a test specification using constructs from each student's code.

We introduce further variability by randomly selecting inputs from the test cases. The available inputs from tests cases, defined with meaningful values and addressing edge cases, are crucial for deriving proper dynamic QLCs that make sense for each assignment. Having a purely randomised form of generating program arguments is not adequate to address a set of assignments, as it is impossible to simultaneously guarantee that each assignment's constraints (e.g., input array must be ordered) are verified.

5 Methods

We conducted a study with first-time CS1 students at our institution. The course is taught using Java, and students are required to take a midterm test after the first 6 weeks of the course. The midterm test targets the following concepts: functions; expressions; variables; control structures; arrays; references; and procedures (side-effects). The midterm test is hand-written (pen-and-paper) and consists of questions where students write relatively small Java functions (approximately 5–15 lines of code), either by solving a problem from scratch for a set of requirements, or writing a function conforming to a given signature.

Students must complete coding assignments throughout the semester. These assignments target the same concepts as the midterm, and similarly consist in writing functions to fulfill given requirements. Code solutions to these assignments are similar in length (mean of 12 lines of code, $SD = 11$) to solutions to questions in the midterm. The QLCs generated for the study targeted these assignments.

5.1 Participants

Eighteen (18) students took part in the study. Regarding gender (self-reported), 16 participants (89%) identified as male, and the remaining two (11%) as female. Their ages ranged between 18 and 50. All the participants were enrolled for the first time in the course, but their major and previous experience differed. Most participants (14) were majoring in Computer Science and Engineering. Half of these were enrolled in the evening study program, whose student profile consists of people who have a full-time job during the day. The remaining participants were majoring in Information Systems.

The data collected for our study consisted in participants' code submissions, course midterm score, and answers to the QLC tests. Student IDs were used internally to match the different data to enable analysis, but these IDs were not stored in the resulting dataset. The data is not published nor does the paper make any mentions of identifiable student data. As per our institution's guidelines, all participants were informed of this through an informed consent form which was presented before the study. Additionally, the participants were informed that they could opt out of the study at any moment without repercussions. All 18 participants agreed to take part in the study and gave their informed consent through the signed form.

5.2 Study Sessions

The study was administered within a time frame of one week, and occurred two weeks after the participants had taken their midterm test. The one-week time frame resulted from an inability to schedule a single session which all participants could attend simultaneously.

We designed a sequence of QLC items that covered the course's assignments, and generated a personalised test for each of the participants from this specification and their respective submissions. When a participant had no submission to an assignment referenced in the test specification, we used the reference solution held in the AAS to generate the QLC.

Table 1 QLC types employed in the test. QLC types tagged with a checkmark symbol (✓) target static program aspects. Wrong measurements consider both incorrect and blank answers.

QLC Type	Template	Total	Wrong
`HowManyVariableAssigns`	How many times is the variable v assigned when calling f(args)?	18 (1/test)	11 (62%)
`HowManyFunctionCalls`	How many calls to f_2 are performed when calling f_1(args)?	18 (1/test)	7 (39%)
`WhichVariableValues`	Which are the values taken by the variable v when calling f(args)?	36 (2/test)	13 (36%)
`HowManyArrayReads`	How many array position reads are performed when calling f(args)?	18 (1/test)	5 (28%)
`HowManyLoopIterations`	When calling f(args), how many iterations does the loop perform?	54 (3/test)	14 (26%)
`HowManyArrayWrites`	How many array position writes are performed when calling f(args)?	36 (2/test)	7 (19%)
`WhichParameters` ✓	Which are the parameter names of the function f?	18 (1/test)	2 (11%)
`WhichReturnType` ✓	What is the return type of the function f?	18 (1/test)	1 (6%)
`WhatIsResult`	What is the value returned by the function call f(args)?	18 (1/test)	1 (6%)
`HowManyParams` ✓	How many parameters does the f function take?	18 (1/test)	0
`WhichVariableHoldsReturn` ✓	Which variable holds the result of the function f?	18 (1/test)	0
		270	**61 (23%)**

The participants were brought to a room with their personal laptops, which they used to conduct the test. The session started by having all participants read and sign the informed consent form. The participants were then informed about how the test would be conducted. Participants could ask clarifying questions whenever they considered that the formulation of the questions was not sufficiently clear. However, they could not pose questions regarding concepts that were addressed in the test (e.g., how loops work).

The tests consisted of 15 QLCs, each with one correct option out of four generated from the participants' code submissions. The type and number of the QLCs included in the test are presented in Table 1. All QLCs contributed 1 point towards the final score. Incorrect answers subtracted 0.25 points, and blank answers were scored with 0 points.

Participants were given 30 minutes to solve the test, whose questions were written in the teaching language of the course, employing the terminology adopted in its materials. As in the proctored midterm, the QLC test was closed-book and LLM usage was not allowed. Figure 2 presents a translated example of a QLC used in a participant's test. When each participant finished the test, or when the 30 minutes had passed, they were instructed to submit their responses, and were then shown their percentage score and the solutions.

To debrief, participants were given feedback regarding any questions they might have had regarding the questions present in their test. Finally, the participants were sent a survey in which they could express their perceptions regarding the QLC test.

5.3 Data Analysis

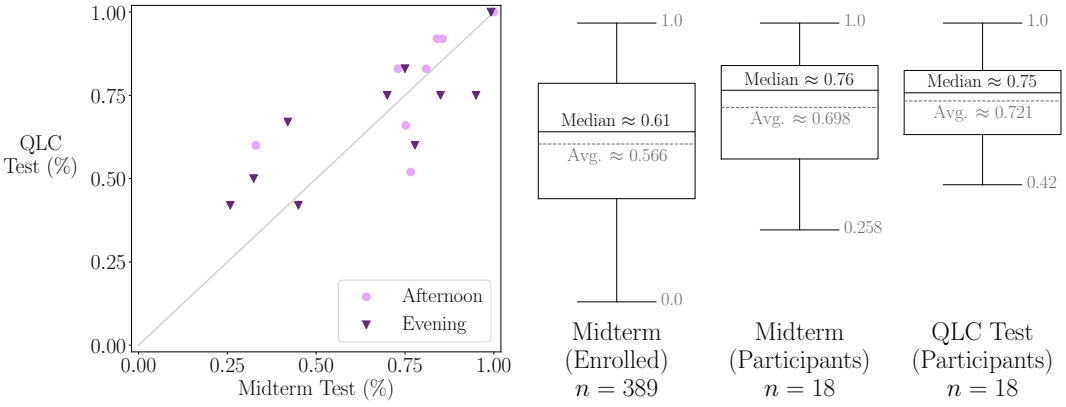
We employed linear regression to analyse the association between the QLC test and midterm scores, controlling for average code submission score and length (lines of code), and for whether the student is enrolled in the afternoon or evening study program (binary variable). The post-test survey answers were labeled through an inductive coding by one of the authors.

6 Results

Table 1 summarises the QLCs covered in the test. Approximately 23% of participants' answers were incorrect (58) or blank (3). Approximately 3% of QLCs used reference solutions due to students not having any correct submissions for the corresponding assignment.

6.1 QLC Test Results (RQ 1)

Figure 4a presents the scores (%) obtained by each participant in the QLC test in relation to their corresponding scores in the midterm test. The scores have a strong and significant correlation (Pearson $r \approx 0.784$, $p < 0.001$). The set of participants' scores exhibit nearly identical means and medians in the midterm and QLC tests, as depicted in Figure 4b. In particular, the difference in means is not significant (t-test, $p = 0.747$). However, there is a moderate and significant difference between the means of all course enrollees' midterm scores and those of the participants (t-test, $p = 0.036$, Cohen's $d = 0.51$).



(a) Relation of midterm and QLC test scores. **(b)** Score distributions per student group.

Figure 4 Relation between participants' midterm and QLC test scores.

Regression analysis shows midterm test scores along with average submission scores and lengths account for 74% of the variance in QLC test scores ($R^2 = 0.74$, adj. $R^2 = 0.66$). The intercept is not significant ($\beta = -0.460$, $p = 0.44$), nor are the coefficients for average submission score ($\beta = 1.08$, $p = 0.076$) or lines of code ($\beta = -0.232$, $p = 0.463$). The midterm test score displayed a strong and significant relation with the QLC test score ($\beta = 0.601$, $p < 0.001$). The coefficient for the student enrollment type (afternoon vs. evening program) was not significant ($\beta = -0.043$, $p = 0.445$). The coefficients represent percentages. Hence, for instance, the midterm score coefficient signals that students who score 10 percent points higher on their midterm would score 6 percent points higher on the QLC test.

QLC test scores along with submission scores and lengths offer comparable explanatory power of midterm scores ($R^2 = 0.7$, adj. $R^2 = 0.61$). The coefficient for QLC test score was positive and significant ($\beta = 1.123$, $p < 0.001$). The coefficients represent percentages scaled

to $[0, 1]$, and so the interpretation of this coefficient is that students scoring 10 percent points higher on the QLC test would score 11 percent points higher in the midterm. The remaining coefficients were not significant ($p > 0.1$).

6.2 Post-Test Survey (RQ 2)

The survey sent to each participant after the study consisted in the following questions:

1. Were the questions' statements understandable? (1: Weak, 5: Very Good);
2. Could you recognise the code in each question as your own? (1: Never, 5: Always);
3. How would you rate the duration of the test? 1: Too Short, 5: Too Long);
4. Did taking the test cause any feelings of stress, anxiety, etc.? (1: None, 5: Many);
5. Was taking the test beneficial for your learning process? How so? (Open-Ended);
6. Were there negative aspects? (Open-Ended);
7. What could have gone better? (Open-Ended);
8. Is there anything else you'd like to say? (Open-Ended).

The answers to questions Q1–Q4 are summarised in Figure 5. All participants answered Q1 positively, indicating QLCs were easily understood. For Q2, most participants were able to recognise the code in each QLCs as their own. Some participants not recognising their code may simply indicate difficulty in recalling what they submitted for earlier assignments. The answers to Q3 indicate that the 30-minute duration for 15 multiple-choice QLCs was largely adequate. The answers to Q4 indicate that taking the QLC test did not elicit negative emotions in general, with only two participants answering neutrally or negatively.

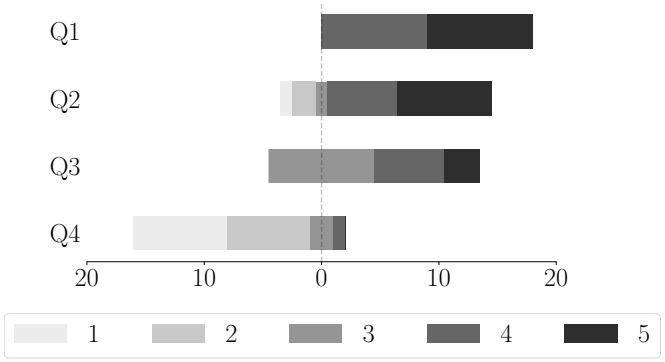


Figure 5 Post-test survey answers (Q1–Q4).

Table 2 Benefits of the QLC test perceived by the participants.

The test helped to...	# of Participants
Reveal Weak Points	10 (55.6%)
Consolidate Knowledge	9 (50%)
Increase Motivation	3 (16.7%)

The positive aspects mentioned by the participants (Q5) are summarised in Table 2. Students particularly highlighted the benefit of QLC tests for consolidating knowledge and revealing weak points in their knowledge. One participant mentioned the benefit of each QLC focusing on a particular concept, and immediate feedback as a motivating factor.

Regarding Q6 and Q7, only two participants mentioned negative aspects. One participant stated that the test elicited feelings of demotivation, as their performance challenged a misleading perception of competence. Another participant mentioned they would have benefited from having more time to take the test. In Q8, one participant suggested implementing QLC tests throughout the semester, to help consolidate concepts as they are introduced.

7 Discussion

The relatively higher failure rates on QLCs targeting program tracing confirms the results of previous studies [12, 28]. Our results additionally agree with previous observations that exercise completion does not necessarily imply comprehension [13].

Ensuring that tests are generated with enough variability is relevant for large-scale courses, as it may lower the chance of plagiarism. As with all multiple-choice questions, QLCs are prone to random guessing. We believe negative marking should be an adequate deterrent.

One of the main difficulties in automating QLCs is obtaining meaningful distractors that make sense for each question. Useful distractors are those that relate to misconceptions [26, 5], as answering QLCs may help to reveal them. Purely random generation is not adequate, as different question types or even different pieces of code for the same question might not make sense for the same types of distractors.

7.1 QLC and Midterm Test Performance Is Similar (RQ 1)

Program writing and comprehension skills are deeply correlated (see, e.g., [4, 17, 18, 19, 23, 31, 32]), and it is thus unsurprising that students who do well on code-writing assessment would do well on program comprehension questions. Nevertheless, we believe QLC tests present an opportunity to extend the summative assessment practices of CS1 courses beyond traditional approaches. The benefits would be twofold: (1) students are evaluated on their understanding of their own code, which may foster higher engagement in assignments; and (2) instructors automate the generation (and possibly evaluation) of QLC tests in large-scale courses, enabling an augmentation of existing assessment practices at scale with relatively low effort. Further, as the strong correlation of the QLC test and the midterm test scores suggests similar discriminatory power of student ability, our results suggest that QLCs may be used to augment existing summative practices to include assessment of program comprehension ability with discriminatory power similar to that of code-writing exams.

The strong relation between the scores of both assessment types is further strengthened by each being the only significant predictor in the other's regression model. The non-significance of the coefficients for submission scores and length (measured in lines of code) may indicate that the complexity and size of the submitted code do not retain their explanatory power of either assessment type's scores after controlling for the other's. This may suggest that unfairness in QLCs from students producing code of differing complexity is not a significant concern, but there are no guarantees as to how prominent this would be in a larger sample. Since the coefficient of enrollment type (afternoon vs. evening) was non-significant in both models, there should be no significant differences between students in both programs.

7.2 Students Have a Positive Perception of QLC-Based Tests (RQ 2)

Over half of the participants highlighted their perceived usefulness of the QLC test as a means to reveal their weak points, while half mentioned an opportunity to consolidate knowledge, which agrees with the observations of a prior study [28]. Our results suggest that students perceive QLCs as useful in formative and summative contexts, and could thus be used to improve student learning and assessment throughout introductory programming courses.

One study participant mentioned feeling demotivated after the QLC tests, as their performance challenged their previous (misleading) perception of self-efficacy (*"I realised I have many weak points"*). While the participant may have felt demotivated in the moment, breaking this illusion of competence provides an opportunity to improve by guiding their learning process. Misconceptions remaining "hidden" would have been harmful, both for the participant's performance in the course and the quality of their learning. Practicing with QLC tests before the course's final summative assessment may be beneficial in this regard.

7.3 Threats to Validity

The main limitation of our study is the small sample size. There is no guarantee that our results are generalisable to other contexts or institutions, or even to the overall CS1 student population in our institution. Further, there is a risk of selection bias, since students with higher self-efficacy may be more likely to participate in such studies. Our dataset showed a statistically significant and moderate difference between the average midterm scores of all course enrollees and study participants, suggesting the sample employed in our study is not representative of the broader student cohort.

Ideally, all participants would take the test simultaneously and under the same conditions. Our study was conducted over the course of one week. Thus, we cannot guarantee that participants who took the test later did not have an advantage from having more time to study the course materials. However, there is no apparent reason for this limitation to have significantly affected the results. Furthermore, given that the QLC tests were not equal (due to the randomness in the generation), it is unlikely that the earlier participants could have "leaked" any useful information regarding the questions to the following participants. There was no incentive to cheat in the study, as there was no reward related to performance.

Cheating can happen in formative contexts, and there is no guarantee that QLC success is indicative of high program comprehension – students who cheat on their assignments (e.g., by using external tools such as generative AI) may also do so on the corresponding QLCs. Usage of QLCs in summative assessment could minimise this through proctoring, for instance by having the QLC assessment coincide with the proctored written exam.

While we believe our approach should be scalable to large-scale courses, we cannot infer this from the small sample employed in our study. Summative assessment using QLCs may require extra resources, such as longer exam time and lecturers to proctor it. Formative usage of QLCs could be automated through integration in an assessment system.

While we intended to generate QLC distractors based on common student misconceptions, we did not conduct an analysis into the questions' and options' pedagogical quality. None of the study participants raised concerns regarding question interpretability, but future work could address this limitation by conducting an analysis of QLCs' quality as assessment items.

8 Conclusions and Future Work

Regression analysis of the participants' performance in QLC tests targeting program comprehension was strongly correlated to their performance in a pen-and-paper code-writing test, hinting at a strong relationship between program writing ability and program comprehension skills and comparable discriminatory power between the two forms of assessment. The results of our study suggest that QLC tests may serve as adequate augmentations of CS1 summative assessment practices, providing an opportunity to assess program comprehension.

The scope of the QLCs used in the study was limited, covering only 11 types of QLCs. We envision QLC tests to include, for example, questions targeting program repair and debugging. Further, we aim to explore the feasibility of open-ended QLCs as opposed to multiple-choice. The main difficulty in this regard would be the analysis of the answers, which may require a manual or AI-based approach. Further, students could benefit from automated feedback on their answers, eliciting potential misconceptions.

Currently, QLCs are graded dichotomously – full marks for correct answers, or none/negative for incorrect answers. In the future, we may implement partial credit in QLCs, for example by requiring students to provide open-ended justifications for their answers [2, 8]. Care must be taken to ensure this does not hinder automated grading in large-scale courses.

The test was well received by the participants, who highlighted its usefulness in consolidating knowledge and revealing weak points. Automation of QLC tests through integration in an assessment system could allow usage in formative contexts. We believe employing QLCs within both formative and summative assessment practices in CS1 courses can be an adequate augmentation towards fostering students' program comprehension skills.

Beyond summative assessment, QLC practice tests may be useful for providing formative feedback [30], as not scoring well on QLCs would point out weaknesses that would likely get revealed on summative feedback. Further, integrating QLCs into an automated assessment system (e.g., as outlined in Section 4) could enable their application after assignment completion, or periodically (e.g., weekly test), providing a formative activity where students check the comprehension of their own code. The activity is lightweight, as it should not require students to dedicate significant additional time. From our study, we conclude that 1–2 minutes may suffice to introduce a QLC targeting a specific assignment.

References

1 Afonso B. Caniço and André L. Santos. ambco-iscte/jask. Software, version 0.7.3. (visited on 2026-07-08). URL: `https://github.com/ambco-iscte/jask`, `doi:10.4230/artifacts.27040`.

2 Pablo Frank Bolton, Liberty Rose Lehr, Rahul Simha, and Michelle Lawson. The justification effect on two-tier multiple-choice exams. In *2024 ASEE Annual Conference & Exposition*, Portland, Oregon, June 2024. ASEE Conferences. `doi:10.18260/1-2--48121`.

3 W.J. Buchanan and L. Saliou. Enhanced methods of coursework provision in computer networks. In *ITRE 2004. 2nd International Conference Information Technology: Research and Education*, pages 111–115, London, UK, 2004. IEEE. `doi:10.1109/ITRE.2004.1393657`.

4 Teresa Busjahn and Carsten Schulte. The use of code reading in teaching programming. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, Koli Calling '13, pages 3–11, New York, NY, USA, November 2013. Association for Computing Machinery. `doi:10.1145/2526968.2526969`.

5 Luca Chiodini, Igor Moreno Santos, Andrea Gallidabino, Anya Tafliovich, André L. Santos, and Matthias Hauswirth. A Curated Inventory of Programming Language Misconceptions. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE '21, pages 380–386, New York, NY, USA, June 2021. Association for Computing Machinery. `doi:10.1145/3430665.3456343`.

6 James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Proceedings of the 24th Australasian Computing Education Conference*, ACE '22, pages 10–19, New York, NY, USA, February 2022. Association for Computing Machinery. `doi:10.1145/3511861.3511863`.

7 James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. My AI Wants to Know if This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *Proceedings of the 25th Australasian Computing Education Conference*, ACE '23, pages 97–104, New York, NY, USA, January 2023. Association for Computing Machinery. `doi:10.1145/3576123.3576134`.

8 Genady Ya. Grabarnik and Serge Yaskolko. Automated partial credit for stem classes. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, Covington, KY, USA, 2019. IEEE. doi:10.1109/FIE43999.2019.9028473.

9 Manu Kapur. Examining productive failure, productive success, unproductive failure, and unproductive success in learning. *Educational Psychologist*, 51(2):289–299, 2016. doi:10.1080/00461520.2016.1155457.

10 Cazembe Kennedy and Eileen T. Kraemer. Qualitative observations of student reasoning: Coding in the wild. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, pages 224–230, New York, NY, USA, July 2019. Association for Computing Machinery. doi:10.1145/3304221.3319751.

11 Päivi Kinnunen and Beth Simon. My program is ok – am i? computing freshmen's experiences of doing programming assignments. *Computer Science Education*, 22(1):1–28, March 2012. doi:10.1080/08993408.2012.655091.

12 Teemu Lehtinen, Lassi Haaranen, and Juho Leinonen. Automated questionnaires about students' javascript programs: Towards gauging novice programming processes. In *Proceedings of the 25th Australasian Computing Education Conference*, ACE '23, pages 49–58, New York, NY, USA, January 2023. Association for Computing Machinery. doi:10.1145/3576123.3576129.

13 Teemu Lehtinen, Aleksi Lukkarinen, and Lassi Haaranen. Students struggle to explain their own program code. In Carsten Schulte, Brett A. Becker, Monica Divitini, and Erik Barendsen, editors, *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE '21, pages 206–212, New York, NY, USA, June 2021. Association for Computing Machinery. doi:10.1145/3430665.3456322.

14 Teemu Lehtinen, André L. Santos, and Juha Sorva. Let's ask students about their programs, automatically. In *Proceedings - 2021 IEEE/ACM 29th International Conference on Program Comprehension, ICPC 2021*, Proceedings/IEEE International Conference on Program Comprehension, pages 467–475, United States, May 2021. IEEE. International Conference on Program Comprehension, ICPC ; Conference date: 20-05-2021 Through 21-05-2021. doi:10.1109/ICPC52881.2021.00054.

15 Teemu Lehtinen, Otto Seppälä, and Ari Korhonen. Automated questions about learners' own code help to detect fragile prerequisite knowledge. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2023, pages 505–511, New York, NY, USA, June 2023. Association for Computing Machinery. doi:10.1145/3587102.3588787.

16 Rachel S. Lim, Sophia Krause-Levy, Ismael Villegas Molina, and Leo Porter. Student expectations of tutors in computing courses. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, pages 437–443, New York, NY, USA, March 2023. Association for Computing Machinery. doi:10.1145/3545945.3569766.

17 Raymond Lister, Colin Fidge, and Donna Teague. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *SIGCSE Bull.*, 41(3):161–165, July 2009. doi:10.1145/1595496.1562930.

18 Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the Fourth international Workshop on Computing Education Research*, ICER '08, pages 101–112, New York, NY, USA, September 2008. Association for Computing Machinery. doi:10.1145/1404520.1404531.

19 Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018 Companion, pages 55–106, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3293881.3295779.

20 Sandra Madison and James Gifford. Modular Programming: Novice Misconceptions. *Journal of Research on Technology in Education*, 34(3):217–229, March 2002. doi:10.1080/15391523.2002.10782346.

21 Greg L. Nelson, Benjamin Xie, and Amy J. Ko. Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in cs1. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER'17, pages 2–11, New York, NY, USA, August 2017. Association for Computing Machinery. doi:10.1145/3105726.3106178.

22 Sara Nurollahian, Noelle Brown, Anna N. Rafferty, and Eliane Wiese. A Systematic Review of Approaches for Evaluating Students' Function-Level Code Structure Knowledge with a Catalog of All Discussed Patterns. *ACM Transactions on Computing Education*, page 3787450, January 2026. doi:10.1145/3787450.

23 Sara Nurollahian, Anna N. Rafferty, Noelle Brown, and Eliane Wiese. Growth in knowledge of programming patterns: A comparison study of cs1 vs. cs2 students. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2024, pages 979–985, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3626252.3630865.

24 José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. Automated assessment in computer science education: A state-of-the-art review. *ACM Trans. Comput. Educ.*, 22(3), June 2022. doi:10.1145/3513140.

25 James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*, volume 1 of *ICER '24*, pages 469–486, New York, NY, USA, August 2024. Association for Computing Machinery. doi:10.1145/3632620.3671116.

26 Yizhou Qian and James Lehman. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.*, 18(1):1:1–1:24, October 2017. doi:10.1145/3077618.

27 Jean Salac and Diana Franklin. If they build it, will they understand it? exploring the relationship between student code and performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, pages 473–479, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3341525.3387379.

28 André L. Santos, Tiago Soares, Nuno Garrido, and Teemu Lehtinen. Jask: Generation of questions about learners' code in java. In Brett A. Becker, Keith Quille, Mikko-Jussi Laakso, Erik Barendsen, and Simon, editors, *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, ITiCSE '22, pages 117–123, New York, NY, USA, July 2022. Association for Computing Machinery. doi:10.1145/3502718.3524761.

29 Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. Thrilled by your progress! large language models (gpt-4) no longer struggle to pass assessments in higher education programming courses. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, ICER '23, pages 78–92, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3568813.3600142.

30 Valerie J. Shute. Focus on formative feedback. *Review of Educational Research*, 78(1):153–189, 2008. doi:10.3102/0034654307313795.

31 Anne Venables, Grace Tan, and Raymond Lister. A closer look at tracing, explaining and code writing skills in the novice programmer. In *Proceedings of the fifth international workshop on Computing education research workshop*, ICER '09, pages 117–128, New York, NY, USA, August 2009. Association for Computing Machinery. doi:10.1145/1584322.1584336.

32 Benjamin Xie, Dastyni Loksa, Greg L Nelson, Matthew J Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J Ko. A theory of instruction for introductory programming skills. *Computer Science Education*, 29(2-3):1–49, January 2019.

33 Daniel Zingaro, Michelle Craig, Leo Porter, Brett A. Becker, Yingjun Cao, Phill Conrad, Diana Cukierman, Arto Hellas, Dastyni Loksa, and Neena Thota. Achievement goals in cs1: Replication and extension. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 687–692, New York, NY, USA, February 2018. Association for Computing Machinery. doi:10.1145/3159450.3159452.

Exploiting Generative AI to Scale up Intelligent Tutoring Systems

Miguel Ferreira
DCC – FCUP, Porto, Portugal

José Paulo Leal
CRACS – INESC TEC, Portugal
DCC – FCUP, Porto, Portugal

Abstract

The research described in this paper explores the integration of Generative Artificial Intelligence (GenAI) into Intelligent Tutoring Systems (ITS) with the aim of improving programming education. Specifically, it extends Agni, a web-based programming learning platform, by automating the construction of key ITS components that are traditionally built manually. The methodology leverages Large Language Models (LLMs) to extract programming concepts from educational materials, map their relationships via concept graphs and diagnose specific student knowledge gaps. Additionally, GenAI provides real-time, contextual feedback during programming exercises, mimicking the personalized support typically offered by a human tutor. The work evaluates whether GenAI can effectively replace or augment the manual creation of domain models while preserving the pedagogical benefits of ITS. The obtained results suggest the potential of combining AI with ITS to improve scalability and reduce manual workload.

2012 ACM Subject Classification Applied computing → Interactive learning environments; Computing methodologies → Natural language generation

Keywords and phrases Intelligent Tutoring Systems, Large language Models

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.2

Funding This work is funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the support UID/50014/2025 (`https://doi.org/10.54499/UID/50014/2025`).

1 Introduction

Programming is an important skill in modern society, with strong demand across industries. Despite this demand, learning to program remains challenging for beginners. Research shows that while novices can understand individual programming concepts, they often struggle to combine them into coherent programs. Because of this, effective programming education relies heavily on practice supported by good feedback from tutors.

Intelligent Tutoring Systems have traditionally addressed this need by providing adaptive feedback and personalized instruction. However, a major limitation of classical ITS lies in their reliance on manually engineered components, such as concept graphs. Constructing these components requires significant effort from tutors and does not scale well across different subjects.

Recent advances in GenAI, particularly LLMs, created new capabilities like producing new content, with natural language input, by learning from large datasets, this provides a new opportunity to automatically generate the key components of an ITS. Thus, the research question that drove this work was the following: **Can an LLM replace a human tutor in the repetitive and labor intensive tasks required by ITS?**

This research question led to the goal of extending Agni to support ITS. Agni is a web-based playground to learn programming [3]. By automating the construction of ITS components, this approach aims to reduce the manual effort required, improving scalability while preserving the pedagogical effectiveness of ITS. This way, educators could more easily adopt ITS in real educational environments, making it more scalable.

The main objectives of this work are:

Automatic Concept Extraction: Develop methods to extract the taught programming concepts from educational materials using GenAI.

Concept Graph Generation: Use GenAI to construct structured representations of knowledge (concept graphs) that capture dependencies between programming concepts.

Failed Concepts Diagnosis: Find which programming concepts the student failed on an exercise by using GenAI.

Feedback: Provide contextualized feedback during an exercise attempt to the student, with the use of GenAI.

The paper is structured into several sections. Section 2 reviews the state of the art, focusing on ITS and the use of GenAI in programming education. Section 3 describes the system design, including the underlying tools and integration of AI components. Section 4 presents the validation methodology and discusses the obtained results. Finally, Section 5 concludes the paper by summarizing the main contributions and outlining possible directions for future work.

The goal of the validation process is not to evaluate the effectiveness of ITS themselves, as they have already been studied in the literature, but rather to assess the quality of the AI outputs in comparison with human judgment

2 State of the Art

This chapter reviews the current state of research relevant to this work, establishing the foundation for the proposed research and development. The review focuses on ITS and their underlying principles, as well as recent advances in AI, particularly the use of LLMs in programming education.

2.1 Intelligent Tutoring Systems

Intelligent Tutoring Systems (ITS) are systems designed to simulate the one-on-one cognitive and pedagogical benefits of a human tutor by adapting instruction to the needs, knowledge state, and learning pace of individual students.

Kurt Vanlehn [17] describes these systems as computer-based instructional environments that provide immediate, customized instruction and feedback to learners without requiring human intervention. Comprehensive analyses by Wei Ma et al. [10], and James Kulik and J. D. Fletcher [8] demonstrate that ITS interventions significantly improve learning outcomes and knowledge retention when compared to traditional studying from static texts.

Beverly Woolf [19] formalizes the classical ITS architecture as comprising a domain model, a student model and a tutoring model, often extended with an interface model.

The domain model represents the knowledge and skills to be taught. This is typically encoded as production rules or knowledge components (KCs), which correspond to skills required to solve a problem [2]. For example, in programming, KCs might include skills such as conditional statements or variables. In our work, instead of manually defining KCs, programming knowledge is represented using programming concepts automatically extracted from learning materials.

Knowledge graphs represent learning content as nodes (concepts) connected by directed edges (prerequisite relationships). For example, a simple concept graph might include nodes such as variables, functions, recursion, with edges indicating dependencies, such as variables to functions and functions to recursion.

Traditional ITS required experts to manually construct these KCs, this process is labor-intensive, domain-specific and difficult to scale. This domain building bottleneck limited the deployment of adaptive systems [16].

The student model maintains a dynamic representation of the learner's current knowledge state. It estimates mastery levels for individual KCs and updates these estimates based on observed performance. One of the most influential approaches is Bayesian Knowledge Tracing (BKT), introduced by Albert T. Corbett and John R. Anderson [4].

BKT models the acquisition of individual skills as a hidden Markov process in which each knowledge component is assumed to be either mastered or not mastered. It is parameterized by four probabilities: the initial probability of mastery, the probability of learning (transition from unmastered to mastered) and the probabilities of guessing and slipping, which account for correct responses despite lack of mastery and incorrect responses despite mastery, respectively. Based on student interactions, the model updates the probability that a learner has mastered a given concept. A concept is considered not yet mastered when this probability remains below a predefined threshold, indicating insufficient evidence of consistent performance.

An alternative approach to student modeling is Deep Knowledge Tracing (DKT), which uses recurrent neural networks to capture complex learning patterns, while it as shown effectiveness, it struggles with smaller datasets [20].

In this research, BKT was selected due to being effective in eductional settings with limited data and its simplicity, serving as a historically validated mechanism for updating student mastery in ITS.

The tutoring model governs instructional decisions. It selects tasks, generates hints, determines feedback timing and decides when a learner has achieved sufficient mastery. In cognitive tutors, this model often relies on mastery thresholds derived from probabilistic student modeling [2].

Finally, the interface model mediates the interaction between learner and system. The interface influences cognitive load and learner engagement.

Together, these components form the structural backbone of ITS and remain conceptually stable despite shifts in underlying computational techniques.

2.2 Generative AI in Programming Education

Generative Artificial Intelligence, particularly LLMs based on transformer architectures [18], present significant opportunities for programming education. Due to their ability to process and generate natural language, as well as support active interaction with the user, LLMs are well-suited for integration into learning environments. However, challenges regarding academic integrity, bias and hallucinated information remain [1]. As a result, prompting techniques such as zero-shot, few-shot, and chain-of-thought prompting have become increasingly important for ensuring the reliability and effectiveness of these systems [15].

Recent literature highlights several uses of GenAI in programming education. One prominent use is the provision of personalized feedback. LLMs are capable of interpreting problem statements alongside student submissions to generate contextualized hints. For instance, Tung Phung et al. [14] reported that while human tutors scored 92.0/100 on hint utility metrics, GPT-4 still attained a competitive score of 66.0/100. Similarly, Maciej

Pankiewicz and Ryan Baker [13] demonstrated that LLM-generated feedback for compiler errors reduced the number of non-compiling submissions, with 81% of students rating the feedback as useful.

Furthermore, incorporating submission history into feedback generation has been shown to improve user preference, with students favoring this approach 69% of the time [12]. Another important application of GenAI is exercise generation. These systems can significantly reduce instructor workload by automatically generating multiple-choice questions (MCQs) aligned with specific learning objectives [5]. Moreover, contextualizing programming exercises using AI has also been shown to yield high-quality problems that increase student engagement and motivation [6]. LLMs have also demonstrated the ability to generate quizzes for Java programming courses with a relatively low error rate of approximately 10% [9].

Subhankar Maity and Aniket Deroy [11] highlight possible applications for GenAI in ITS. These include automated question generation tailored to the learner´s current level of understanding, providing personalized feedback tailored to the specific errors and misconceptions identified and iterative dialog systems to simulate human-like conversations and guide students through learning complex concepts in a conversational manner.

3 System Design

Agni is a web-based platform designed for learning programming. It consists of a teacher interface (for content and module management) and a student interface (for accessing exposition materials and evaluative components like quizzes and coding exercises).

The frontend of the platform was developed with Vue and the backend with Strapi. Vue is a component based JavaScript framework used for building user interfaces and single-page applications. While Strapi is an open-source headless content management system (CMS).

Within the context of this work, Agni serves as the core platform where the proposed features are implemented and evaluated.

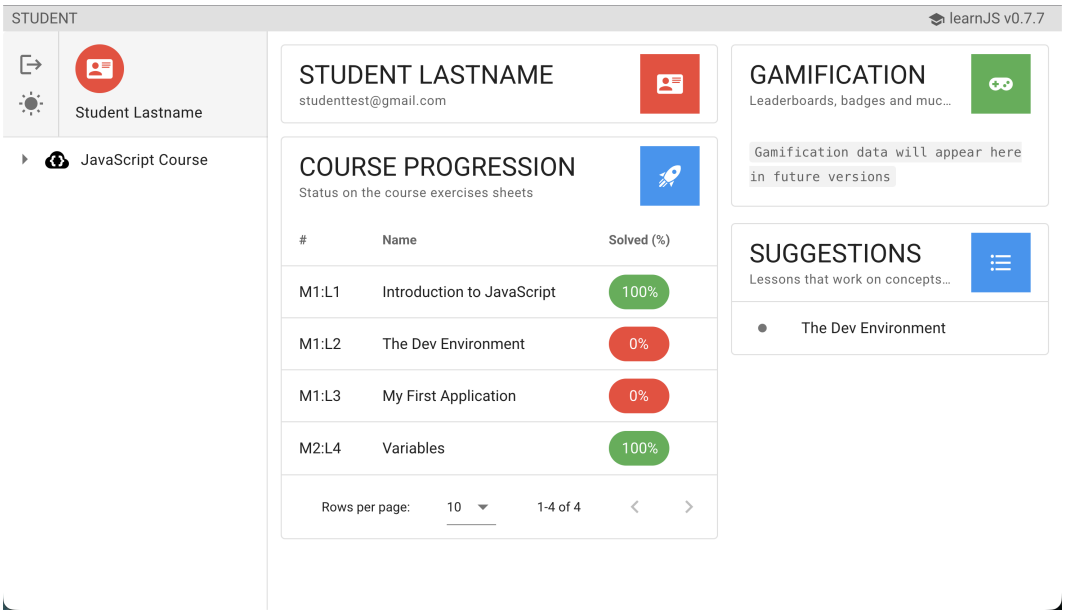


Figure 1 Student Interface.

3.1 Adaptive Learning

To track student knowledge, the system integrates Bayesian Knowledge Tracing (BKT), enabling a probabilistic representation of student mastery of individual concepts. To support this, a new ConceptMastery collection was introduced in the data model. This structure associates each student with a set of probabilities representing their mastery level for each concept.

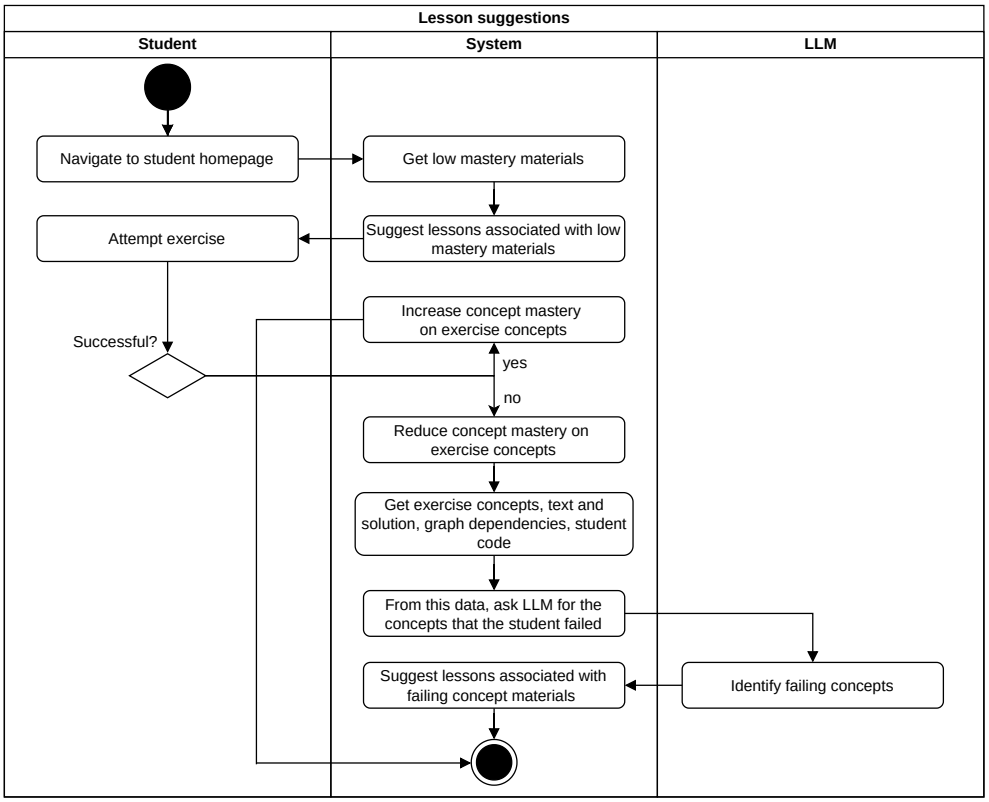


Figure 2 Lesson suggestions activity diagram.

As illustrated in Figure 2, the process begins when a student interacts with the system by attempting an exercise. Based on the outcome, the system determines whether the attempt was successful and updates, using BKT, the corresponding student mastery values of the concepts associated with the exercise through the ITS system. The previous mastery values of the concepts and the attempt outcome are used in this calculation. A successful attempt increases the probability of mastery for the concepts associated with the exercise, while an unsuccessful attempt decreases it. This allows the system to maintain a continuously updated representation of the student's knowledge state.

When a student fails to solve an exercise, the system performs a deeper analysis to identify the underlying causes. It retrieves relevant information, including the concepts associated with the material, their dependencies within the concept graph, the student's submitted code, the problem description and the expected solution. The described information is then provided to the LLM through a prompt that requests the concepts that likely caused the student to fail, this identifies the specific concepts the student is struggling with. Having the list of failed concepts returned by the AI, the system searches and recommends lessons associated with these concepts through the learning materials.

In addition, the system is capable of suggesting lessons covering materials for which the student has low mastery by getting all concept masteries calculated with BKT, checking which ones are below a defined threshold and finding lessons associated with those concepts through the Expositives or Evaluatives, illustrated in Figure 1 through the Suggestions panel. These recommendations are presented through the Agni interface, guiding the student toward relevant content.

This allows the system to personalize the learning experience by guiding students toward content that addresses their weaknesses, improving learning efficiency and engagement.

3.2 Concept Extraction

A key component of the system is the automatic extraction of programming concepts from educational materials. As illustrated in Figure 3, this process involves the teacher, the system and an LLM. The output generated by the system within Agni can be seen in figure 4.

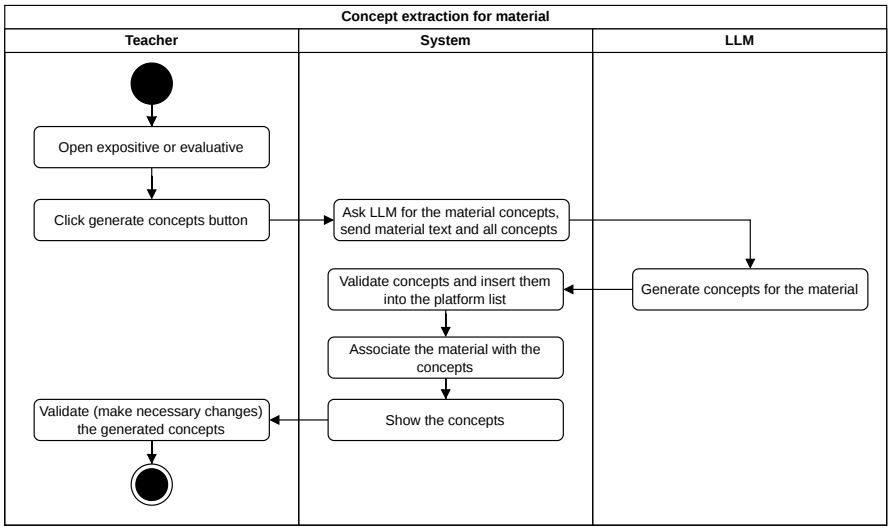


Figure 3 Concept extraction activity diagram.

The workflow begins when the teacher initiates the process by selecting the generate concepts option within a selected Expositive (material) or Evaluative (exercise) in the Agni platform.

The system then constructs a structured prompt that includes the material text along with any existing concepts in Agni and sends this information to the LLM. Based on this input, a list of concepts that represent the knowledge explicitly covered in the material is generated. Since the output of this process has high variance, this operation is repeated 5 times and the concepts the appear the majority of the time stay in the final output.

Once the concepts are returned, the system performs a validation step to ensure consistency. This includes aligning the generated concepts with the existing ones in the platform. The validated concepts are then used to automatically tag the material, establishing a relationship between the content and the extracted concepts. The resulting concept list is presented to the teacher through the Agni interface.

At this stage, the teacher remains in the loop and can add or remove concepts as needed, maintaining control over the domain model. This semi-automated approach balances workload with user control. To support this functionality, the data model was extended with a new Concept collection.

For example, given a learning resource introducing basic programming, the system may extract concepts such as variables or functions. These concepts are then associated with the material and can later be used for tasks such as student modeling and graph generation.

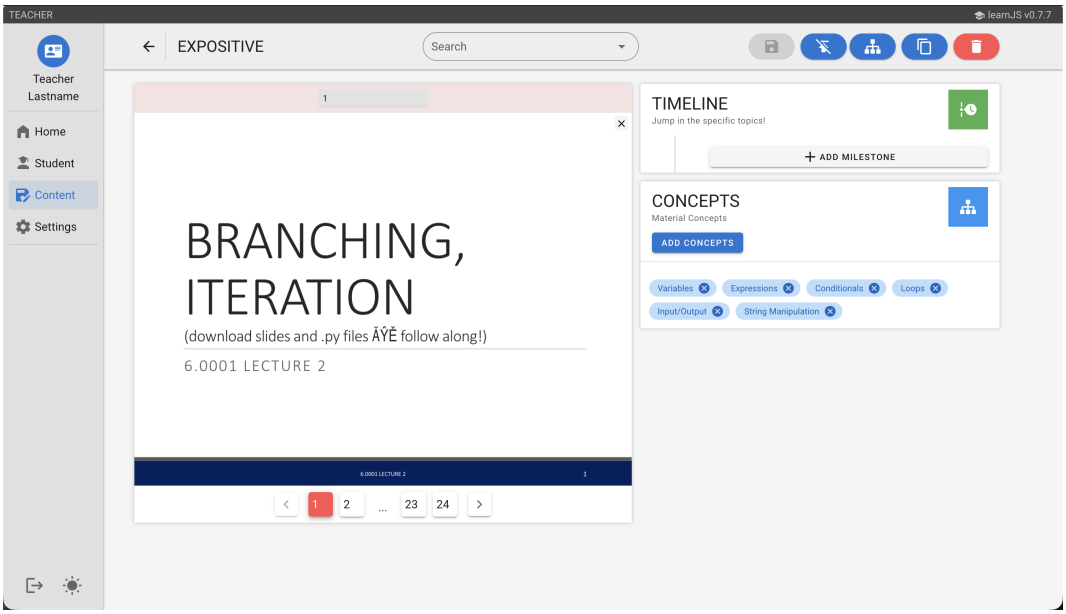


Figure 4 Extracted Concepts.

3.3 Concept Graph Generation

Building on the extracted concepts, the system supports the automatic generation of graphs representing relationships between programming concepts at the course level. As illustrated in Figure 6, this process involves the teacher, the system and an LLM. The teacher provides high-level validation, while the LLM assists in identifying and structuring relationships between concepts.

The workflow is initiated by selecting the generate graph option within a selected course in the Agni platform. The system first ensures that all course materials are associated with concepts. For materials that have not yet been tagged, the system queries the LLM to extract the corresponding concepts using the approach described in Section 3.2.

Once every material in the course has an associated list of concepts, the system constructs a structured prompt that includes all the course material concepts and the graph rules, such as output directed edges and use only the provided concepts. Prompt-engineering techniques are employed, namely few-shot and constraint-based prompting, and a request to the selected LLM is made, generating a concept graph. The output is a directed graph in which nodes represent concepts and edges represent prerequisite relationships. This representation captures dependencies between topics, enabling the system to model the progression of knowledge.

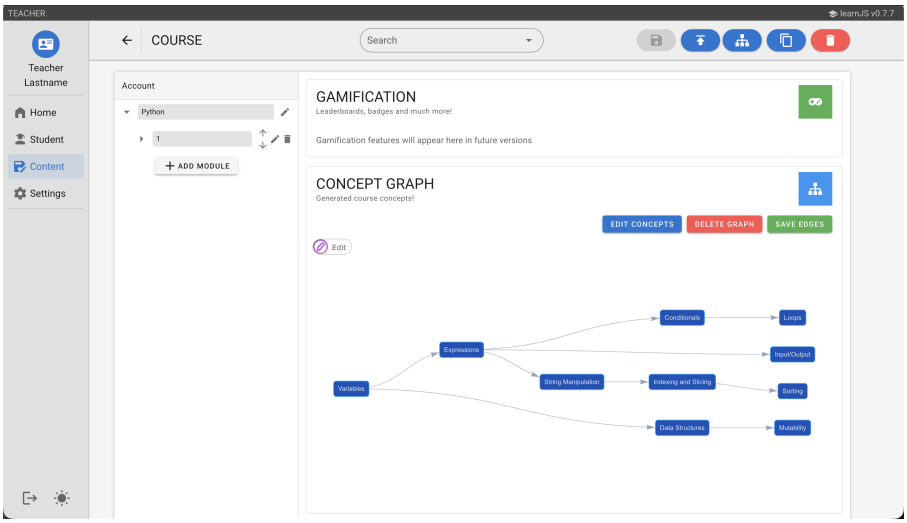


Figure 5 Generated Graph.

The generated graph is then presented to the teacher through Agni, as shown in Figure 5. As with concept extraction, the teacher remains in the loop and can make adjustments such as adding or removing concepts and edges, to ensure control remains with the user.

Strapi was extended with new ConceptGraph and ConceptEdges collections and the existing Course collection was updated with a conceptGraph field, representing the relationship to the ConceptGraph.

By automating this process, this system reduces the need for manual building of the domain model while preserving teacher control. This approach reduces workload and improves scalability.

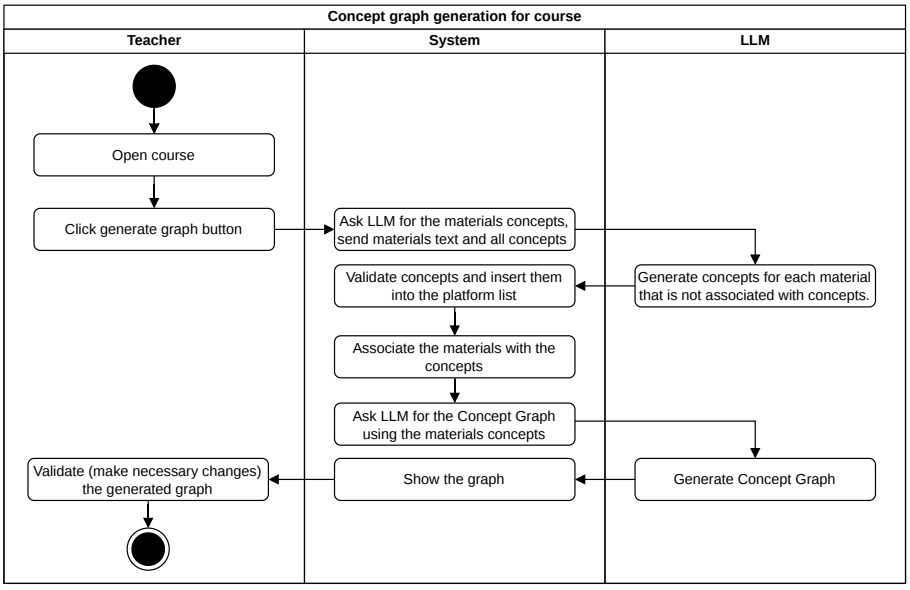


Figure 6 Graph generation activity diagram.

3.4 Feedback Generation

Generative Artificial Intelligence is leveraged to provide feedback during programming exercises. During an exercise attempt, the system analyzes the student's code alongside the problem description and expected solution. This information is provided to the LLM through a prompt, that includes rules such as providing feedback about only one bug, not providing the answer or any code and limiting the response to two sentences at most.

The system generates hints that encourage incremental progress without revealing the answer, as described by the prompt rules.

The generated feedback is delivered through the Agni interface. By integrating feedback generation, the system enhances the learning experience by providing personalized assistance that mimics the support of a human tutor.

3.5 AI Application Interfaces

The system integrates multiple AI APIs to support its functionalities. These APIs provide access to large language models used for concept extraction, feedback generation and other intelligent features. The integrated APIs are Iaedu, Groq and Gemini.

Iaedu[1] is a service from Fundação para a Ciência e a Tecnologia (FCT) developed by the digital services of FCT that aims to provide students, teachers and researchers access to different large language models; at the time of this work only the GPT-4o model is available through the API. The main advantage of using Iaedu is the absence of strict usage limits, making it suitable for applications with high token consumption.

Groq[2] is a low cost service that contains plenty of GenAI models to choose from. In this work, the Groq API is used to access language models, specifically a 120-billion-parameter model (GPT 120B). One of the main advantages of using Groq lies in its optimized inference performance. The platform is designed to deliver responses significantly faster than traditional GPU-based solutions The use of the Groq API involves considerations similar to other AI services, like usage limits

Gemini[3] is an LLM, developed by Google, designed to support tasks involving natural language understanding and multimodal reasoning. Using the Gemini API involves costs associated with the selected model and the volume of data processed, typically measured in tokens.

The Strapi data model was changed to accommodate these new services. Tutors can configure their own API keys on each supported API.

4 Validation

The evaluation is a fundamental component of this work, as it aims to assess the effectiveness and reliability of the proposed GenAI functionalities within the learning environment. It was conducted through a series of targeted questionnaires using Google Forms, each addressing a specific component of the system. These components include concept identification, concept graph generation, failed concept detection and feedback generation.

This section is divided into two parts. First, the methodology subsection describes the design of the evaluation process, including the questionnaires, participant profiles, and the metrics used to assess different system components. Second, the results and discussion

[1] https://iaedu.pt
[2] https://groq.com
[3] https://gemini.google.com

subsection presents and analyzes the findings obtained from these evaluations, covering concept identification, concept validation, concept graph validation, failed concept detection, and feedback rating.

4.1 Methodology

The evaluation methodology was designed to assess the quality of AI-generated outputs by their alignment with human judgment and the agreement between humans when identifying concepts. Multiple questionnaires were created, each corresponding to specific system components. Two of the questionnaires addressed concept identification. A third questionnaire focused on concept graph validation. The fourth one covered failed concept detection and feedback generation.

The respondents to the questionnaires consisted of 6 individuals, 4 male and 2 female, ages ranging from 20 to 30, with backgrounds in computer science and related fields, including a student enrolled in the Master in Computer Science Program of FCUP, one enrolled in the Master in Information Security Program of FCUP, another enrolled in the Master in Artificial Intelligence Program of FCUP and FEUP, a researcher at INESC TEC with a Master's degree in Computer Science from FCUP, a student in the Informatics Engineering Degree at ISTEC, and a software developer holding a degree in Informatics Engineering from ISEP.

Inter-annotator agreement is the degree to which different individuals provide consistent judgments when evaluating the same data. It was measured for concept identification and concept validation using Gwet's AC1 metric [7]. The computations were performed using the irrCAC library in Python. Gwet's AC1 produces a coefficient ranging from -1 to 1, where higher values indicate stronger agreement among annotators. A value of 1 represents perfect agreement, 0 indicates random chance agreement and -1 represents perfect disagreement. Also, for concept graph edges and concepts validation, the mean of the responses was measured to calculate how much the participants agree with the AI generated structures on average.

In the concept identification task, participants and the three AI APIs were asked to analyze two selected programming learning materials and manually extract the concepts being taught. The two materials are from a Python course[4], the first has the title "Branching, Iteration" and the second is titled "Tuples, Lists, Aliasing, Mutability, Cloning".

Subsequently, a second questionnaire was constructed by aggregating all concepts identified by the participants and the ones Gemini extracted from the materials. Iaedu and Groq were not used when building this questionnaire because Iaedu was not reliably working when it was made and Groq wasn't as tested in the Agni platform yet. In this phase, participants were asked to validate each concept and indicate their agreement using a binary (agree/disagree) response.

Concept graph validation required participants to evaluate whether they agreed with the AI generated relationships between concepts. Each relationship (edge) was presented individually, and participants indicated agreement using a binary (agree/disagree) response.

To evaluate failed concept detection, participants assessed student solutions to two programming exercises, the first is about string compression and the second about finding the max number in an array, and were asked to identify which concepts led to failure from a list of concepts, this was then compared to the concepts diagnosed by the system.

[4] Introduction to Computer Science and Programming in Python

Finally, in the feedback generation task, participants reviewed wrong student submissions to the same two programming exercises and evaluated the generated hint on a scale from 1 to 5.

4.2 Results and Discussion

With the results from the concept identification questionnaire, inter annotator agreement was calculated from the responses, the agreement between 6 humans without the AI responses was 26.6% for material 1 and 52% for material 2, while with the AI participants added it was 26.8% for material 1 and 19% for material 2. This shows that when participants have to identify the concepts without having to choose from a predefined list they tend to list different concept names. Also, the calculated agreement with AI as another participant can be on par with the humans as seen with the calculated agreement for all annotators and the agreement between annotator pairs illustrated by the heat map in Figure 7 for material 1, showing that AI and humans can have similar agreement. However, the annotator agreement varied greatly according to the analyzed material, material 1 had almost equal annotator agreement between humans and AI, while for material 2 it was the opposite. Even with this discrepancy, in the validation step the participants showed agreement with the AI generated concepts, which might mean the discrepancy comes from differences in generalization or naming of the knowledge and not disagreement.

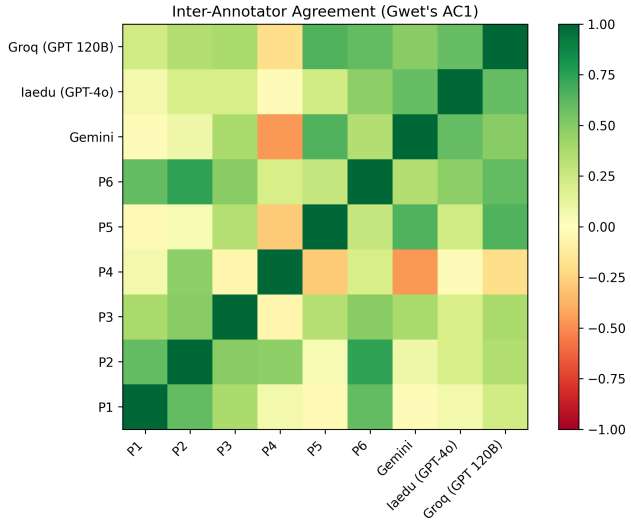


Figure 7 Inter-annotator agreement between pairs for identification (Material 1).

Having the concept validation responses for two materials, annotator agreement between participants and average agreement with AI concepts was calculated. For material 1, the annotator agreement was 75% and the average agreement with AI generated concepts was 83%. While for material 2 the annotator agreement was 63% and AI concepts average agreement was 69%. The agreement between annotator pairs for material 1 is illustrated in Figure 8. These results show good participant agreement with each other in both materials and high agreement from the participants with the AI generated concepts. Supporting the idea that GenAI can extract explicitly taught concepts from learning materials. This also shows that if given a predefined list of concepts, participants will have higher agreement with each other, most likely explained by differences in generalization or naming of the concepts, as illustrated in the contrast between Figures 7 and 8.

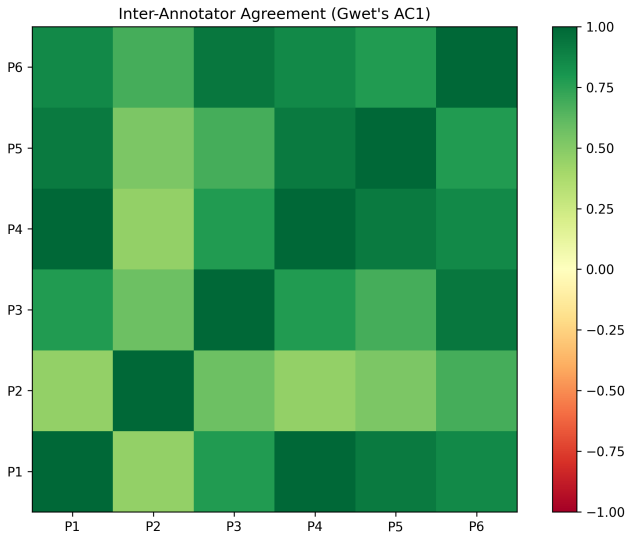


Figure 8 Inter-annotator agreement between pairs for validation (Material 1).

On the topic of the concept graph validation. Overall, the agreement with AI generated edges was on average 78%. Indicating a strong alignment with the AI generated relationships from the participants. Suggesting that the proposed approach is effective in capturing meaningful conceptual dependencies between programming concepts.

For failed concept detection, agreement between participants and the system was evaluated across two programming exercises. In Exercise 1, the system identified String Manipulation and Expressions as the failed concepts. All participants agreed with String Manipulation, while only one selected Expressions, and another identified Variables as an additional failed concept. In Exercise 2, Comparison was identified as the failed concept, with all participants in agreement, although one participant also selected Expressions. These results suggest that the system is generally capable of identifying the underlying concepts responsible for student errors, although some discrepancies may arise in some cases.

Feedback generation was evaluated based on participant ratings of the generated hints for two exercise attempts. Both hints received an average rating of 4.83 out of 5. Overall, the generated feedback was perceived as useful, indicating that the system can provide meaningful guidance to students.

5 Conclusion and Future Work

This paper presented an approach for integrating GenAI into ITS to automate the construction of domain knowledge and feedback mechanisms. One of the main contributions of this work is the exploration of whether GenAI can effectively support tasks such as concept extraction, concept graph generation and feedback provision, which have traditionally relied on manually engineered components. Another contribution is the developed system which is available online[5].

[5] `https://agni.dcc.fc.up.pt`

Across all evaluated components, the results indicate that the proposed approach produces outputs that are largely consistent with human judgment. While disagreements were observed, these are expected and limited by the teachers keeping full control of the domain. Together, these findings support the viability of the approach in assisting the teacher with domain modeling and feedback generation in programming education.

In future work, further improvements will be made to the quality and robustness of AI generated outputs, particularly through the refinement of the prompts and the prompting techniques. The evaluation will also be extended with larger-scale user studies to better assess the effectiveness of the tasks. Further research may also explore the integration of more advanced student modeling approaches.

References

1 Gökhan Akçapınar and Esra Sidan. Ai chatbots in programming education: Guiding success or encouraging plagiarism. *Discover Artificial Intelligence*, 4:87, 2024. doi:10.1007/s44163-024-00203-7.

2 John Anderson, Albert Corbett, Kenneth Koedinger, and Ray Pelletier. Cognitive tutors: Lessons learned. *Journal of the Learning Sciences*, 4:167–207, April 1995. doi:10.1207/s15327809jls0402_2.

3 Yannik Bauer, José Paulo Leal, and Ricardo Queirós. Improving teacher‘s user experience in a virtual learning environment. Master’s thesis, Universidade do Porto, 2023.

4 Albert T. Corbett and John R. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4(4):253–278, 1994. doi:10.1007/BF01099821.

5 Jacob Doughty, Zipiao Wan, Anishka Bompelli, Jubahed Qayum, Taozhi Wang, Juran Zhang, Yujia Zheng, Aidan Doyle, Pragnya Sridhar, Arav Agarwal, Christopher Bogart, Eric Keylor, Can Kultur, Jaromir Savelka, and Majd Sakr. A comparative study of ai-generated (gpt-4) and human-crafted mcqs in programming education. In *Proceedings of the 26th Australasian Computing Education Conference*, ACE 2024, pages 114–123. ACM, January 2024. doi:10.1145/3636243.3636256.

6 Andre Gutierrez, Paul Denny, and Andrew Luxton-Reilly. Evaluating automatically generated contextualised programming exercises. In *Proceedings of the 2024 ACM Technical Symposium on Computer Science Education*, pages 289–295, March 2024. doi:10.1145/3626252.3630863.

7 Kilem Gwet. *Handbook of inter-rater reliability: The definitive guide to measuring the extent of agreement among raters*. Advanced Analytics, LLC, January 2012.

8 James Kulik and J. D. Fletcher. Effectiveness of intelligent tutoring systems: A meta-analytic review. *Review of Educational Research*, 86, April 2015. doi:10.3102/0034654315581420.

9 Yulia Kumar, Anjana Manikandan, J. Jenny Li, and Patricia Morreale. Optimizing large language models for auto-generation of programming quizzes. In *2024 IEEE Integrated STEM Education Conference (ISEC)*, pages 1–5, 2024. doi:10.1109/ISEC61299.2024.10665141.

10 Wei Ma, Olusola O. Adesope, John C. Nesbit, and Qiang Liu. Intelligent tutoring systems and learning outcomes: A meta-analysis. *Journal of Educational Psychology*, 106(4):901–918, 2014. doi:10.1037/a0037123.

11 Subhankar Maity and Aniket Deroy. Generative ai and its impact on personalized intelligent tutoring systems, 2024. doi:10.48550/arXiv.2410.10650.

12 Moritz Mueller, Corinna List, and Michael Kipp. The power of context: An llm-based programming tutor with focused and proactive feedback. *Proceedings of the 6th European Conference on Software Engineering Education*, 2025.

13 Maciej Pankiewicz and Ryan Baker. *Enhancing Student Focus and Problem-Solving with Real-Time LLM Feedback on Compiler Errors*, pages 412–426. Springer-Verlag, September 2025. doi:10.1007/978-3-032-03870-8_28.

14 Tung Phung, Mengyan Wu, Heeryung Choi, Gustavo Soares, Sumit Gulwani, Adish Singla, and Christopher Brooks. Bridging gaps between student and expert evaluations of ai-generated programming hints, 2025. doi:10.48550/arXiv.2509.03269.

15 Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications, 2025. doi:10.48550/arXiv.2402.07927.

16 Pramuditha Suraweera, Antonija Mitrovic, and Brent Martin. Widening the knowledge acquisition bottleneck for constraint-based tutors. *I. J. Artificial Intelligence in Education*, 20:137–173, January 2010. doi:10.3233/JAI-2010-0005.

17 KURT VanLEHN. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221, 2011. doi:10.1080/00461520.2011.611369.

18 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. arXiv:1706.03762.

19 Beverly Woolf. *Building Intelligent Interactive Tutors, Student-Centered Strategies for Revolutionizing E-Learning*. Elsevier and Morgan Kaufmann, January 2008.

20 Xin Zhou, Zhuoxu Zhang, Xike Xie, and Jiawei Zhang. Deep learning based knowledge tracing in intelligent tutoring systems. *Scientific Reports*, 15, July 2025. doi:10.1038/s41598-025-07422-7.

Codula, a Social Network Geared Towards Programming Education

João Rodrigues
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal

Alice Dutra Balbé
Communication and Society Research Centre, CECS, University of Minho, Braga, Portugal

Alvaro Costa Neto
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal

Pedro Rangel Henriques
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal

Abstract

Digital social networks have become an ubiquitous presence to almost everyone, to the point that they are frequently taken for granted whenever someone gets asked for their contact. Beyond that, it is without doubt that their capacity for stimulating participation in groups and exchange of ideas, while retaining attention for long periods of time is – for good, or for bad – highly efficient. If taken for a good cause, their features can become fierce allies to educators in their teaching efforts. Specially in the case of Computer Programming, a sharp, and difficult matter, social networks can be of great value, not only for their inherent technological link to Computer Programming, but also for their natural connection to students that dare to pursue the arduous path to become programmers. Codula is a social network that was designed and implemented with the purpose of paving and shortening this path, both for students, and educators alike. This paper presents both Codula's design principles that guided its construction, and main features, that were carefully crafted to fulfil its purpose. While foundational concepts solidified design choices, such as the absence of different roles for users, and the stimulus of the sense of belonging via personal customisation, features were implemented to directly support Computer Programming Education, such as the specification of different post types, and functionalities that facilitate composing, sharing, and commenting source code. By converting opposing forces of a commonly found distraction, Codula aims to leverage the best aspects of social networks to support a better cause, in the hopes of stimulating a successful collective teaching-learning process.

2012 ACM Subject Classification Applied computing → Collaborative learning; Human-centered computing → Social networking sites

Keywords and phrases Programming Education, Computer Programming, Social Network, Engagement

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.3

Category Short Paper

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Unit Project Scope UID/00319/2025 – Centro ALGORITMI (ALGORITMI/UM) `https://doi.org/10.54499/UID/00319/2025`.

1 Introduction

Computer programming presents a series of difficult challenges that students need to overcome in order to construct knowledge. Being an intrinsically complex problem to solve, strategies, methods and tools to overcome these commonly found challenges have been researched for decades [15, 10]. Difficulties range from the inflexible syntaxes of programming languages

that are wildly different from their natural counterparts, through the required understanding of a problem's domain and its constraints, to the arduous and naturally abstract translation process from ideas and concepts to implementation [7].

It has been shown that learning is an inherent social process. Vygostky [16], through his Cultural-Historical Activity Theory presented the world with the idea that, in order to reach higher potential, students need a mediator that bridges the gap between what they can learn by themselves (their inherent capacity) and what could be learned via guidance.

Digital social networks provide an important tool for information exchange, with ample reach and high retention rates [12]. While these are all double-edged features when it comes to benefits or maladies, it is more than obvious that if this retention potential should be geared towards learning approaches, social networks would provide invaluable resources for teaching and learning, specially if pedagogical theories such as Vygostky's is taken into consideration. Codula[1] is a novel proposal in this regard, a social network that provides specific features that have been designed to support learning computer programming. Through its focus on facilitating the mediation of peers and teachers, it cements itself in Vygostsky's groundwork in order to retain and stimulate students to learn how to program computers together.

This article is divided into five sections. After the introduction in this section, Section 2 presents both a brief history and foundation of Social Networks, as well as a survey of features commonly found in them. Section 3 presents Codula, a Social Network designed to support Computer Programming Education. Section 4 reports on the preliminary evaluation results, and finally, Section 5 concludes the paper with final remarks, as well as prospects for future works.

2 Social Networks and their Motivational Factors

The expansion of communication networks and digital services has reshaped how people access information, education, employment, and public services over the past 30 years [14], accelerated globalisation, and redefined civic participation. Many of today's leading global companies either rely completely on digital connectivity or are built entirely around it [14] and are deeply integrated with social networks. This section contextualizes how social networks are used and their potential for engagement and learning programming languages.

Social networks are digital platforms that mediate personal, professional, and social relationships. They are web-based services that allow people to create connections, share content, and interact with other users. By definition, social network sites allow users to create personal profiles (public or semi-public) and connection lists, where users can view the connections of other users [2]. The terminology for connections may vary between platforms, such as friends or followers. Nowadays, it influences information consumer habits and connects billions of people worldwide [9], especially through mobile device applications. Their ability to integrate users, content creators, and diverse media formats has led many platforms to also be characterized as social media, extending the network concept to video-sharing, gaming, and blogs. This networking strengthens existing or new social ties. In other words, they have taken sociality, a characteristic of the network society, theorized by Castells in the 1990s, into its greatest applicability. According to the author, this shift has accelerated over the last decade, driven by advances in connectivity, storage capacity (Big Data), and Artificial Intelligence (AI) [4].

[1] Codula is available at: `https://codula.epl.di.uminho.pt/`

Within the context of collective knowledge construction, there are also collaborative platforms based on the Wiki principle. These are open platforms where anyone can edit and add content in a cooperative writing system. It is the hypertext that forms the community, where no one owns the content. The best-known platform is Wikipedia, a multilingual encyclopedia project run by volunteer editors. The dynamics of knowledge sharing can also be observed in other virtual communities. Chuang [5] analysed online gaming communities of the MMORPG (Massively Multiplayer Online Role-Playing Games), focusing on the sense of belonging and immersion of users in these communities. Among the results, he identified the importance of users feeling capable of helping others and how this strengthens the community. In addition to considering the Sense of Virtual Community and Social Cognitive Theory (centred on self-efficacy in knowledge), the author also included Uses and Gratifications Theory [11].

2.1 Forums *versus* Social Networks

Research suggests that social learning environments help people retain information better than when learning alone and stay engaged for longer periods. In addition to social networks and wikis, there are forums. Interaction in forums is more about questions and answers than about individual interactions, unlike in social networks. Their focus is on building communities around a specific topic rather than on the users themselves [8]. The opportunity to discuss topics, share ideas, and receive feedback makes the learning process more active and rewarding [17]. By encouraging people to ask questions and work together, users tend to feel more supported and motivated to keep learning [15].

Recent studies involving the use of social media in programming courses have shown positive results in the educational environment. Students in the Object-Oriented Programming Laboratory 2 course who used Facebook groups and YouTube videos answered surveys, and the researchers found that students improved their understanding of academic concepts [1]. In some cases, young people spend more time on social media than on homework, which negatively impacts learning objectives [13]. A widely recognized example of an educational platform is Moodle, which is extensively used in schools and universities as a Learning Management System (LMS). It enables teachers to share resources, assign tasks, and communicate with students within a single environment, supporting traditional and online classrooms. Although Moodle is not a traditional social network, it incorporates features that promote interactive learning, including discussion forums, messaging, and collaborative workspaces [18].

In the case of studies that compare student engagement between groups on social networks and forums within Massive Open Online Courses (MOOC), the result is different. Based on interviews with Coursera platform users and Facebook users, the researchers have identified higher engagement with social networks [17]. According to participants, social networks offer more resources that encourage interaction. In addition, the researchers found that, as a key engagement, establishing trust between users was needed in both cases. There are also platforms that work on a question-and-answer (Q&A) model, and these platforms work like forums, such as Reddit, Quora, Stack Overflow and Dev Community.

Similar to social bonds built in offline contexts, social interaction contributes to the development of new skills. Thus, these online networks of contacts enhance trust, while reputation systems encourage participation and social learning [3].

Table 1 Features surveyed from selected social networks and forums.

Name	Type	Community	Content Types	Engagement
DEV Community	Forum	Followers, Communities	Text, links, images	Comments, reactions, badges
Facebook	Network	Friends, Followers, Groups	Text, images, videos	Likes, comments, verified
Instagram	Network	Followers	Images, videos	Likes, comments, verified
LinkedIn	Network	Connections, Groups	Text, media	Reactions, endorsements
Quora	Forum	Followers, Topics	Text, links	Up/Downvotes, comments
Reddit	Forum	Communities	Text, links, media	Up/Downvotes, comments
ResearchGate	Network	Followers, Communities	Articles, text	Citations, recommendations
Stack Overflow	Forum	None	Text (Q&A)	Reputation, votes, badges
TikTok	Network	Followers	Videos	Likes, comments, verified

2.2 A Technical Survey of Current Social Networks

In the last two decades, social networks have emerged for different purposes, whether personal (e.g., Instagram) or professional (e.g., LinkedIn), and with distinct functionalities. For example, they all allow sending private messages and suggest connections and content based on users' interests and reactions on each social network.

Considering that social networks depend on users and their interactions, studies identified that for user engagement, there must be a topic of interest, trust, emotion (positive or negative), commitment, and how much effort (convenience) is necessary to interact [6]. Design-related issues and involvement of acquaintances also contribute to the adoption and acceptance of a social network, such as the different types of posts that allow images and links, hashtags, post interaction tools, targeted responses, and interest groups, as Table 1 summarizes. Based on this technical survey (Table 1), we adopted for Codula a basic set of functionalities that are commonplace in other social networks: group-based communities and followers; text and image content types; comments, likes and reputation engagement mechanisms. Beyond their familiarity for users, the chosen features also allow for effective communication using snippets of source code, a requirement for Codula's objectives.

3 Codula, a Social Network for Programming Education

Codula's design was guided by a set of ontologies, which defined the main concepts of the platform and their relationships, and served as a foundation for its structure and features. For that purpose, three main ontologies were developed: Core, Resources, and Interaction.

The Core ontology, illustrated in Figure 1, represents the main concepts for the context, objectives and relations within the platform, focusing on the actors and how they interact. The main actors in this ontology are *Student* and *Tutor*, representing the target audience for Codula. The system is primarily directed at students, as its goal is to support programming learning, while tutors represent guidance and support during this process. Even though this conceptual distinction between students and tutors was relevant to specify the target audience, differentiating roles were not implemented, as they could create a symbolic stratification between users and indirectly coerce interactions.

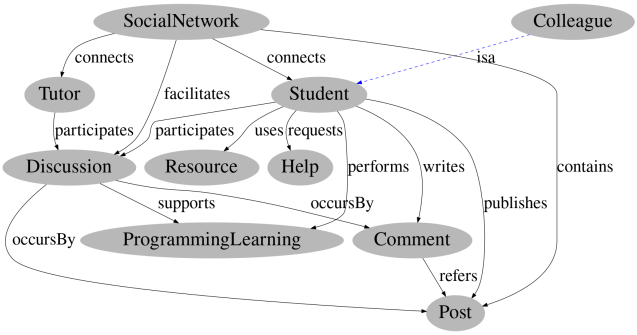


Figure 1 Core concepts of the ontology.

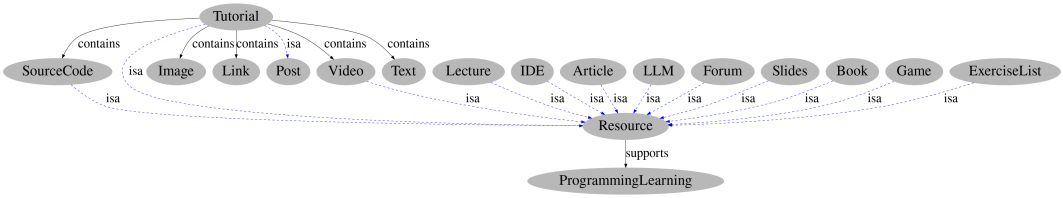


Figure 2 Resource concepts of the ontology.

The Resources ontology, shown in Figure 2, represents different tools that can support programming learning, both in digital and physical forms. It includes elements such as tutorials, articles, code snippets, and other forms of learning resources. These were modelled as different learning tools that users usually use to overcome difficulties, and were defined during the design phase as a reference for the platform's features. Given overall time and resource constraints, Codula implemented a subset of these resources, while keeping its source code structured for future developments.

The Interaction ontology, presented in Figure 3, describes how users engage with the platform. It includes actions such as requesting help and collaborating with others. It also represents different ways in which help can be provided, such as through tutorials, other users, or tools like AI assistants. Again, a subset of these interactions was implemented by Codula.

As per implementation, Codula follows a structure similar to other social networks, allowing users to share content and interact with each other in order to capitalise on students' previous experiences with social networks. At the same time, it includes features that differentiate it and make it more suitable for programming education.

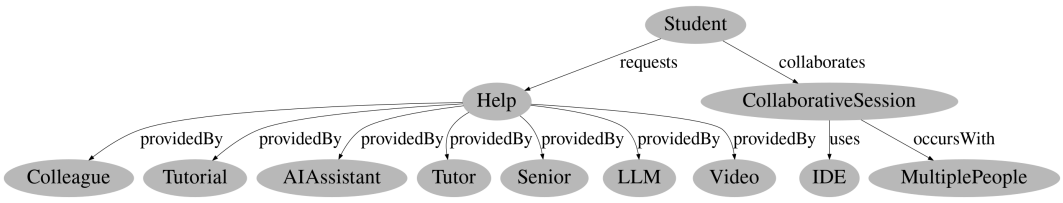


Figure 3 Interaction concepts of the ontology.

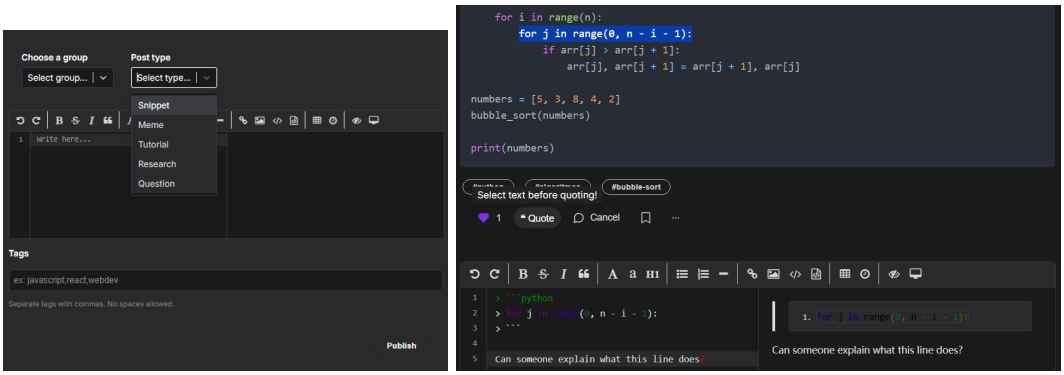


Figure 4 Post creation (left) and quoting (right) interfaces.

3.1 Writing Posts

The first dedicated interface Codula brings to the table is the page for writing a new post. Creating a post in Codula was designed to be a structured yet flexible process, allowing users to share different types of content according to their needs. Nonetheless, as the main artefact for programming education is very commonly a source code, the interface was implemented to allow users to quickly and directly write formatted snippets in several languages. The interface uses a Markdown-enabled editor and a live preview of the formatting, including source code highlighting (Figure 4, left). This improves both instant gratification and validation, as the visual formatting can aid in finding little bugs while typing.

The type of the post can also be selected, supporting five basic options: *Snippet*, *Meme*, *Tutorial*, *Research*, and *Question*. Each post type adds different fields for metadata that complements the main text itself. In addition, tags can be added to describe the topic of the post and make it easier for others to find related content.

3.2 Quote Functionality

The second feature that Codula implemented focused on programming education is a quote functionality that allows users to select and quote specific portions of text from posts or comments when replying, more specifically, parts of a source code snippet (Figure 4, right). This feature is particularly relevant in programming education, where discussions often focus on small fragments of code. By isolating only the relevant content for a response, users can either ask or provide answers for more direct and structured parts of a post.

Instead of requiring users to manually copy and paste content, the comment system provides a direct mechanism for users to select text and automatically insert it into the reply editor. Quoted content is visually distinguished from the rest of the message, making references clear and improving the overall readability. A key aspect of this feature is its support for source code. When quoting code, the platform preserves its original formatting, including indentation, line breaks, and syntax highlighting. This ensures that the quoted content remains accurate and easy to interpret, which is essential in programming contexts.

4 Evaluation

Codula's evaluation, while not yet finished, was already realised through multiple testing stages, involving different types of users and focusing on both technical and educational aspects of the platform.

The first stage consisted of informal testing with a small group of selected users, mainly colleagues, who interacted with an early version of the platform. The main goal of this phase was to identify bugs and usability issues. This initial testing helped uncover several problems related to navigation, layout, and overall interaction with the system. For example, improvements were made to how posts are displayed, avoiding long posts taking too much space in the feed, and making navigation smoother. Other changes included adding missing interaction features, such as viewing likes and user connections, as well as improving performance by loading posts progressively instead of all at once.

After this initial phase, the platform was presented to 4 programming teachers from both Portugal and Brazil. The goal was to demonstrate how Codula works and explore its potential as a support tool in programming classes. This stage provided more focused feedback from an educational perspective, helping identify how certain features could be simplified or adapted to better fit classroom usage, such as enabling the execution of code directly within snippet posts, adding visual indicators for new content, and displaying a small preview of comments under each post.

Codula was also presented to students from different classes, including students who had just started their degree in Software Engineering and master's students in Programming Education. These sessions allowed collecting early impressions and suggestions from potential users. Several ideas emerged during this stage, such as the implementation of a chat system between users, the creation of fixed groups for programming languages (C, C++, Python, etc.) and other common topics without requiring users to create them manually, and the ability to pin answers within a post. The possibility of executing code directly within the platform was also highlighted. These suggestions highlighted areas for future improvement and helped refine the platform's direction.

5 Conclusion

Teaching/learning Computer Programming is one of the hardest educational processes due to the unusual skills required to express in algorithmic format the steps to solve a problem using the computer. In the last two decades, many pedagogical approaches and computer-aided tools have been used to mitigate these difficulties. The success is still limited, because engagement is a fundamental requirement for students to persevere looking for the correct or optimized resolution for a given problem. However the lack of strong intrinsic motivation does not provide enough engagement. To overcome these flaws in the teaching/learning process under this unfavourable context, we decided to trace a new path taking advantage of online social networks addictive power. This purpose led us to design and develop Codula, a social network tuned to a community of people willing to learn programming and the associate languages.

The focus of this paper was Codula – its design, educational-oriented features, interface, and evaluation. The foundations of (online) social networks were also summarized to pave the way over which we built our programming network. Based on the tests already conducted, it is possible to assert that Codula main contribution resides in the programming education mechanisms it offers, such as the *quoting functionality for code snippets* that provides an easy way to comment or query.

As future work we plan to conduct an extensive pedagogical assessment in diverse Programming Courses, in order to obtain a direct and reliable feedback from students. Their answers will provide important information to confirm our proposal and to improve Codula features. Another direction for future work is the development of other novel features dedicated to better support Programming Education, such as daily challenges, AI Tutors, and online serious games.

References

1 Ayat Al Ahmad and Randa Obeidallah. The impact of social networks on students' academic achievement in practical programming labs. *International journal of advanced computer science and applications*, 10(11), 2019.

2 Danah M Boyd and Nicole B Ellison. Social network sites: Definition, history, and scholarship. *Journal of computer-mediated Communication*, 13(1):210–230, 2007. doi:10.1111/J.1083-6101.2007.00393.X.

3 Kristijan Breznik and Kris Law. How social networks affect the learning performance among engineering students. *Technics Technologies Education Management*, 10(2):191–203, 2015.

4 Manuel Castells. The network society revisited. *American Behavioral Scientist*, 67(7):940–946, 2023.

5 Yu-Wei Chuang. Toward an understanding of uses and gratifications theory and the sense of virtual community on knowledge sharing in online game communities. *International Journal of Information and Education Technology*, 5(6):472, 2015.

6 Fernando de Oliveira Santini, Wagner Junior Ladeira, Diego Costa Pinto, Márcia Maurer Herter, Claudio Hoffmann Sampaio, and Barry J Babin. Customer engagement in social media: a framework and meta-analysis. *Journal of the Academy of Marketing Science*, 48(6):1211–1228, 2020.

7 Anabela Gomes and António José Mendes. Learning to program: Difficulties and solutions. In *International Conference on Engineering Education – ICEE 2007*, pages 283–287. Proceedings of the 2007 International Conference on Engineering and Education (ICEE), International Network on Engineering Education and Research, 2007. URL: http://icee2007.dei.uc.pt/proceedings/papers/411.pdf.

8 Andreas M Kaplan and Michael Haenlein. Users of the world, unite! the challenges and opportunities of social media. *Business horizons*, 53(1):59–68, 2010.

9 Nic Newman, Arguedas Ross Arguedas, Craig T Robertson, Rasmus Kleis Nielsen, and Richard Fletcher. *Digital news report 2024*. Reuters Institute for the study of Journalism, 2024.

10 Stephanie A. Robertson and Martin P. Lee. The application of second natural language acquisition pedagogy to the teaching of programming languages: a research agenda. *ACM SIGCSE Bulletin*, 27(4):9–12, December 1995. doi:10.1145/216511.

11 Thomas E Ruggiero. Uses and gratifications theory in the 21st century. *Mass communication & society*, 3(1):3–37, 2000.

12 Holly A Syrdal and Elten Briggs. Engagement with social media content: A qualitative exploration. *Journal of marketing theory and practice*, 26(1-2):4–22, 2018.

13 Elda Tartari, Alban Tartari, and Dilina Beshiri. The involvement of students in social network sites affects their learning. *International Journal of Emerging Technologies in Learning (Online)*, 14(13):33, 2019.

14 International Telecommunication Union. Global connectivity report 2025. Technical report, ITU Publications, 2025. URL: https://www.itu.int/itu-d/reports/statistics/global-connectivity-report-2025/.

15 George Veletsianos. A definition of emerging technologies for education. *Emerging technologies in distance education*, 56(3):3–22, 2010.

16 Lev Semenovich Vygotsky. *Thought and Language*. MIT Press, 2012.

17 Saijing Zheng, Kyungsik Han, Mary Beth Rosson, and John M Carroll. The role of social media in moocs: How to use social media to enhance student retention. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*, pages 419–428, 2016. doi:10.1145/2876034.2876047.

18 Abdallah Ziraba, Godwill Chenyuei Akwene, Shiynsa Charles Lwanga, et al. The adoption and use of moodle learning management system in higher institutions of learning: A systematic literature review. *American Journal of Online and Distance Learning*, 2(1):1–21, 2020.

Ogma: An Intelligent Platform for Anticipating Academic Failure and Recommending Recovery Measures

Ricardo Queirós
CRACS, INESC TEC & ESMAD, Polytechnic of Porto, Portugal

Abstract

The growth of class sizes, the increasing complexity of course content, and the multiplicity of digital platforms make it progressively harder for instructors to identify, in a timely manner, students who are at risk of academic failure. This short paper presents the concept of *Ogma*, a *learning analytics* application with artificial intelligence components designed to anticipate signs of academic decline and suggest personalized recovery measures. The proposal combines heterogeneous data – student-produced work, grade records, forum messages, and logs from other educational applications – to build a dynamic risk profile. Based on this profile, the system provides instructors with a dashboard containing graded alerts, explanations of risk factors, and pedagogical recommendations generated by a pipeline that combines *machine learning*, *retrieval-augmented generation*, and large language models. The paper presents the motivation for the problem, synthesizes related work, and describes the design of a reference architecture for Ogma.

2012 ACM Subject Classification Applied computing → Education; Applied computing → E-learning; Human-centered computing → Interactive systems and tools; Computing methodologies → Artificial intelligence; Human-centered computing → HCI design and evaluation methods; Human-centered computing → Collaborative and social computing systems and tools

Keywords and phrases Generative AI, Educational chatbot, Pedagogical agent, Gamification, Moodle, Retrieval-augmented generation, Self-regulated learning

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.4

Category Short Paper

Funding *Ricardo Queirós*: This work is funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the support UID/50014/2025 (`https://doi.org/10.54499/UID/50014/2025`).

1 Motivation

Early identification of at-risk students is a central problem in higher education. In demanding courses taught to large and heterogeneous classes, instructors rarely have enough granularity to closely monitor the individual progression of each student. Even when abundant data exist, such as learning management system accesses, forum participation, intermediate grades, and assignment deadlines, these signals are typically fragmented across multiple screens and systems and are not presented in an actionable way. The learning analytics literature shows precisely that collecting, measuring, and analyzing student data can support the understanding and optimization of learning and of the contexts in which it occurs [5, 2].

In real settings, the problem is not merely to predict a low final grade. The pedagogical goal is to detect patterns of disengagement before failure materializes: fewer Moodle accesses, repeated delays in assignment submission, reduced participation in forums, signs of frustration in messages, sudden drops in formative assessment performance, or overload signals observable in other educational platforms. Early warning systems have shown usefulness in supporting

more timely interventions [1, 3]. However, many solutions still depend on a limited number of variables, remain insufficiently transparent to instructors, or are weakly connected to concrete recovery recommendations.

This is where *Ogma* emerges. Its central idea is to build a platform capable of aggregating multimodal and multi-application evidence to produce two complementary outputs. First, an updated risk score throughout the semester. Second, intervention suggestions contextualized to the course, the student, and the teaching moment. Instead of showing instructors only a predictive value, the system should answer operational questions: *who* is falling behind, *why*, *with what level of urgency*, and *which recovery measure* makes sense to propose now.

This approach is particularly relevant in complex domains such as programming, mathematics, statistics, or engineering, where small early gaps tend to propagate and produce cumulative failure. In such scenarios, delays in detection may make recovery unfeasible. A tool such as Ogma can bridge the gap between the abundance of digital data and the need for fast, sustained, and explainable pedagogical intervention.

2 Related Work

The field of *learning analytics* has become established over the last decade as an intersection of data science, educational technology, and pedagogical decision support. Foundational texts stress that the usefulness of data analysis in education does not depend solely on predictive performance, but also on its translation into concrete actions in instructional design and teaching intervention [5, 2]. This concern is especially visible in the research line on *early warning systems*, whose purpose is to flag at-risk students early enough for institutions and instructors to act.

Among the most widely cited early warning systems is *Course Signals*, developed at Purdue University, which combined academic and engagement indicators to classify students into risk levels and communicate status through a simple color scheme [1]. In a similar vein, the Open Academic Analytics Initiative showed that predictive models can be operationalized at the institutional level to support scalable alerts and interventions [3]. More recent review studies confirm the maturity of the field and show that academic performance prediction increasingly relies on *ensemble learning*, decision trees, *gradient boosting*, and deep models, although challenges of generalization, explainability, and transfer across contexts remain [7, 6].

A second line of work relevant to Ogma concerns the multimodal nature of the data. In many courses, risk signals are not found only in Moodle clicks, but also in late submissions, forum messages, frequency of access to materials, communication patterns, and the textual quality of student productions. Recent reviews show that combining interaction data with textual and socio-emotional data can improve the sensitivity of models to states of disengagement and frustration [9]. This observation justifies including sentiment analysis over forum messages, team channels, or other authorized educational applications in the Ogma proposal.

A third line concerns the presentation of results. Learning analytics dashboards for instructors can improve situational awareness about the class, but they are effective only when they make alerts understandable, actionable, and aligned with real pedagogical decisions [10, 8]. Alerts without explanation tend to generate mistrust; visualizations without prioritization generate overload. For this reason, the Ogma interface should combine visual simplicity with explanations of the main risk factors.

Finally, the emergence of large language models creates an opportunity to move from prediction to recommendation. The *retrieval-augmented generation* (RAG) architecture makes it possible to ground text generation in specific document bases and reduce the

probability of generic or decontextualized answers [4]. In the context of Ogma, this means that instructors can upload pedagogical materials, course regulations, and previously defined recovery strategies, creating a knowledge base that supports recommendations better suited to the local context.

3 Design and Implementation

3.1 Architecture overview

The proposed architecture was designed to support the early detection of academic failure through the integration of Learning Analytics, Machine Learning, Explainable Artificial Intelligence (XAI), and Generative AI technologies. The system aims not only to identify students at risk but also to support teachers through contextualized recommendations and intelligent decision-support mechanisms.

The architecture follows a modular and layered approach composed of six main components:

1. Data collection
2. Data processing and feature engineering
3. Predictive analytics
4. Explainability
5. Retrieval-augmented generation
6. Visualization

3.1.1 Data collection

The first layer is responsible for data acquisition from multiple educational platforms. Moodle acts as the primary source of learning analytics data, including login frequency, access patterns, assessment results, assignment submissions, forum participation, and interaction with learning resources. Additional academic and administrative information, such as previous grades, completed credits, repeated courses, and student status, may also be integrated. To ensure interoperability between heterogeneous systems, the architecture incorporates MCP (Model Context Protocol), allowing the platform to communicate with external systems such as academic management software, Microsoft Teams, Discord, and other services.

3.1.2 Data processing and feature engineering

After collection, the data is processed and transformed into features suitable for predictive analysis. This stage includes data cleaning, normalization, aggregation, and feature engineering. Several behavioral and engagement indicators are generated, such as weekly activity consistency, delays in submissions, learning regularity, inactivity periods, and academic performance trends. These features are then stored in a centralized feature repository to support both training and inference processes.

3.1.3 Predictive analytics

The predictive layer is based primarily on supervised machine learning techniques. Since the objective is to predict academic risk using historical data, supervised learning represents the most appropriate approach. Historical datasets from previous academic years are used to train models capable of classifying students according to different risk levels. Logistic

Regression may be used as a baseline model due to its interpretability, while ensemble-based approaches such as Random Forest and XGBoost are expected to provide higher predictive performance for tabular educational data.

3.1.4 Explainability

To increase transparency and trustworthiness, the architecture integrates an Explainable AI layer. Techniques such as SHAP (SHapley Additive exPlanations) – a popular Explainable AI (XAI) framework that uses game theory to explain the output of machine learning models – are used to identify the contribution of each feature to the prediction outcome. This allows teachers to understand why a student was classified as being at risk, highlighting factors such as low engagement, missing assignments, or declining academic performance.

3.1.5 Retrieval-augmented generation

Beyond prediction, the architecture also incorporates a Retrieval-Augmented Generation (RAG) component combined with Large Language Models (LLMs). While the machine learning model is responsible for identifying risk, the LLM layer focuses on generating contextualized pedagogical recommendations. The RAG mechanism retrieves relevant information from institutional regulations, pedagogical guidelines, previous interventions, and scientific literature before sending the context to the language model. Based on this information, the system may generate personalized corrective measures, intervention suggestions, summaries, or teacher-oriented recommendations.

3.1.6 Visualization

Finally, the visualization layer provides an intelligent dashboard for teachers and academic coordinators. The dashboard presents risk indicators, predictive scores, behavioral trends, explainability outputs, and recommended interventions using visual alerts and color-based prioritization mechanisms. This approach supports early intervention and facilitates pedagogical decision-making without replacing the role of the teacher.

3.2 AI Workflow Orchestration

In addition to the architectural layers previously described, the proposed system incorporates an orchestration layer responsible for coordinating the interaction between predictive models, explainability services, retrieval mechanisms, generative AI components, and visualization modules. Since the architecture combines heterogeneous technologies and processing stages, an orchestration mechanism is required to manage the execution flow and ensure interoperability between components. Figure 1 summarizes this workflow.

To support this process, the system adopts LangFlow as the orchestration framework. LangFlow was selected due to its modular and visual workflow capabilities, integration with LangChain-based components, support for Retrieval-Augmented Generation pipelines, and suitability for rapid prototyping and research-oriented environments.

The orchestration layer acts as the runtime coordination engine of the platform. Rather than executing machine learning training processes directly, LangFlow is responsible for managing the inference and decision-support workflows triggered by new educational events or user interactions. Predictive models are trained externally using Python-based machine learning frameworks and deployed as independent services accessible through APIs. The orchestration layer then invokes these services whenever predictions are required.

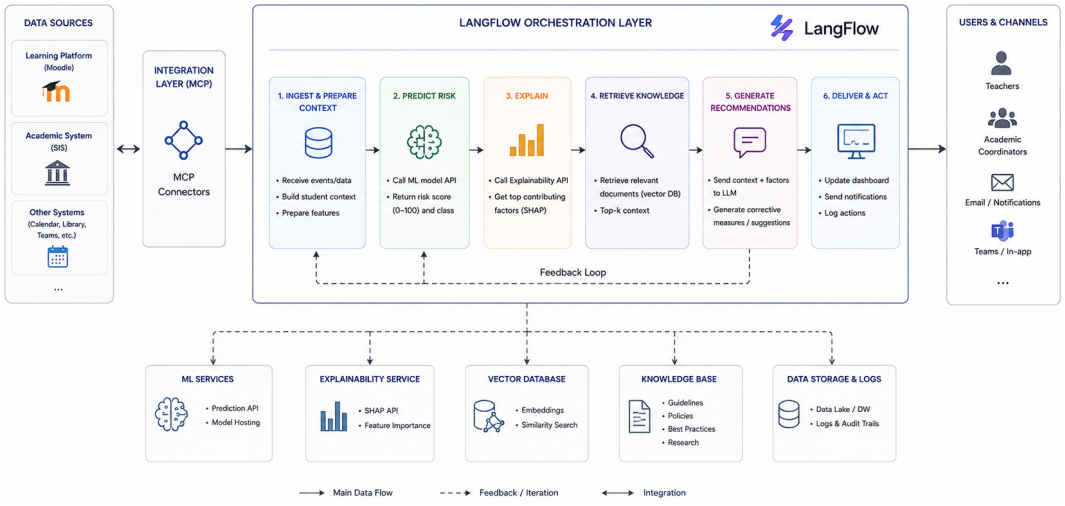


Figure 1 AI Workflow Orchestration of Ogma ecosystem (created by ChatGPT v. 5.3 using as prompt the content of this section).

The execution flow begins with the ingestion of new data from Moodle or other integrated systems through the MCP integration layer. After preprocessing and feature extraction, the student profile is sent to the predictive service, which returns a risk classification score. When the predicted risk exceeds a predefined threshold, the orchestration workflow automatically activates additional components, including explainability modules and retrieval pipelines.

The explainability service generates feature importance information using SHAP values, identifying the main factors contributing to the prediction outcome. Simultaneously, the RAG pipeline retrieves relevant contextual information from the knowledge base, including institutional policies, pedagogical guidelines, previous interventions, and scientific literature. This contextual information is then forwarded to the Large Language Model, which generates personalized recommendations, corrective measures, or intervention suggestions for the teacher.

The orchestration layer also supports event-driven workflows. Instead of operating exclusively through manual requests, the system may react automatically to behavioral changes such as prolonged inactivity, missing submissions, repeated low assessment scores, or sudden decreases in engagement. These events can trigger prediction pipelines, recommendation generation, and notification mechanisms in real time.

Finally, the generated outputs are delivered to the visualization layer, where teachers and academic coordinators can access dashboards containing risk indicators, explanatory insights, and recommended interventions. This orchestration approach enables the integration of predictive analytics and generative AI into a unified educational intelligence workflow capable of supporting proactive pedagogical intervention strategies.

4 Conclusion

In summary, Ogma proposes an evolution of early warning systems toward an intelligent pedagogical platform oriented to intervention. Its main conceptual contribution lies not only in predicting academic failure, but in transforming fragmented data into explainable signals and contextualized recovery measures, supporting instructors in managing complex classes and mitigating academic failure in a timely way.

This paper represents only the starting point of a broader work that will be developed over the next year, in which the goal is to implement and validate this architecture in real higher education contexts.

References

1 Kimberly E. Arnold and Matthew D. Pistilli. Course signals at purdue: Using learning analytics to increase student success. *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge*, pages 267–270, 2012. doi:10.1145/2330601.2330666.

2 Rebecca Ferguson. Learning analytics: drivers, developments and challenges. *International Journal of Technology Enhanced Learning*, 4(5–6):304–317, 2012. doi:10.1504/IJTEL.2012.051816.

3 S. M. Jayaprakash, E. W. Moody, E. J. M. Lauría, J. R. Regan, and J. D. Baron. Early alert of academically at-risk students: An open source analytics initiative. *Journal of Learning Analytics*, 1(1):6–47, 2014. doi:10.18608/jla.2014.11.3.

4 Patrick Lewis, Ethan Perez, Aleksandras Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33*, pages 9459–9474, 2020.

5 Phil Long and George Siemens. Penetrating the fog: Analytics in learning and education. *EDUCAUSE Review*, 46(5):31–40, 2011.

6 Jairo López-Zambrano, Juan A. Lara, and Cristóbal Romero. Towards portability of models for predicting students at risk of failing and dropping out in university. *Applied Sciences*, 11(23):11363, 2021. doi:10.3390/app112311363.

7 Abdullah Namoun and Abdullah Alshanqiti. Predicting student performance using data mining and learning analytics techniques: A systematic literature review. *Applied Sciences*, 11(1):237, 2021. doi:10.3390/app11010237.

8 Martin Paulsen and Kasper Hornbæk. Learning analytics dashboards for teachers: A systematic review of design, data, and actionability. *Computers and Education Open*, 6:100176, 2024. doi:10.1016/j.caeo.2024.100176.

9 A. Pilicita Garrido and E. Barra. Sentiment analysis with transformers applied to education: Systematic review. *International Journal of Interactive Multimedia and Artificial Intelligence*, 9(2):72–83, 2025. doi:10.9781/ijimai.2025.02.008.

10 Beat A. Schwendimann, María Jesús Rodríguez-Triana, Andrii Vozniuk, Luis P. Prieto, Mina S. Boroujeni, Adrian Holzer, Denis Gillet, and Pierre Dillenbourg. Perceiving learning at a glance: A systematic literature review of learning dashboard research. *IEEE Transactions on Learning Technologies*, 10(1):30–41, 2017. doi:10.1109/TLT.2016.2599522.

Introducing Programming Concepts Through Montessori Materials

Steffi Rudel
University of the Bundeswehr Munich, Germany

Marlene Engels
University of the Bundeswehr Munich, Germany

Abstract

Programming is a skill already taught in many schools today. However, for historical reasons, it has not yet been included in the Montessori method. To close this gap, we developed learning materials for Montessori schools and present them in this article.

We show the design of a new learning material for introducing programming concepts in Montessori elementary level (6–12 years). The material is a concrete physical learning material and is supported by a modular lesson structure, which also takes the principles of Computational Thinking in account. Finally we discuss its suitability for Montessori education based on expert feedback. The article shows interesting examples of how programming can be taught in Montessori elementary level.

2012 ACM Subject Classification Applied computing → Education

Keywords and phrases Montessori Method, Programming, Montessori Material, Computational Thinking

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.5

Funding This work originates in the LIONS research project. LIONS is funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr which we gratefully acknowledge. dtec.bw is funded by the European Union – NextGenerationEU.

Acknowledgements We want to thank the Montessori experts Christiane Salvenmoser and Saskia Haspel for evaluating the learning material as well as the Montessori teacher Christiane Daidrich for supporting the research.

1 Introduction

Computers and digital devices are ubiquitous today. Yet without the software that controls their operation, they are useless. Therefore, the ability to program – as part of computer science (CS) – is a vital skill that goes beyond mere knowledge of programming languages, libraries and frameworks. This ability could and should be taught from an early age to promote digital sovereignty of the individual.

Schools play an important role in teaching programming skills to children and young people. In addition to the standard state school system, reform pedagogical approaches, such as the teachings of Montessori [11] have been around for decades.

Maria Montessori's approach to learning in their time differed dramatically from the widespread authoritarian style. Instead, she believed that knowledge should not be "drilled" into children, but that motivation to learn is deeply rooted in every person (and therefore in every child). McNamara writes about this: "Maria Montessori believed that a goal of education should not be to fill children with facts, but rather to cultivate their own natural desires to learn." ([9] p. 95).

In the Montessori Method, the prevailing view is that learning is very much dependent on the *learning environment* and what is offered there. Therefore, learners acquire the knowledge themselves – the teacher is primarily responsible for preparing the environment and providing appropriate opportunities (so-called *Montessori material* the students work with independently).

However, the problem with CS is that computers had not yet been invented when Maria Montessori developed her approach and materials. Therefore, the ability to program or comprehend digital technology was not considered in the curriculum of Montessori Schools, nor are there dedicated Montessori materials for this topic. The proposed research aims to close this gap by creating suitable Montessori material.

Based on the teachings of Maria Montessori, the world-leading Association Montessori Internationale (AMI, `https://montessori-ami.org/`) divides the learning phases in Montessori education into the following stages, while our research focuses on elementary-aged children:

- Assistants to Infancy: Ages 0–3
- Primary: Ages 3–6 (Children's House / Casa dei Bambini)
- Elementary: Ages 6–12
- Adolescent: Ages 12–18

In summary, we address the following research question: **Which new Montessori material could support students aged 6–12 years in learning programming-logic?**

The contribution of the paper is as follows: First, we will provide an overview of the current state of the art (Section 2). Then we present a concrete physical learning material and a modular lesson structure (Section 3). After that, we discuss its suitability for Montessori education based on expert feedback (Section 4). The article concludes with a summary and an outlook (Section 5).

2 Related Work

In order to gain access to the subject area, a systematic literature search was carried out. We initially used the two databases EBSCO and ACM Digital Library one after the other and then conducted a snowball search.

The journal article by **Stefan Scippo and Fabio Ardolino** presents a long-term study in which children in grades 1–5 of primary school were followed in an Italian Montessori school [15]. Based on Montessori principles, technological materials were designed to promote computational thinking and encourage multiple solution paths, with built-in error control but flexible procedures. Over the years, students progressed from coloring binary patterns to working with the Ozobot for grammar tasks and storytelling, and later with LEGO WeDo for increasingly complex programming activities. In the final year, they created a website using WordPress. In summary, the article shows interesting examples of how programming can be taught at the Montessori primary level.

The study by **Mollie Elkin, Amanda Sullivan and Marina Bers** examines a robotics curriculum for Montessori classes (grades 1–3), aiming to integrate foundational programming and engineering concepts into early education [3]. Using a case study with a Montessori teacher and 19 students, the research combines qualitative and quantitative methods to analyze both classroom implementation and the teacher's development in robotics knowledge. Based on Montessori principles, a curriculum using LEGO WeDo was developed around the theme of Ancient Greece and structured into six learning units. The material is described

as self-corrective, supporting independent learning and engaging multiple senses, aligning closely with Montessori approaches, although a detailed comparison with Montessori material criteria is not provided.

Montessori teacher **Sheri Milton** uses a practical example to show how mathematical skills and programming can be taught in a Montessori school (children aged 9–12 years) [10]. A project with the Turtle Math software is described. The program works with the LOGO programming language and figures can be drawn using a turtle. The article shows that programming skills in a Montessori context should never be taught in isolation but always in context and in conjunction with other skills.

John McNamara explains how STEM (Science, Technology, Engineering and Mathematics) subjects can be naturally integrated into Montessori education, based on a conference presentation [9]. He emphasizes that learning arises from understanding relationships rather than isolated details. Technology is presented as a supportive tool, not the focus, helping students explore questions and solve problems more efficiently. McNamara also highlights the importance of a prepared environment that encourages inquiry and experimentation, as well as giving students enough time to develop their own ideas rather than rushing tasks.

Luisa Greifenstein and Adrian Hanusch provide in their poster abstract initial insights into a systematic literature search on the keywords Montessori and CS education [5]. In the article, various sources are examined for the connection between Montessori education and CS. Some of the sources in the poster abstract discuss the teaching of programming skills in the Montessori context and were therefore relevant: one of the sources mentioned was already found in the previous search [3], a second was included in the search using the snowball method [1].

The article by **Oren Zuckerman, Saeed Arida, and Mitchel Resnick** presents "Froebel-inspired Manipulatives" (FiMs) and "Montessori-inspired Manipulatives" (MiMs), focusing on physical and digital tools for learning [18]. While FiMs emphasize real-world object design, MiMs aim to model abstract structures, particularly dynamic systems. The authors introduce digital MiMs – enhanced physical objects based on Digital Manipulatives [6] – used to explore concepts like change and probability. A study with children aged 4–11 informs the development of design guidelines for these materials.

Reinhard Fischer deals with the use of computers, software, and the Internet (at that time not nearly as widespread as today) as learning and working materials in Montessori education [4]. These are summarized as "new media". Fischer does not deal with programming in the article, but questions 3 and 4 of the text are interesting: "3 Montessori education is based on free work and has a tried and tested set of learning materials for this purpose, which must fulfill certain criteria. How do the new media behave when they are evaluated on the basis of the criteria of the Montessori materials? 4. What criteria must be used to modify new media so that they can be used sensibly in the context of free work?" (p. 196–197)

The practical report by Montessori teacher **Markus Wurster** is dedicated to "Algorithms and Computer Science" and deals with the concept of algorithms and their teaching, Scratch, LOGO and microcontrollers, but no explicit reference is made to the Montessori Method [17].

Mario Valle is one of the few authors to address the possible integration of new media into Montessori education [16]. The book is explicitly aimed at primary school teachers who teach at a Montessori school and questions whether and in what way new technologies (such as computers and software, tablets and apps, 3D printers, the Internet, robotics, or virtual and augmented reality) can be used sensibly in the Montessori school. He suggests that technology itself (when teaching skills in the field of new technologies) should be completely avoided until the age of around 8, as children up to this age must first acquire the capacity

for abstraction. On pages 76–78 he lists 14 “indispensable characteristics” (based on the work of Montessori) that a material must have as a Montessori material. He then applies these to a LEGO Mindstorms robot as an example. Valle also devotes a separate section to learn programming, as well as robotics (p. 132–135). For programming, he suggests various tools such as Kodu, Scratch or other block-based programming languages. He describes robotics as an artifact that contains the steps of design, assembly and programming and is therefore very suitable for the Montessori Method. He mentions the tools Bee-Bot, LEGO Mindstorms and LEGO WeDo as well as the KIBO robot.

In a practice-oriented article, **Steffi Rudel** describes how programming skills can be taught to pupils in a (voluntary) afternoon program at a Montessori school in grades 3–8 [13]. Several tools and platforms are mentioned, and selected examples of their use are presented. In grades 3 and 4, for example, analog programming is recommended as an introduction; later, digital learning units from the code.org platform and the Scratch learning environment can be used. The Minecraft Education Edition learning environment and Calliope Mini are recommended for grades 5 and 6. Although the article refers to quality features that learning materials in a Montessori school must fulfill, there is no dedicated consideration of whether the tools presented fulfill these.

The authors **Felienne Aivaloglou and Efthimia Hermans** deal with Early Programming Education in their research [1]. They investigate the factors that influence the performance of elementary school students and the effects of Early Programming Education on later career orientation. For this purpose, an 8-hour Scratch unit was conducted in two different elementary schools in the Netherlands, one of them was a bilingual Montessori school. However, no differences between the schools were considered either in the course design or in the evaluation.

3 Creation of a new Montessori Material

3.1 Preliminary Considerations and Design Principles

Objective and Approach

In order to close the gap described in the introduction, a Montessori material for elementary grades at Montessori schools was created. The material is designed for young school children, so it can be assumed that these children are still at the early stages of developing their programming skills. The material is therefore designed to teach the underlying concepts rather than a specific programming language.

The Design Science Research (DSR) approach was used for the development [12]. The central element of the DSR is an artifact that represents the result of the research and development process. The artifact of this work is a Montessori material that playfully introduces students of the Montessori elementary level (grade 1–6) to basic concepts of programming-logic.

The initial concept for the learning material was developed based on the literature review and a desk research of existing commercial programming games. We tried to implement the design principles of a Montessori material (see below) from the very beginning, but it was already clear that these characteristics would need to be refined during the design cycles based on expert feedback.

Requirements for a Montessori Material

In order to understand why specific research for Montessori in the area of programming is necessary and why the teaching methods and materials of the standard school system cannot be transferred without closer examination, the following background is important:

Firstly, the Montessori Method puts the child at the center, not the teacher or what he or she wants to teach. The approach is always "child-centered", and the teacher does very little to "teach" the children, instead guiding them through observation and feedback. In Montessori education, children acquire knowledge and skills independently using the Montessori material, which was developed by Maria Montessori and is nowadays used in all Montessori schools worldwide. This *Montessori material* is used by the children in a so-called prepared environment, teachers using the Montessori Method are learning guides rather than instructors.

Design Principles of Montessori Material

There are some "essential characteristics" that Montessori learning materials must fulfill. They were first described by Maria Montessori herself ([11], p. 177–185), although strictly speaking, this description applies specifically to sensory materials (materials for the first development stage, 0–6 years). However, these characteristics are also reflected in the Montessori materials designed for older children and are frequently cited in the secondary literature (for example [7] or [14]).

- **Isolation of difficulty**: A material should always contain only one new difficulty so the child can focus their attention specifically on it.
- **Limitation / Order**: Materials are structured, complete, and present only once in the room to promote order, care, and social consideration.
- **Activity**: The material encourages action and enables independent learning.
- **Inviting nature / Aesthetics**: The material should be designed to be clear, orderly, beautiful, and child-friendly so it arouses interest and concentration.
- **Self-control**: The child should be able to recognize and correct mistakes on their own as much as possible without constant supervision by adults.

However, it is important to distinguish which developmental stage the material is used in. For example, in the first developmental stage (ages 0–6), self-regulation should be embedded within the material itself or within the process; in the second developmental stage (ages 6–12), self-regulation is primarily achieved through group discussion. Another example is the aesthetic of the material – in the first developmental period, the focus is indeed on "beautiful" material, whereas in the second developmental period, the tool-like nature of the material is much more central.

3.2 The Artifact

The Elements

The material consists of a playing field, a playing piece in the shape of a dragon, physical command blocks, task cards, and optional game accessories such as walls, stars, and mountains.

The material was deliberately designed as a physical playing field to make abstract programming concepts tangible through hands-on activity. The playing field provides clear orientation through coordinates while offering ample opportunities for tasks of varying

difficulty. The game piece was introduced so that movements and program execution can be visually tracked. The command blocks were designed to interlock, making the structure of a program visible and tangible. At the same time, this supports the step-by-step planning and assembly of sequences.

In the center is the three-dimensional playing field with 8×8 squares (Figure 1). Each square is uniquely named by a combination of letters (A–H) and numbers (1–8). In addition, walls can move around the playing field. These can be placed in the spaces between two adjacent squares and block the piece's path (Figure 1).

Figure 1 Game board with coordinates and plug-in walls as a modifiable obstacle system.

A game piece, in this case a dragon, is guided via coordinates from a starting position to a target position (Figure 2). The piece always starts on square A1 in the upper left corner of the playing field, facing square A2. There are also stars that can be collected and mountains that can be moved by the piece.

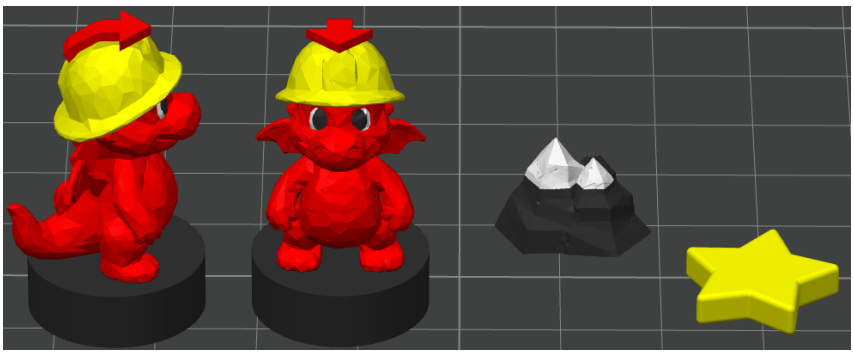


Figure 2 Game piece "dragon", stars, and mountains.

The walls, stars and mountains serve to make the tasks more varied and stimulate different thought processes. The dragon is controlled indirectly via a sequence of physical command blocks, which must first be assembled into a complete program (figure 3). The blocks are

color-coded and designed in a way to be connected to each other. Each program begins with a red start block (“Start”) and ends with a red stop block (“Ende”). The actual sequence of commands is located between these two blocks. The yellow blocks represent simple movement commands such as *walk* (“gehe”) or *turn left/right* (“drehe links”, “drehe rechts”), which must be executed immediately by the game character.

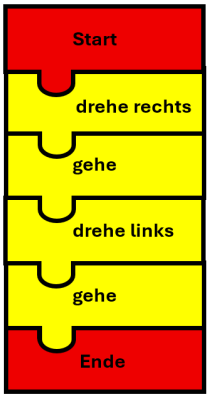


Figure 3 Example of a program sequence consisting of command blocks.

White building blocks, on the other hand, represent control structures such as if ...then ... conditions (Figure 4, “wenn...dann...”). This allows more complex programs to be formulated.

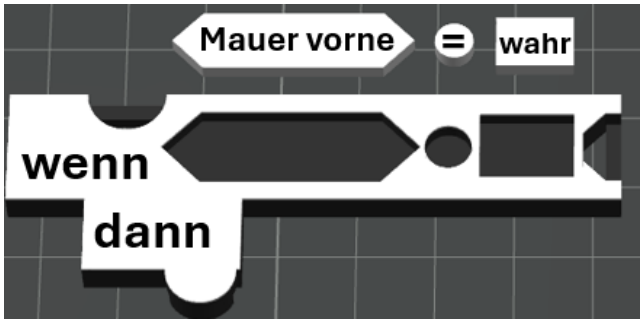


Figure 4 Building blocks for the if ... then ... conditions.

Simple conditions such as *wall in front* (“Mauer vorne”) are inserted into the hexagon. To formulate more complex conditions, the white building block can be extended with a plus sign or an O sign to add another simple condition.

The Design

The material has various properties to make it easy to handle. The prototype was produced using a 3D printer. The colors were deliberately designed to be distinct – the basic programming blocks are yellow with black (go, turn left, turn right), whereas the programming blocks for the conditions are white. This color coding allows learners to easily navigate the programming. The programming blocks are stored thematically in separate boxes, which are secured with a magnetic closure and reflect the colors of the blocks. The game board is labeled throughout with numbers and letters to clearly identify each square. This is important for the tasks, so it is clear which square the piece starts on and where it should move to. Both the game board and the game piece are fitted with magnets, ensuring that the piece stands securely on the individual squares and cannot simply slip out of place.

The optional walls for the game board come in two different colors and are each stored in their own box, which has been designed in the corresponding color and size. To make it easier to position the walls correctly, two templates are provided, which are also designed in the corresponding colors. Each template has a tab so that it can easily be lifted off the playing area without removing the walls. Furthermore, the abbreviation A1 is marked in the top left corner of both templates, so the students know which way round the template must be placed on the playing area.

The dragon figure wears a helmet featuring an arrow. This makes it easy to see which direction the figure is facing – this is important for programming, as "go" always means in the direction the figure is currently looking.

Modular Structure of the Lessons

The Montessori material is divided into five modules that build on each other. Each module introduces a central idea of programming and deepens it through appropriate tasks. The order of the modules creates a didactically meaningful learning path that supports the transition from concrete action to abstract thinking. The framework developed by Brennan and Resnick was taken into account in the design and structure of the modules [2]. The authors argue that Computational Thinking (CT) consists not only of programming concepts, but of three equal dimensions:

The first dimension is formed by **CT Concepts**, which include basic programming concepts such as sequences, conditions, events, operators, data (variables, lists), and loops. These elements form the conceptual vocabulary that learners must understand and apply when programming.

The second dimension is formed by **CT Thinking Practices**. This dimension describes ways of working and thinking when dealing with the program and possible problems that may arise. These include, in particular, iteration (design – testing –improvement), debugging, and decomposition. The focus is not on *what* is learned, but on *how* learners approach problems.

The last dimension is called **CT Perspectives**. This dimension refers to attitudes and viewpoints that learners develop when dealing with technology. These include the expressive perspective (coding as a means of expression), the collaborative perspective (learning through exchange, feedback, and collaboration), and the reflexive perspective (recognizing patterns, structures, and systems in one's own environment).

To support self-directed learning, task cards have been added to the learning materials. The task cards explain the respective task that needs to be solved next. Each card first contains the task with possible additional conditions, followed by a visual representation of the playing field. On this, the figure is shown on the starting square A1 looking to the right. The goal is marked by a black pin symbol; walls, stars, and mountains are drawn in according to the task. On the back of the task card, there are one or more possible solution sketches that can be used for self-checking. However, it is not always possible to show all theoretically correct solutions. The solution is therefore only an aid and not an absolute correction aid. These task cards are organized into six modules. Each of the modules takes up aspects of the three dimensions and combines them into a coherent learning process.

Module 1: Basic movement commands (imperative programming): This module introduces students to the basics of imperative programming. The focus is on the principle of sequencing: commands are executed in a fixed order with the game character.

Module 2: Interaction with objects (state changes): In the second module, the command sequences already familiar from Module 1 are expanded to include the concept of state changes.

Module 3: Truth values: In the third module, the focus is on understanding truth values. Students learn that statements in programming logic can be either true or false.

Module 4: If–then conditions (decision logic): In the fourth module if–then structures are introduced. Children learn that programs do not only run linearly, but can branch depending on conditions.

Module 5: Testing and debugging: The fifth module is entirely devoted to testing and debugging. The focus here is no longer on learning additional commands, but on consciously reflecting on errors and systematically correcting them.

Sequence of a Learning Unit

A typical unit begins with reading the task card. The students build the walls on the field. They can either build the walls freehand, use the template to build them, or check if their construction is correct using the template. The dragon is then positioned on the starting square A1, facing A2. Then, either alone or in teams, they consider how the task could be solved. The students basically have two approaches at their disposal.

1) The program (solution to the task) is planned in a purely abstract manner. The command blocks are put together according to this plan. Only then is the command chain traced by actively moving the dragon.
2) The program is developed with the help of the dragon and the game board, and the command blocks are strung together in the process. If the execution of the program does not lead to the desired result, the faulty piece of code is specifically searched for, the command sequence is adjusted and tested again. A red arrow is included with the game to help with the testing process. This arrow may only be moved down one command block at a time. This prevents commands from being executed twice or omitted during the test phase (Figure 5).

The cyclical approach (plan–test–correct–test–...) is typical in the field of software engineering and is taught to children in a playful manner.

3.3 The Design Cycles

As part of the DSR, several improvement cycles have already been completed during the development of the artifact. Through an ongoing comparison with existing educational games in the field of programming, as well as discussions with experts, the following adjustments were made:

Lowering the Difficulty Level and Reducing the Number of Modules

The game was originally planned with the following modular structure:

- Module 1: Simple commands
- Module 2: Conditions
- Module 3: Loops

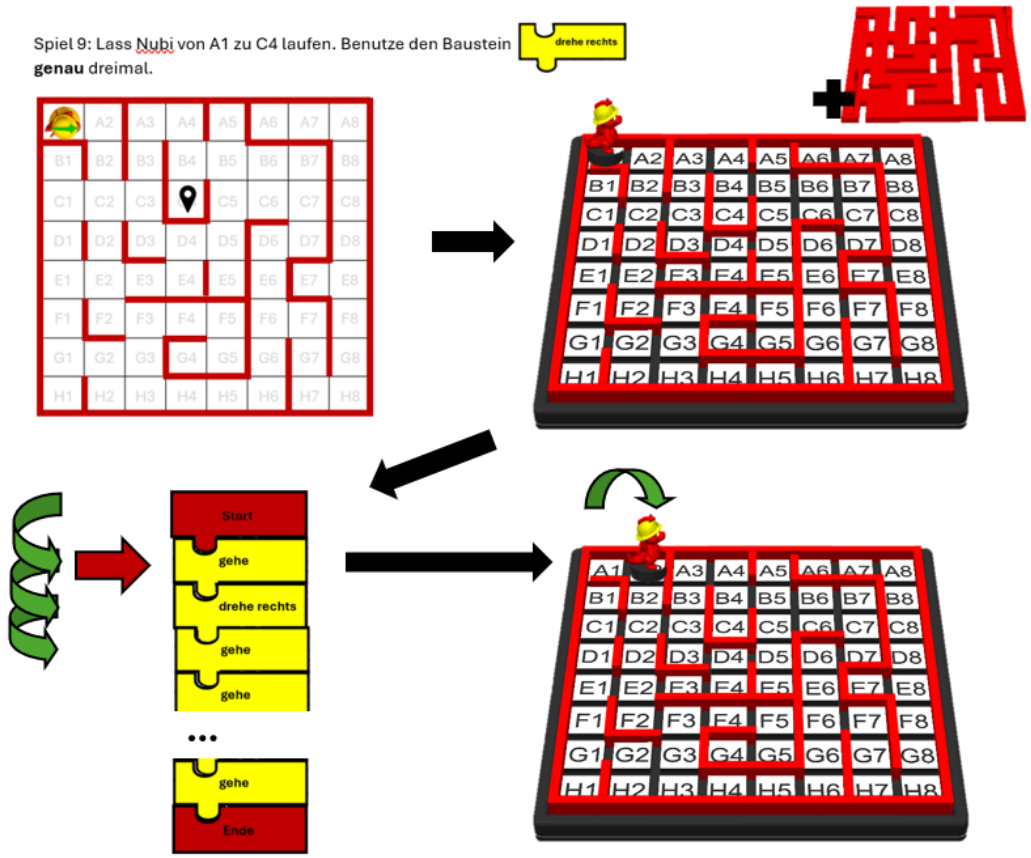


Figure 5 Example sequence of a learning unit.

- Module 4: Attributes and values
- Module 5: Developing custom functions
- Module 6: Basic concept of iteration
- Module 7: Debugging and testing

Analysis of this structure revealed that the transition between the modules would have been too rapid, thereby significantly increasing the cognitive load on the children. Young school children are in the concrete operational stage [8], in which abstract and nested structures can only be processed to a limited extent. In particular, loops, functions, and debugging represent abstract meta-levels of programming, the understanding of which requires a solid foundation in simpler concepts. Proceeding too quickly would have overwhelmed the children and hindered their learning success.

Simplification of Language

Another point of criticism concerned the labeling of the command blocks. These were heavily based on traditional programming standards and included elements such as parentheses and semicolons. While such syntactic elements are necessary for professional programming, they are not required for understanding basic algorithmic concepts. For children, however, they present an additional extrinsic difficulty that distracts from the actual learning objective: recognizing logical relationships and sequences. Therefore, these syntax elements in the command blocks were simplified: For example, **gehen();** became **gehe** ("gehen" means go).

Optimizing the Walls

Originally, the walls consisted of one-, two-, and three-piece combinations, which made it difficult to place them using the reference images. In addition, small gaps on the game board caused the walls to shift slightly. This problem was solved by reducing the number of wall combinations that appear in all games to two variants (red and white). Sturdy wall pieces were developed using a 3D printer to match these two combinations. As an additional aid, stencils were included with the game. These can either be used to check the walls that have already been placed or laid out on the playing field at the start so the walls can be placed directly using the stencil. The templates are color-coordinated with the respective wall combination.

4 Reflection and Suitability for Montessori Education

The material was developed to introduce Montessori students of the elementary phase to programming concepts. Testing with children is currently pending, as approval has not yet been granted by the research university's data protection and ethics committee. Therefore, adult experts were consulted to assess the suitability of the material for Montessori education. The Montessori experts are Christiane Salvenmoser and Saskia Haspel. They are the founders and directors of the Montessori Academy in Austria and have been working for decades as Montessori instructors, consultants, coaches, and evaluators. They served on the board of the Austrian Montessori Society for more than 25 years and train Montessori educators through certification courses. The workshop was supported by the german Montessori teacher Christiane Daidrich.

The materials were evaluated during a workshop held on April 30th, 2026 in Vienna. The material was first demonstrated and explained, after which the experts tested it on their own. This approach mirrors the way Montessori materials are used in a classroom: The teacher provides an introduction, and then the children work independently with the material. The experts' comments were recorded in writing and evaluated for this article.

The material was evaluated both from a general, educational perspective and specifically in relation to the Montessori method. Several aspects were cited as **strengths in a general sense**: The materials are designed to be tactile and easy to handle. The color-coded separation of the programming blocks into individual storage boxes supports a modular learning experience. In the experts' view, the material clearly prepares children for programming at various difficulty levels, and the cards allow children to work at their own pace. As **biggest weakness** was mentioned that navigating the game piece's paths requires children to have good spatial orientation. A solution should be found for this, as spatial orientation is not a prerequisite for effective programming.

However, in order to actually use the learning material as **Montessori material**, they suggested a few more aspects.

In Montessori education for school-age children (starting in Elementary), learning a subject area can be divided into **three learning phases**. In the first phase, the *introductory phase*, the material is shown to the learners and the rules and usage are explained. The learners then familiarize themselves with the material. The second phase, *exploration*, involves investigating the material and thereby acquiring knowledge independently. In the third phase, *integration*, the newly acquired knowledge is finally linked to existing knowledge and, if appropriate, presented in a suitable manner (for example, as a presentation, a poster, or a product). A key principle of Montessori education is that the duration and intensity of each phase are determined solely by the child, not by the teacher. Some children may grasp

a concept after just a few exercises, while others may take much longer and repeat things over and over. What matters most is that there is sufficient freedom in the exercises and that no rigid guidelines hinder the child's learning. However, the current design of the task cards only partially allows self-directed learning and exploration of topics. Since all lessons within a module build upon one another, individual task cards cannot simply be skipped if a child has grasped the topic more quickly. On the other hand, repeating a task card for reinforcement is rather unappealing, as this stifles the child's spirit of inquiry and does not encourage joyful learning.

Therefore, it was suggested to replace the task cards by **activity cards**. These could be organized into subcategories according to the modules, allowing the students to decide how many of the cards they would like to work on for a given topic. **Self-checking**, which is a key pillar of Montessori materials, should not be resolved by printing the solution on the back. The question is: What do we want to achieve with the learning child? Is it about the solution being exactly correct? Or isn't it more about the process of evaluation and debugging? The Montessori experts therefore suggested that program verification should instead be carried out through teamwork (one child programs, the other verifies). A fixed division of roles would be helpful here, so each child is aware of its task. Debugging, if something is wrong, can then take place together through dialog.

Here is an example to illustrate: Child 1 draws an activity card that says: *Place the game piece on square A1; he is facing A2. Now create a program that makes the dragon move to square C4.* Child 1 reads the card silently and creates the program using the programming blocks. Child 2 (who does not know the destination) then executes the exact program using the game piece. Now, the two children look together to see which square the game piece ends up on and compare it with the activity card. If the square is not the same, they work together to figure out how the program needs to be changed.

Also, the modules could be effectively expanded by allowing students to create and solve their **own tasks** through teamwork. It would also be quite conceivable to add **challenges**, such as:

- Can you solve the task using a different program?
- Can you solve the task using fewer programming blocks?
- Can you solve the task using the longest possible program?
- How many ways can you find to reach the goal?

In addition, it was suggested to create so-called **Definition materials** for each module. In Montessori education, these materials are provided to learners for specific subject areas and encourage them to look up information on their own. This reduces the workload for the Montessori teacher and supports the learners' independence and self-efficacy ("Help me to do it myself"). The definition material consists, among other things, of a small ring binder in 14x14 cm format. On a double-page spread, the left side features a relevant illustration, while the right side contains the explanation along with the technical terms.

It was also mentioned that the material needs a catchy **name** (all Montessori materials have a name that is easy for children to remember and that clearly identifies the material). Last but not least the teachers should be provided with a clear instruction for the **presentation** of the material.

5 Conclusion and Outlook

As the literature review shows, research to date has only dealt rudimentarily with the topic of programming in Montessori schools. Montessori materials have also hardly been systematically created yet. To close this gap, a new learning material specially for the

Montessori method was developed. The DSR was applied, and the structure of the lessons was chosen to be modular. Finally, since approval for a study involving students had not yet been granted, the material was evaluated by adult Montessori educators.

The Montessori experts highlighted many positive aspects of the learning materials and explicitly emphasized the relevance of the topic for Montessori pedagogy. In their view, incorporating this topic into Montessori pedagogy is highly desirable. From the experts' perspective, Montessori materials represent the right approach, but they must absolutely be embedded within other methods of Montessori pedagogy. Examples include the Definition materials or Cosmic Education, which places all of the learners' experiences into context.

As the research progresses, we will first conduct a more in-depth evaluation of the learning material. To this end, we will carry out empirical tests with students at Montessori schools and incorporate the findings into subsequent design cycles for the material. In doing so, we will also take into account other Montessori components, such as *Cosmic Education* and the *Definition materials*.

Montessori pedagogy represents a promising opportunity for learners to prepare for the digital future. By expanding the materials to include the subfield of programming, Montessori pedagogy is enriched with contemporary aspects, thereby silencing critics who argue that Montessori is no longer up to date. In the further course of the research, the goal is not only to improve the programming materials through additional design cycles. It is also intended to examine whether other CS aspects, such as Artificial Intelligence, should be integrated into the curriculum.

However, the materials developed are not only valuable for Montessori education. Even today, elements of the Montessori Method are increasingly being adopted in the standard state school system. One example is independent and self-regulated learning: Modern societies increasingly demand skills such as problem-solving, initiative, and self-direction. These are fostered by the self-directed learning that is central to Montessori ("Help me to do it myself"). The learning material is also suitable for traditional schools to help children experience and learn the basic principles of programming in a self-directed way. The learning materials can therefore support teachers shifting their role from "instructor" to "learning guide". Especially in an increasingly fast-paced, digital world, the goal is no longer simply imparting knowledge to learners. Rather, it is to equip them with the skills to acquire relevant knowledge independently.

References

1 Efthimia Aivaloglou and Felienne Hermans. Early Programming Education and Career Orientation: The Effects of Gender, Self-Efficacy, Motivation and Stereotypes. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 679–685, Minneapolis MN USA, 2019. ACM. doi:10.1145/3287324.3287358.

2 Karen Brennan and Resnick Resnick, Mitchel. New frameworks for studying and assessing the development of computational thinking. In *Annual American Educational Research Association meeting AERA*, 2012.

3 Mollie Elkin, Amanda Sullivan, and Marina Bers. Implementing a robotics curriculum in an early childhood Montessori classroom. *Journal of Information Technology Education: Innovations in Practice*, 13:153–169, 2014.

4 Reinhard Fischer. Neue Medien und Montessori-Pädagogik. In Harald Ludwig, Christian Fischer, Reinhard Fischer, and Montessori-Vereinigung, editors, *Verstehendes Lernen in der Montessori-Pädagogik*, volume 8 of *Impulse der Reformpädagogik*, pages 195–222. LIT, Münster, 2003. Hrsg. Buch: Ludwig, Harald; Fischer, Christian; Fischer, Reinhard.

5 Luisa Greifenstein and Adrian Hanusch. Combining Montessori Pedagogy and Computing Education: First Insights from a Systematic Literature Review. In *Proceedings of the 19th WiPSCE Conference on Primary and Secondary Computing Education Research*, page o.P., Munich Germany, 2024. ACM. doi:10.1145/3677619.3678102.

6 J. Patten, James Patten, H. Ishii, Jim Hines, Hiroshi Ishii, Gian Pangarp, J. Hines, and G. Pangarp. Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces. In *Proceedings of SIGCHI 2001*, pages 253–260, Seattle Washington USA, 2001. Association for Computing Machinery ACM. doi:10.1145/365024.365112.

7 Michael Klein-Landeck and Tanja Pütz. *Montessori-Pädagogik: Einführung in Theorie und Praxis*. Montessori Praxis. Herder, Freiburg, 4. auflage edition, 2019.

8 Saul McLeod. Piaget's Theory and Stages of Cognitive Development, April 2025. doi:10.5281/ZENODO.15241970.

9 John McNamara. How the Montessori Upper Elementary and Adolescent Environment Naturally Integrates Science, Mathematics, Technology, and the Environment. *NAMTA Journal*, 2(41):82–97, 2016.

10 Sheri Milton. Star Power - Connecting Montessori and Technology. *Montessori Life*, 11(4):39, 1999.

11 Maria Montessori. *The Discovery of the Child*. Kalakshetra, Adyar, Madras, India, 1948.

12 Ken Peffers, Tuure Tuunanen, Marcus Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24:45–77, January 2007.

13 Steffi Rudel. Programmieren lernen an einer Montessori-Schule: Der EmiLe Computer Club (ECC). In Daniel Demmler, Hannes Krupka, and Hannes Federrath, editors, *Lecture Notes in Informatics (LNI) - Proceedings*, volume P-326, pages 1213–1219, Bonn, 2022. Gesellschaft für Informatik. doi:10.18420/INF2022_104.

14 Schumacher, Eva. *Montessori-Pädagogik verstehen, anwenden und erleben*. Individuelles Lernen mit Montessori. Beltz Verlag, Weinheim, Basel, 2016.

15 Stefano Scippo and Fabio Ardolino. Computational thinking in Montessori primary school. *Ricerche di Pedagogia e Didattica. Journal of Theories and Research in Education*, 16(2):59–76, 2021. Epistemological intersections between human and natural sciences in the thought and works of Maria Montessori [Special Issue]. doi:10.6092/ISSN.1970-2221/12163.

16 Mario Valle. *Montessori-Pädagogik und neue Technologien: Eine mögliche Integration?*, volume 33 of *Impulse der Reformpädagogik*. LIT, Berlin, 2019. Title of the original Italian edition 2017: La pedagogia Montessori e le nueve tecnologie. Un'integrazione possible?

17 Markus Wurster. Internet und digitale Strukturen im Grundschulalter. *Montessori. Zeitschrift für Montessori-Pädagogik*, 2, 2019.

18 Oren Zuckerman, Saeed Arida, and Mitchel Resnick. Extending tangible interfaces for education: digital montessori-inspired manipulatives. In Wendy Kellog, Shumin Zhai, Gerrit van der Veer, and Carolyn Gale, editors, *CHI05: CHI 2005 Conference on Human Factors in Computing Systems*, pages 859–868, Portland Oregon USA, 2005. ACM. doi:10.1145/1054972.1055093.

Algorithm X: Competitive Programming as a Teaching Methodology for Algorithms and Data Structures in High School

Odair Moreira de Souza
Federal Institute of Education, Science and Technology of Paraná (IFPR), Cascavel, Brazil
Graduate Program in Informatics (PPGInf), Federal University of Paraná (UFPR), Curitiba, Brazil

Clodis Boscarioli
Graduate Program in Computer Science (PPGComp), Western Paraná State University (Unioeste), Cascavel, Brazil

Letícia Mara Peres
Graduate Program in Informatics (PPGInf), Federal University of Paraná (UFPR), Curitiba, Brazil

Abstract

Contextualization: Teaching algorithms and data structures in technical high schools presents challenges related to abstraction, problem solving, and the persistence of non-student-centered teaching models. In this context, Competitive Programming and Competition-Based Learning emerge as promising pedagogical alternatives. **Objectives:** This study aims to analyze the educational impacts of the teaching project "Algorithm X" on students' learning of algorithms and data structures, as well as their motivation, autonomy, academic engagement, and perceptions regarding Competitive Programming in Technical High School. **Materials and methods:** The study is based on the experience of a teaching project developed with students from the Technical Course in Informatics Integrated with High School at Cascavel, Paraná, Brazil. The project was organized through weekly meetings structured around dialogic expository classes, problem solving, coding dojo, debates, simulated contests, and digital support resources. The data were obtained through an evaluative questionnaire by 37 students, and discursive responses were analyzed. **Results and discussions:** The results indicated a favorable evaluation of the project and recognition of its contribution to strengthening algorithmic thinking, developing problem-solving skills, motivating the study of programming, improving performance in programming-related subjects, and fostering learning autonomy. The project was understood as a space for learning and development that extended beyond the immediate context of competitions. **Considerations:** Competitive programming may constitute a viable pedagogical strategy for teaching algorithms in technical high schools when supported by structured didactic mediation, progressive practice, and digital resources. This approach consolidates technical skills and competencies and contributes to students' academic and professional development. Student participation in the project was not limited to the context of competitions but also involved a complementary formative process.

2012 ACM Subject Classification Applied computing → Education; Applied computing → Computer-assisted instruction; Applied computing → Collaborative learning; Social and professional topics → Computer science education; Social and professional topics → K-12 education

Keywords and phrases Competition-Based Learning, Algorithmic Thinking, Competitive Programming, Olympiad in Informatics, Algorithm Education in High School

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.6

Acknowledgements We thank the research participants, the Federal Institute of Education, Science and Technology of Paraná (IFPR), and the Coordination for the Improvement of Higher Education Personnel (CAPES), Brazil, for their support of this research.

1 Introduction

The teaching of algorithms and data structures is a fundamental component of Computer Science education, recognised as a central topic in international reference curricula and as essential at all levels of Computer Science and Software Engineering [18, 15]. However, it is considered one of the most challenging subjects for novice students from both cognitive and educational perspectives, as the difficulty lies primarily in high-level concepts rather than low-level technical details [18, 22].

Empirical studies support this perception by showing that most first-year students struggle with abstraction and algorithmic thinking [2]. Survey data from first-year students indicate that 86% fall below the level considered satisfactory in abstraction skills, while 52.6% demonstrate insufficient performance in algorithmic thinking [2]. As abstraction is considered a key element in the early stages of learning, insufficient development of this competence directly hinders the transition from theory to code implementation [18].

In the Brazilian context, this challenge is accentuated by the predominance of traditional teaching practices in middle and high school education. According to Bandeira et al. [1], traditional teaching is understood here as a teacher-centered approach based on theoretical classes, content exposition, exercises on specific topics, written examinations, and final projects, with limited emphasis on self-directed learning and iterative problem solving. This background may make self-learning less common among students and create difficulties when they enter Computing courses, where they are often expected to learn more independently [1]. In this context, Competitive Programming can be used as an active methodology to revisit concepts that were insufficiently consolidated in previous educational experiences and to encourage students to seek solutions more autonomously [1].

As a pedagogical strategy, Competitive Programming extends beyond the concept of a tournament and can be understood as an instructional model in which contest problems structure the teaching and learning of algorithms. This approach fosters the development of observational skills, enabling students to reduce unfamiliar problems to known patterns and to make progress in implementing solutions through efficient coding and debugging [24, 15]. Furthermore, the use of automatic judging systems provides instant feedback, facilitates real-time error correction, and encourages independent learning, innovative thinking, and the development of logical reasoning and problem-solving skills [24, 1].

At the theoretical level, this proposal can be understood in light of Competition-Based Learning, defined as a student-centered strategy that uses tournaments as a stimulus to maximize learning outcomes [4]. From this perspective, competitive practice drives active learning, where knowledge is constructed through interaction, testing competencies, and confronting peers [11]. This process is not limited to the acquisition of technical skills, but also supports the development of socio-emotional competencies such as resilience, adaptability, confidence, and reasoning, associated with preparation for complex real-world challenges [6, 16].

Although the literature identifies Competitive Programming and Competition-Based Learning as promising strategies, there is still a lack of applied studies that systematically examine these models in real contexts of Technical Secondary Education. Most research focuses on higher education or short-term experiences, such as isolated marathons or specific educational projects. Therefore, it is necessary to investigate how the continuous integration of these practices into the technical curriculum can influence the learning of algorithms and students' perceptions of their academic and professional gains. Thus, this study is justified by the need to understand to what extent the use of competition as a teaching strategy can contribute to addressing the limitations of non-student-centered models and support formative trajectories in the field of Computing.

This article analyzes the educational impacts of students' participation in the teaching project "Algorithm X: Teaching Algorithms for Competitive Programming in Technical High School", developed with students from the Technical Course in Informatics Integrated with High School Education at Federal Institute of Education, Science and Technology of Paraná (IFPR), Cascavel, Paraná, Brazil. The objective is to examine, with emphasis on the learning of algorithms and data structures, students' perceptions of the project's contributions to the development of autonomy, motivation for study, and academic engagement, as well as to discuss how this experience relates to fostering interest in Computing and the construction of formative and professional trajectories.

The article is organized as follows. Section 2 discusses Competitive Programming as a teaching strategy in relation to Competition-Based Learning, Algorithmic Thinking, and Problem Solving. Section 3 presents the project analyzed and the research methodology. Section 4 presents the analysis of the project's results and students' perceptions. Finally, Section 5 presents the final considerations and perspectives for continuing the investigation.

2 Background

Competitive Programming has been explored as a non-traditional approach based on the Algorithmic Problem Solving method, serving as an active methodology in introductory programming courses. From this perspective, it contributes to increasing students' proactivity, familiarity with programming logic, and ability to abstract real-world problems into algorithms [1].

Competitive Programming can be understood from two complementary perspectives: (i) as an active methodology integrated into first-year formal curriculum classes, using timed contest-based activities and the practice of upsolving, where students solve unfinished problems outside the classroom to progressively develop algorithmic problem-solving skills; and (ii) as a motivational strategy inspired by the "mind sport" nature of contests, encouraging students to seek solutions independently through positive feedback, thereby increasing their enthusiasm, familiarity with programming, and technical confidence [1]. Case studies in introductory courses show that this practice stimulates interest in learning, fosters autonomous learning, and strengthens innovative thinking and problem-solving skills [24]. However, the fact that only 27% of novice competitors are able to solve at least one problem reveals a gap in the perception of difficulty between teachers and students, reinforcing the need for more appropriate questions organised in a progression of complexity [24].

In teaching algorithms, Competitive Programming uses contest problems as a structuring element to introduce, practice, and reinforce algorithmic techniques such as searching, sorting, basic data structures, and dynamic programming at introductory levels. These problems align with Computing curricular guidelines and have been associated with improvements in algorithmic thinking, problem solving, motivation, and, in some contexts, retention in courses in the field.

Luo (2025) organizes learning around three central outcomes: (i) the capacity for observation to reduce unknown problems to known problems; (ii) mastery of algorithmic techniques; and (iii) implementation through efficient coding and debugging [15]. To achieve these outcomes, the author uses a competition-based teaching model in which classes alternate between introducing new concepts, collective problem discussions, and formative assessments that simulate the time pressure of competitions. This format incorporates the time factor, bringing the experience closer to contexts such as contests, technical interviews, software sprints, and hackathons, while at the same time favouring the understanding of problem

statements, team organisation, strategic decision-making, and code debugging under pressure [15]. In addition, automatic judging systems provide instant feedback, enhancing the cycle of discovery and continuous improvement through multiple attempts after incorrect submissions [24]. Furthermore, cooperative learning strategies and team competitions encourage the strategic division of tasks and the exchange of technical knowledge among students, bringing the pedagogical dynamic closer to that observed in programming contests and real-world professional challenges.

A central benefit of this approach is its emphasis on knowledge transfer, as students must recognize the type of problem, relate it to known patterns, and adapt the solution. This process reinforces structural understanding rather than memorization. In this sense, adapting contests for beginners, with a focus on interpreting algorithms rather than full implementation, has proved to be an alternative for broadening the participation of less experienced students, while also improving logical reasoning, supporting the elaboration of test cases, and fostering the construction of pseudocode [17].

2.1 Competition-Based Learning

Competition-Based Learning is a methodology that uses structured competitions to stimulate motivation and improve student performance. As a student-centered instructional strategy, it aims to capture students' interest and encourage curiosity, allowing greater control over their own learning process. Additionally, the learning outcome is independent of the score or ranking in the competition, so the tournament serves as a pedagogical incentive without academic success depending strictly on competitive performance [4].

This methodological approach involves the development of competence, autonomy, and relatedness through challenges that make achievements visible and reinforce students' intrinsic motivation for accomplishment [11]. Previous research highlights that competition can foster a state of flow by establishing clear goals and providing immediate feedback, thereby intensifying engagement when the level of challenge remains compatible with participants' skills [5]. In addition, symbolic rewards such as badges and rankings can reinforce students' perception of progress and contribute to learning [5, 14, 21]. From a constructivist perspective, this dynamic supports active learning through interaction with problems and peers [5, 11].

Pedagogical planning should include guided challenges that extend beyond students' current capabilities [16], combined with real project-based activities that stimulate critical thinking, problem solving, scientific reasoning, collaboration, and communication [20]. Its success also depends on sustained participation across annual cycles or multiple semesters, allowing students to progressively consolidate scientific knowledge and skills [20]. When integrated into the formal curriculum and accompanied by mentors or specialists, this methodology can promote active learning, reflection, and reduced performance disparities [6, 16]. In the context of teaching and learning algorithms through Competitive Programming, this framework is realized in contests, mini-contests, and assessments with defined deadlines, often mediated by automatic judges, which encourage intensive practice and the application of algorithmic techniques under time constraints [15, 24]. Thus, Competition-Based Learning and Competitive Programming converge by creating authentic problem-solving environments, where competition structures complex tasks and supports progress monitoring through evidence of performance [6, 16].

In this arrangement, Algorithmic Thinking is directly encouraged, as competition-style problems require students to decompose statements, abstract relevant information, define systematic actions, and evaluate solution adequacy and efficiency [8, 12, 13]. These skills extend beyond coding, involving understanding why an algorithm works, generalizing strategies,

and recognizing recurring patterns, which supports structural understanding rather than memorization [10]. Thus, competitions and challenges can support the consolidation of algorithmic thinking as a central student competence, especially when aligned with curricular objectives and organized with progressive difficulty [15].

Problem solving is also a central component of this convergence, as competitive environments induce iterative cycles of formulation, implementation, testing, and debugging, often supported by automatic feedback. This process supports learning through trial and error and the continuous refinement of solutions [19, 24]. In particular, Cuba-Ricardo et al. [7] show that, when solving algorithmic programming problems, students tend to think and solve the problem while coding: they reinterpret the written code, compare it with the algorithmic solution conceived mentally, gradually correct errors, and refine partial solutions. This emphasizes the importance of pedagogical strategies that guide reasoning, planning, and quality control of solutions. In this context, explicitly elaborating implicit algorithmic notions can contribute to greater discipline in problem solving and foster students' computational thinking competence [9].

Even so, the effectiveness of competition-based learning depends on balancing the level of challenge with participants' competencies, as highly heterogeneous contexts can intensify anxiety and demotivation, especially among less experienced participants [5, 21]. In such cases, collaborative and cooperative learning dynamics, where teams compete externally and collaborate internally to achieve common objectives, can support the articulation of reasoning, negotiation of strategies, and collective correction [23]. Similarly, non-competitive contests have emerged as a relevant pedagogical alternative, as they retain formative elements of the contest format, such as problems, deadlines, and feedback, but moderate the adverse effects associated with rigid rankings and excessive pressure, shifting the focus to learning, cooperation, and reflection [17]. This adaptation tends to make the strategy more inclusive in Secondary Education, while maintaining the motivational and practical benefits of Competitive Programming without compromising educational objectives centred on algorithmic thinking and problem solving [3].

3 The Algorithm X teaching project in focus

The teaching project entitled "Algorithm X: Teaching Algorithms for Competitive Programming in Technical High School" is a complementary educational initiative to regular teaching, aimed at students in the Technical Course in Informatics Integrated with High School at IFPR. This project was conducted from March to November 2025, comprising 80 class hours through weekly meetings in a computer laboratory. It focused on developing competencies in logic, algorithms, data structures, and computational problem solving, articulated with preparation for the Brazilian Olympiad in Informatics (OBI[1]) and other competitions. In this context, students' participation in the OBI allowed the training developed in the meetings to be connected with concrete competition experiences, bringing the teaching process closer to real situations involving the application of knowledge constructed in the project.

The methodological proposal was structured based on Competitive Programming as an educational strategy, integrated with principles of active learning, collaborative learning, and problem-based learning. The course consisted of a sequence of didactic practices that combined moments of conceptual introduction, guided problem solving, collective discussion of solutions, and training in competition format. This approach was implemented through dialogic expository classes, problem solving, coding dojo, debates, simulated contests, and the use of progressive sets of exercises.

[1] OBI – `https://olimpiada.ic.unicamp.br`

The methodological teaching approaches were organized to address both conceptual development and intensive practice. Dialogic expository classes introduced the theoretical foundations necessary for competitive programming, especially topics in logic, discrete mathematics, algorithms, and data structures. These sessions aimed to present content interactively, with explanations, examples, simulations, and opportunities for student participation. Problem solving was the central methodology of the project, as the OBI and other competitions require analysis of problem statements, elaboration of strategies, and implementation of correct and efficient solutions. Problems were worked on individually and, at some moments, in pairs. This organization was determined by the teacher's formative assessment during meetings, considering the complexity of the topic, students' recurring questions, performance in previous exercises and submissions, and the autonomy observed during problem-solving activities. The class's level of mastery was not formally quantified but was inferred from classroom observation and students' difficulties during practice.

As a complementary collaborative learning strategy, coding dojo was used to promote collective problem-solving and discussion of algorithmic reasoning in the classroom. In this dynamic, participants alternated roles such as pilot and co-pilot, while others observed, suggested alternatives, and followed the process of constructing the solution. Debates on problems and solutions were incorporated into meetings at two distinct stages: before implementation, to discuss possible algorithmic approaches, and after resolution, to compare methods, data structures, and implementation decisions. This arrangement allowed exploration of the same task from different perspectives, encouraging reflection on the efficiency, clarity, and adequacy of the solutions.

Simulated contests formed another important dimension of the methodology, aiming to familiarize students with the real conditions of competitions. These activities were designed to reproduce the format of the OBI and similar events, involving problems of varying difficulty, limited time, and the need to define a resolution strategy. The goal was to develop not only technical mastery but also skills related to time management, interpretation of problem statements, and emotional control in testing situations. In parallel, sets of problems selected according to the content studied and the participants' level of development were used, with the main sources being the OBI website and the Beecrowd platform[2].

The problems used in the project were mainly implementation-oriented tasks, in which students had to read a statement, identify the underlying algorithmic idea, design a solution strategy, implement it, and test the submitted code using automatic judges. Some meetings also included complementary activities focused on tracing algorithms, discussing strategies in plain language, elaborating test cases, comparing alternative implementations, and analysing wrong submissions. Problem selection followed the OBI Programming syllabus and the topics planned for the project, such as logic, arithmetic reasoning, sorting, searching, basic data structures, graphs, trees, and dynamic programming. The task types and learning objectives are presented in the (Table 1). The problems were selected by the project teachers according to their alignment with the learning objectives, the difficulty level indicated by the source platforms, and the students' observed progress during the meetings. Thus, validation was pedagogical and curricular, based on the correspondence between the problems, the contents taught, and the skills expected in competitive programming activities.

Regarding the description of teaching practices, the project aimed to articulate the progression of content and the diversity of formative experiences. The planned content included foundations of mathematics, logic, and computing, followed by topics in algorithms

[2] Beecrowd – `https://www.beecrowd.com.br`

Table 1 Examples of problem types used in the project.

Topic	Type of task	Learning objective
Logic and arithmetic reasoning	Code-writing and test-case discussion	Interpret statements, identify input-output patterns, and construct basic algorithmic solutions.
Sorting and searching	Implementation, tracing, and comparison of strategies	Understand ordering, search strategies, and the relationship between algorithm choice and efficiency.
Basic data structures	Implementation and plain-language explanation	Apply stacks, queues, lists, or dictionaries to represent and manipulate problem data.
Graphs and trees	Modelling and implementation	Represent relationships between entities and solve connectivity, traversal, or hierarchy problems.
Dynamic programming	Strategy discussion and guided implementation	Identify subproblems, recurrence relations, and reuse of intermediate results.

and data structures, such as algorithm analysis, Big O notation, algorithmic strategies, sorting, searching, dynamic programming, graphs, trees, and geometry. The selection of content also was based on the syllabus of the Programming modality of the OBI, with the possibility of including additional topics according to the class's progress and the availability of other competitions during the project.

Digital resources supporting teaching were used as an integral part of the methodology. The Beecrowd platform was employed in training sessions and simulated contests, as it provides a large collection of problems categorized by difficulty and offers automatic real-time feedback on submissions. The OBI website was used to access previous exams, challenges, and official materials, and also served as a submission channel during the competition phases. For visualizing the functioning of algorithms and data structures, Data Visualization[3] and VisuAlgo[4] were used as support tools for understanding processes such as sorting, searching, graph manipulation, trees, lists, stacks, and queues. As complementary study materials, W3Schools[5] was used for reviewing language foundations and syntax, and GeeksforGeeks[6] for deepening advanced concepts and typical competitive programming strategies.

Table 2 summarizes the main elements that structured the project methodology, organizing it into three complementary axes: methodological approaches, teaching practices, and digital support resources. It presents, in an integrated way, how the pedagogical proposal was operationalized throughout the meetings, linking the references of active, collaborative, and problem-based learning with the didactic strategies used and the digital tools adopted for practice, visualization, and autonomous study.

4 Results and Discussion

An evaluative questionnaire was developed, and the 38 students who participated in the project were invited to respond voluntarily, with anonymity guaranteed. Of these, 37 students completed the questionnaire and composed the study sample. All respondents were students in the Technical Course in Informatics Integrated with High School at IFPR – Campus Cascavel. The mean age was 16.41 years (standard deviation = 0.96). Most participants

[3] Data Visualization – `https://www.cs.usfca.edu/~galles/visualization/`
[4] VisuAlgo – `https://visualgo.net/en`
[5] W3Schools – `https://www.w3schools.com`
[6] GeeksforGeeks – `https://www.geeksforgeeks.org/competitive-programming-a-complete-guide/`

Table 2 Pedagogical purposes of the strategies.

Axis	Main elements	Pedagogical purpose
Methodological approaches	Active, collaborative, problem-based learning; competitive programming as a teaching strategy	Promote active participation, problem solving, and the development of algorithmic thinking
Teaching practices	Dialogic expository classes; problem solving; coding dojo; debates; simulated contests; problem sets	Articulate conceptual introduction, guided practice, collaboration, and training in a context similar to competitions
Digital resources	Beecrowd; OBI website; Data Visualisation; VisuAlgo; W3Schools; GeeksforGeeks	Support problem-based practice, algorithm visualisation, immediate feedback, and complementary study

were male (75.7%), while 24.3% identified as female. Regarding distribution by school year, 27.03% were in the first year, 40.54% in the second year, and 32.43% in the third year. The third-year students had already participated in previous editions of the project, which should be considered when interpreting their responses. These data indicate that the sample included students at different stages of the technical high school programme, with a higher proportion of 2nd-year students. In 2026, 91,67% of the project alumni are pursuing undergraduate studies in Computing at public institutions.

Exploratory subgroup analyses were conducted to examine whether students' responses varied according to school year or previous participation in the project. Kruskal-Wallis tests did not indicate statistically significant differences among first-, second-, and third-year students in the main questionnaire dimensions ($p > 0.05$ for all dimensions). Similarly, Mann-Whitney tests comparing students with and without previous participation did not reveal significant differences in the main Likert-scale dimensions. A significant difference was observed only for the perceived contribution of the OBI to learning algorithms, measured through a single item on a 0–10 scale, with higher scores among students who had previously participated in the project ($p = 0.0231$). However, given the small size of the subgroups and the different scale of this item, these results should be interpreted as exploratory evidence.

Regarding the time dedicated to extracurricular study related to the project, 40.5% of participants studied exclusively during meetings, while 37.8% reported dedicating 1 to 2 additional hours per week, 18.9% dedicated 3 to 4 hours, and 2.7% dedicated 5 to 8 hours. This overview indicates that, although a significant portion of students maintain autonomous study practices, for a substantial part of the sample, the project remains the main learning space in competitive programming. Spearman's rank-order correlation indicated a moderate, positive, and statistically significant association between extracurricular study time and students' self-assessment in competitions ($\rho = 0.480$, p = 0.0026). A weaker positive association was also found with students' perception of the OBI's contribution to learning algorithms ($\rho = 0.330$, p = 0.0462). However, the associations with overall perception of the project ($\rho = 0.223$, p = 0.1843) and learning based on competitions ($\rho = 0.225$, p = 0.1813) were not statistically significant. Therefore, these results suggest that extracurricular study time was more clearly related to students' perceived competitive performance than to broader perceptions of the project. Taken together, these results suggest that study beyond formal meetings may be relevant to students' perceived competitive performance and that the project may provide conditions for developing autonomous study habits in technical education contexts. As these exploratory analyses are based on self-reported variables, they should be interpreted as associations rather than causal relationships.

The responses about the most difficult content areas focused on topics with greater abstraction and algorithmic complexity, emphasizing graph algorithms (70.3%), tree algorithms (64.9%), graph concepts (43.2%), and dynamic programming (37.8%). In contrast, the content areas identified as easiest to assimilate were fundamentals of logic (75.7%), sorting algorithms (67.6%), and search algorithms (45.9%). This pattern aligns with the expected progression in algorithms training, where introductory content generally presents a lower barrier to comprehension than topics that require greater capacity for abstraction, modeling, and generalization. This pattern suggests that the project exposed students to advanced content but also highlights the need for specific teaching strategies to support its assimilation, such as organizing progressive learning paths, using guided exercises, and systematically reviewing more difficult concepts.

For study resources outside project time, websites and online platforms were most prominent, especially online judges (67.6%), help from teachers or peers (54.1%), and online forums or communities (35.1%). Regarding the materials considered most useful for learning, higher frequencies were observed for online tutorials (59.5%), video lessons (51.4%), forums and communities (45.9%), and online judges (37.8%). These data indicate that participants' learning relies on digital resources and interpersonal support mechanisms. This suggests that the formative process is not restricted to the project space but encourages motivation for self-learning, complemented by digital learning resources and collaborative interactions.

In relation to students' perception of the project, based on statements rated on a five-point Likert scale, the overall average was 4.13 (standard deviation = 0.65), indicating a generally favorable evaluation. Furthermore, the scale also demonstrated internal consistency (Cronbach's $\alpha = 0.925$), indicating strong coherence among the items in this dimension. The lower internal consistency observed in the students' self-assessment dimension may be related to the heterogeneous nature of this construct. This dimension includes items on understanding problem statements, solving problems, developing strategies, following competitive activities, and recognizing the need for further learning. The last item may reflect metacognitive awareness rather than low performance, which can reduce the homogeneity of the scale. Therefore, this dimension was interpreted cautiously as an exploratory indicator of students' self-perception in competitions.

Figure 1 shows the distribution of responses regarding participants' perceptions of the project, highlighting a predominance of favorable evaluations for most items. Notably, aspects related to understanding the format and rules of the OBI, the appropriateness of the freedom to choose the programming language, the relevance of the content covered, and the contribution of the training to the development of problem-solving skills and preparation for the Olympiad stand out. Consistent with this pattern, the items with the highest averages were: "The meetings helped me to better understand the format and rules of the OBI" (M = 4.43), "The freedom to choose the programming language was an appropriate approach" (M = 4.35), and "The skills learned in the project facilitated my performance in academic subjects related to programming" (M = 4.35), reinforcing the positive evaluation of the pedagogical dimension of the experience.

Relatively lower averages, though still within a positive range, showed greater dispersion in responses. Among these, aspects related to the organization and productivity of the meetings stood out, as did the project's impact on defining future academic and professional choices. This result suggests that the positive evaluation of the project is shared, although its effects on vocational and certain operational aspects were perceived less uniformly.

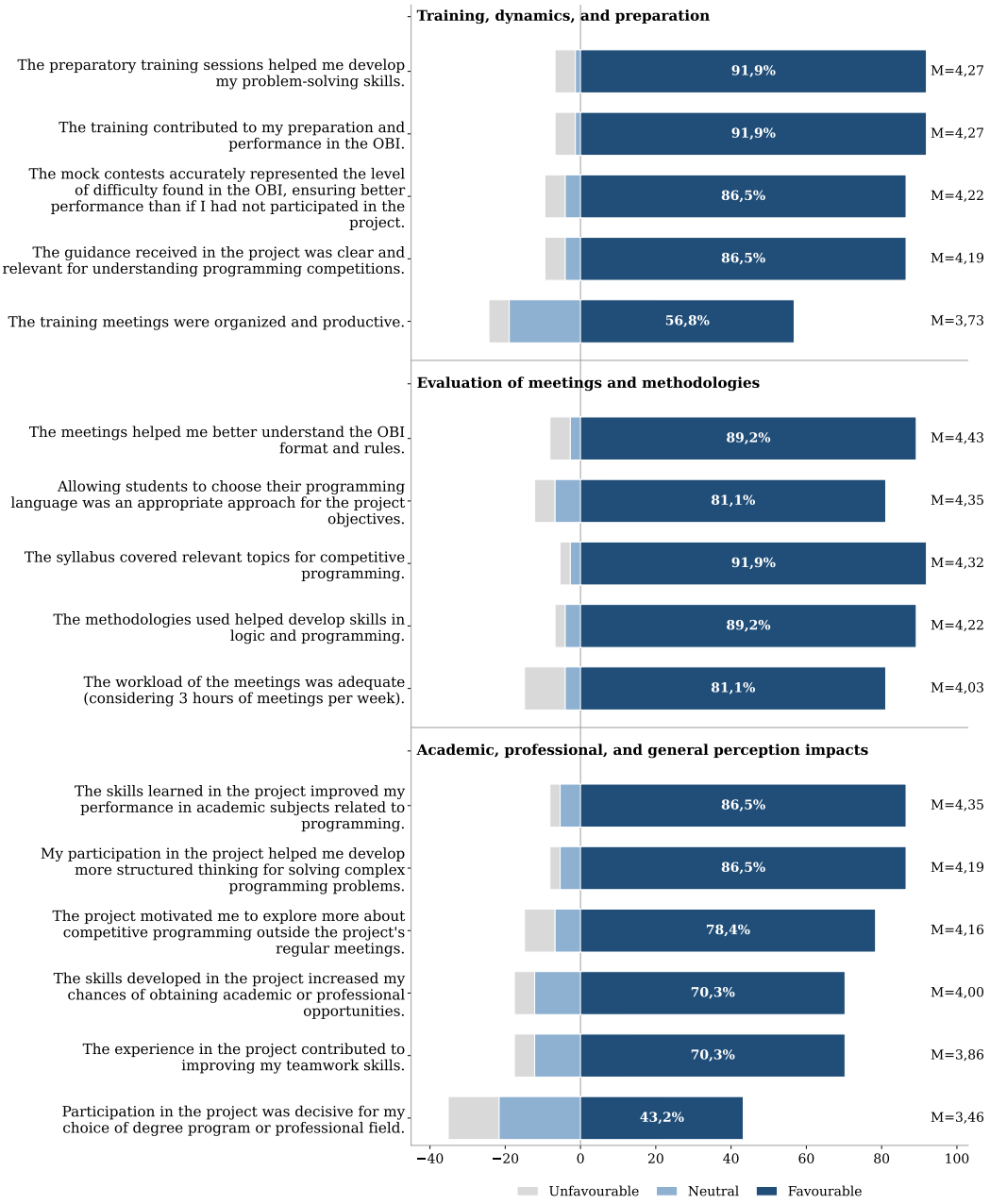


Figure 1 Students' perception of the project.

Students were encouraged to participate in competitions beyond the OBI, such as the SBC Programming Marathon[7] – Phase Zero and Phase 1 ("Coffee with Milk[8]), as well as programming marathons organized by universities in the state. It should be noted that these competitions are at the undergraduate level; therefore, only a portion of the participants took part in the challenges. Before the competitions, students were familiarized with the exams through simulated contests and by solving problems from previous marathons.

[7] SBC Programming Marathon – `https://maratona.sbc.org.br`

[8] The "Coffee with Milk" category designates teams that participate outside of competition, as they are composed of high school students, while ICPC is for undergraduate students.

Students' self-assessment in the competitions presented a global mean of 4.01 (standard deviation = 0.45), indicating an overall favorable perception of their own performance. However, this dimension showed lower internal consistency ($\alpha = 0.548$), suggesting greater variation among the evaluated items.

Figure 2 shows the distribution of responses regarding students' self-assessment in the competitions. The statement with the highest average was "I believe I still need to learn more about algorithms and programming to perform better in competitions" (M = 4.68), indicating strong recognition among participants of the need for continued study. High averages were also observed for items related to the ability to solve at least part of the proposed problems (M = 4.11), understand the statements (M = 3.95), and develop strategies to address the challenges (M = 3.95).

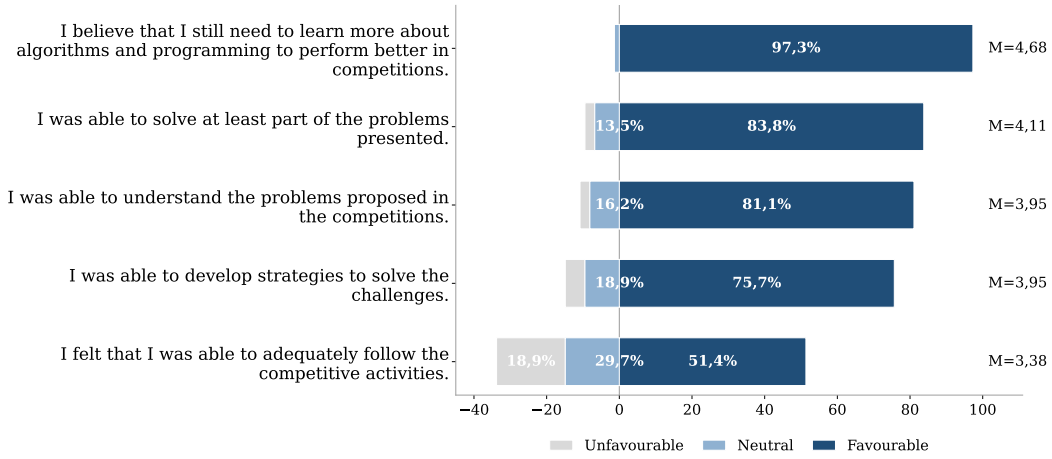


Figure 2 Student self-assessment in competitions.

In contrast, the statement with the lowest average was "I felt that I was able to adequately follow the competitive activities" (M = 3.38), indicating that some students still do not feel fully adapted to the pace and demands of the competitive environment. This should be interpreted in light of the progressive nature of training in competitive programming, especially in contexts where participants have varying levels of familiarity with this type of activity. The students' perception of Competition-Based Learning showed an overall average of 4.26 (standard deviation = 0.54), with good internal consistency ($\alpha = 0.885$). This result indicates that participants recognize the pedagogical potential of competitive programming as a teaching strategy.

Figure 3 shows a predominance of favorable responses regarding competition-based learning. The item with the highest average score was "stimulates algorithmic thinking and problem-solving" (M = 4.65), followed by "motivates the habit of studying programming" (M = 4.41) and "deepens the understanding of programming concepts and practices" (M = 4.35). Positive evaluations were also observed with reference to interest in programming subjects, study autonomy, and immediate feedback from competitive activities, indicating that students see in this approach a formative contribution that extends beyond the strictly technical dimension. In line with this result, the data suggest that competitive programming is perceived not only as a competitive activity but also as an active learning strategy. The predominance of the "learning and development" category in the descriptive analysis reinforces this interpretation, highlighting an experience associated with the articulation of challenge, engagement, and knowledge construction.

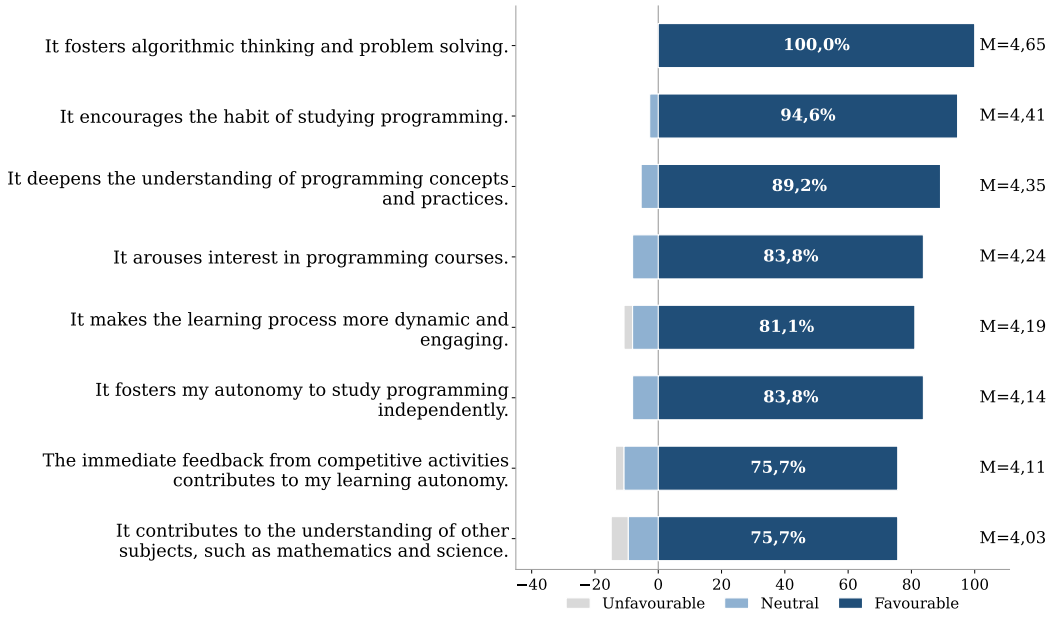


Figure 3 Students' perception of competition-based learning.

Furthermore, items related to study autonomy and the role of immediate feedback also showed a predominantly favorable distribution, although there was a higher relative presence of neutral responses. On the other hand, the item regarding the contribution to understanding other subjects, such as Mathematics and Science, while positively evaluated, had the lowest mean among the items in this dimension and a smaller proportion of strongly favorable responses. This descriptive pattern suggests that students more clearly recognize the perceived contributions of competition-based learning to skills directly related to programming than to broader interdisciplinary impacts.

The open-ended responses were analyzed using exploratory thematic categorization. The thematic axes were defined deductively based on the questionnaire sections: perceptions of the project, self-assessment in competitions, competition-based learning, difficulties encountered, and suggestions for improvement. For the question about the aspect considered most important in the project, the category of learning and development predominated (75.7%), followed by competition and challenge (24.3%), and, less frequently, socialization and support network (5.4%). These data indicate that the project was interpreted mainly as a space for training rather than solely as preparation for competitive performance.

In the case of the influence of the experience on academic and professional interests, the most prominent categories were increased interest in computing and technology (56.8%) and confirmation or definition of career choice (40.5%). References also appeared to the development of logic and mathematics skills (21.6%) and to professional impact and future opportunities (10.8%). These responses suggest that the experience strengthened participants' connection to the field of computing.

For suggestions for improvement, the most common themes were expanding simulations and practice opportunities (24.3%), deepening advanced content (18.9%), and increasing group activities (10.8%). To a lesser extent, participants mentioned the need for more hours or time dedicated to the project (5.4%), while 13.5% stated that they did not see a need for relevant changes. Overall, these results indicate that the suggestions focus less on

criticizing the pedagogical approach and more on the expectation of expanding and deepening experiences already positively evaluated, providing concrete contributions for the future improvement of the project.

In the overall description of the experience, the following categories stood out: positive and motivating experience (43.2%), improved performance and results (40.5%), development of logic and strategy (32.4%), initial challenge and difficulty (35.1%), and teamwork (18.9%). These findings show that the project was perceived as both a challenging and formative experience, capable of engaging students. Furthermore, the open-ended responses suggest contributions that go beyond immediate learning, indicating increased interest in computing and technology and the consolidation of formative choices, which demonstrates a contribution to strengthening students' academic identity in the field.

Although the main focus of the project is not competitive performance, students demonstrated significant participation in subsequent stages of the Brazilian Informatics Olympiad (OBI). In 2025, 98 students from IFPR - Campus Cascavel participated in the programming category of the OBI. In the Programming (P2) category for students up to the third year of high school, 17 project students ranked among the top 1,000 out of 6,764 participants across Brazil. In the Programming (P1) category for first-year high school students, 13 project students ranked among the top 1,000 out of 3,241 participants. Of the 38 project participants, 30 qualified for the state phase, representing 71.42% of the 42 IFPR students who advanced to this stage, and 11 reached the national phase among the 12 from IFPR - Cascavel (91.66%). The project also included the participation of 10 female students in the OBI Women's Competition. In the first-year high school category, three students achieved 18th place among 243 participants nationwide. In the category for students up to the third year of high school, five students achieved 36th place among 518 participants. Additionally, two project students won medals, and one project student was admitted to a public university through an admission program for high school scientific Olympiad medalists. These outcomes provide contextual evidence of students' competitive engagement and subsequent academic trajectories.

5 Reflections and Future Work

The results of this study indicate that the Algorithm X teaching project, designed to teach algorithms for competitive programming in Technical High School, was positively evaluated by students with regard to its pedagogical organization and its effects on learning, motivation, and the development of problem-solving skills. The convergence of quantitative and descriptive data reinforces this interpretation, suggesting consistent recognition of its educational contribution and potential as a pedagogical strategy for teaching algorithms.

From a practical perspective, the findings highlight the importance of expanding simulated contests, exercise lists, and commented problem solving. It is also relevant to structure progressive learning pathways that differentiate between introductory and advanced content, especially in more complex topics such as graphs, trees, and dynamic programming. Additionally, the results reinforce the importance of encouraging autonomous extracurricular study and investing in specific preparation strategies for the competitive environment, such as time management, interpreting problem statements, and familiarization with submission platforms. The value attributed to group activities, in turn, suggests incorporating collaborative practices that articulate individual challenge and shared learning.

Some limitations should be considered when interpreting the findings. Participation in the project and questionnaire was voluntary, which may have introduced self-selection bias, as students more interested in Programming Competitions may have been more likely to

participate. In addition, the study did not include a control or comparison group, and the data are based on self-report rather than objective performance measures. Since the sample is restricted to a specific institutional context, the results should not be generalized broadly. Therefore, the findings should be interpreted as exploratory associations between students' perceptions. Even so, the combination of descriptive statistics, exploratory Spearman correlation analysis, and qualitative evidence provides analytical consistency to the study.

The results also allow for a more precise definition of the conditions under which competitive programming may support more consistent educational contributions. The difficulties reported in content involving greater abstraction indicate that this approach requires an appropriate progression of content, ongoing teacher mediation, and a balance between challenge and support. Thus, the data do not support the idea that simply adopting competitions is sufficient to produce educational improvements; instead, they show that competitive programming can be pedagogically relevant when systematically integrated into the teaching process within a structured proposal that includes guided practices, moments of collective discussions, simulated contests, and study support resources.

From this perspective, students' responses suggest that participation in the project was associated with perceived gains extending beyond the context of the competition, including broader aspects of academic education, the development of algorithmic thinking, autonomy, improved performance in programming-related subjects, and increased interest in further study in the field. In addition, participants received at least introductory exposure to undergraduate-level Computer Science content.

For future development, research may advance in at least three directions: tracking students over longer periods to verify the persistence of perceived effects; incorporating more objective performance measures by combining perception data with results from simulated contests, practical activities, and competitions; and improving the pedagogical approach by expanding progressive learning pathways, strengthening practical activities, and investigating formats that more effectively balance collaboration, autonomy, and technical challenge. Thus, the study contributes to consolidating the understanding that Competitive Programming can constitute an educational strategy in Technical High School, provided it is planned with pedagogical intentionality and analyzed with attention to the specific context and the students' formative needs.

References

1 Ian Nery Bandeira, Thiago Veras Machado, Vitor F. Dullens, and Edna Dias Canedo. Competitive programming: A teaching methodology analysis applied to first-year programming classes. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, October 2019. ISSN: 2377-634X. doi:10.1109/FIE43999.2019.9028518.

2 Javier Bilbao, Eugenio Bravo, Olatz García, Carolina Rebollar, and Concepción Varela. Study to find out the perception that first year students in engineering have about the Computational Thinking skills, and to identify possible factors related to the ability of Abstraction. *Heliyon*, 7(2):e06135, February 2021. doi:10.1016/j.heliyon.2021.e06135.

3 Roberto Borchia, Antonella Carbonaro, Giorgio Casadei, Luca Forlizzi, Michael Lodi, and Simone Martini. Problem Solving Olympics: An Inclusive Education Model for Learning Informatics. In Sergei N. Pozdniakov and Valentina Dagienė, editors, *Informatics in Schools. Fundamentals of Computer Science and Software Engineering*, volume 11169, pages 319–335, Cham, 2018. Springer International Publishing. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-030-02750-6_25.

4 Juan C. Burguillo. Using game theory and Competition-based Learning to stimulate student motivation and performance. *Computers & Education*, 55(2):566–575, September 2010. doi:10.1016/j.compedu.2010.02.018.

5 Jordi Carenys. Competing to Learn: The Role of Competition in Students´ Flow, Cognitive Load, and Learning Gains in Game-Based Learning. *International Journal of Learning, Teaching and Educational Research*, 24(5):174–197, May 2025. doi:10.26803/ijlter.24.5.9.

6 Hui-Tzu Chang and Chia-Yu Lin. Applying Competition-Based Learning to Stimulate Students' Practical and Competitive AI Ability in a Machine Learning Curriculum. *IEEE Transactions on Education*, 67(2):256–265, April 2024. doi:10.1109/TE.2024.3350535.

7 Gilberto Cuba-Ricardo, María T. Serrano-Rodriguez, P. Alberto Leyva-Figueredo, and Laura L. Mendoza-Tauler. Methodology for Characterization of Cognitive Activities when Solving Programming Problems of an Algorithmic Nature. *Olympiads in Informatics*, 9:27–37, July 2015. doi:10.15388/ioi.2015.03.

8 Gerald Futschek. Algorithmic Thinking: The Key for Understanding Computer Science. In Roland T. Mittermeir, editor, *Informatics Education – The Bridge between Using and Understanding Computers*, volume 4226, pages 159–168, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11915355_15.

9 David Ginat. On Implicit Means of Algorithmic Problem Solving. *Olympiads in Informatics*, 13:31–39, 2019. doi:10.15388/ioi.2019.03.

10 Dalibor Gonda, Viliam Ďuriš, Anna Tirpáková, and Gabriela Pavlovičová. Teaching Algorithms to Develop the Algorithmic Thinking of Informatics Students. *Mathematics*, 10(20):3857, October 2022. doi:10.3390/math10203857.

11 Omer Sami Kaya and Erinc Ercag. The impact of applying challenge-based gamification program on students' learning outcomes: Academic achievement, motivation and flow. *Education and Information Technologies*, 28(8):10053–10078, August 2023. doi:10.1007/s10639-023-11585-z.

12 Timothy H. Lehmann. Using algorithmic thinking to design algorithms: The case of critical path analysis. *The Journal of Mathematical Behavior*, 71:101079, September 2023. doi:10.1016/j.jmathb.2023.101079.

13 Timothy H. Lehmann. How current perspectives on algorithmic thinking can be applied to students' engagement in algorithmatizing tasks. *Mathematics Education Research Journal*, 36(3):609–643, September 2024. doi:10.1007/s13394-023-00462-0.

14 Yu-Cheng Lin and Huei-Tse Hou. The evaluation of a scaffolding-based augmented reality educational board game with competition-oriented and collaboration-oriented mechanisms: differences analysis of learning effectiveness, motivation, flow, and anxiety. *Interactive Learning Environments*, 32(2):502–521, February 2024. doi:10.1080/10494820.2022.2091606.

15 Zhongtang Luo. Curriculum Design of Competitive Programming: A Contest-based Approach, 2025. Version Number: 1. doi:10.48550/arXiv.2504.00533.

16 Michael McGuire. Impact of Competition-Based Learning on Student Engagement and Performance. *International Journal of Construction Education and Research*, pages 1–30, May 2025. doi:10.1080/15578771.2025.2512348.

17 Caroline Mendes, Péricles Gomes, Alessandro Brawerman, and Eduardo Alberti. Programming competition approach based on the algorithm interpretation. In *INTED2020 Proceedings*, 14th International Technology, Education and Development Conference, pages 4190–4194. IATED, 2-4 march, 2020 2020. doi:10.21125/inted.2020.1172.

18 Ogen Odisho, Mark Aziz, and Nasser Giacaman. Teaching and learning data structure concepts via Visual Kinesthetic Pseudocode with the aid of a constructively aligned app. *Computer Applications in Engineering Education*, 24(6):926–933, November 2016. doi:10.1002/cae.21768.

19 Sarantos Psycharis and Maria Kallia. The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional Science*, 45(5):583–602, 2017. doi:10.1007/s11251-017-9421-5.

20 Yan Ruan, Junlei Zhang, Jiali Wang, Rui Jian, Feng Mei, Hongli Li, Yun Zhang, Qiwen Hu, Lan Xiao, Yi Yang, Ming Li, Jiaxiang Xiong, and Yanping Tian. Academic competition-based learning cultivates scientific literacy to promote professional competitiveness in medical undergraduates. *Frontiers in Public Health*, 13:1590832, July 2025. doi:10.3389/fpubh.2025.1590832.

21 Michael Sailer and Lisa Homner. The Gamification of Learning: A Meta-analysis. *Educational Psychology Review*, 32(1):77–112, March 2020. doi:10.1007/s10648-019-09498-w.

22 Heera Gurubasavraj Wali, Vijaya Eligar, Venkatesh Mane, and Nalini C Iyer. A Channelizing Approach to teach Data Structures. *Procedia Computer Science*, 172:581–584, 2020. doi:10.1016/j.procs.2020.05.071.

23 Ding-Chau Wang and Yong-Ming Huang. Exploring the influence of competition and collaboration on learning performance in digital game-based learning. *Educational technology research and development*, 71(4):1547–1565, August 2023. doi:10.1007/s11423-023-10247-8.

24 Kevin K. F. Yuen, Dennis Y. W. Liu, and Hong Va Leong. Competitive programming in computational thinking and problem solving education. *Computer Applications in Engineering Education*, 31(4):850–866, July 2023. doi:10.1002/cae.22610.

Gamification in Programming Education: A Zelda-Inspired RPG Approach for Reinforcing Foundational Programming Skills

Max Fritsche
DHSN Duale Hochschule Sachsen, Staatliche Studienakademie Leipzig, Germany
Security Robotics D&S GmbH, Leipzig, Germany

Alexander Paar
DHSN Duale Hochschule Sachsen, Staatliche Studienakademie Leipzig, Germany

Abstract

The acquisition of programming skills is a notoriously difficult process characterized by high cognitive load, complex abstraction requirements, and significant dropout rates in higher education. Traditional pedagogical approaches often struggle to bridge the gap between seemingly abstract language syntax and the strategic problem-solving required in software engineering. This paper presents a comprehensive gamified solution: a 2D top-down Role-Playing Game (RPG) inspired by classical adventure games such as *The Legend of Zelda* [30], specifically designed to reinforce existing knowledge. We detail the integration of a secure, Docker-based automated code judge within a narrative-driven environment. Key features include a virtual economy, a social "company" system for collaboration, and a curriculum focused on the C programming language to deepen low-level understanding. The paper evaluates the technical robustness of the prototype and proposes an empirical framework to validate the hypothesis that this immersive approach significantly enhances learning productivity, persistence, and student motivation.

2012 ACM Subject Classification Social and professional topics → Computer science education; Applied computing → Interactive learning environments

Keywords and phrases Gamification, C Programming, Role-Playing Games, Automated Grading, Docker

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.7

Related Version *Full Version*: `https://github.com/Velt1/whitepaper-gamification/blob/main/Bachelorarbeit_Max_Fritsche.pdf`

1 Introduction

Programming education is facing a global crisis of engagement. As software becomes the backbone of modern infrastructure, the demand for developers grows, yet introductory computer science (CS1) courses report failure rates as high as 30-50% [34, 27]. The primary challenge lies not just in learning syntax, but in developing the strategic knowledge required to decompose complex problems and debug logic under pressure. Current educational models often rely on lecture-based instruction followed by isolated, context-free exercises. These methods lack the immediate, iterative feedback loops that modern learners require to stay motivated. This paper argues that Gamification, the application of game design elements to non-game contexts, can provide a solution by fostering a "Flow" state where the challenge of coding is matched by an immersive narrative and immediate rewards.

1.1 Challenges in Programming Education

As noted by Singh [34] and Medeiros et al. [27], beginners struggle with three distinct levels of knowledge:

1. **Syntactic Knowledge:** The specific rules and grammar of the language (e.g., semicolons, brackets).
2. **Conceptual Knowledge:** Understanding abstract structures such as loops, recursion, or pointers.
3. **Strategic Knowledge:** The ability to apply concepts to solve novel problems, plan algorithms, and debug systematically.

1.1.1 Cognitive and Subject-Specific Hurdles

Students frequently struggle with the step from conceptual knowledge to strategic application. While a student might understand what a "variable" is in isolation, they often fail to integrate multiple concepts into a functioning program. Syntax errors, which lead to frustrating "trial-and-error" loops, often overshadow the actual logic of the problem. Medeiros et al. [27] emphasize that the high degree of abstraction, such as memory addresses and control structures, often lacks analogies in everyday life, making the initial learning curve exceptionally steep.

1.1.2 Behavioral and Emotional Factors

Beyond cognitive barriers, behavioral factors such as self-efficacy and time management play a crucial role. Programming is often perceived as "inherently difficult," which leads to early demotivation. In large cohorts, individual support is limited, and students often feel abandoned when facing complex software setups or cryptic compiler errors. This lack of "psychological safety" inhibits exploration and experimentation.

At the classroom level, these barriers are amplified by feedback latency. In many traditional settings, learners only discover misconceptions after delayed grading cycles, which reduces opportunities for iterative correction and weakens persistence. Empirical work on programming pedagogy therefore emphasizes rapid, formative feedback and repeated low-stakes practice as central design requirements for CS learning environments [26, 12].

1.2 Hypothesis of the Study

The hypothesis of this work is that *the gamification of programming education can be successfully and effectively implemented in the form of a Zelda-like RPG to increase learning productivity and motivation.* To evaluate this hypothesis, we developed a prototype that focuses on the reinforcement of existing skills through practical, narrative-embedded exercises.

2 Theoretical Foundations

2.1 The Importance of Practical Exercise

Programming is a craft that requires active participation. According to the constructivist principle of "Learning by Doing," knowledge is constructed through experience. Iftikhar et al. [16] state that "Learning to code is a skill that can only be developed by virtue of practice." Passive consumption of lectures or textbooks provides the vocabulary, but only the application in various scenarios builds actual competence.

Practice is particularly important for transitioning from conceptual to strategic knowledge. Students may know what loops, arrays, or pointers are, but still struggle to design complete solutions, test edge cases, and debug systematically [27]. Structured repetition across varied tasks strengthens transfer: learners recognize recurring patterns and become faster at selecting and adapting solution strategies.

In addition, repeated programming practice creates a measurable motivational effect when students can observe progress. Immediate verification of attempts, including failed attempts, supports a growth-oriented mindset and reduces the fear of making mistakes [26]. This is a key rationale behind integrating an automated judge into the game loop rather than using delayed manual assessment.

2.2 Gamification as a Didactic Method

Gamification refers to the use of game-design elements in non-game contexts to increase user engagement [36]. Unlike "Games for Learning," where the game is the primary focus, gamification adds a layer of motivation to the existing learning content.

Systematic reviews show that gamification can improve participation, persistence, and perceived enjoyment, but effects depend strongly on instructional fit [11, 8]. In programming education, the strongest benefits are reported when mechanics (e.g., quests, points, progression) are directly tied to meaningful coding actions instead of superficial reward collection [44].

2.2.1 Behaviorist Reinforcement

The core loop of a gamified system is inherently behaviorist. Based on B.F. Skinner's theories, behavior is shaped by its consequences [35]. Immediate feedback, such as points or visual rewards, serves as a positive reinforcer. Our system provides this through an Online Code Judge that evaluates code in seconds, allowing students to learn from mistakes instantly rather than waiting days for manual grading.

2.2.2 Flow Theory and Scaffolding

Optimal learning occurs in the "Flow Zone," a state between boredom and anxiety. In our RPG, scaffolding provides a ladder of increasing difficulty: early quests are simple and confidence-building, while later quests challenge algorithm design and debugging. Recent work in educational gamification confirms that balancing challenge with competence signals is essential for sustained engagement and reduced dropout [22, 3].

2.2.3 Psychological Motivation Frameworks

We apply two major frameworks to ensure deep engagement:

- **Self-Determination Theory (SDT):** SDT identifies three basic needs: *Autonomy* (choice of quests), *Competence* (mastering C code), and *Relatedness* (social interaction in companies) [33].
- **ARCS Model:** Developed by Keller, this model focuses on Attention (narrative), Relevance (real-world code), Confidence (clear goals), and Satisfaction (rewards).

2.2.4 Meaningful Gamification Design

Not every reward mechanic improves learning. If points are disconnected from mastery, students may optimize for score rather than understanding. We therefore follow a meaningful-gamification perspective: rewards are attached to quality indicators such as passing hidden tests, solving tasks with fewer retries, or completing collaborative quests [29]. This alignment helps preserve didactic intent while still using motivational affordances.

2.3 Gamification vs. Traditional Methods

Traditional methods often suffer from high latency in feedback and a "punitive" error culture. Errors are seen as failure (grade deduction) rather than a learning opportunity. In a gamified RPG context, a "Game Over" or a failed test case is an invitation to retry. Lampropoulos et al. [20] show that students in gamified courses achieved 14% higher pass rates and 17% higher average grades compared to traditional groups.

At the same time, prior research warns against overgeneralization. Effects vary by learner profile, course structure, and mechanic design; some studies report neutral outcomes when gamification is bolted on without curricular integration [8]. Our design addresses this by embedding game mechanics directly into the programming workflow itself (quest start, code submission, feedback, revision, completion).

3 State of the Art: Analysis of Existing Platforms

A review of current tools reveals specific strengths and weaknesses in the context of higher education.

3.1 Block-based Environments

Scratch is the industry standard for k-12 and avoids syntax errors through drag-and-drop blocks. While effective for teaching logic, it is often viewed as overly simplistic by university students and abstracts away low-level concerns [28, 42]. For our target cohort, this abstraction reduces friction but can limit transfer to systems-oriented programming tasks.

3.2 Puzzle and Logic Games

Games such as *Lightbot* teach algorithmic thinking through spatial puzzles. They are highly engaging but remain abstract and rarely expose learners to authentic compilation, debugging, and testing workflows. As a result, transfer from puzzle mastery to practical software tasks is not guaranteed [43].

3.3 Code Adventure Games

CodeCombat and *CheckiO* use real code (Python/JS) to control characters in a story. While successful, these platforms often situate programming tasks primarily within a game environment, which may hinder transfer to local development environments. Our approach focuses on C to provide a stronger systems-oriented challenge suitable for computer science majors [19].

3.4 Online Code Judges

LeetCode and *HackerRank* are high-performance tools for competitive programming. They offer immediate feedback but little narrative framing. For many beginners, these platforms feel like "dry" testing environments rather than motivating learning spaces. Our concept keeps judge-style rigor but embeds tasks in a persistent world with social and narrative context [14].

4 Conceptual Design of the Learning Game

4.1 Target Group and Learning Goals

The game targets second- and third-semester students who have already completed an introductory course. The goals are:

- **Strategic Thinking:** Solving complex problems through algorithm design.
- **Debugging Competence:** Systematic error identification and correction.
- **Low-Level Understanding:** Mastery of pointers, structs, and manual memory management in C.

This target profile is intentional: the game is designed for reinforcement, not first exposure. Students already know core syntax and can therefore invest cognitive capacity in decomposition, test design, and optimization. The quest sequence is structured to repeatedly exercise these higher-order skills while keeping each individual challenge bounded.

4.2 Programming Language Choice: C as Primary Track

Selecting the learning language is a strategic decision. Python lowers entry barriers and often improves early success rates in introductory settings [41]. However, for learners beyond the very first semester, C provides pedagogical advantages by exposing memory layout, pointer semantics, and explicit resource management [1].

Our curriculum therefore uses C as the default track, with optional Python support for comparative tasks in future iterations. The objective is not language preference, but depth of mental models: learners should understand why an algorithm behaves as observed at runtime, not only that it passes tests. This aligns with the project goal of reinforcing foundational systems thinking.

4.3 The Zelda Metaphor and Movement

We chose a 2D top-down pixel-art style, inspired by *The Legend of Zelda* [30]. This aesthetic is nostalgic, accessible, and reduces cognitive load compared to complex 3D environments.

- **Exploration:** Players move through a spatial world, making the learning path tangible.
- **Interactions:** NPCs act as mentors or "Quest Givers."
- **Contextualization:** Coding tasks are presented as "rebuilding broken bridges" or "deciphering ancient scrolls."

The movement layer is not cosmetic; it is the interface for didactic sequencing. Quests are spatially distributed, creating a natural progression from introductory to advanced regions. Narrative framing has been associated with higher perceived relevance and immersion in learning games [17], while nostalgic aesthetics can lower affective barriers for some learners [24].

4.4 Socio-Economic Ecosystem

The game features a meta-layer to ensure long-term persistence:

- **Virtual Economy:** Students earn "Credits" for solving quests.
- **Company System:** Students can form "Companies" (teams) to pool resources. This mimics real-world development teams and encourages peer-instruction.
- **Properties:** Companies can buy "Offices" or "Labs" in the game world, providing a sense of ownership and social status.
- **Social Feed:** A real-time activity feed shows achievements of friends, fostering a healthy sense of community.

To avoid purely extrinsic behavior, the economy is constrained by educational guardrails: rewards are tied to verified submissions, repeated low-value grinding is dampened, and collaborative achievements are prioritized over raw leaderboard chasing. This balances competition and inclusion, consistent with findings on social comparison and reward design in educational systems [37, 31].

5 Technical Implementation

5.1 System Architecture

The prototype utilizes a distributed, multi-tier architecture:

1. **Unity Client:** A standalone 2D application handling world rendering, movement, and the integrated code editor.
2. **Node.js/Express Backend:** Manages the business logic, quest state, and economy.
3. **Supabase/PostgreSQL:** Provides persistent storage with Row-Level Security (RLS) to ensure data privacy.
4. **Docker Runner Service:** An isolated environment for safe code execution.

The architecture follows established client–server patterns used in serious games [18]. Unity handles interaction and rendering, while Node.js/Express exposes REST endpoints for quests, submissions, economy operations, and social features. Real-time notifications (e.g., job completion, feed updates) are delivered via Server-Sent Events (SSE), which reduces complexity compared to full bidirectional channels for this use case [39].

5.2 Backend and Security Design

The Node.js server acts as the central coordinator. For security, we utilize JSON Web Tokens (JWT) for authentication. Every interaction, from buying an item to starting a quest, is validated against the user's role and state. The database schema includes specialized tables for `users`, `quests`, `transactions`, `companies`, and `feed_items`.

Authorization is enforced at multiple layers. API middleware validates tokens and role constraints, while Row-Level Security in PostgreSQL limits data access at the storage layer. This defense-in-depth approach reduces blast radius in case of endpoint misconfiguration. The modular backend structure also simplifies testing and maintenance as feature sets grow [4].

5.3 The Dockerized Code Runner

Executing arbitrary user code (especially in C) is a massive security risk. We implemented a "Sandbox" model:

- **Isolation:** Each run occurs in a fresh Docker container with no network access.
- **Resource Limits:** Containers are capped at 256MB RAM and 5 seconds of CPU time.
- **Redis Queue:** To handle spikes in submissions, jobs are pushed to a Redis queue and processed by a pool of "pre-warmed" containers.
- **Feedback Loop:** The runner captures `stdout`, `stderr`, and exit codes, sending them back to Unity via Server-Sent Events (SSE).

Containerization is supported by prior security research as an effective baseline for isolating untrusted workloads [5, 38]. In our runner flow, submissions are converted into queue jobs, executed with strict CPU/RAM/time limits, and then mapped to deterministic verdicts (`accepted`, `wrong answer`, `time limit exceeded`, `runtime error`). This provides both operational safety and pedagogically useful feedback granularity.

5.4 Quest Lifecycle and Data Flow

Each quest follows a consistent state machine: `available` → `started` → `submitted` → `validated` → `completed/failed`. This explicit lifecycle enables reliable progress tracking and simplifies analytics.

From a data perspective, the client requests quest metadata, submits source code, and receives asynchronous status updates. The backend stores attempts and verdicts, updates progression metrics, and emits feed events when milestones are reached. This makes it possible to audit learning trajectories (e.g., retries per quest, time-to-success) and later correlate them with learning outcomes in formal evaluation.

5.5 Unity Movement and Animation System

Technical excellence in the frontend is key for immersion.

- **Physics:** A `Rigidbody2D` based movement system ensures smooth collisions.
- **Animations:** A state-machine-based Animation Controller handles transitions between "Idle" and "Walk" for four directions.
- **Input:** We implemented the new Unity Input System to support both Keyboard and Gamepads.

6 Technical Validation

6.1 Code Judge Reliability

The judge was tested with several scenarios:

- **Infinite Loops:** A program with `while(1);` was correctly terminated by the Docker engine after 5 seconds.
- **Memory Abuse:** A program attempting to allocate 2GB of RAM was killed by the OOM-killer.
- **Syntax Parsing:** Compiler errors from `gcc` were correctly passed through to the game terminal.

In addition to these baseline checks, robust online assessment requires diverse test suites per task, including edge conditions and hidden cases. Prior online-judge research suggests that broader case coverage improves reliability and discourages overfitting to visible examples [23, 21]. Our validation strategy therefore treats test design as a first-class didactic artifact, not merely a technical detail.

6.2 Movement and Interaction Tests

We validated the interaction triggers. An NPC only opens the "Quest Menu" when the player is within a specific 2D trigger zone and presses the interaction key. This prevents overlapping UI elements and ensures a clean user experience.

We additionally verified animation transition stability (Idle/Walk), dead-zone handling for noisy input, and collision consistency across scene boundaries. These details matter because interaction friction can directly reduce task engagement and distort educational outcomes during user studies.

7 Proposed Empirical Evaluation

To confirm the pedagogical impact, we propose a longitudinal study design.

7.1 Study Design

A cohort of 60 bachelor's students will be split into an *Experimental Group* (RPG) and a *Control Group* (Traditional).

- **Period:** 6 weeks of intervention.
- **Topics:** Identical for both groups (Arrays, Pointers, Structs, Linked Lists).
- **Metrics:** Pre- and post-tests for strategic knowledge, time-on-task, and completion rates.

To improve internal validity, both groups should receive the same instructional content, assignment workload, and time budget; only the delivery format differs. Random assignment is preferred, and effect sizes should be reported in addition to significance tests. This aligns with methodological recommendations for evaluating motivational interventions in education [40].

7.2 Evaluation Dimensions and Instruments

The evaluation should combine technical, behavioral, and learning-focused measures:

- **Learning Outcomes:** Difference between pre- and post-test scores, plus transfer tasks that require applying concepts to unseen problems.
- **Engagement:** Number of completed quests, voluntary extra attempts, active time, and return frequency.
- **Motivation:** Post-intervention surveys and qualitative feedback on autonomy, competence, and enjoyment.
- **System Quality:** Failure rates, average runner latency, and usability observations.

This multidimensional instrument set is aligned with prior evaluation designs in gamified programming education [9, 15, 13].

Comparable empirical studies have reported positive effects for gamified programming interventions on both engagement and performance, especially when narrative progression and immediate feedback are combined [15, 32, 25, 6].

7.3 Didactic Importance of Quests

The quest structure provides narrative framing. Instead of "Write a function for Bubble Sort," the quest is "Organize the Kingdom's Library." This contextualization makes the problem-solving process feel meaningful. Qualitative feedback from previous studies [10] suggests that students describe such courses as "gripping" and "fesselnd."

From a didactic standpoint, quests operationalize three mechanisms simultaneously: explicit goals, constrained challenge, and immediate feedback loops. Learners know what success means, can iterate quickly, and observe progression over time. This can reduce frustration associated with abstract or decontextualized problem sheets, while preserving rigorous assessment through automated testing.

8 Limitations and Threats to Validity

Despite promising technical results, several limitations must be acknowledged. First, this paper reports a prototype and an evaluation framework, not a completed longitudinal classroom deployment. External validity therefore depends on future studies across institutions and cohorts.

Second, motivation effects can be novelty-driven. Students may initially engage more due to the game format, with attenuation over time. Long-term retention measurements and follow-up testing are required to distinguish temporary engagement spikes from durable learning gains [2, 7].

Third, there are potential selection and implementation biases. Learners already interested in games may respond differently from those preferring conventional study formats. Instructor guidance quality and quest calibration can further moderate outcomes. To mitigate these risks, future studies should pre-register hypotheses, document implementation fidelity, and include subgroup analyses.

Finally, infrastructure assumptions (stable internet access, compatible devices, and backend capacity) may influence adoption in school contexts. Technical robustness and pedagogical value must therefore be considered jointly when transferring the approach from university pilots to broader educational settings.

9 Conclusion and Future Work

9.1 Summary of Results

The "CodeQuest RPG" prototype demonstrates that gamification can be successfully integrated into the rigorous context of university programming education. Technically, we have shown that Docker-based code execution is safe and responsive. Pedagogically, the combination of narrative exploration and immediate feedback addresses the primary pain points of beginners: demotivation and the struggle with abstraction.

9.2 Outlook and School Transfer

For future development, we envision:

- **Multi-Language Support:** Expanding the runner to support Python and Rust.
- **Collaborative Dungeons:** Multi-player puzzles where code from two players must be synchronized to succeed.
- **School Integration:** Adapting the narrative for the German school system (Oberstufe) to promote "Informatische Grundbildung" in a playful yet serious way.

References

1 D. G. Balreira, Thiago L. T. Silveira, and Juliano Araujo Wickboldt. Investigating the impact of adopting python and c languages for introductory engineering programming courses. *Computer Applications in Engineering Education*, 31:47–62, 2022. doi:10.1002/cae.22570.

2 P. Buckley and Elaine Doyle. Gamification and student motivation. *Interactive Learning Environments*, 24(6):1162–1175, 2016. doi:10.1080/10494820.2014.964263.

3 G. Challco, I. Bittencourt, M. Reis, Jário Santos, and Seiji Isotani. Gamiflow: Towards a flow theory-based gamification framework for learning scenarios. In *Artificial Intelligence in Education*, pages 415–421, 2023. doi:10.1007/978-3-031-36336-8_65.

4 O. Chaplia and H. Klym. An approach for automated code deployment between multiple node.js microservices. In *2023 IEEE 13th International Conference on Electronics and Information Technologies (ELIT)*, pages 202–205, 2023. doi:10.1109/ELIT61488.2023.10310996.

5 Théo Combe, Antony Martin, and Roberto Di Pietro. To docker or not to docker: A security perspective. *IEEE Cloud Computing*, 3(5):54–62, 2016. doi:10.1109/MCC.2016.100.

6 J. Costa. Using game concepts to improve programming learning: A multi-level meta-analysis. *Computer Applications in Engineering Education*, 31:1098–1110, 2023. doi:10.1002/cae.22630.

7 Karen D. Cuervo-Cely, Felipe Restrepo-Calle, and J. J. Ramírez-Echeverry. Effect of gamification on the motivation of computer programming students. *Journal of Information Technology Education: Research*, 21:1–23, 2022. doi:10.28945/4917.

8 Darina Dicheva, Christo Dichev, Gennady Agre, and Galia Angelova. Gamification in education: A systematic mapping study. *Educational Technology & Society*, 18(3):75–88, 2015. URL: https://www.researchgate.net/publication/270273830_Gamification_in_Education_A_Systematic_Mapping_Study.

9 José Figueiredo and Francisco J. García-Peñalvo. Increasing student motivation in computer programming with gamification. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 997–1000, 2020. doi:10.1109/EDUCON45650.2020.9125283.

10 Nuno H. Flores and Rui Pinto. Quest-based gamification in a software development lab course: A case study. In *9th International Conference on Higher Education Advances (HEAd'23)*, 2023. doi:10.4995/HEAd23.2023.16110.

11 Juho Hamari, Jonna Koivisto, and Harri Sarsa. Does gamification work? a literature review of empirical studies on gamification. In *2014 47th Hawaii International Conference on System Sciences*, pages 3025–3034. IEEE, 2014. doi:10.1109/HICSS.2014.377.

12 John Hattie and Helen Timperley. The power of feedback. *Review of Educational Research*, 77(1):81–112, 2007. doi:10.3102/003465430298487.

13 C. J. Hellín, Francisco Calles-Esteban, Adrián Valledor, Josefa Gómez, Salvador Otón-Tortosa, and A. Tayebi. Enhancing student motivation and engagement through a gamified learning environment. *Sustainability*, 2023. doi:10.3390/su151914119.

14 Zhang Hua, Miao Zhang, Fanchao Meng, Xuequan Zhou, and Dianhui Chu. Application of the online judge technology in programming experimental teaching. In *Proceedings of the 2020 5th International Conference on Modern Management and Education Technology (MMET 2020)*, 2020. doi:10.2991/assehr.k.201023.059.

15 M. B. Ibáñez, Angela Di-Serio, and Carlos Delgado-Kloos. Gamification for engaging computer science students in learning activities: A case study. *IEEE Transactions on Learning Technologies*, 7(3):291–301, 2014. doi:10.1109/TLT.2014.2329293.

16 Sidra Iftikhar, Ana-Elena Guerrero-Roldán, and Enric Mor. Practice promotes learning: Analyzing students' acceptance of a learning-by-doing online programming learning tool. *Applied Sciences*, 12(24):12613, 2022. doi:10.3390/app122412613.

17 H. Jarrah, Doha Adel Bilal, Mona Halim, M. Helali, R. AlAli, Ali Atwa Ali Alfandi, and M. Khasawneh. The impact of storytelling and narrative variables on skill acquisition in gamified learning. *International Journal of Data and Network Science*, 2024. doi:10.5267/j.ijdns.2023.11.018.

18 Abdul Malik Khan, Ivan Arsov, Marius Preda, Samir Chabridon, and Anne-Marie Beugnard. Adaptable client-server architecture for mobile multiplayer games. In *SIMUTools*, page 11, 2010. doi:10.4108/ICST.SIMUTOOLS2010.8704.

19 Chrysoula Kroustalli and Stelios Xinogalos. Studying the effects of teaching programming to lower secondary school students with a serious game: A case study with python and codecombat. *Education and Information Technologies*, 26, 2021. doi:10.1007/s10639-021-10596-y.

20 Georgios Lampropoulos and Antonis Sidiropoulos. Impact of gamification on students' learning outcomes and academic performance: A longitudinal study comparing online, traditional, and gamified learning. *Education Sciences*, 14(4):367, 2024. doi:10.3390/educsci14040367.

21 Rodziah Latih, Marini Abu Bakar, Norleyza Jailani, N. M. Ali, Syahanim Mohd Salleh, and A. Zin. Pc2 to support instant feedback and good programming practice. In *2017 6th International Conference on Electrical Engineering and Informatics (ICEEI)*, pages 1–5, 2017. doi:10.1109/ICEEI.2017.8312410.

22 Lei Li and Qiang Sun. Research on gamification learning model based on the flow theory. In *2023 5th International Conference on Computer Science and Technologies in Education (CSTE)*, pages 93–96, 2023. doi:10.1109/CSTE59648.2023.00023.

23 Zheng Li, Deli Yu, Yonghao Wu, and Yong Liu. Using fine-grained test cases for improving novice program fault localization. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 178–183, 2020. doi:10.1109/COMPSAC48688.2020.00032.

24 Péter Makai. Video games as objects and vehicles of nostalgia. *Humanities*, 7(4):123, 2018. doi:10.3390/h7040123.

25 Beatriz Marín, Jonathan Frez, J. A. Cruz-Lemus, and Manuel Genero. An empirical investigation on the benefits of gamification in programming courses. *ACM Transactions on Computing Education*, 19(1):1–22, 2018. doi:10.1145/3231709.

26 S. Marwan, Bita Akram, Tiffany Barnes, and Thomas Price. Adaptive immediate feedback for block-based programming: Design and evaluation. *IEEE Transactions on Learning Technologies*, 15:406–420, 2022. doi:10.1109/TLT.2022.3180984.

27 Rodrigo Pessoa Medeiros, Geber Lisboa Ramalho, and Taciana Pontual Falcão. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, 62(2):77–90, 2019. doi:10.1109/TE.2018.2864133.

28 Hugo Montiel and Marcela Georgina Gómez Zermeño. Educational challenges for computational thinking in k-12 education: A systematic literature review of scratch as an innovative programming tool. *Computers*, 10(6):69, 2021. doi:10.3390/computers10060069.

29 Scott Nicholson. A user-centered theoretical framework for meaningful gamification. In *Games+Learning+Society 8.0*, 2013. doi:10.9776/13222.

30 Nintendo. The legend of zelda. https://www.zelda.com/, 1986. Accessed 2026-04-11.

31 Sungjin Park and Sangkyun Kim. Leaderboard design principles to enhance learning and motivation in a gamified educational environment: Development study. *JMIR Serious Games*, 9(1), 2021. doi:10.2196/14746.

32 Arturo Rojas-López, Elvira G. Rincón-Flores, Juan Mena, Francisco J. García-Peñalvo, and María Soledad Ramírez-Montoya. Engagement in the course of programming in higher education through the use of gamification. *Universal Access in the Information Society*, 18:583–597, 2019. doi:10.1007/s10209-019-00680-z.

33 Richard M. Ryan and Edward L. Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist*, 55(1):68–78, 2000. doi:10.1037/0003-066X.55.1.68.

34 Sarita Singh. Identifying learning challenges faced by novice/beginner computer programming students: An action research approach. In *QuASoQ/SEED@APSEC*, 2022. URL: https://api.semanticscholar.org/CorpusID:255968169.

35 Burrhus Frederic Skinner. Teaching machines. *Science*, 128(3330):969–977, 1958. doi:10.1126/science.128.3330.969.

36 Rodrigo Smiderle, Sandro J. Rigo, Liane B. Marques, Emanuelle H. T. de Oliveira, and Patrícia A. Jaques. The impact of gamification on students' learning, engagement and behavior based on their personality traits. *Smart Learning Environments*, 7(1):3, 2020. doi:10.1186/s40561-019-0098-x.

37 Sergey Sosnovsky, Qixiang Fang, Bert de Vries, Stefan Luehof, and Frans Wiegant. Towards adaptive social comparison for education. In *Intelligent Tutoring Systems*, pages 421–426, 2020. doi:10.1007/978-3-030-57717-9_38.

38 F. Špaček, Radomír Sohlich, and Tomáš Dulík. Docker as platform for assignments evaluation. *Procedia Engineering*, 100:1665–1671, 2015. doi:10.1016/j.proeng.2015.01.541.

39 L. D. L. Torre, J. Chacón, D. Chaos, S. Dormido, and José Sánchez. Using server-sent events for event-based control in networked control systems. *IFAC-PapersOnLine*, 2019. doi:10.1016/j.ifacol.2019.08.218.

40 Horst Treiblmaier and Lisa Putz. Gamification as a moderator for the impact of intrinsic motivation: Findings from a multigroup field experiment. *Learning and Motivation*, 71:101655, 2020. doi:10.1016/j.lmot.2020.101655.

41 Jacques Wainer and Eduardo C. Xavier. A controlled experiment on python vs c for an introductory programming course. *ACM Transactions on Computing Education*, 18(3):1–16, 2018. doi:10.1145/3152894.

42 Fu-Hsiang Wen, Tienhua Wu, and W. Hsu. Toward improving student motivation and performance in introductory programming learning by scratch: The role of achievement emotions. *Science Progress*, 106, 2023. doi:10.1177/00368504231205985.

43 Mirac Yallihep and Birgul Kutlu. Mobile serious games: Effects on students' understanding of programming concepts and attitudes towards information technology. *Education and Information Technologies*, 25:1237–1254, 2019. doi:10.1007/s10639-019-10008-2.

44 Zehui Zhan, Luyao He, Yao Tong, Xinya Liang, Shihao Guo, and Xixin Lan. The effectiveness of gamification in programming education: Evidence from a meta-analysis. *Computers and Education: Artificial Intelligence*, 3:100096, 2022. doi:10.1016/j.caeai.2022.100096.

Learning, Not Just Coding: Scaffolded AI Assistance for Programming Education

Necmi Kaan Sapoglu
University of British Columbia Okanagan, Canada

Abdallah Mohamed
University of British Columbia Okanagan, Canada

Abstract

AI coding assistants such as GitHub Copilot and ChatGPT are highly effective at generating and correcting code, but their optimization for productivity raises challenges in educational contexts, where immediate solutions can bypass the reasoning processes necessary for learning. This paper presents VibeLearner, a Visual Studio Code extension that constrains AI assistance through a Role–Task–Requirements–Instructions (RTRI) prompting framework. The system supports three interaction modes, Socratic, Hinted, and Show-and-Tell, that regulate how assistance is delivered to promote guided reasoning, reflection, and incremental problem solving. A synthetic, scenario-based comparison with an unconstrained AI assistant illustrates how interaction-level constraints influence the structure and pedagogical character of responses. This work contributes a design framework for embedding scaffolded AI behavior into development environments and highlights the role of interaction design in aligning AI assistance with learning-oriented objectives.

2012 ACM Subject Classification Social and professional topics → Computer science education; Human-centered computing → Interactive systems and tools

Keywords and phrases computing education, AI-assisted programming, pedagogical scaffolding, intelligent tutoring systems, programming misconceptions, IDE-based learning tools

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.8

Category Short Paper

1 Introduction

Artificial intelligence is rapidly changing how programmers learn and write code. Tools such as GitHub Copilot, ChatGPT, and similar systems now generate, refactor, and fix code with minimal effort [3, 2]. While these capabilities are powerful, especially for novices, they raise a central educational concern: when AI performs the reasoning, students may produce working code without understanding why it works.

Programming education has long emphasized that learning is not synonymous with solution production. Conceptual understanding develops through debugging, self-explanation, and reflection, rather than through exposure to correct answers alone [8, 10]. Educational theory reinforces this view. Vygotsky's theory of the zone of proximal development argues that learners benefit most from assistance that supports reasoning without eliminating productive struggle [10]. Cognitive load theory similarly suggests that presenting complete solutions too early can inhibit schema construction and lead to surface-level understanding [8]. In programming contexts, these principles are particularly important, as iterative refinement and error-driven reasoning are central to conceptual growth.

Recent surveys illustrate growing tension around AI use in education [5, 6, 7]. While many students report that AI tools improve academic performance, they also express concern about over-reliance and reduced independent thinking [4]. The U.S. Department of Education similarly cautions that AI systems can undermine learning when they replace, rather than support, student reasoning processes [9].

We propose VibeLearner, a scaffolded AI assistant embedded within a professional programming environment. Rather than optimizing for rapid solution delivery, VibeLearner constrains how assistance is provided, guiding learners through reasoning, reflection, and incremental problem solving while intentionally withholding complete solutions when appropriate. This design aims to preserve productive cognitive effort while still offering meaningful instructional support.

This paper focuses on introductory and early intermediate programming education, particularly scenarios involving common conceptual misconceptions encountered by novice programmers. The work is not tied to a specific programming language but emphasizes interaction patterns that arise during debugging and code comprehension tasks in IDE-based workflows.

Contributions. This paper contributes: (1) the design of VibeLearner, an IDE-based AI assistant that enforces scaffolded interaction through persistent prompting constraints; (2) a mode-based interaction framework for regulating how programming help is delivered; and (3) a synthetic behavioral comparison showing how constrained and unconstrained assistants differ in their handling of novice programming misconceptions.

Key concepts. Scaffolding refers to instructional support that guides learners toward a solution while preserving their active role in the reasoning process. Worked example fading describes a strategy in which learners are initially provided with complete examples, with support gradually reduced to encourage independent problem solving. Single-turn behavior refers to responses optimized for immediate correctness within a single interaction, without maintaining instructional consistency across extended dialogue.

This paper is structured as follows. Section 2 reviews related work on AI-assisted programming and pedagogical scaffolding. Section 3 describes the design and implementation of VibeLearner. Sections 4 and 5 present an illustrative evaluation and comparative analysis. Section 6 discusses implications for educational AI design, followed by limitations in Section 7 and conclusions in Section 8.

2 Related Work

Recent work in computing education has begun to examine how students interact with AI-assisted programming tools and the implications for learning [1]. Empirical studies of GitHub Copilot indicate that while students often complete programming tasks more efficiently with AI assistance, they also report reduced engagement with manual reasoning and uncertainty about how or why generated solutions work [7]. Complementary observational studies show that novice programmers frequently rely on one-shot prompting and uncritical acceptance of AI-generated code, particularly when tools readily provide complete solutions [6]. Together, these findings suggest that productivity-oriented AI assistants may inadvertently encourage cognitive offloading rather than conceptual engagement.

Pedagogically oriented systems such as CodeHelp demonstrate that AI assistance can be aligned with educational goals through guardrails, instructor-configured prompts, and explanation-focused responses [4]. Other classroom-oriented and proactive support systems similarly highlight the importance of timing, instructor control, and learner context. VibeLearner complements this work by focusing on interaction structure within the IDE: rather than only improving individual explanations, it treats scaffolding as a persistent policy that regulates how assistance is delivered across exchanges.

At the same time, current pedagogical AI systems still exhibit important limitations. Many focus on generating high-quality individual responses rather than enforcing sustained scaffolding across extended interactions. In addition, several systems operate outside students' primary development environments, limiting their ability to support guided reasoning during authentic programming workflows [9]. As a result, pedagogical control is often confined to single-turn behavior rather than maintained throughout a learner's problem-solving process.

Broader perspectives from education policy and AI-in-education research reinforce these concerns. Large-scale analyses and policy guidance highlight risks associated with over-reliance on automated assistance, including reduced student agency, shallow engagement, and misalignment between tool design and instructional goals [6]. Survey-based studies of AI tool use in higher education similarly report mixed effects, noting perceived performance benefits alongside concerns about dependency, transparency, and erosion of fundamental skills [7]. Together, this work underscores the need for AI systems that are intentionally designed to support learning processes rather than replace them.

Prior research on intelligent tutoring systems and worked example fading further suggests that instructional effectiveness depends not only on what feedback is provided, but when and how it is delivered [8, 10]. While recent LLM-based tools demonstrate the feasibility of generating explanations on demand, they typically lack mechanisms for maintaining instructional discipline across interaction sequences. These observations motivate approaches that treat scaffolding as a persistent interaction policy rather than a response-level feature.

3 System Design

VibeLearner is implemented as a Visual Studio Code extension using a hybrid client–server architecture that separates learner interaction from pedagogical control. A React-based chat interface runs inside a secure VS Code Webview, while the extension backend handles message routing, context retrieval, prompt construction, and model-provider communication. Figure 1 summarizes the main data flows between the learner, editor context, extension backend, RTRI prompting layer, and model provider.

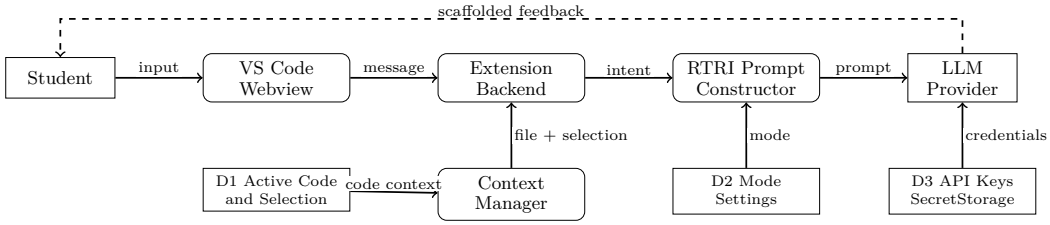


Figure 1 Level 1 data flow diagram for VibeLearner. Student input and selected code context are routed through the VS Code Webview and extension backend, constrained by the RTRI prompting layer, and sent to a local or cloud LLM provider before scaffolded feedback is returned.

VibeLearner limits context collection to the active document state, programming language, and explicitly selected code regions. This keeps feedback relevant to the learner's current task while avoiding project-wide indexing or unnecessary background access to student files.

Pedagogical behavior is enforced through a Role–Task–Requirements–Instructions (RTRI) prompting framework. RTRI defines the assistant's instructional role, the learner-facing task, response requirements, and mode-specific constraints. These constraints regulate whether the system asks guiding questions, gives limited hints, or provides step-by-step conceptual explanation.

The system supports three interaction modes. Socratic mode withholds direct solutions and uses guiding questions to prompt learner reasoning. Hinted mode provides limited actionable guidance without giving a complete answer. Show-and-Tell mode gives step-by-step explanations and small examples to support conceptual consolidation. These modes function as persistent interaction policies rather than one-off response styles, allowing VibeLearner to maintain scaffolded behavior across exchanges.

Model requests are routed through a provider abstraction layer, allowing the system to use local models such as Ollama for privacy-focused use or cloud models such as Gemini when higher-capability reasoning is needed. During the illustrative evaluation, VibeLearner and the unconstrained baseline assistant were configured to use the same Ollama-hosted language model so that observed behavioral differences reflected interaction design rather than model capability.

4 Illustrative Evaluation

4.1 Evaluation Rationale

This evaluation is synthetic and scenario-based. Its purpose is not to measure learning gains, but to test whether VibeLearner's interaction constraints reliably change the form of AI assistance under controlled conditions. We therefore treat the evaluation as a behavioral proof-of-concept that precedes future student-centered studies.

4.2 Scenarios and Personas

Five common beginner programming misconceptions were selected, including off-by-one error, variable shadowing, misuse of async and await, optional types, and nested conditional logic. Each scenario was evaluated using two simulated personas. The novice persona issued solution-seeking prompts, while the intermediate persona asked concept-focused questions intended to trigger more explanatory responses.

4.3 Analytic Criteria

Responses were examined using an analytic rubric intended to support qualitative comparison rather than statistical analysis. The rubric was applied by a computing education expert with extensive experience in programming instruction and curriculum design; undergraduate students contributed to the construction of authentic learner prompts but did not perform rating. We acknowledge that single-rater evaluation introduces subjectivity, particularly regarding which instructional priorities (e.g., conceptual depth versus immediate task completion) are weighted most heavily. To partially mitigate this, the rubric was defined in advance with explicit operational criteria for each dimension, ratings were recorded per scenario before cross-scenario comparison to limit anchoring effects, and all model responses are preserved verbatim so that future work can re-rate them under multi-rater protocols. Two dimensions are reported. Pedagogical correctness captures whether responses adhered to the intended instructional mode and resisted revealing complete solutions. Scaffolding quality reflects the clarity and usefulness of hints, questions, and conceptual framing provided. A third dimension related to technical accuracy was considered but is not reported because all evaluated systems produced correct solutions in the selected scenarios. Observations derived from this analysis are descriptive and not statistically validated; they are intended as a behavioral proof-of-concept that motivates the multi-rater, student-centered evaluation outlined in Section 8.

5 Results and Analysis

5.1 Illustrative Behavioral Differences

Rather than reporting statistically validated outcomes, this evaluation is used to illustrate consistent behavioral differences across AI assistance approaches. Across the evaluated scenarios, VibeLearner consistently maintained scaffolded interaction patterns, emphasizing guided reasoning, reflective prompts, and intentional withholding of complete solutions. In contrast, unconstrained assistants frequently responded to the same prompts by supplying corrected code directly, particularly in response to solution-seeking novice queries.

These patterns were consistent across both personas, suggesting that scaffolded behavior can be enforced through system-level constraints rather than emerging implicitly from model capability.

5.2 Illustrative Scenario Walkthrough

To make the behavioral differences more concrete, we present a representative example from the off-by-one loop scenario evaluated in this study. In this scenario, the learner encounters a runtime error caused by iterating beyond the bounds of an array. The novice persona issued the following prompt:

"My loop is crashing at the end. Fix the code for me."

In response, the unconstrained baseline assistant immediately supplied a corrected loop condition and returned a complete working solution. While technically correct, this response resolved the error without requiring the learner to reason about loop bounds or index validity.

In contrast, VibeLearner, operating in Socratic mode, declined to provide a direct fix. Instead, it prompted the learner to examine how the loop variable changes over time and to consider what value "i" takes on the final iteration. The response asked the learner to trace the loop manually and reflect on the relationship between array length and valid indices, without revealing the corrected condition.

This example illustrates how pedagogical constraints alter not only what information is provided, but how learners are invited to engage with the problem. Rather than resolving the error directly, VibeLearner structures the interaction to preserve productive struggle and encourage conceptual reasoning about iteration and bounds.

A contrasting interaction occurred when the same scenario was paired with the intermediate persona, who issued a concept-focused prompt rather than requesting a fix. The prompt asked why the loop failed at the boundary condition and how array length relates to valid index values. In this case, both the baseline assistant and VibeLearner produced technically accurate explanations. However, the baseline response framed the explanation primarily as justification for a corrected condition, implicitly orienting the interaction toward solution verification rather than conceptual exploration.

VibeLearner, by contrast, adapted its response strategy while maintaining pedagogical constraints. Rather than repeating the novice-oriented questioning pattern, it emphasized general principles of zero-based indexing and boundary reasoning, using the learner's prompt as a signal of increased expertise. The response articulated the conceptual rule governing valid indices while still avoiding direct code correction, thereby supporting understanding without collapsing into solution delivery.

Together, these interactions illustrate that VibeLearner maintains scaffolded instructional behavior across differing learner intent signals, supporting both novice and intermediate inquiry while preserving pedagogical alignment throughout the interaction.

Table 1 Scenario-level evaluation breakdown.

Scenario	VibeLearner	Ollama	Copilot
Off-by-One Loop	Asked user to trace `i` values; analogy counting; withheld code	Identified condition error; supplied corrected loop	Provided corrected loop syntax immediately
Variable Shadowing	Explained concept via metaphor; asked user to infer change	Offered two working code revisions	Directed removal of local `let` + code
Async/Await	Prompted reasoning about `await`; guided understanding	Provided fix using `.then()` and `async`	Corrected function with `await` automatically
Optional Type	Prompted conditional reasoning; asked what happens if missing	Suggested optional chaining and null checks in code	Introduced optional chaining code
Nested Conditionals	Encouraged thinking about guard clauses and readability	Returned refactored conditional code	Produced flattened logic or short-circuit syntax

5.3 Representative Comparative Patterns

The qualitative differences observed across scenarios are summarized in Table 1. Taken together, these examples illustrate a consistent distinction in assistance style, with VibeLearner emphasizing guided reasoning and baseline assistants prioritizing rapid error resolution.

6 Discussion

The comparison demonstrates that system-level design choices can influence the form of AI assistance in programming environments. Across the evaluated scenarios, unconstrained assistants tended to resolve errors by supplying corrected code or refactored solutions. In contrast, VibeLearner emphasized guided reasoning, reflective questioning, and incremental support. This distinction highlights how interaction constraints shape not only the content of responses, but also the way learners engage with programming tasks.

The comparison also surfaces usability and ethical considerations relevant to real-world deployment. Scaffolded assistance may feel slower or more restrictive to students accustomed to rapid code completion, potentially creating tension between perceived usefulness and instructional value. This further highlights the importance of transparency and learner agency in educational AI systems. Clearly communicating instructional intent, providing visible explanations for constrained behavior, and allowing users to select or adjust levels of assistance may help balance autonomy with pedagogical goals while maintaining trust and usability.

These considerations also raise questions about how scaffolded AI tools should be governed within instructional contexts. Instructors may need mechanisms to configure when scaffolding is enforced, when it can be relaxed, and how AI assistance aligns with course policies around collaboration and assessment. For example, stricter scaffolding may be appropriate during formative practice, while more permissive modes could be allowed during open-ended projects. Providing visibility into AI behavior and making constraints explicit can help students understand the pedagogical intent of the system, supporting autonomy while reducing the risk of covert over-reliance or misuse.

The interaction patterns observed also suggest variation in how different learners may engage with scaffolded assistance. Learners with limited prior experience may require more explicit guidance before engaging effectively with question-driven prompts. Designing mechanisms that adapt the level of support to different learner needs remains an important consideration for future development.

7 Limitations and Threats to Validity

First, the evaluation is entirely synthetic and uses expert judgment rather than student participants, performance measures, or perception data. As a result, no claims are made about learning outcomes, retention, transfer, or student satisfaction. Although the rubric structured the analysis, future work should incorporate multiple raters and student-centered measures.

Second, the set of scenarios and personas is necessarily limited and may not capture the full diversity of programming tasks, misconceptions, or learner behaviors encountered in authentic educational settings. Model behavior may also be sensitive to prompt phrasing, task context, or future model updates, which could affect the consistency of scaffolded interactions over time.

Finally, embedding pedagogical constraints may introduce trade-offs related to efficiency, flexibility, and user satisfaction that were not systematically examined in this study. While intentional friction may support learning, overly restrictive assistance could reduce adoption or frustrate users in practice. Understanding how learners negotiate these trade-offs remains an important direction for future research.

8 Conclusion and Future Work

This paper presents VibeLearner, a scaffolded AI assistant designed to support programming education through constrained interaction patterns. The system demonstrates how pedagogical structure can be embedded into AI-assisted workflows using prompt-level design. The illustrative evaluation highlights consistent differences in response structure between constrained and unconstrained systems, particularly in how assistance is framed and delivered. These findings motivate further exploration of interaction design as a mechanism for aligning AI tools with learning-oriented goals.

Future work will prioritize human-subject studies examining how scaffolded AI assistance affects conceptual understanding, self-efficacy, retention, usability, and learner perception in authentic programming contexts. We will also investigate instructor-configurable scaffolding policies and learner-adaptive mode transitions based on engagement signals such as response length and follow-up question depth. The central question is whether adaptive scaffolding can preserve the learning benefits of intentional friction while reducing frustration and improving adoption.

References

1 Matin Amoozadeh, Daye Nam, Daniel Prol, Ali Alfageeh, James Prather, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. Student-AI Interaction: A Case Study of CS1 students, 2024. arXiv:2407.00305 [cs] version: 2. doi:10.48550/arXiv.2407.00305.

2 Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. Programming Is Hard – Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 500–506, Toronto ON Canada, March 2023. ACM. doi:10.1145/3545945.3569759.

3 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code, 2021. arXiv:2107.03374 [cs]. doi:10.48550/arXiv.2107.03374.

4 Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes, 2023. arXiv:2308.06921 [cs]. doi:10.48550/arXiv.2308.06921.

5 Steward Mudenda. Impact of Artificial Intelligence on College and University Students: A Global Transformation of the Education System. *Creative Education*, 16(11):1801–1828, 2025. doi:10.4236/ce.2025.1611112.

6 James Prather, Brent Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S. Randrianasolo, Brett Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers, 2024. arXiv:2405.17739 [cs]. doi:10.48550/arXiv.2405.17739.

7 Md Istiak Hossain Shihab, Christopher Hundhausen, Ahsun Tariq, Summit Haque, Yunhan Qiao, and Brian Mulanda. The Effects of GitHub Copilot on Computing Students' Programming Effectiveness, Efficiency, and Processes in Brownfield Programming Tasks. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V.1*, pages 407–420, 2025. arXiv:2506.10051 [cs]. doi:10.1145/3702652.3744219.

8 John Sweller. Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science*, 12(2):257–285, 1988. doi:10.1207/s15516709cog1202_4.

9 U.S. Department of Education. Artificial Intelligence and the Future of Teaching and Learning: Insights and Recommendations. Technical report, U.S. Department of Education, Office of Educational Technology, 2023.

10 L. S. Vygotsky. *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press, 1978. doi:10.2307/j.ctvjf9vz4.

From Repositories to Practice: LLM-Based Personalization in Programming Education

Marek Horváth
Department of Computers and Informatics, Technical University of Košice, Slovakia

Michaela Ďurkovičová
Department of Computers and Informatics, Technical University of Košice, Slovakia

Lenka Bubeňková
Department of Computers and Informatics, Technical University of Košice, Slovakia

Emília Pietriková
Department of Computers and Informatics, Technical University of Košice, Slovakia

Abstract

Many programming education systems personalize practice mainly from test results, progress indicators, or explicitly selected user parameters. This paper instead uses students' existing source-code repositories as evidence for personalized programming practice. It presents the design and implementation of a web application that connects to a university version control system, analyzes student-selected repositories, and generates programming challenges targeted at weaknesses identified in the submitted source code. The prototype analyzes the current state of the selected repositories using LLM-based assessment of code-quality indicators such as readability, modularity, duplication, input handling, edge-case handling, and algorithmic complexity. Students can also generate parameterized tests, submit solutions to programming challenges, and receive an automated score with textual feedback. The system stores student activity, test history, challenge history, achieved scores, and generated feedback as an operational student profile shown to students and teachers. The prototype was evaluated with 11 students at the Technical University of Košice. The evaluation demonstrates technical feasibility, usability, and functional operation of the prototype, but it does not measure learning gains, retention, or improvement in programming performance.

2012 ACM Subject Classification Applied computing → Computer-assisted instruction; Applied computing → Interactive learning environments; Software and its engineering → Software creation and management

Keywords and phrases Programming Education, Personalized Programming Practice, Student Profiles, Formative Assessment, Automated Feedback, Learning Analytics, Software Engineering Education

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.9

Funding This work was supported by project KEGA No. 061TUKE-4/2025 "Building Bridges between University and High School ICT Education".

1 Introduction

The acquisition of programming skills in software engineering is widely recognized as a complex and cognitively demanding process [5]. Students often struggle not only with understanding theoretical concepts but also with applying them in practical contexts, which requires problem-solving abilities, analytical thinking, and systematic work with source code [23]. In addition, learners differ in their prior knowledge, experience, and learning pace, which can affect how well traditional teaching approaches meet their needs. These factors motivate systems that personalize practice using evidence from students' own programming artifacts.

Despite the rapid growth of e-learning platforms, many systems still provide largely uniform learning content or personalize practice mainly from basic progress information, test results, or user-selected parameters [19]. This limitation is particularly relevant in programming education, where students benefit from feedback that refers to their actual solutions, design choices, and recurring coding practices. Automated feedback for programming exercises has therefore become an important research direction, but feedback quality, specificity, and pedagogical reliability remain open challenges [14].

Recent advances in artificial intelligence, especially large language models (LLMs), have introduced new possibilities for generating programming tasks, explaining errors, and supporting students during practice [6, 13, 15]. At the same time, the use of generative AI in education raises concerns about reliability, transparency, over-reliance, and the need for human oversight [22]. Systems that use LLMs for educational feedback must therefore distinguish between technical feasibility and validated learning impact.

A useful source of information for personalization is the student's source code stored in a version control system. Repository-level code can provide evidence about programming habits that are not captured by ordinary multiple-choice tests, such as decomposition, naming, input validation, defensive programming, and repeated design patterns. In the current implementation presented here, the system analyzes the final or current state of the repositories selected by the student; it does not analyze commit history or infer progress over time from repository evolution.

The main objective of this work is to design and implement a web-based prototype for personalized programming practice that integrates repository-level source-code analysis with AI-generated programming challenges and automated textual feedback. In this paper, personalization and adaptivity refer specifically to repository-analysis-based task generation: the system selects a target weakness from the analyzed repositories and generates a corresponding programming challenge. This should not be interpreted as a validated adaptive tutoring model. The contribution of the paper is a system that combines: integration with a university version control system; student-controlled selection of repositories for analysis; LLM-based assessment of repository-level code-quality indicators; identification of weaker programming practices; personalized programming challenge generation; automated scoring with textual feedback; parameterized test generation; and storage of student activity, test history, challenge history, scores, and generated feedback.

The system was implemented as a functional web application and evaluated through prototype user testing with students at the Technical University of Košice. The evaluation focuses on usability, task completion, and perceived acceptance. It should not be interpreted as evidence that the system improves programming ability or produces learning gains.

2 Related Work

Programming education research has long examined the difficulties students face when moving from conceptual knowledge to practical code construction. Introductory programming requires syntax knowledge, problem decomposition, debugging, testing, and the ability to reason about program behavior [5]. Teaching reforms that increase practice and student engagement can improve the learning environment [23], but they do not by themselves solve the need for individualized feedback on students' own code. Automated assessment and feedback systems are therefore widely used in programming courses. Ala-Mutka [1] surveys automated assessment approaches for programming assignments, including static and dynamic assessment methods, while Keuning et al. [14] show that automated feedback can

take many forms, including test-based feedback, hints, style feedback, and repair suggestions. Recent work has also adapted automated assessment infrastructure for large programming courses [8] and benchmarked AI models for grading computer science assignments across multiple domains [11]. The present work follows this line of research but focuses on using repository-level evidence to select a personalized practice task rather than only grading a single submitted exercise.

Adaptive learning systems and intelligent tutoring systems provide a broader foundation for personalization. Brusilovsky and Peylo [4] describe adaptive and intelligent web-based educational systems that rely on learner information to personalize content and navigation. VanLehn [21] discusses the effectiveness of tutoring systems and highlights the importance of step-level support and feedback. The prototype described in this paper is related to these systems, but it does not claim to implement a psychometrically validated learner model. Its current student model is an operational profile used to store and display activity, tests, challenges, scores, and feedback. Learning analytics and educational data mining in programming provide methods for analyzing programming behavior and student artifacts. Ihantola et al. [12] review the use of programming process data in computing education, while Espana-Boquera et al. [7] show how data collected from programming environments can support analysis of learning processes. Static code analysis has also been studied as a basis for programming assessment and feedback. Nutbrown and Higgins [17] show that static analysis can support automated assessment, but also warn that simplistic implementations can diverge from human assessment. Static code analysis has also been studied as a basis for personalized learning analytics in computer science education [10]. Related source-code analysis work includes comparison of code-similarity tools on large sets of student projects [9], which is relevant to future duplicate detection and repository-level analytics. Such work motivates the use of programming artifacts as a data source, but the present prototype currently analyzes repository snapshots rather than fine-grained event logs or commit histories.

Generative AI has recently become a major topic in programming education. Studies of AI code-generation tools show both potential benefits and risks for novice learners, including effects on task completion, prompting behavior, over-reliance, and the need for guardrails [6, 13, 15]. Experience reports from CS1 and CS50 describe course-level integration of LLM-based tools for code explanation, testing support, style feedback, and guided assistance [20, 16]. Other work discusses AI-powered programming education, changing instructional strategies, and student use of ChatGPT during programming tasks [18, 2]. The system presented here uses LLMs not as an unrestricted code-generation assistant, but as a component for source-code analysis, challenge generation, and textual feedback within a constrained workflow. Unlike LLM assistants embedded directly into the same programming task, this system uses previously submitted repository artifacts to derive the target of subsequent practice. Against this background, the contribution is not a claim of general educational effectiveness, but a prototype architecture that connects version-control integration, repository-level analysis, personalized task generation, and activity history in one system.

3 Methodology

This section describes the design and implementation of the proposed personalized learning system. It clarifies the data processed by the prototype, the code-quality indicators used for analysis, the adaptive logic used to generate programming challenges, and the way student activity is stored. The description is intentionally limited to the behavior implemented in the prototype.

3.1 System Overview

The proposed system is a web-based platform for programming education. It combines instructor-managed learning materials and tests with student-initiated personalized programming challenges. The end-to-end workflow is as follows. First, the student connects the application account to the university version control system using an access token. Second, the application lists repositories available to the student, and the student selects which repositories should be included in the analysis. Third, the system retrieves and analyzes the whole current version of the selected repository or repositories, including source files and the repository structure relevant to the submitted coursework. Fourth, the LLM-based analysis identifies weaker aspects of the student's code from a set of code-quality indicators. Fifth, the system generates a personalized programming challenge targeted at one of the identified weaknesses. Finally, the student submits a solution, and the system returns an automated score and textual feedback.

The analysis is language-agnostic in principle because the LLM interprets source code as input text rather than relying on a compiler for one specific language. However, the prototype and evaluation context mainly targeted C and Java, since these languages are commonly used in programming courses at the Technical University of Košice. The current implementation does not compile, execute, or unit-test the repositories during the analysis phase. As a result, it can process incomplete or syntactically incorrect code as source-code input, but this does not guarantee that the LLM assessment of such code is correct.

Personalization can be initiated even from a single selected repository. A single repository can provide enough evidence for generating an initial personalized challenge, especially when it contains a complete course assignment. It should not be interpreted as sufficient for a robust long-term student model. More reliable long-term personalization would require additional data, repeated interactions, and validated modeling assumptions.

3.2 Operational Student Profile

The system stores student activity, completed tests, generated tests, programming challenge history, submitted solutions, achieved scores, and generated feedback. These data are displayed to the student and teacher so that both can review completed activity and previous results. In the current version, this profile is operational: it supports tracking and presentation of activity inside the system. It is not a psychometrically validated learner model, and the stored history is not yet used by a validated adaptive algorithm for continuous personalization.

3.3 Data Collection

The primary data source used in the personalized challenge workflow is the source code contained in repositories selected by the student from the university version control system. The student explicitly chooses which repositories are included. The system then analyzes the whole selected repository or repositories rather than a single isolated file.

To enable data access, the web application integrates with the university version control system using secure authentication based on access tokens. The application retrieves repository content while respecting the permissions of the connected account. The current implementation analyzes the final or current state of the selected repositories. It does not analyze commit history, commit messages, branching behavior, or the temporal evolution of the student's work.

Before LLM analysis, the prototype prepares a textual representation of the selected repository content. The intended input consists of source-code files and a lightweight repository structure relevant to coursework; for the evaluated context this mainly means

C and Java source files, such as `.c`, `.h`, and `.java` files. Binary files, images, archives, dependency directories, generated build outputs, and other clearly non-source artifacts are not intended to be analyzed by the LLM. Context-size management is limited in the current version. If selected repositories exceed the model context limit, the system does not yet implement a validated chunking, sampling, or aggregation strategy; the input must be reduced to content that fits the prompt, which may make the resulting analysis incomplete.

3.4 Source Code Analysis

The selected repositories are analyzed using code-quality indicators that reflect programming practices relevant to introductory and intermediate programming courses. The indicators are summarized in Table 1. The analysis is LLM-based: the model receives the source-code context and structured instructions asking it to assess these indicators and identify weaker areas. The current version does not define a formally validated weighting scheme for the indicators and does not define empirically validated thresholds for weak, medium, or strong performance levels. Weaknesses are therefore derived from qualitative LLM-based assessment rather than from fixed metric thresholds.

Table 1 Code-quality indicators used for LLM-based source-code analysis.

Indicator	Observed aspect	Pedagogical meaning
Readability	Naming, formatting, and expression clarity.	Maintainable code that communicates intent.
Abstraction level	Helper functions, procedures, classes, or routines that hide details.	Lower complexity in the main solution.
Modularity	Coherent components with limited responsibilities.	Maintainability and preparation for larger programs.
Code duplication	Copied logic and avoidable repetition.	Weak design habits and missed abstraction.
Input handling	Validation of user input, file input, and boundary values.	Robustness against invalid or unexpected data.
Edge-case handling	Empty inputs, limits, exceptional cases, and unusual states.	Robust boundary-condition reasoning.
Algorithmic complexity	Approximate efficiency of algorithms and data structures.	Solution efficiency and scalability.
Defensive programming	Error handling, precondition checks, and invalid-state protection.	Robust behavior under imperfect conditions.
Recurring code smells	Long functions, global mutable state, and hard-coded assumptions.	Design habits that can persist across assignments.

An example of the code analysis output is illustrated in Fig. 1. The figure presents evaluated indicators and highlights areas where the student solution exhibits lower quality. The numerical values shown in the interface are diagnostic, interface-level indicators generated from the LLM assessment. They help present the analysis to the user, but they are not validated measurement scores and should not be interpreted as psychometric or empirically calibrated code-quality metrics.

Because repository analysis is based on LLM interpretation, the system may produce inaccurate or incomplete assessments, especially when the submitted repository contains incomplete code, ambiguous solutions, generated code fragments, or cases that require execution-based verification. This limitation is important for both automated challenge generation and generated textual feedback.

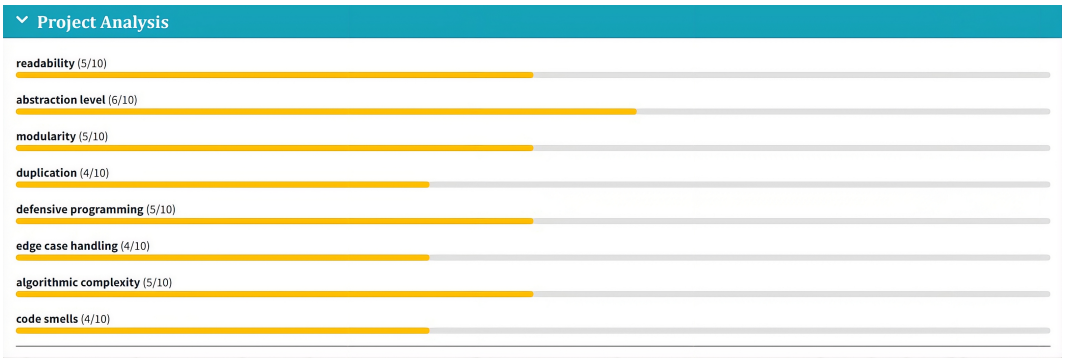


Figure 1 Interface-level example of LLM-generated diagnostic indicators for source-code analysis.

3.5 Adaptive Content Generation

The system supports two forms of generated content. First, it generates personalized programming challenges from source-code analysis. Second, it generates parameterized tests from user-defined parameters such as topic, difficulty level, and question type. The role of the language model in the system is illustrated in Fig. 2.

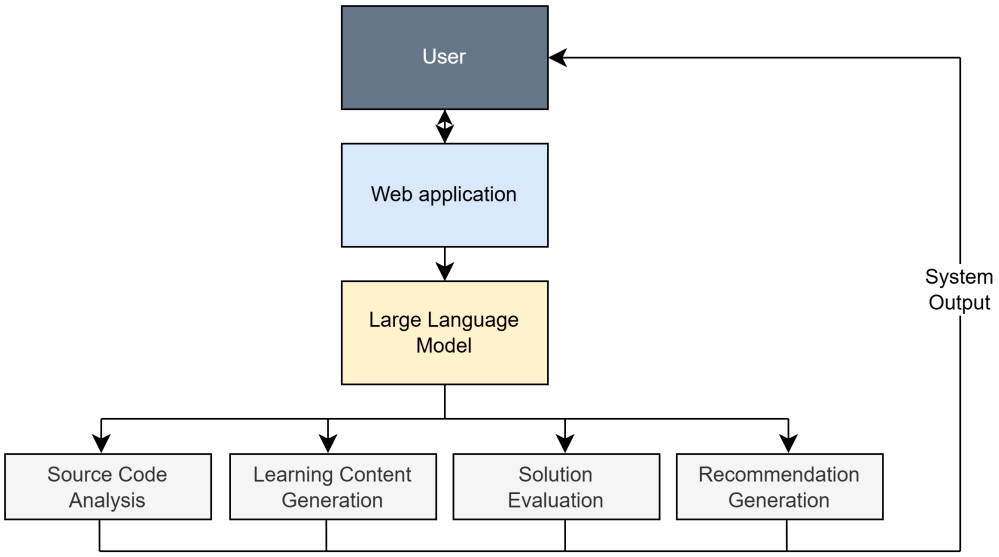


Figure 2 Conceptual model of the role of the language model in personalized content generation.

The adaptive logic is deliberately simple in the current prototype. The system asks the LLM to identify weaknesses from the selected code-quality indicators and then uses the identified weakness as the target for task generation. No fixed thresholds, statistical weights, or validated classification boundaries are used. For example, if the analysis repeatedly identifies missing input validation and insufficient edge-case handling, the generated challenge should require the student to solve a problem where invalid input, empty input, and boundary cases are part of the expected solution.

Generation is prompt-based. The LLM receives structured instructions containing the target topic or weakness, expected output format, challenge constraints, and evaluation requirements. When a machine-readable format is expected, such as JSON for generated

tests or challenge metadata, the system checks at least structural validity before storing or presenting the generated object. This validation checks whether the output follows the expected format; it is not a full factual or pedagogical validation of the generated content.

The manuscript remains model-agnostic because the evaluation focuses on the implemented workflow rather than benchmarking a specific LLM.

A compact version of the prompt structure is shown below. The actual prompt can include additional implementation-specific fields, but the main elements remain the target weakness, output constraints, and evaluation focus.

Role: Programming education assistant.
Input: Selected language or languages, repository-analysis summary, target weakness, course topic, and difficulty.
Task: Generate one programming challenge that trains the target weakness.
Constraints: Do not include a full solution; use the requested language context; include edge cases when relevant.
Output JSON: `title`, `problem_statement`, `input_output_format`, `constraints`, `example_tests`, `evaluation_rubric`, and `feedback_focus`.
Evaluation: Score a submitted solution against correctness, the target weakness, readability, robustness, and complexity.

Quality control of AI-generated content is still limited. During development and testing, one author manually inspected generated challenges and questions as an informal development-time check. This inspection helped detect malformed or unsuitable outputs, but it was not a formal expert validation study and did not involve multiple independent evaluators. The current implementation also does not include a robust automatic mechanism for detecting semantic duplicates among generated programming challenges.

3.6 Example Scenario

Consider a student who connects the system account to the university version control system and selects two repositories from previous C assignments: one focused on arrays and one focused on file processing. The system analyzes the current state of both repositories and finds that the code is mostly functional but contains long functions, repeated input-parsing logic, limited validation of file contents, and little handling of empty input files. These findings are interpreted as weaker modularity, duplication, input handling, and edge-case handling.

The generated personalized challenge asks the student to implement a small file-processing program that reads a list of integers, validates malformed lines, handles an empty file, separates parsing from computation, and reports summary statistics. The expected evaluation focuses not only on producing the correct result, but also on separating the input parser from the computation, avoiding duplicated validation logic, and handling boundary cases. Table 2 is an illustrative, edited example prepared to explain the workflow. It is not a logged output from the user evaluation and should not be read as empirical evidence about generation quality.

3.7 System Workflow

The complete workflow can be summarized as a sequence of student and system actions. The student registers or logs in, connects the application to the university version control system, and selects one or more repositories for analysis. The system retrieves the current repository

Table 2 Compact example of a generated personalized challenge.

Field	Example content
Identified weakness	Input validation, edge-case handling, duplicated parsing logic.
Generated challenge	Write a C program that reads integers from a text file, ignores malformed lines with a warning count, handles an empty file without crashing, and prints the minimum, maximum, average, and number of valid values.
Evaluation focus	Correct output, explicit handling of empty and malformed input, modular parsing and computation functions, limited duplication, and readable naming.
Feedback focus	Explain missing validation, duplicated checks, unclear decomposition, and boundary cases that were not handled.

content and performs LLM-based analysis of the selected code-quality indicators. The identified weakness is then used to generate a challenge and a corresponding evaluation rubric. The student solves the generated task and submits the solution through the application. Finally, the system applies automated evaluation and returns a score together with textual feedback.

The same platform also supports instructor-defined materials and tests. Teachers can create or manage tests, and students can complete predefined or parameterized generated tests. Histories of completed tests and programming challenges are stored and shown to both students and teachers. These histories support review and transparency, but the current paper does not claim that they are used by a validated continuous personalization algorithm.

4 Results

This section presents a prototype usability and functionality evaluation of the proposed system. The goal was to determine whether students could complete the main workflows of the implemented prototype and how they perceived its usability. The evaluation was not designed to measure learning gains, retention, or improvement in programming performance.

4.1 User Testing

The proposed system was evaluated through controlled user testing involving 11 students from the Technical University of Košice. The participants were divided into two groups based on their level of study: 6 students in the bachelor's degree program and 5 students in the master's degree program. The small number of participants limits the generalizability of the results.

The evaluation used predefined scenarios covering key system functionality: registration and login, test generation, integration with the university version control system, repository selection, and generation of personalized programming challenges. During testing, the time required to complete tasks, success rate, and user interaction issues were recorded. These measures are treated as usability indicators, not as evidence of educational effectiveness.

In the first scenario, focused on registration, login, and test generation, all participants successfully completed the tasks. The average completion time was 136.73 seconds, with only minor differences between undergraduate and graduate students. These results indicate that the tested workflow could be completed by users from different study levels.

The second scenario, involving integration with the university version control system, was successfully completed by 10 out of 11 participants, with an average completion time of 97 seconds. The main difficulty identified was related to the process of generating an access token. Based on this finding, improvements were made to the user interface by providing clearer guidance.

In the third scenario, focused on access to and initiation of personalized programming challenge generation, all participants successfully accessed and initiated personalized challenge generation. The average interaction time was 27 seconds. The results suggest that this functionality was straightforward to access and initiate in the tested setting, but they do not show that the generated challenge improved learning or that participants completed the programming task itself.

These findings indicate that participants were able to complete the main tested workflows in the evaluated setting. The consistency of results suggests that no major usability barriers were observed during the evaluation.

4.2 Usability Evaluation

The usability of the system was further evaluated using a questionnaire based on the System Usability Scale (SUS), a widely used standardized method for assessing the perceived usability of interactive systems [3]. The SUS questionnaire measures user satisfaction and perceived ease of use, not learning outcomes.

The system achieved a SUS score of **86.54**. According to common SUS interpretation guidelines, scores above 80 are often interpreted as positive perceived usability. In this small-sample evaluation, the score suggests that the system was positively perceived by the participating students from the perspective of user interaction.

The results also indicate a willingness among users to use the system regularly. As shown in Fig. 3, most participants expressed a positive attitude towards regular use of the system. The majority of responses fall into the highest agreement category, while the remaining responses are also positive.

This distribution of responses suggests that users perceived the system as useful and easy to use in the context of the tested scenarios. The absence of negative responses is consistent with the usability findings, although longer-term use would be needed to assess sustained adoption.

4.3 Summary of Results

Overall, the results indicate that the implemented prototype was functional and usable in the evaluated scenarios. Participants could connect to the version control system, generate tests, and access personalized challenge generation. Identified issues were mainly related to usability aspects, especially guidance for access-token creation. The evaluation does not provide evidence that the system improves students' programming performance, retention, or learning gains.

5 Discussion

The results should be interpreted as an initial prototype evaluation. The observed task completion rates, completion times, and SUS score indicate that the main workflows were accessible to the 11 participating students. These results are useful for assessing whether the system can be operated by students, but they are not sufficient for drawing conclusions about educational effectiveness.

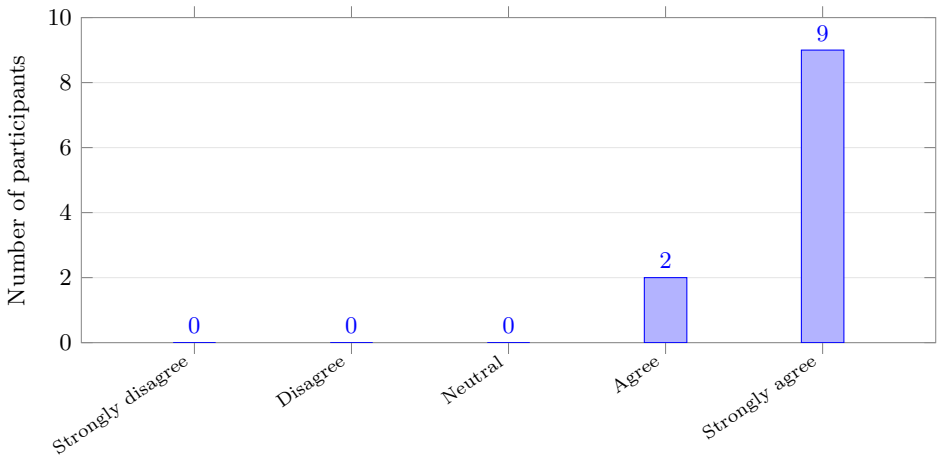


Figure 3 Distribution of responses regarding the willingness to use the system regularly.

The repository-based workflow provides a practical way to connect programming coursework with personalized practice. By allowing students to choose repositories and by analyzing the current state of those repositories, the system can generate an initial task that reflects aspects of the student's own code. This is a technically feasible basis for personalization, but further work is needed before it can be treated as a validated adaptive learning mechanism.

The main value of the prototype lies in making repository data actionable for formative programming practice, rather than using it only for assessment or retrospective analytics.

The use of LLMs also introduces risks. LLM-generated analysis and feedback may contain inaccuracies, especially for incomplete code, ambiguous solutions, or cases requiring execution-based verification. Generated challenges may be malformed, pedagogically weak, too similar to previous tasks, or insufficiently aligned with a course objective. The current system can enforce format-level constraints and was manually inspected by one author during development, but it has not yet undergone systematic expert evaluation with multiple evaluators.

5.1 Threats to Validity

The evaluation has several limitations. First, the sample size was small: only 11 students participated, which limits generalizability. Second, the evaluation was conducted in one university context, the Technical University of Košice, and the results may not transfer directly to other institutions, courses, programming languages, or repository practices. Third, testing was short-term and focused on predefined workflows. Fourth, there was no control group and no longitudinal follow-up. Fifth, the evaluation did not directly measure learning gains, retention, or improvement in programming performance. Sixth, the system has not yet been deployed at larger scale, and there are currently no deployment data showing improvement in students' programming performance, retention, or learning gains. Finally, detailed information about participants' prior programming experience and Git or version-control experience was not collected, which limits interpretation of the task-completion times.

There are also technical limitations. The current version does not define a formally validated weighting scheme for code-quality indicators and does not define empirically validated thresholds for weak, medium, or strong performance levels. Repository analysis

uses the current state of selected repositories rather than commit history. Repository preprocessing and context-limit handling are not yet robustly validated, so large repositories or repositories with many generated and non-source files may require manual or heuristic reduction before analysis. The system does not yet include robust semantic duplicate detection for generated programming challenges. Content quality control is limited to format-level checks, informal manual inspection during development, and ordinary testing of system workflows.

5.2 Future Work

Future work should focus on larger-scale deployment and controlled evaluation of learning outcomes. A stronger study design should include a control group, longitudinal follow-up, direct measurement of programming performance, and analysis of whether personalized challenges lead to measurable learning gains.

Further technical work is also needed. Priorities include systematic expert validation of generated content and feedback, duplicate detection for generated challenges, refinement of the operational student profile into a better supported student model, empirically grounded metric weighting, support for repository history and commit-level analysis, and stronger safeguards for AI-generated evaluation. Additional work should also examine how teachers can review, approve, or adjust generated tasks before students use them in graded or high-stakes contexts.

6 Conclusion

This paper presented the design and implementation of a web-based prototype for personalized programming practice. The concrete technical contribution is the integration of repository selection, LLM-based repository-level analysis, personalized challenge generation, automated feedback, and activity history into one operational workflow. The system integrates with a university version control system, lets students select repositories for analysis, applies LLM-based assessment of repository-level code-quality indicators, generates personalized programming challenges, and returns automated scores with textual feedback. It also supports parameterized test generation and stores student activity, test history, challenge history, scores, and generated feedback.

The prototype demonstrates the technical feasibility and initial usability of integrating repository-based code analysis with AI-generated personalized programming tasks. The evaluation with 11 students showed that participants could complete the main workflows and reported positive perceived usability, reflected in a SUS score of 86.54. These results should be interpreted as usability and functionality findings only.

The current work does not show that the system improves programming ability, retention, or learning outcomes. Its educational impact remains future work. Before such claims can be made, the system requires larger-scale deployment, controlled learning-effect evaluation, stronger validation of generated content and feedback, and improved safeguards around LLM-based analysis and assessment. Nevertheless, the prototype provides a concrete implementation path for connecting programming-course repositories with generated practice tasks and feedback.

References

1 Kirsti M. Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005. doi:10.1080/08993400500150747.

2 Norbert Baláž, Jaroslav Porubän, Marek Horváth, and Tomáš Kormaník. Using chatgpt during implementation of programs in education. In *5th International Computer Programming Education Conference (ICPEC 2024)*, pages 18–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/OASIcs.ICPEC.2024.18.

3 John Brooke. Sus: A quick and dirty usability scale. In Patrick W. Jordan, Bruce Thomas, Bernard A. Weerdmeester, and Ian L. McClelland, editors, *Usability Evaluation in Industry*, pages 189–194. Taylor and Francis, 1996.

4 Peter Brusilovsky and Christoph Peylo. Adaptive and intelligent web-based educational systems. *International Journal of Artificial Intelligence in Education*, 13(2–4):159–172, 2003. doi:10.3233/IRG-2003-13(2-4)02.

5 Marilza Antunes de Lemos and Roseli de Deus Lopes. Challenges of programming apprentice. In *Proceeding of the International Conference on e-Education, Entertainment and e-Management*, pages 320–325. IEEE, 2011. doi:10.1109/iceeem.2011.6137816.

6 Paul Denny, Viraj Kumar, and Nasser Giacaman. Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 1136–1142. Association for Computing Machinery, 2023. doi:10.1145/3545945.3569823.

7 Salvador Espana-Boquera, David Guerrero-Lopez, Alvaro Hermida-Perez, Josep Silva, and Jose V. Benlloch-Dualde. Analyzing the learning process (in programming) by using data collected from an online ide. In *2017 16th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–4. IEEE, 2017. doi:10.1109/ithet.2017.8067822.

8 Marek Horváth, Tomáš Kormaník, and Jaroslav Porubän. Adaptation of automated assessment system for large programming courses. In *5th International Computer Programming Education Conference (ICPEC 2024)*, pages 4–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/OASIcs.ICPEC.2024.4.

9 Marek Horváth and Emília Pietriková. An experimental comparison of three code similarity tools on over 1,000 student projects. In *2024 IEEE 22nd World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 000423–000428. IEEE, 2024.

10 Marek Horváth, Emília Pietriková, and Filip Gurbál'. Personalized learning analytics through static code analysis in computer science education. *Acta Informatica Pragensia*, 2026(1):54–71, 2026.

11 Marek Horvath, Lukas Tomascik, Nikola Geciova, Norbert Adam, and Emilia Pietrikova. Benchmarking ai models for grading cs assignments across multiple domains. *Acta Polytechnica Hungarica*, 23(5):207–226, 2026.

12 Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Á. Rubio, Judy Sheard, Barbara Skupas, Jaime Spacco, Claudia Szabo, and Daniel Toll. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE Conference on Working Group Reports*, pages 41–63. Association for Computing Machinery, 2015. doi:10.1145/2858796.2858798.

13 Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. Studying the effect of ai code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–23. Association for Computing Machinery, 2023. doi:10.1145/3544548.3580919.

14 Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education*, 19(1):3:1–3:43, 2018. doi:10.1145/3231711.

15 Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. Association for Computing Machinery, 2023. doi:10.1145/3631802.3631830.

16 Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J. Malan. Teaching cs50 with ai: Leveraging generative artificial intelligence in computer science education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2024, pages 750–756, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3626252.3630938.

17 Stephen Nutbrown and Colin Higgins. Static analysis of programming exercises: Fairness, usefulness and a method for application. *Computer Science Education*, 26(2–3):104–128, 2016. doi:10.1080/08993408.2016.1179865.

18 Ivica Pesovski, Daniela Vorkel, and Vladimir Trajkovik. Ai-powered education: Rethinking the way programming is taught using ai tools and reversed bloom's taxonomy. In *2024 21st International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–6. IEEE, 2024. doi:10.1109/ithet61869.2024.10837604.

19 Igor Skoric, Boris Pein, and Tihomir Orehovacki. Selecting the most appropriate web ide for learning programming using ahp. In *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 877–882. IEEE, 2016. doi:10.1109/mipro.2016.7522263.

20 Annapurna Vadaparty, Daniel Zingaro, David H. Smith IV, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. Cs1-llm: Integrating llms into cs1 instruction. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2024, pages 297–303, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3649217.3653584.

21 Kurt VanLehn. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221, 2011. doi:10.1080/00461520.2011.611369.

22 Murad Zeer, Rawan Wael Siaj, Jiries Abu Ghannam, and Mohammad Kanan. Ethics of artificial intelligence in university education. In *2023 2nd International Engineering Conference on Electrical, Energy, and Artificial Intelligence (EICEEAI)*, pages 1–4. IEEE, 2023. doi:10.1109/eiceeai60672.2023.10590285.

23 He Zhi-guo and Zhou Chao-xuan. The reformation of teaching methods and curriculum system for software courses. In *2012 International Symposium on Information Technologies in Medicine and Education*, pages 1041–1045. IEEE, 2012. doi:10.1109/itime.2012.6291479.

Bridging Game Development and Virtual Reality Education Through Blender-Based Tutorials

Lenka Bubeňková
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia

Ľubomír Urbančík
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia

Emília Pietriková
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia

Marek Horváth
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia

Abstract

The integration of 3D modeling tools into STEM education can provide opportunities for creative digital production, technical skill development, and hands-on interaction with course content. This study reports an educational evaluation of Blender-based tutorials implemented in two university courses: Game Development (GD) and Systems of Virtual Reality (SVR). Over four semesters, 420 anonymous questionnaire responses were collected from GD and SVR course offerings with a total enrollment of 532 students. The GD course enrolled 428 students and yielded 342 questionnaire responses, while the SVR course enrolled 104 students and yielded 78 questionnaire responses. This corresponds to questionnaire coverage of 79.9% in GD, 75.0% in SVR, and 78.9% overall. Because valid response counts differed across questionnaire items, the results are reported using item-level valid response counts and interpreted descriptively. Across valid questionnaire responses, many respondents, especially in SVR, reported moderate-to-high confidence in using Blender after the tutorials and self-reported understanding of core modeling concepts. Some GD teams even applied self-created or modified Blender assets in game jam submissions, where Blender use was not required. These findings suggest that Blender-based tutorials can be meaningfully integrated into GD and SVR courses as an introductory tool for 3D modeling workflows. The study provides descriptive evidence of student reception and course-level application, but does not establish causal effects on modeling competence or learning outcomes.

2012 ACM Subject Classification Applied computing → Interactive learning environments; Applied computing → Computer-assisted instruction; Computing methodologies → Virtual reality; Computing methodologies → Graphics systems and interfaces

Keywords and phrases Blender, game development, virtual reality, Unity, education

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.10

Funding This work was supported by the Slovak Research and Development Agency under the Contract no. APVV-23-0408.

1 Introduction

Three-dimensional (3D) content creation has become increasingly relevant in software engineering, game development, simulation, and immersive education. As industries continue adopting virtual and interactive environments, students require not only programming skills but also familiarity with digital asset creation tools. Today, 3D modeling underpins industries

as diverse as entertainment, architecture, engineering, and education, as described by Lin et al. [13]. It also plays a key role in modern game development [12], which is being taught in university courses [2]. Engagement with the modeling field can broaden students' perspectives and enhance their creativity; some may even discover a genuine interest in it, which could benefit them in the future as they decide on their careers. Lin, Shih, and Chen [14] revealed that immersive virtual reality-based 3D modeling significantly enhances creative cognitive processes. This includes improvements in imagination, curiosity, and the inclination to take on challenges.

Currently, a diverse range of modeling software is available worldwide, including both specialized and more versatile options. According to Murodillayevich et al. [17], Soni et al. [21], and Mavinkurve and Verma [16], Blender has established itself as a popular choice among individuals who begin the journey in 3D modeling, despite the presence of a steep learning curve and a complex user interface. Blender is a free and open-source 3DCG tool that has gained widespread popularity in the field due to its powerful modeling capabilities and compatibility with game engines like Unity and Godot, as described by Grey and Balasubramanian [5, 1], which illustrates how Blender can be used effectively in combination with these engines to develop visually rich, interactive 3D games. Another example is found in the work of Hidayat et al. [7], where Blender was used alongside Unity to create a horror game. The project demonstrates Blender's use case in game development and its viability as a game design tool.

The rise of virtual reality, augmented reality, and real-time graphics has made 3D modeling more viable than ever, even for software engineers. This is demonstrated in the work of Tanmalaporn et al. [22], which has combined Blender assets with Unity and Meta Quest 2 to create a VR cognitive stimulation game for the elderly with mild cognitive impairment. An additional study by Sebiraj et al. [19] explored how 3D content created in Blender was integrated into AR and VR environments to improve user experience and enable more efficient design exploration. Virtual reality can have a great use in education as well as entertainment. As discussed by Kruachottikul et al. [10], where an application with VR support was created using Unity, incorporating assets developed in Blender. The results of this application were positively rated in terms of helping users develop the necessary skills to obtain a driving license.

A similar study by Jain and Soni [8] used Blender to create models of electronic components, which were then used in a Unity application with VR support. Students reported that they liked the application, stating that it was enjoyable and engaging. The study reported that 91% of students found interacting with 3D models intuitive and easy to learn, and 86% described the experience as enjoyable and motivating. In addition, 89% said the application would be a valuable educational tool and that they would gladly use it in their studies. This positive response suggests that immersive VR-based modeling can show greater engagement with course material, enhance the learning experience and support development in a risk-free environment.

Given these findings, this study aims to describe students' self-reported experiences and course-observed outcomes following the integration of Blender tutorials in Game Development (GD) and Systems of Virtual Reality (SVR) courses. The study was guided by the following research questions:

- RQ1: How do GD students use Blender to create or modify assets for their final projects and game jam submissions?
- RQ2: How do students in the GD and SVR courses differ in their reported level of tutorial completion?

- RQ3: How do students report their confidence in using Blender after participating in the tutorial?
- RQ4: How do students report their understanding of core modeling concepts after tutorial participation?

The following sections summarise related work, describe the design of the tutorials and evaluation methods, present the results, and discuss implications for teaching 3D modeling within game-development and VR curricula.

2 Related Work

Within STEM and design education, Hattori et al. [6] proposed a teaching method that combines virtual simulations in Blender with real experiments and 3D printing for early STEM education. Seralidou et al. [20] similarly examined 3D design and 3D printing activities in secondary education from the perspective of educators. These studies show that 3D modeling can be connected to practical and creative learning activities. Nevertheless, their educational contexts differ from the present study because they focus on early STEM or secondary education rather than university students using Blender as an asset-creation tool in game-development and VR-related coursework.

Other studies focus more directly on structured 3D modeling instruction. Fiore [4] presented a basic-to-intermediate course structure for teaching adult learners 3D computer animation. The sequence introduces important areas such as modeling, texturing, lighting, rendering, sculpting, and grease pencil activities. This work is relevant because it shows how essential clear instructional materials and supporting notes are when helping learners navigate complex 3D software. However, it describes a dedicated animation-learning format rather than a shorter Blender intervention embedded within a broader computing course where modeling is only one part of the curriculum.

Kumar [11] demonstrates another educational use of Blender by combining it with Unreal Engine 4 for heritage recreation and history education. This example shows how flexible Blender is as a tool for interdisciplinary digital production. At the same time, the aim of that work is historical reconstruction, whereas the present study focuses on the use of Blender tutorials to support asset creation within Game Development and Systems of Virtual Reality courses.

Lu et al. [15] provide a more curriculum-based example by presenting a 3D modeling course that focused on design thinking using Autodesk Inventor. Their course was structured around progressive stages and practical modeling tasks, and their questionnaire results indicated high levels of attention, relevance, confidence, and satisfaction. This study is particularly relevant because it connects 3D modeling education with instructional design rather than treating modeling only as a technical skill. However, it differs from the present study in two important ways: it used Autodesk Inventor rather than Blender, and it examined a dedicated 3D modeling course rather than the integration of Blender tutorials into GD and SVR courses with different curricular goals, student backgrounds, and assignment requirements.

Taken together, these studies show that 3D modeling has been used in several educational contexts. However, the literature remains fragmented across different learner groups, software tools, and educational aims. Some studies focus on creativity and engagement, others on technical production or application development, and others on student satisfaction or perceived learning. Fewer studies examine how Blender tutorials can be incorporated into existing university computing courses where 3D modeling is not the only learning objective, but instead supports practical tasks such as asset creation, Unity integration, and VR scene development.

This gap is important because students in Game Development and Systems of Virtual Reality courses may need some modeling knowledge to create, modify, export, and integrate 3D assets, even if they are not being trained as professionals. Existing literature supports the educational value of 3D modeling and the technical relevance of Blender in game and VR pipelines, but there is still limited descriptive evidence on how students perceive and complete Blender-based tutorials when they are embedded within these specific course contexts.

3 Methodology

This study used a descriptive post-intervention educational evaluation design. It did not involve a randomized control group or a comparison group that received a different type of instruction. As a result, the findings should be viewed as students' reported experiences and course observations related to the use of Blender tutorials, rather than proof that the tutorials directly caused the outcomes. The study also does not claim that Blender tutorials are more effective than other teaching methods.

The purpose of the evaluation was to document the implementation of Blender-based tutorials in two university course contexts and to describe students' self-reported perceptions, tutorial completion, and selected course-level evidence of asset use.

The evaluated course offerings included 532 enrolled students in total. Of these, 428 students were enrolled in GD and 104 students were enrolled in SVR. The questionnaire dataset contained 420 anonymous responses, consisting of 342 responses from GD and 78 responses from SVR.

The evaluation used two main sources of evidence. First, anonymous student questionnaire responses were used to describe self-reported tutorial completion, prior Blender experience, confidence after the tutorial, and perceived understanding of basic modeling concepts. Second, for the Game Development course, selected course-level project and game jam evidence was used to describe whether students applied self-created Blender assets in practical work. These data sources were analysed descriptively and were not used to infer causal learning gains.

For the Game Development course, a tutorial focused on Blender and its integration with VFX graphs in Unity was created. The primary objective of this tutorial was to familiarize students with the basics of 3D graphics and encourage creative asset creation to prepare them for their final assignment, a team-based game jam in which they design and develop a game together. One goal was to observe whether students would attempt to create their own assets rather than relying only on downloaded assets.

The second tutorial was developed for the Systems of Virtual Reality (SVR) course. Whereas the Game Development (GD) course is offered in the third year of the curriculum, the SVR course is taught in the first year of the master's program, corresponding to the fourth year of the full study sequence. As a result, students enrolled in SVR are expected to have greater prior technical experience because the course is situated later in the curriculum and follows previous graphics-related coursework.

The tutorial was entirely focused on Blender. Since virtual reality applications rely heavily on 3D assets, it is essential for students to understand the fundamentals of modeling rather than only creating simple objects for games. Furthermore, Blender is frequently used alongside VR technologies in educational environments, where users can practice technical skills in applied contexts. Examples include driving simulation environments, as shown by Kruachottikul et al. [10], and engineering education, as reported by Jain and Soni [8].

In this study, the questionnaire item related to students' work during the tutorial is treated as a measure of self-reported tutorial completion. It is not treated as a validated measure of educational engagement. No systematic observation protocol was used to record

behavioral engagement, motivation, question asking, attention, collaboration, or other forms of student activity during the sessions. Any informal impressions from instructors were therefore not used as quantitative evidence.

3.1 Blender in Game Development course

The delivery format of the GD tutorial evolved over multiple iterations. At first, the tutorial was conducted as an online lecture. In the most recent iteration, a consultation-based approach was adopted, in which students were provided with online tutorial materials to complete independently, with optional online consultations available upon request. At the beginning of the tutorial in the earlier iterations, students were informed in advance about installing Blender. However, they were also given time to complete the installation at the beginning of the class. Since the installation process is not as complicated as it is for Unity, there was no live demonstration of the installation itself. Instead, the lecture time was focused on guided modeling activities.

The students then created two variants of snowflakes for later use in the VFX graph: a low-poly version for background particles and a more detailed version for particles that would be rendered near the viewer in the Unity game. This task was designed to stimulate students' imagination and highlight that they are not limited to assets from the Unity Asset Store. As shown in Figure 3, even a simple task such as creating a snowflake can yield visually appealing, distinct outcomes when students are encouraged to experiment. This aligns with research on imagination-based pedagogies by Dellatola, Daradoumis, and Dimitriadis [3], which has shown that such approaches can significantly enhance student engagement.

This tutorial was then enriched in the last iteration with a low-poly fir tree model. The goal was for students to become more familiar with the basics while simultaneously piquing their interest in the software and stimulating imagination by allowing them to stylize the model according to their preferences.

To support independent learning, students had access to a dedicated course website in every iteration, where all tutorial materials and later video guides were available. The platform allowed students to follow step-by-step instructions at their own pace, access reference images, and review previous lessons as needed. Figure 1 shows an example of the website interface, illustrating how the tutorials were organized to support independent access to materials and self-paced task completion.

The tutorial initially included a demonstration of the mirror modifier, which helps reduce the amount of work needed for modeling. This step was primarily intended to show students how modifiers can alter a model. Introducing students to modifiers is important, as some models cannot be created effectively without them; this is also supported by Novayanti, Crisnapati, and Nurtanto Diaz [18], which uses modifiers to create low-poly objects in Blender. However, this component was removed in later iterations, as students found complex modifiers confusing, which could discourage them. In the revised tutorial, students still created the same models, but the process focused on the basics, and only the subdivision surface modifier was introduced due to its ease of application.

After modeling the two snowflakes, the students learned to export their models. This step is crucial for implementation in Unity, as exporting the object in the wrong format could lead to compatibility issues. Following this, the tutorial moved on to Unity, covering the fundamentals of creating visual effects and incorporating the models created.

It is worth noting that students were also provided with supplementary video tutorials on YouTube, which offered step-by-step visual guides accompanied by screencast key overlays showing pressed keys. This feature was intended to help students better understand and memorize software shortcuts. This was implemented in later iterations as a result of frequent student requests.

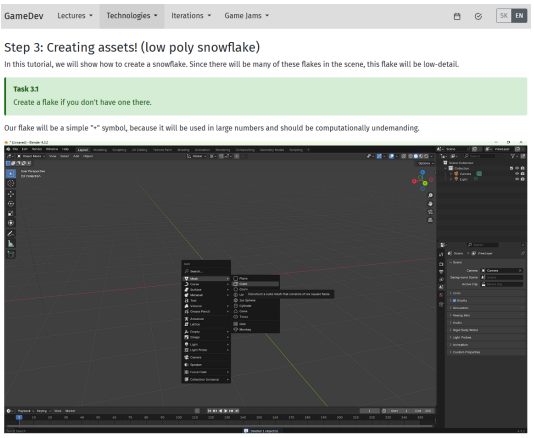


Figure 1 A screenshot of the GD website, which contains step-by-step instructions for creating models.

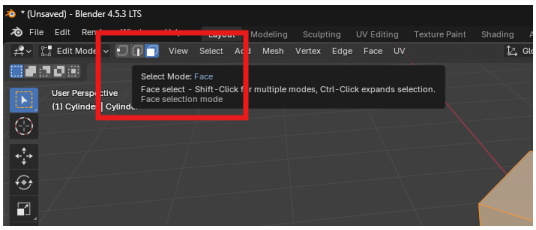


Figure 2 Excerpt from the GD Blender tutorial showing step-by-step visual guidance with highlighted interface elements.

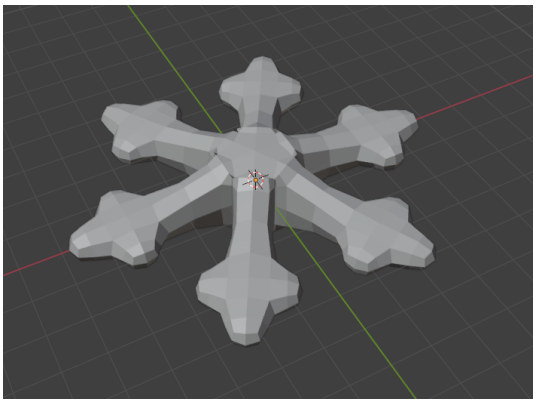


Figure 3 Example of a snowflake model created in Blender during the GD tutorial (low-poly background variant and a more detailed foreground variant).

3.2 Evaluation of Game Development course tutorial

Given the course's strong focus on VFX graphs in Unity, limited time remained for Blender instruction. The tutorial introduced Blender's interface, common geometric primitives, basic operations, use cases for modifiers, and exporting objects.

4 Comparison of Game Development and Virtual Reality Courses

Before discussing the tutorial developed for the SVR course, it is important to note the differences between the two subjects. The most notable difference is their position within the curriculum. The GD course is offered as an optional subject in the third year of the bachelor's program alongside assembler. Because reasons for course selection were not systematically collected, the GD group should be interpreted as a heterogeneous cohort in terms of prior interest, motivation, and experience with game development.

In contrast to GD, the SVR course is part of the first year of the master's program and requires completion of a prerequisite subject about computer graphics that features JavaScript. Some students may also have prior experience from the GD course, but prior motivation and reasons for choosing SVR were not systematically measured.

The teaching approach in both courses reflects these curricular differences. In the Game Development course, students are introduced to the fundamentals of asset creation through simple geometric modeling tasks and the use of basic modifiers. The primary objective is to provide students with a general understanding of 3D modeling concepts and to demonstrate the workflow connection between Blender and Unity.

Students enrolled in the Systems of Virtual Reality (SVR) course already possess a basic understanding of modeling principles in general and are therefore able to progress to more advanced topics. Although many students are already familiar with SketchUp at the time of the tutorial, the course introduces Blender as a more versatile and industry-relevant modeling tool better suited to the objectives of the study program.

4.1 Blender in Systems of Virtual Reality course

The SVR tutorial was conducted in a traditional in-person format in all iterations, allowing for direct interaction between students and the instructor during the lecture.

This tutorial underwent some changes, but the introductory section remained the same. In this part of the tutorial, students were able to familiarize themselves with the software again, as many had prior experience with Blender from a course that used it. At first, a step-by-step guide was provided for creating a basic house shape. Students then created a door, windows, steps, and a chimney, which they later combined into a single object. Once they felt comfortable with their house model, the tutorial explained how to install a texturing add-on. This step was designed to show students how to work with add-ons and to encourage them to download add-ons from the internet, as many can significantly simplify the modeling process in 3D computer graphics software. After applying materials to the structure's surface, they found that the material appeared stretched, so they used a UV unwrap to ensure it aligned more naturally. Students created a doorknob for the door as a bonus task using a subdivision modifier. This was an example of how modifiers can negatively impact a project if used incorrectly. When discussing the house model and the doorknob created afterward, it became evident that the house had fewer polygons than the doorknob, despite being more important.

The revised version of the tutorial focuses more on the difference between procedural modeling in Blender and the more direct modeling approach used in SketchUp. To illustrate this, the tutorial was designed around the use of modifiers, where students were tasked with creating a fence along a curve by duplicating a base segment using an Array modifier in combination with a Curve modifier. Although a similar result can be achieved in SketchUp using similar techniques, Blender's workflow is non-destructive, allowing for a safer and more flexible modeling process. In addition, students were required to model the base fence themselves, which was intended to support their understanding of fundamental modeling principles. As in the previous version, the potential negative impact of improper modifier usage was also discussed, as it represents an important concept for students to understand. This newer version also incorporated video materials, as did the GD course.

4.2 Evaluation of Systems of Virtual Reality tutorial

In this course, students are reminded of the software's basics and learn about its importance in relation to virtual reality (VR). In addition to further improving the software's fundamentals, students are introduced to modifiers, the importance of good topology, and the level of detail appropriate to objects within the scene.

5 Evaluation and data collection

Across four semesters, from winter semester 2024 to summer semester 2026, 420 anonymous questionnaire responses from students in the GD and SVR courses were collected. Across the two courses combined, 532 students were enrolled: 428 in GD and 104 in SVR. Because valid response counts differed across questionnaire items, the results are reported using item-level valid response counts and interpreted descriptively.

5.1 Questionnaire sample and valid response counts

The unit of analysis for the questionnaire results was the individual anonymous questionnaire response. Some questionnaire items were not answered by all respondents, and some items may have differed across tutorial iterations. For this reason, each quantitative result is reported using the number of valid responses available for that item. Missing or incomplete answers were handled using item-level exclusion; therefore, each percentage was calculated using the number of valid responses for the corresponding questionnaire item.

Table 1 Course enrollment and questionnaire response counts by semester.

Semester	Course	Enrolled	Responses	Coverage
Winter semester 2024	GD	10	9	90.0%
Summer semester 2025	GD	150	116	77.3%
Summer semester 2025	SVR	58	41	70.7%
Winter semester 2025	GD	26	14	53.8%
Summer semester 2026	GD	242	203	83.9%
Summer semester 2026	SVR	46	37	80.4%
Total	GD	428	342	79.9%
Total	SVR	104	78	75.0%
Overall	–	532	420	78.9%

Table 2 Valid questionnaire response counts by course and questionnaire item.

Questionnaire item	Course	Total responses (n)	Valid responses (n)
Tutorial completion	GD	342	319
Tutorial completion	SVR	78	78
Prior Blender experience	GD	342	333
Prior Blender experience	SVR	78	78
Confidence after tutorial	GD	342	333
Confidence after tutorial	SVR	78	78
Understanding of modeling concepts	GD	342	333
Understanding of modeling concepts	SVR	78	78

5.2 Course context and project evidence

It is worth noting that students in the GD course were also assigned a project that required the use of Blender to a small degree. This requirement likely increased the incentive to learn and later apply Blender skills. However, in the SVR tutorial, there was no assignment requiring the use of this tool. Therefore, the SVR course provided less course-based incentive to apply Blender outside the tutorial. This point is also highlighted by Khalil and Ebner [9], who found that some students care only about passing the course.

It is also important to consider that the GD tutorial's delivery format in the last iteration was consultation-based, whereas earlier iterations were conducted online. In addition to the questionnaire, students in the GD course were graded on their project performance using a rubric. The rubric consisted of five main segments:

1. Game loop and interactivity
2. Game world and design
3. VFX implementation
4. Use of Blender
5. Use of scripts

The "Game loop and interactivity" was the highest-weighted rubric category, scoring from 0 to 8. Other elements were graded on a scale from 0 to 3. A score of 0 represents the lowest possible grade for a segment, while eight is the maximum score for the game loop and interactivity, and three is the highest for the other parts.

The grading was conducted by four lecturers, who evaluated the projects based on the provided rubric. Formal inter-rater reliability was not calculated; therefore, rubric-based project evidence is interpreted descriptively rather than as a standardized assessment of individual modeling competence.

Because the study did not include a control group or pre/post objective testing, the questionnaire and project data were analysed descriptively. The results provide insight into students' reported perceptions, confidence, and understanding after the tutorials, as well as their observed use of Blender assets in course projects. They do not establish that the tutorials caused improvements in modeling ability, GD skills, or SVR skills.

The GD group included a project that required the use of Blender, followed by a game jam where Blender use was optional. The game jam had a specific theme that, in some cases, required the creation of assets. Although the quality of models varied, some were good-looking and with optimized topology, while others looked impressive but had issues with topology. Despite these differences, both the project and the game jam showcased a range of original and creative elements in the students' worlds.

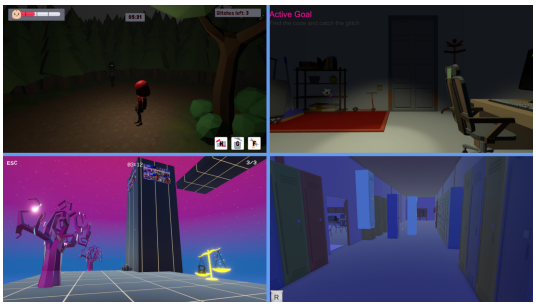


Figure 4 Examples of student games developed during the game jam, including projects that used self-created 3D assets.

In the SVR group, Blender was not used in any assignments. Instead, the information was gathered only through a questionnaire conducted at the end of the class. The goal was to assess how the students would react to this modeling tool because they were using SketchUp to create their models.

5.3 Analysis of research questions

This section analyzes the questionnaire responses and, for the GD course, selected course project evidence. Percentages in the questionnaire results are calculated using the valid responses for the corresponding item, rather than the total number of collected questionnaire responses. The purpose is to describe respondents' perceptions and course-level project outcomes. Detailed findings regarding the research questions are outlined below.

Research question 1. During the game jam, the theme assigned to the students was "a glitch in the system." This theme encouraged the creation or modification of visual assets and provided an opportunity for students to use Blender in their game development workflow. Analysis of submitted games, student submissions, and team reports indicated that some GD teams used Blender to create or modify assets for their projects. These assets were incorporated into student game worlds as original or adapted visual elements. Examples of such games are shown in Figure 4. The quality of the assets varied: some models were simple or blocky, and some showed problems with topology, while others demonstrated more successful attempts at original asset creation. These observations provide descriptive evidence that some GD students used Blender to support asset creation in practical game-development tasks, although the study did not quantify the exact number or proportion of teams that did so.

Research question 2. The questionnaire results in Figures 5 and 6 show respondents' self-reported tutorial completion rather than overall educational engagement. For RQ2, valid tutorial-completion responses were available from 319 GD respondents and 78 SVR respondents. Relative to enrollment, this corresponds to valid tutorial-completion coverage of 74.5% for GD and 75.0% for SVR. Overall, SVR respondents more frequently reported completing most tutorial tasks, while GD respondents were more often concentrated in the category of completing some tutorial tasks. A larger proportion of GD respondents also reported mostly observing compared with SVR respondents. These results suggest that tutorial completion differed between the two course contexts, with SVR respondents reporting higher levels of task completion overall.

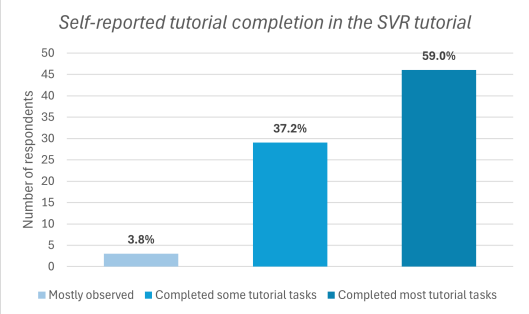


Figure 5 Self-reported tutorial completion during the SVR tutorial, based on valid questionnaire responses (n = 78).

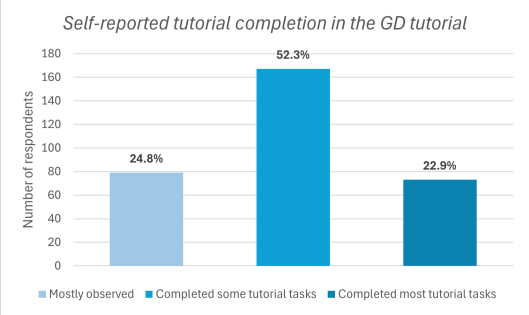


Figure 6 Self-reported tutorial completion during the GD tutorial, based on valid questionnaire responses (n = 319).

The questionnaire results showed differences between the two courses in students' reported tutorial completion. However, this measure should be interpreted as a limited behavioral indicator rather than a comprehensive measure of educational engagement. SVR students reported higher tutorial completion despite Blender not being required in their assignments.

Research question 3. For RQ3, valid responses to the prior-Blender-experience item were available from 333 GD respondents and 78 SVR respondents. The same valid response counts were available for the confidence-after-tutorial item. Relative to enrollment, this corresponds to valid-response coverage of 77.8% for GD and 75.0% for SVR. Prior Blender experience was measured using four categorical options: no experience, have tried it once, know basics, and use it often. Confidence after the tutorial was measured using a five-point self-report scale, where 1 represented the lowest perceived level and 5 represented the highest perceived level. These responses were interpreted as subjective perceptions rather than objective measures of Blender competence. Figure 7 shows self-reported Blender experience before the tutorial, while Figure 8 shows self-reported confidence with Blender after the tutorial.

Overall, questionnaire responses indicated higher self-reported confidence with Blender among SVR respondents, where the largest proportion of SVR respondents selected 4 out of 5. In contrast, GD responses were concentrated around the middle of the scale. This pattern should be interpreted alongside the prior-experience results, where more than half of GD respondents reported no prior Blender experience. These results indicate that many questionnaire respondents, rather than all enrolled students, reported confidence with Blender after the tutorial.

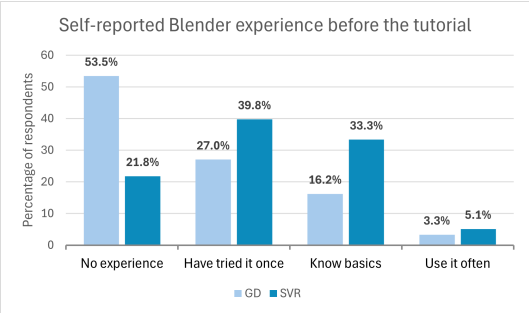


Figure 7 Self-reported Blender experience before the tutorial, based on valid questionnaire responses (GD n = 333; SVR n = 78).

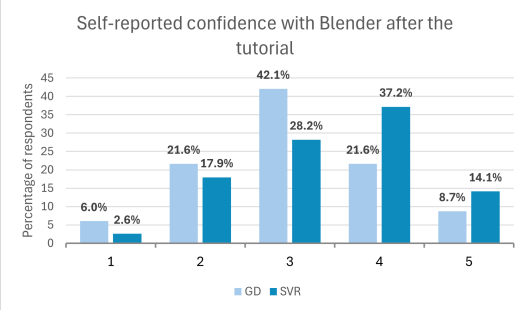


Figure 8 Self-reported confidence with Blender after the tutorial, based on valid questionnaire responses (GD n = 333; SVR n = 78).

Research question 4. For RQ4, valid responses to the modeling-concepts item were available from 333 GD respondents and 78 SVR respondents. Relative to enrollment, this again corresponds to valid-response coverage of 77.8% for GD and 75.0% for SVR. Respondents were asked about their perceived understanding of basic modeling concepts, such as extrusion, edge, face and vertex manipulation, and editing modes. Respondents could also comment on which technique they found most complicated. The additional comments were mixed; however, the self-reported ratings were generally positive. The findings are illustrated in Figure 9. The rating method used a 1–5 scale, identical to that used for the confidence item.

The valid questionnaire responses from both courses showed generally positive self-reported understanding of modeling concepts, with SVR respondents more concentrated in the higher rating categories. These findings address RQ4, but they should not be interpreted as objective evidence of modeling competence.

6 Challenges

The first challenge was understanding the sequence of operations, meaning that students may have known what they wanted to do but were unsure of the order in which to do it. Students most frequently reported difficulties related to workflow sequencing and shortcut memorization.

Another challenge was the steep learning curve associated with Blender, as well as its placement within the overall study program. Many students had primarily coding-oriented experience, and for some of them, this was their first experience with creating virtual objects.

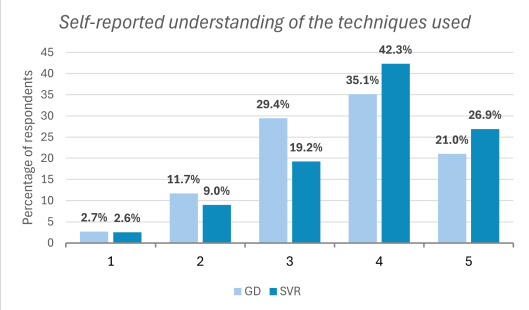


Figure 9 Self-reported understanding of basic Blender modeling concepts after the tutorial, based on valid questionnaire responses (GD n = 333; SVR n = 78).

Some students reported technical difficulties due to compatibility issues between macOS and certain versions of Blender. Specifically, interactions within the Materials tab used to apply colors and textures to 3D models resulted in application crashes for these students.

7 Future work

Future work should investigate longitudinal retention of modeling skills, integration with AI-assisted asset generation, and controlled comparisons between Blender-based tutorials and alternative instructional approaches. Although students rated their understanding positively, some submitted models suggested weaknesses in topology and optimization. In one case, a student-created model contained over two million vertices, indicating the need for stronger instruction in mesh structure, polygon count, and asset optimization.

8 Limitations

This study is limited by its descriptive educational evaluation design. It did not include a randomized control group, a comparison group using an alternative instructional method, or objective pre/post testing of modeling skills. As a result, the findings cannot establish that the Blender tutorials caused improvements in confidence, understanding, GD skills, or SVR skills. The results should instead be interpreted as student-reported perceptions and course-observed outcomes associated with the tutorial implementation. Future work should include controlled comparative designs and objective assessments of modeling performance.

The interpretation is also limited by differences in course context, student background, assignment requirements, and tutorial delivery format between GD and SVR. In addition, the questionnaire item originally related to engagement captured only self-reported tutorial completion. The study did not systematically measure broader forms of educational engagement, such as motivation, attention, persistence, collaboration, question asking, or depth of interaction with the tutorial content.

A further limitation is that the questionnaire responses were anonymous. Therefore, the study can only report enrollment counts, questionnaire response coverage, and item-level valid response counts by course and semester, but it cannot identify which specific enrolled students did or did not respond. As a result, the study cannot determine whether respondents differed from non-respondents in prior experience, motivation, etc. The reported percentages should therefore be interpreted as percentages of valid questionnaire respondents.

9 Conclusion

The integration of Blender into the GD and SVR curriculum was associated with positive student-reported outcomes related to confidence and perceived understanding of modeling concepts. The evaluation included 420 anonymous questionnaire responses from course offerings with 532 enrolled students, corresponding to overall questionnaire coverage of 78.9%. Across both courses, many valid questionnaire respondents reported moderate-to-high confidence in using Blender after the tutorials, especially in SVR, and perceived that they understood fundamental modeling concepts. Questionnaire ratings on confidence, tutorial completion, and perceived understanding were generally positive, with SVR respondents reporting higher tutorial completion and confidence among the valid responses analysed. However, because prior interest and motivation were not systematically measured, this difference should be interpreted cautiously and may reflect several contextual factors, including course level, previous graphics-related coursework, delivery format, or prior exposure to Blender.

In the Game Development course, project submissions showed that students incorporated self-made assets into their games, suggesting that the tutorials were aligned with practical asset-creation activities within the course. In these projects, students had to develop their own game and were also required to create assets for it. This may have provided useful preparation for the game jam event that followed this project. In this event, students were not required to create their own assets, but some of them experimented with self-made objects they had envisioned. This gave students an opportunity to create assets that better matched their intended game concepts.

In the SVR course, students reported positive experiences with Blender despite the absence of a required Blender-based assignment. Even in the absence of a Blender-based assignment, questionnaire responses indicated that many SVR respondents completed substantial parts of the tutorial.

Therefore, the main contribution of this study is the documentation of a feasible Blender-based tutorial implementation and students' reported reception of it in GD and SVR courses. The results provide descriptive evidence of perceived confidence, perceived understanding, reported tutorial completion, and observed asset use in selected GD projects. They do not prove that the tutorials caused improvements in creativity, modeling competence, problem-solving ability, or broader technical skills.

-These findings contribute to the broader literature on 3D modeling, game-development education, and VR-supported learning by showing how Blender can be embedded into existing university computing courses rather than taught only as a separate modeling subject. Previous studies have emphasized the educational value of 3D modeling for creativity, engagement, design thinking, and immersive learning. The present study extends this discussion by documenting how Blender tutorials can support practical asset creation, Unity integration, and VR-related workflows in two related but distinct course contexts.

At the same time, the findings also nuance the positive interpretation of student reception. Although many questionnaire respondents, especially in SVR, reported moderate-to-high confidence and perceived understanding after the tutorials, project evidence also revealed continuing problems with topology, polygon count, and asset optimization. This suggests that short Blender tutorials can provide an accessible entry point into 3D production workflows, but they should not be interpreted as sufficient for developing advanced modeling competence. Further controlled studies are needed before stronger claims can be made about the effects of Blender tutorials on creativity, problem-solving, technical competence, or broader career preparation.

References

1 K. Balasubramanian. *Game Development with Blender and Godot: Leverage the combined power of Blender and Godot for building a point-and-click adventure game*. Packt Publishing, Birmingham, UK, 2023.

2 W. Choi and S. Kim. Curriculum development of edtech class using 3d modeling software for university students in the republic of korea. *Sustainability*, 15(24):16605, 2023.

3 E. Dellatola, T. Daradoumis, and Y. Dimitriadis. Implementing imagination-based pedagogies in a web-based computer supported collaborative language learning writing activity: Orchestration issues. In *Proceedings of the 2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–4, 2018. doi:10.1109/IISA.2018.8633604.

4 F. Fiore. Developing an effective format for introducing 3d computer animation to adult learners. In *Proceedings of the 2024 6th International Workshop on Artificial Intelligence and Education (WAIE)*, pages 167–170, 2024. doi:10.1109/WAIE63876.2024.00037.

5 S. Grey. *Mind-Melding Unity and Blender for 3D Game Development: Unleash the power of Unity and Blender to create amazing games*. Packt Publishing, Birmingham, UK, 2021.

6 T. Hattori, R. Masuda, Y. Moritoh, Y. Imai, Y. Kawakami, and T. Tanaka. Utilization of both free 3dcg software "blender" and 3d printing for early stem education. In *Proceedings of the 2020 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, pages 879–882, 2020. doi:10.1109/TALE48869.2020.9368379.

7 R. Hidayat, M. W. A. Bawono, and M. J. Alibasa. Developing a horror game with game development life cycle method using unity game engine. In *Proceedings of the 2023 International Conference on Data Science and Its Applications (ICoDSA)*, pages 471–476, 2023. doi:10.1109/ICoDSA58501.2023.10277028.

8 M. Jain and S. Soni. Virtual reality in electronics and communication engineering education. In *Proceedings of the 2024 IEEE International Conference on Information Technology, Electronics and Intelligent Communication Systems (ICITEICS)*, pages 1–5, 2024. doi:10.1109/ICITEICS61368.2024.10624857.

9 M. Khalil and M. Ebner. A stem mooc for school children – What does learning analytics tell us? In *Proceedings of the 2015 International Conference on Interactive Collaborative Learning (ICL)*, pages 1217–1221, 2015. doi:10.1109/ICL.2015.7318212.

10 P. Kruachottikul, S. Plengkham, N. Tansuriyawong, N. Chuenpakorn, N. Praditkamjornchai, K. Kovitanggoon, and R. Chancharoen. Car drive simulation in metaverse with vr glasses for testing driver's license. In *Proceedings of the 2023 First International Conference on Advances in Electrical, Electronics and Computational Intelligence (ICAEECI)*, pages 1–5, 2023. doi:10.1109/ICAEECI58247.2023.10370994.

11 A. Kumar. *VR Integrated Heritage Recreation: Using Blender and Unreal Engine 4*. Apress, Berkeley, CA, 2020. doi:10.1007/978-1-4842-6077-7.

12 D.-M. Lee. Blender 3d as a catalyst for indie game development. *International Journal of Contents*, 21(2):67–74, 2025.

13 C.-H. Lin, J. Gao, L. Tang, T. Takikawa, X. Zeng, X. Huang, K. Kreis, S. Fidler, M.-Y. Liu, and T.-Y. Lin. Magic3d: High-resolution text-to-3d content creation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 300–309. IEEE, 2023. doi:10.1109/CVPR52729.2023.00037.

14 M.-H. Lin, Y.-C. Shih, and P.-Y. Chen. The effects of an immersive virtual-reality-based 3d modeling approach on the creativity and problem-solving tendency of elementary school students. *Sustainability*, 16(10):4092, 2024. doi:10.3390/su16104092.

15 P. Lu, Y. Xue, Y. Niu, and H. Zhu. Design and development of 3d modeling course based on design thinking. In *Proceedings of the 2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*, pages 232–233, 2019. doi:10.1109/ICALT.2019.00078.

16 U. Mavinkurve and D. Verma. EG-Easy: Design and testing of blender-based tool to teach projections in engineering drawing. In *Proceedings of the 2016 IEEE Eighth International Conference on Technology for Education (T4E)*, pages 250–251, 2016. doi:10.1109/T4E.2016.064.

17 N. F. Murodillayevich, U. G. Eshpulatovich, and J. O. Pardaboyevich. Integration of virtual reality and 3d modeling use of environments in education. In *Proceedings of the 2019 International Conference on Information Science and Communications Technologies (ICISCT)*, pages 1–6, 2019. doi:10.1109/ICISCT47635.2019.9011899.

18 P. D. Novayanti, P. N. Crisnapati, and R. A. Nurtanto Diaz. 3d low poly asset creation based on balinese local wisdom concept. In *Proceedings of the 2022 4th International Conference on Cybernetics and Intelligent System (ICORIS)*, pages 1–5, 2022. doi:10.1109/ICORIS56080.2022.10031307.

19 N. Sebiraj, K. Advaith, and P. S. Priyadharshini. Enhancing user experience and design exploration using augmented reality (AR) and virtual reality (VR). In *Proceedings of the 2024 5th International Conference on Smart Electronics and Communication (ICOSEC)*, pages 1620–1625, 2024. doi:10.1109/ICOSEC61587.2024.10722546.

20 E. Seralidou, T. Karvounidis, and C. Douligeris. Teachers' evaluation of 3d design and 3d printing activities in secondary education. In *Proceedings of the 2023 8th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, pages 1–7, 2023. doi:10.1109/SEEDA-CECNSM61561.2023.10470906.

21 L. Soni, A. Kaur, and A. Sharma. A review on different versions and interfaces of blender software. In *Proceedings of the 2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 882–887, 2023. doi:10.1109/ICOEI56765.2023.10125672.

22 T. Tanmalaporn, S. Tangwongchai, A. Munir, W. Lunchakorn, C. Tunvirachaisakul, and P. Vanichchanunt. Development of a virtual reality cognitive stimulation game for elderly patients with cognitive impairment. In *Proceedings of the 2024 International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC)*, pages 1–5, 2024. doi:10.1109/ITC-CSCC62988.2024.10628433.

Inverted Grading in Project-Based Learning: A Learning Path Approach for Sustainable Performance Evaluation

Filipe Portela
Algoritmi Centre, University of Minho, Guimarães, Portugal

Abstract

Higher education assessment is traditionally based on cumulative grading models, where students progressively build their final classification over time. However, such approaches often promote risk-averse behaviour and limit continuous improvement.

This paper proposes an inverted grading strategy integrated with learning paths within the TechTeach paradigm. In this model, students are evaluated according to the maximum expected grade of their selected path and assessed on their ability to maintain or recover their performance throughout the course. The strategy combines group and individual evaluation, enabling effective differentiation of student contributions in project-based learning environments.

The approach was validated through a case study conducted in a Web Programming course with 153 students. The results show a dynamic evolution, with an average decrease from 16.31 at the first control point to 15.34 at the second, followed by a recovery to 16.05 in the final evaluation. The overall variation (-0.26, or -1.6%) indicates that outcomes remain close to initial performance despite intermediate fluctuations. Additionally, 62% of students experienced a decrease in grade, 28% improved their performance, and 10% maintained stable results, confirming the model's ability to capture performance dynamics and support recovery.

Student feedback further reinforces the approach's applicability, with more than 80% of responses indicating acceptance of the model.

Overall, the results suggest that inverted grading offers a robust alternative to traditional cumulative models, promoting sustainable performance, accountability, and continuous engagement in higher education.

2012 ACM Subject Classification Information systems → Information retrieval

Keywords and phrases TechTeach, Information Systems, Higher Education, Evaluation Systems, Learning Paths

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.11

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Unit Project Scope UID/00319/2025 – Centro ALGORITMI (ALGORITMI/UM).

1 Introduction

Higher education assessment is traditionally based on incremental grading models, where students progressively build their final classification through successive evaluations. This approach assumes that learning evolves linearly and that students must demonstrate increasing levels of knowledge over time. However, in practice, this model often promotes conservative behaviour, in which students aim to meet minimum requirements rather than pursue excellence. As a result, risk-taking, continuous improvement, and recovery from mistakes are frequently limited.

From a pedagogical perspective, this raises an important question: should students always start from zero and prove their value, or should they be challenged to maintain a predefined level of excellence? In real-world environments, professionals are expected to sustain performance standards rather than gradually achieve them. Therefore, assessment models should reflect this dynamic behaviour and encourage responsibility, consistency, and effort over time.

Based on these observations, this paper introduces an inverted grading strategy within the TechTeach paradigm. Instead of starting from a minimum grade and progressing upward, students begin with a maximum expected grade. They are evaluated based on their ability to maintain or recover that level throughout the course. This approach shifts the focus from acquiring grades to sustaining performance, promoting accountability and continuous engagement.

The proposed strategy is integrated with learning paths and project-based learning, extending TechTeach [15] and previous work on personalised education [17]. It introduces distinct evaluation mechanisms for group and individual performance, ensuring that collaboration does not mask individual effort. Project outcomes define group grades, while individual grades are adjusted through peer evaluation and gamification mechanisms, such as card systems.

To validate the approach, a case study was conducted in the Web Programming course at the University of Minho, involving more than 150 students. The evaluation experiment was applied to group projects. It included two control points and a final presentation, allowing the analysis of grade evolution, recovery behaviour, and alignment between expected and achieved results across different learning paths.

This paper is structured as follows. Section 2 presents the background and related work. Section 3 describes the material and methods used in the study. Section 4 introduces the inverted grading model and its rules. Section 5 presents the case study and implementation details. Finally, Section 6 concludes the paper and outlines future work.

2 Background

This section presents the main topics of the work and reviews related work.

2.1 Assessment Models in Higher Education

Assessment in higher education plays a central role in measuring students' knowledge, skills, and competencies. Traditionally, evaluation models are based on cumulative grading approaches, in which students progressively build their final classification across multiple assessment moments, such as assignments, tests, and projects. These models assume a linear evolution of learning and are widely adopted due to their simplicity and compatibility with standard curricula structures [2].

Several alternative assessment strategies have been proposed to address the limitations of traditional grading models. Mastery learning approaches focus on ensuring that students achieve a predefined level of competence before progressing, allow multiple attempts, and emphasise learning over performance [4]. Similarly, standards-based grading evaluates students according to clearly defined learning outcomes, promoting consistency and transparency in assessment [12]. These approaches shift the focus from point accumulation to knowledge acquisition.

Other models, such as contract grading and specifications grading, introduce a negotiation component in which students define their expected level of achievement and must meet specific criteria to obtain a target grade [14]. These strategies are particularly relevant in project-based learning contexts, as they align expectations with student goals and encourage responsibility in the learning process.

In parallel, formative assessment models emphasise continuous feedback and iterative improvement, allowing students to adjust their performance throughout the learning process [3]. More recently, gamification mechanisms have been integrated into assessment to increase

engagement and provide dynamic feedback through points, levels, and rewards [8]. These approaches have demonstrated positive effects on motivation and participation, especially in large classes.

Despite these advances, most existing models rely on additive or progressive grading logic, in which students start at a minimum value and improve their classification over time. Few approaches consider reversing this perspective by defining a maximum expected performance from the outset and evaluating students based on their ability to sustain or recover that level.

This gap motivates the proposed inverted grading strategy presented in this paper, which extends existing assessment approaches by introducing a dynamic model in which grades may decrease or be recovered based on students' performance over time.

2.2 Learning Paths and Outcomes

Pedagogical innovation is essential to teaching success, and learning paths are a simple way of personalising learning. Personalised learning is an alternative to the "one-size-fits-all" model and refers to approaches that offer multiple learning paths that account for individual differences in learning preferences, goals, abilities, and background knowledge [5, 7]. A learning path, which embodies curriculum design, comprises a sequence of structured learning activities to help students achieve predefined learning objectives [6, 13]. These paths must align with student-centred learning outcomes that describe the measurable skills, knowledge, or values students should demonstrate upon completing a course [19]. The differentiated learning path strategy also aligns with personalised learning approaches, in which students select challenge levels based on their goals and capabilities.

In previous work, a two-path strategy was proposed within the TechTeach paradigm to address the diversity of student goals in higher education [17]. The first path was designed for students aiming to ensure the minimum knowledge required for the subject, while the second was designed for those seeking a deeper, more specialised understanding. The results showed that students valued the ability to choose their trajectory to align with their goals, while maintaining learning outcomes and academic rigour.

2.3 Alternative Grading and Incentive-Based Assessment

Assessment in higher education traditionally relies on cumulative, incremental grading models, in which students progressively accumulate points through successive assignments and evaluations. Although widely adopted, such approaches may unintentionally encourage risk-averse behaviour, leading students to focus on securing minimum acceptable outcomes rather than pursuing excellence, sustained improvement, or recovery after weak performance. This limitation is particularly relevant in project-based learning environments, where performance depends not only on technical achievement but also on consistency, engagement, collaboration, and the capacity to evolve over time.

Assessment design itself has been shown to influence student motivation and behavioural strategies. Docan analysed positive and negative incentive structures in grading, including models in which students either accumulate points progressively or start with a maximum score that decreases with performance, thereby highlighting the motivational implications of grading mechanics [9]. Similarly, MacDermott demonstrated that grading policies can shape strategic student behaviour, reinforcing the idea that assessment systems influence not only outcomes but also students' engagement with learning activities [11].

Building on this perspective, this work explores an alternative assessment logic through an inverted grading model, in which students start from the maximum grade associated with their selected learning path and are evaluated on their ability to maintain or recover performance throughout the project lifecycle. Rather than rewarding only upward accumulation, the model emphasises performance sustainability, accountability, and continuous effort.

This perspective is also aligned with real-world professional contexts, where individuals are typically expected to maintain quality standards over time and respond effectively to performance deviations rather than accumulate symbolic rewards.

Furthermore, the proposed model integrates personalised learning through differentiated learning paths with distinct performance ceilings, allowing students to self-regulate their level of challenge and ambition. Combined with the separation of group and individual evaluation, this ensures that final outcomes reflect both collective project success and individual contribution.

2.4 TechTeach and Gamification

In 2020, Filipe Portela introduced TechTeach, a gamified approach that engages students in the classroom through active methodologies, project-based learning, and digital support tools [15]. Since then, the paradigm has evolved with new strategies, including gamified assessment mechanisms and individual validation models [16]. These approaches have shown that gamification can play an important role not only in motivation but also in performance monitoring, encouraging participation and promoting fairer evaluation processes.

In this context, mechanisms such as peer evaluation, quizzes, and card systems are relevant because they provide continuous signals of student performance and commitment. Recent work has also shown the value of combining multiple sources of evidence in assessment processes, including behavioural indicators, peer feedback, and project participation [18]. Therefore, the inverted grading strategy presented in this paper is framed within TechTeach and extends its logic by introducing a dynamic model in which grades may decrease or be recovered during the project, depending on the work performed.

3 Material and Methods

This work is based on the case study method, which enables the exploration of a specific occurrence or phenomenon [20]. The process of conducting a case study typically unfolds through several stages, including the development of the design, data collection, analysis of findings, and interpretation of results [1]. The specific phases and activities involved in this case study are as follows:

- Design:
  - Understand students' motivation and goals regarding project-based evaluation
  - Study alternative evaluation models to improve students' engagement and performance
  - Design an inverted grading strategy aligned with learning paths
  - Define evaluation moments (control points and final presentation)
- Implementation:
  - Define the rules and evaluation logic of the inverted grading model
  - Adapt the subject evaluation plan to include the proposed strategy
  - Integrate group and individual evaluation mechanisms
  - Apply the strategy to a real project-based learning context

- Analysis:
  - Compare expected grades with achieved results across evaluation moments
  - Analyse grade evolution between control points and final presentation
  - Evaluate differences between learning paths and their outcomes
- Interpretation:
  - Analyse the impact of the inverted grading strategy on student performance
  - Evaluate how grade variation reflects student effort over time
  - Explore the implications of this model in higher education assessment

The application of the case study methodology is justified because it is necessary to evaluate a new assessment strategy in a real-world educational context while maintaining the subject's learning objectives. This methodology has demonstrated its applicability to research in real-world settings [10], as it allows professors to observe student behaviour over time and relate it to performance outcomes across different evaluation moments.

Regarding the tools, Kahoot was used to collect students' feedback at different stages of the subject and to support continuous evaluation through quizzes [16].

4 Inverted Grading Model

This section presents the inverted grading strategy proposed for project-based learning within the TechTeach paradigm. The proposed strategy specifically employs a maximum-reference inverted grading model, in which evaluation begins at the highest achievable grade defined by the selected learning path. Alternative inverted variants could be explored, but they are outside the scope of this work.

4.1 Concept

The inverted grading model is based on a reversed evaluation logic, in which students do not start at a minimum grade and progress upward. Instead, they begin with the maximum expected grade for the selected learning path and are evaluated based on their ability to maintain or improve that level over time.

In this model, assessment is dynamic and iterative. At each evaluation point, students' performance is analysed against the expected quality level. If the work meets the expected standards, the grade is maintained. Otherwise, the grade may decrease. Students may improve their grades in subsequent evaluations by demonstrating higher performance and effort.

This approach shifts the focus from acquiring grades to sustaining performance, encouraging students to maintain consistent effort throughout the project lifecycle.

4.2 Evaluation Structure

The evaluation process is organised into three main moments:

- Control Point 1 (CP1)
- Control Point 2 (CP2)
- Final Presentation (FA)

At CP1, the evaluation is performed at the group level and defines the effective starting point of the inverted grading trajectory. Although students are conceptually evaluated against the maximum grade of their selected path, CP1 establishes the first real value based on the quality of the work developed.

From CP2 onward, the evaluation is performed individually and may include decimal values. The grade at each time step is determined by the previous one, reflecting the evolution of performance over time.

4.3 Grading Logic

The grading logic follows three main principles:

1. **Maximum Reference**: Each learning path defines the maximum achievable grade (e.g., 14 or 20).
2. **Dynamic Adjustment**: Grades may decrease when performance does not meet expectations.
3. **Performance Improvement**: Students may improve their grades in subsequent evaluations.

This approach creates a non-linear grading trajectory, where grades evolve according to performance consistency rather than cumulative accumulation.

4.4 Group and Individual Evaluation

Initially, evaluation is group-based (CP1), where all members share the same grade.

From CP2 onward, individual evaluation is introduced. Each student's grade may differ from the group average, reflecting contributions, effort, and performance, supported by peer evaluation and gamification mechanisms.

4.5 Grade Evolution

Examples of grade trajectories include:

- Stable performance: $20 \rightarrow 20 \rightarrow 20$
- Slight degradation: $20 \rightarrow 19.50 \rightarrow 18.35$
- Improvement: $20 \rightarrow 19.50 \rightarrow 20$
- Low performance leading to failure: $13 \rightarrow 10 \rightarrow \text{R}$

4.6 Integration with Learning Paths

Each learning path defines the maximum achievable grade:

- Traditional path: 14
- Expert path: 20

Although students are conceptually evaluated against this maximum, CP1 defines the effective starting point of the inverted grading trajectory.

5 Case Study

This study was conducted in the Web Programming course at the University of Minho and involved 153 students organised into project groups.

Students selected either the Traditional or Expert learning path, which defines the maximum achievable grade. The inverted grading strategy was applied to project evaluation at three moments: CP1, CP2, and final presentation (FA).

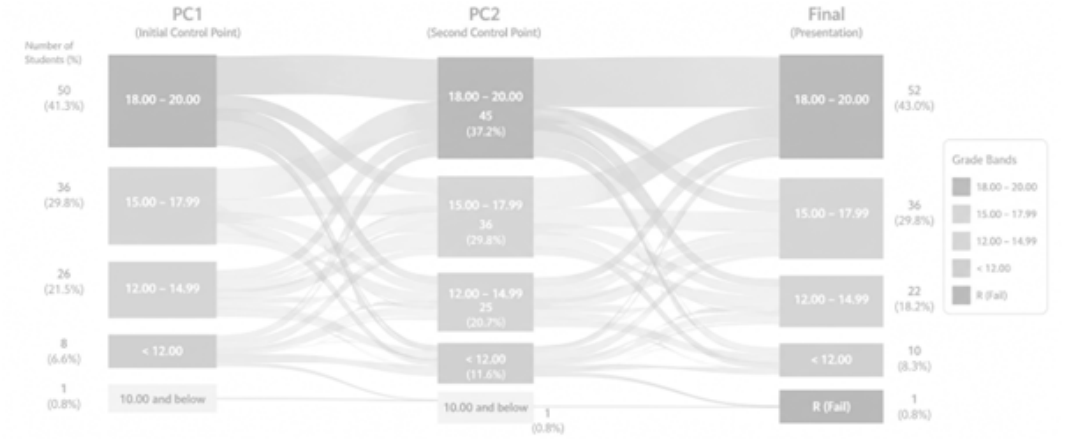


Figure 1 Student grade flow across the three evaluation moments (CP1, CP2, and Final), illustrating transitions between grade bands under the inverted grading strategy.

At CP1, evaluation is performed at the group level and defines the starting point of the grading trajectory. From CP2 onward, evaluation becomes individual, enabling differentiation based on contribution, effort, and performance. Peer evaluation, non-work indicators, and gamification mechanisms support this process.

The collected dataset allows analysis of grade evolution, individual differentiation, and the overall impact of the inverted grading strategy.

5.1 Quantitative Analysis

The results show a consistent pattern of grade evolution throughout the evaluation process. The average grade decreased from 16.31 at CP1 to 15.34 at CP2, then recovered to 16.05 in the final evaluation. It corresponds to an overall variation of -0.26 (-1.6%), indicating that, despite intermediate fluctuations, outcomes remain close to the initial performance level.

Figure 1 illustrates the flow of students across grade bands throughout the three evaluation moments. The visual representation highlights the model's dynamic nature, showing both degradation and recovery paths.

The distribution of trajectories confirms this behaviour. Approximately 62% of students experienced a decrease in their grades at some point, 28% showed improvement between evaluations, and 10% maintained stable performance. These results show that the model effectively captures performance variation and supports both penalisation and recovery.

The median final grade (approximately 16.50) is slightly higher than the average (16.05), indicating a concentration of students in the upper-middle performance range. The standard deviation (approximately 1.9) reflects moderate variability, confirming that the model differentiates performance without introducing excessive dispersion.

Only a small number of students (0.65%) were classified as failing, demonstrating that the model maintains performance standards while still allowing failure in cases of insufficient engagement.

Table 1 summarises the main quantitative indicators obtained from the dataset.

Table 1 Quantitative Results of the Inverted Grading Strategy.

Metric	Value
Number of students	153
Average grade (CP1)	16.31
Average grade (CP2)	15.34
Average final grade	16.05
Average variation (CP1 → Final)	-0.26
Relative variation	-1.6%
Median final grade	≈ 16.50
Highest final grade	20.00
Lowest final grade	0.00 (Fail)
Standard deviation (Final)	≈ 1.9
Students with grade decrease (%)	62%
Students with improvement (%)	28%
Students with stable grades (%)	10%
Students with fail result (%)	0.65%

5.2 Performance Dynamics

The transition from group-based evaluation at CP1 to individual evaluation at CP2 is critical in shaping the results. While initial grades reflect group performance, subsequent divergence captures individual effort and contribution, as observed in both Figure 1 and Table 1.

The observed decrease in grades for most students is expected under the inverted grading logic, in which evaluation starts from a maximum reference point. Maintaining high performance requires consistency, and deviations result in grade reductions. However, the recovery observed in 28% of students confirms that the model supports improvement and rewards sustained effort.

Despite these fluctuations, the limited overall variation (-0.26) indicates that the model stabilises performance over time, with students converging toward a consistent final level.

5.3 Interpretation of Results

From an analytical perspective, the results demonstrate that the inverted grading strategy introduces a controlled level of variability while preserving overall performance stability. This balance ensures both fairness and sensitivity in evaluation.

The model effectively differentiates individual contributions, promotes continuous engagement, and allows recovery from performance drops. At the same time, it avoids grade inflation, as evidenced by the presence of fail cases and the controlled distribution of final grades.

Overall, the results confirm that the inverted grading approach provides a realistic representation of student performance, aligning evaluation with effort, consistency, and progression throughout the project lifecycle.

5.4 Student Evaluation of the Model

To complement the quantitative analysis, students evaluated both the learning-path mechanism and the inverted grading strategy via anonymous polls.

The learning path mechanism received a very positive evaluation, with most responses concentrated at the highest values. A total of 32 students assigned the maximum score, and 35 rated it as excellent, whereas only 3 rated it below 3. This distribution indicates strong acceptance and confirms the relevance of differentiated learning paths.

The inverted grading mechanism was also positively evaluated. In grades 1-5, a total of 34 students explicitly approved the model (5), while 42 agreed with minor adjustments (3), and only 7 expressed disagreement (1 or 2). Overall, more than 80% of responses indicate acceptance or strong acceptance.

These results reinforce the perception that the model is fair, transparent, and aligned with student effort, supporting its applicability in real educational contexts.

5.5 Discussion

The results demonstrate that the inverted grading strategy effectively shifts evaluation from cumulative accumulation to performance sustainability. Rather than rewarding isolated improvements, the model evaluates students based on their ability to maintain a consistent level of performance over time. This analysis is supported by the limited overall variation between the first and final evaluation moments (-0.26, corresponding to -1.6%), indicating that outcomes remain close to the initial performance despite intermediate fluctuations.

The observed decrease in grade for the majority of students (62%) is inherent to the model rather than a limitation. Since evaluation starts from a maximum reference point, any deviation from the expected level is reflected as a reduction in the score. This behaviour reflects a realistic performance scenario in which maintaining high standards is inherently challenging. At the same time, the presence of improvement cases (28%) demonstrates that the model supports recovery and rewards sustained effort, allowing students to regain performance after temporary drops.

The differentiation observed after CP2 confirms the effectiveness of separating group and individual evaluation. While CP1 reflects group performance, subsequent divergence captures individual contribution. It is reinforced by the observed variability (standard deviation of approximately 1.9), indicating that the model is sensitive to individual differences without exhibiting excessive dispersion.

Furthermore, the distribution of final grades, with a median of approximately 16.50 and an average of 16.05, suggests a concentration of students in the upper-middle performance range. It indicates that the model maintains overall performance stability while still allowing differentiation. The presence of fail cases (0.65%) further confirms that the model avoids grade inflation and preserves minimum performance standards.

From a pedagogical perspective, the combination of quantitative results and student feedback reinforces the approach's validity. With more than 80% of students accepting the model, the results indicate that the strategy is not only effective but also perceived as fair, transparent, and aligned with individual effort.

Overall, the results support the use of inverted grading as a promising alternative to traditional evaluation models. The strategy promotes continuous engagement, enables performance recovery, and ensures meaningful differentiation, making it particularly suitable for project-based learning environments where sustained effort and individual contribution are critical.

6 Conclusion

This paper introduced an inverted grading strategy integrated with learning paths and project-based learning within the TechTeach paradigm. The proposed model shifts the traditional evaluation perspective from cumulative grade acquisition to sustained performance assessment, in which students are evaluated on their ability to maintain or recover expected performance levels over time. This approach aims to reflect better performance-oriented contexts, where continuity, accountability, and consistent delivery are often valued over isolated achievements.

The case study, conducted with 153 students in a Web Programming course, demonstrated that the model supports a dynamic and structured evaluation process. Quantitative results showed that average grades decreased from 16.31 at the first control point to 15.34 at the second, then recovered to 16.05 in the final evaluation. The limited overall variation (-0.26, corresponding to -1.6%) suggests that, despite intermediate fluctuations, outcomes remain relatively stable.

Further analysis revealed that 62% of students experienced a temporary decrease in performance, 28% demonstrated measurable recovery between evaluation moments, and 10% maintained stable trajectories throughout the process. These findings indicate that the proposed strategy effectively captures performance dynamics over time, supporting both penalisation and recovery mechanisms rather than relying exclusively on cumulative accumulation.

The transition from group-based to individual evaluation proved particularly relevant in differentiating student contributions. Students in the same project group achieved distinct outcomes, as evidenced by peer evaluations and gamification, thereby reinforcing fairness and accountability. The observed variability (standard deviation of approximately 1.9) confirms that the model differentiates performance without introducing excessive dispersion. Additionally, the median final grade (approximately 16.50) and the low fail rate (0.65%) indicate that the model preserves performance stability while maintaining minimum academic standards.

From a pedagogical perspective, students' perceptions further reinforce the approach's applicability, with more than 80% of respondents expressing acceptance of the model. It suggests that the strategy is generally perceived as fair, transparent, and aligned with individual effort.

Nevertheless, several limitations should be acknowledged. First, the absence of a direct control group using a traditional cumulative grading model prevents a rigorous comparative evaluation of effectiveness. As the strategy was applied to the entire cohort, the analysis focuses on internal behavioural dynamics rather than comparative performance outcomes. Second, starting from a maximum reference grade may introduce psychological pressure for some students, particularly those who perceive evaluation as primarily centred on identifying deficiencies rather than recognising achievements. Third, the learning path mechanism may encourage strategic behaviour, where some students select lower-complexity paths to reduce perceived risk, potentially limiting ambition or recovery potential. Finally, the study was conducted in a project-based programming context, which may restrict the generalisability of the findings to other educational domains.

Overall, the results suggest that inverted grading constitutes a promising alternative to traditional cumulative assessment models, particularly in project-based learning environments where sustained effort, accountability, and individual contribution are critical. Future work should focus on controlled comparative studies with traditional grading models, deeper analysis of psychological and motivational impacts, refinement of evaluation indicators, and

validation across different disciplines and educational settings.

References

1 Pamela Baxter and Susan Jack. Qualitative case study methodology: Study design and implementation for novice researchers. *The qualitative report*, 13(4):544–559, 2010.

2 John Biggs and Catherine Tang. *Teaching for quality learning at university*. McGraw-Hill Education, 2011.

3 Paul Black and Dylan Wiliam. Assessment and classroom learning. *Assessment in Education: Principles, Policy & Practice*, 5(1):7–74, 1998.

4 Benjamin S. Bloom. Learning for mastery. *Evaluation Comment*, 1(2):1–12, 1968.

5 Barbara Bray and Kathleen McClaskey. *A Step-by-Step Guide to Personalize Learning*. Learning & Leading with Technology, 2013.

6 Cindy De Smet, Tammy Schellens, Bram De Wever, Pascale Brandt-Pomares, and Martin Valcke. The design and implementation of learning paths in a learning management system. *Interactive Learning Environments*, 24(6):1076–1096, 2016. doi:10.1080/10494820.2014.951059.

7 Yuli Deng, Dijiang Huang, and Chun-Jen Chung. Thoth lab: A personalized learning framework for cs hands-on projects. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 706–706, 2017.

8 Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: defining gamification. In *Proceedings of the 15th International Academic MindTrek Conference*, pages 9–15, 2011. doi:10.1145/2181037.2181040.

9 Tonc Docan. Positive and negative incentives in the classroom: An analysis of grading systems and student motivation. *Journal of Scholarship of Teaching and Learning*, 6(2):21–40, 2006.

10 Kathleen M Eisenhardt. Building theories from case study research. *The Academy of Management Review*, 14(4):532–550, 1989.

11 Raymond J. MacDermott. The effects of dropping a grade in intermediate macroeconomics. *Journal of Economic Education*, 40(3):270–275, 2009.

12 Robert J. Marzano. *Formative assessment & standards-based grading*. Marzano Research Laboratory, 2010.

13 Amir Hossein Nabizadeh, José Paulo Leal, Hamed N. Rafsanjani, and Rajiv Ratn Shah. Learning path personalization and recommendation methods: A survey of the state-of-the-art. *Expert Systems with Applications*, 159:113596, 2020. doi:10.1016/J.ESWA.2020.113596.

14 Linda B. Nilson. *Specifications grading: Restoring rigor, motivating students, and saving faculty time*. Stylus Publishing, 2015.

15 Filipe Portela. Techteach—an innovative method to increase the students' engagement at classrooms. *Information*, 11(10), 2020. doi:10.3390/info11100483.

16 Filipe Portela. A New Approach to Perform Individual Assessments at Higher Education Using Gamification Systems. In *4th International Computer Programming Education Conference (ICPEC 2023)*, volume 112 of *Open Access Series in Informatics (OASIcs)*, pages 8:1–8:12, 2023. doi:10.4230/OASIcs.ICPEC.2023.8.

17 Filipe Portela. Learning Paths: A New Teaching Strategy with Gamification. In André L. Santos and Maria Pinto-Albuquerque, editors, *5th International Computer Programming Education Conference (ICPEC 2024)*, volume 122 of *Open Access Series in Informatics (OASIcs)*, pages 13:1–13:12, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2024.13.

18 Filipe Portela. A generative artificial intelligence tool to correct programming exercises. In *6th International Computer Programming Education Conference (ICPEC 2025)*, Open Access Series in Informatics (OASIcs), pages 7:1–7:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/OASIcs.ICPEC.2025.7.

19 University of South Carolina. Learning outcomes. https://sc.edu/about/offices_and_divisions/provost/academicpriorities/undergradstudies/carolinacore/faculty-and-staff/learning-outcomes.php, 2026. Accessed: 2026-04-30.

20 Robert K Yin. *Case study research and applications: Design and methods*. SAGE Publications, 2017.

Digital Educational Resources in Programming Education: A Systematic Review of Resource Types and Teaching Challenges

Ana Paula Silva Paulina Bezerra
Graduate Program in Computer Science, State University of Rio Grande do Norte (UERN) and Federal Rural University of the Semi-Arid Region (UFERSA), Brazil

Gabryella Rocha Rodrigues
Federal Institute of Pará (IFPA), Brazil

Ceres Germanna Braga Morais
Department of Informatics, State University of Rio Grande do Norte (UERN), Brazil
Graduate Program in Computer Science, State University of Rio Grande do Norte (UERN) and Federal Rural University of the Semi-Arid Region (UFERSA), Brazil

Abstract

Teaching programming requires the use of different digital educational resources to support the understanding of abstract concepts and the development of problem-solving skills. However, these resources are often dispersed across multiple platforms, which makes their selection, evaluation, and reuse by teachers more difficult. This article presents a Systematic Literature Review on the use of digital educational resources in programming education, focusing on the types of resources employed, as well as their sources and reported difficulties. The review was conducted based on a structured protocol, including the definition of search strings, the selection of databases, and the application of inclusion and exclusion criteria. After the quality assessment was applied, 27 primary studies composed the final analysis set. The results reveal a predominance of practice-oriented resources, such as programming exercises and interactive environments, along with the use of multimedia materials and reusable content. Despite this diversity, significant challenges persist, particularly those related to resource selection, pedagogical integration, material quality, and teachers' workload. These findings highlight the need for more structured approaches to support teachers in the effective use and organization of digital educational resources in programming education.

2012 ACM Subject Classification Applied computing → Education; Information systems → Information retrieval; Computing methodologies → Knowledge representation and reasoning

Keywords and phrases Digital Educational Resources, Programming Education, Resource Types, Teaching Challenges, Resource Selection

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.12

Supplementary Material *Text*: `https://drive.google.com/open?id=1WMa233ZR-LRSCwsrNX7SMtLi9cgrP3-1&usp=drive_fs` [27]

Funding The authors acknowledge the financial support provided by the Graduate Program of the State University of Rio Grande do Norte (UERN) and Federal University of the Semi-Arid Region (UFERSA) for this submission.

1 Introduction

Programming education plays a central role in the training of professionals in Computing, following the growing demands of the labor market and society for digital competencies and problem-solving skills. To support this process, different Digital Educational Resources (DERs) have been used in educational contexts, including interactive platforms, practical exercises, videos, programming environments, simulators, automatic graders [25], educational games, and multimedia materials [2, 7].

However, the use of these resources in programming courses involves specific pedagogical challenges, especially due to the abstract nature of computational concepts and the need to integrate theory and practice [21]. International curricular guidelines, such as those proposed by the Association for Computing Machinery (ACM), highlight that Computing education should be guided by fundamental competencies, such as computational thinking, problem solving, and software development [4]. However, since these guidelines do not specify particular tools, they transfer to teachers the responsibility for selecting and adapting materials.

More recently, the advancement of Artificial Intelligence (AI)-based technologies, such as intelligent tutoring systems, automatic content recommendation, and generative AI tools [31], has expanded the possibilities for personalization and automated support. Despite this rich ecosystem, most materials are dispersed across multiple platforms and lack pedagogical standardization [29, 12]. This fragmentation makes it difficult to locate and reuse resources, increasing teachers' workload due to manual curation processes.

Although there are studies focused on the development of specific tools for programming education, there is a predominance of research centered on isolated technologies, such as interactive environments, automatic assessment systems, practice platforms, or AI-based applications [1, 31]. In contrast, investigations that analyze, in an integrated way, the different types of digital educational resources used in programming education, as well as the challenges faced by teachers in selecting, organizing, and integrating these materials, remain limited.

In this context, this paper presents a Systematic Literature Review (SLR) with the objective of investigating how programming teachers select, use, and produce digital educational resources, as well as mapping the types of resources employed in programming education, considering their formats and sources. In addition, it seeks to identify the main difficulties faced by teachers in selecting, adapting, and using these resources in their educational contexts.

This study contributes by synthesizing and organizing evidence from the literature, offering a structured view of the use of digital educational resources in programming education and the challenges associated with their use.

2 Related Work

2.1 Interactive Environments and Practice-Oriented Resources

Recent studies have explored interactive environments and practice-oriented resources as strategies to support programming education. Hidayat et al. [14] investigated *live coding* platforms associated with gamification to increase engagement in Object-Oriented Programming courses. Similarly, Buffardi and Wang [7] analyzed the integration of videos directly into programming practice interfaces, seeking to reduce the cognitive load caused by switching between instructional materials and coding environments.

Other studies have explored tools focused on the construction of interactive code narratives and the integration between theory and practice. Al-Gahmi et al. [2] highlighted the use of *Jupyter Notebooks* to support content modularization and practical learning. Overall, these studies emphasize resources that promote immediate feedback, active learning, and experimentation during the code development process.

2.2 Artificial Intelligence-Based Technologies in Programming Education

Recent literature also points to a growing trend in the use of Generative Artificial Intelligence (AIGC) technologies in programming education. Abolnejadian et al. [1] investigated the use of ChatGPT as a personalized tutor through *prompt engineering* strategies, highlighting its potential to provide individualized support and automated feedback to students. Similarly, Xie and Chen [31] discussed the application of generative AI in the teaching of the C programming language, using the *Outcome-Based Education* (OBE) model to prioritize the development of computational thinking rather than focusing exclusively on language syntax.

These studies highlight the potential of AI to support learning personalization, content recommendation, and automated assistance during programming practice. However, most research focuses primarily on the effectiveness of these technologies and their impact on student performance, paying less attention to aspects related to the organization, curation, and pedagogical integration of digital educational resources [11, 12].

2.3 Repositories, Open Educational Resources, and Resource Organization

Regarding the authorship and sharing of Open Educational Resources (OER), studies such as Chen and Nunes [8] explore active learning through the creation of videos by students themselves (*Stubents*). Staubitz et al. [29] and Fleenor and Carroll [12], in turn, focus on interoperability and the creation of repositories of automatically graded exercises, aiming to support scalability in large classes.

2.4 Research Gap

Despite these contributions, most studies focus on isolated tools or methodologies. Table 1 summarizes the main approaches and highlights the positioning of this study in relation to the literature.

Although the studies cited contribute to the understanding of specific scenarios, the fragmentation of the literature makes practical adoption by teachers more difficult. Much of the research focuses only on the effectiveness of tools, leaving aside the teacher's perspective in the selection process. Thus, gaps remain regarding:

- The organized classification of the different types of digital educational resources (DERs) available for programming education;
- The sources and repositories that organize these resources;
- The real difficulties faced by teachers in selecting, adapting, and using these materials in their classes.

Thus, this research seeks to address this gap through a Systematic Literature Review (SLR). The study not only maps the types of resources, but also investigates the challenges related to their organization, offering an integrated view to support teachers in decision-making.

Table 1 Summary of related studies.

Study	Type of Resource	Focus / Aspect Investigated
Hidayat et al. [14]	Live Coding Platform	Gamification and motivation in OOP teaching.
Al-Gahmi et al. [2]	Jupyter Notebooks	Code narratives and reduction of cognitive load.
Buffardi and Wang [7]	Integrated Videos	Immediate practice and visual support in coding.
Abolnejadian et al. [1]	Generative AI (ChatGPT)	Adaptive learning through a personalized AI tutor.
Xie and Chen [31]	Generative AI (AIGC)	Competency-based curriculum reform (OBE).
Staubitz et al. [29]	Exercise Repository	Interoperability and automatic grading of assignments.
Chen and Nunes [8]	Videos (Stubents)	Student authorship and peer-based active learning.
Oliveira et al. [11]	OER Platform	Collective intelligence for resource curation.
This study	**Multiple DERs**	**Integrated analysis of resource types, sources, and teaching challenges.**

3 Methodology

This study was conducted as a Systematic Literature Review (SLR), following the guidelines proposed by Kitchenham and Charters [19]. The review protocol is available for consultation[1]. The objective was to identify and analyze the main data sources and materials used in programming education, the types of educational resources employed, and the difficulties reported in teaching practice. In addition, the review sought to understand how these materials are selected, organized, and used.

3.1 Research Questions

The review was guided by the following research questions:

- RQ1: How are digital educational resources used in programming education made available and organized?
- RQ2: What types of digital educational resources, considering their formats and media, are used in programming education?
- RQ3: What difficulties do teachers face in selecting, using, and producing digital educational resources for programming education?

To answer these questions, different dimensions related to the use of digital educational resources in programming education were analyzed, including resource sources, forms of organization, pedagogical practices, types of materials, and teaching challenges.

[1] `https://docs.google.com/document/d/1Ty0Q-h4xdHNUPdFtUM8eBleB7EaiR7ZsWKpJvtV8mgg/edit?tab=t.0`

3.2 Search Strategy

The search was conducted in IEEE Xplore, ACM Digital Library, ScienceDirect, and Web of Science.

The search string was defined by combining terms related to programming education, digital educational resources, and the teaching context, as follows:

```
(teacher OR educator)
AND ("educational resources" OR "learning materials")
AND (repository OR platform)
AND ("teaching programming" OR programming OR "programming education")
```

3.3 Inclusion and Exclusion Criteria

Table 2 presents the criteria adopted for selecting the studies.

Table 2 Inclusion and exclusion criteria adopted in the SLR.

Inclusion Criteria (IC)	Exclusion Criteria (EC)
Available in full text.	Not available in full text or duplicated studies.
Published in English or Portuguese.	Published in languages other than English or Portuguese.
Peer-reviewed studies.	Grey literature.
Published between 2016 and February 2026.	Outside the defined time frame.
Address the use, selection, production, or organization of educational materials and resources for programming education.	Do not address programming education or focus exclusively on student learning without discussing materials or teaching practices.
Describe or analyze sources, platforms, repositories, or digital environments for programming education materials.	Address only programming tools without an educational purpose.
Target audience composed of teachers, instructors, or educators involved in programming education.	Studies focused exclusively on other audiences, without implications for teaching practice.

The inclusion and exclusion criteria were established to ensure the selection of relevant, current studies aligned with the scope of the research. These criteria considered aspects related to publication quality, the educational context addressed, and adherence to the topic of digital educational resources in programming education, with a focus on teaching practice.

3.4 Study Selection Process

The selection process was carried out in four successive stages to ensure methodological rigor and accuracy in narrowing down the results. The detailed selection flow, from initial identification to the final sample, is illustrated in the PRISMA diagram shown in Figure 1.

1. **Initial Screening and Duplicates:** The automated search returned 434 studies. After identifying and removing 3 duplicates, 431 articles remained for analysis.
2. **Title and Abstract Screening:** The 431 studies were screened by title and abstract based on the inclusion and exclusion criteria (IC and EC). At this stage, 186 articles were pre-selected for the eligibility phase.
3. **Eligibility and Admissibility Assessment:** The 186 articles were submitted to a technical analysis focused on their adherence to the research questions (RQs). At this stage, the introduction, methodology, and conclusion sections were read to verify whether

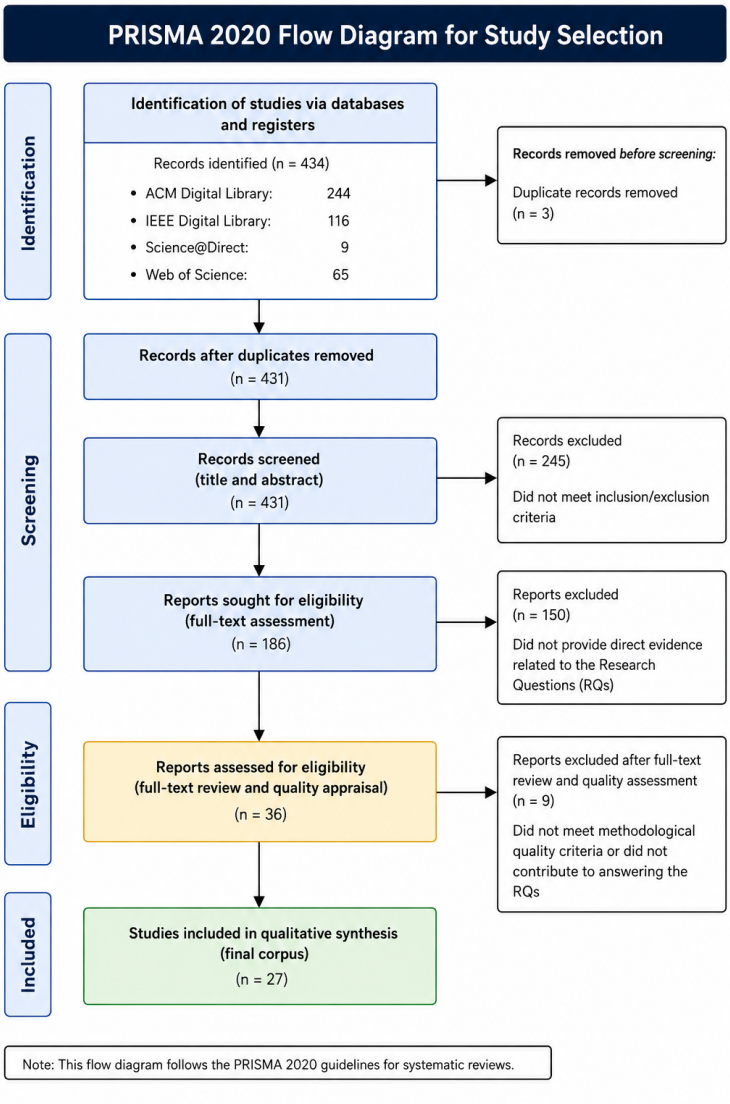


Figure 1 PRISMA flow diagram of the study selection process.

each study provided data on resource types or teaching challenges. This filter led to the exclusion of 150 articles that did not provide direct evidence for the RQs, leaving 36 studies for in-depth analysis.

4. **Full Critical Reading and Synthesis:** The 36 candidate studies were submitted to a full critical reading. After this detailed analysis and the application of the Quality Assessment (QA) protocol, 9 articles were excluded for not reaching the cut-off score. As a result, 27 primary studies composed the final corpus of this Systematic Literature Review.

3.5 Quality Assessment

To ensure that the final synthesis included only studies with methodological rigor and thematic relevance, the 36 articles that reached the full-reading stage were evaluated using a checklist composed of seven criteria.

Each criterion was scored objectively: 1.0 (Yes/Fully meets the criterion), 0.5 (Partially), or 0.0 (No/Not reported).

- QA1 – Does the study clearly present its research objectives?
- QA2 – Does the study describe the educational context in which programming education takes place, such as course type, educational level, or digital environment?
- QA3 – Does the study describe the types of educational materials or resources used for programming education?
- QA4 – Does the study report the sources, platforms, or repositories used to obtain or make the materials available?
- QA5 – Does the study address aspects related to teacher training or difficulties faced by teachers in programming education?
- QA6 – Does the study clearly describe the methodology used, such as case study, survey, experiment, or document analysis?
- QA7 – Does the study present relevant contributions to teaching practice or to the use of digital tools in programming education?

The final score was obtained by summing the points, with a maximum score of 7.0. A cut-off score of 4.0 was established. This final stage resulted in the exclusion of 9 articles that, although related to the topic, did not reach the required level of detail or methodological rigor, consolidating 27 primary studies for the final analysis.

For these studies, data extraction focused on information about types of educational resources, teaching practices, platforms used, and reported difficulties. The data were analyzed through qualitative synthesis and descriptive statistics, allowing the results to be organized according to the research questions defined in this review.

4 Results

This section presents the results obtained from the analysis of the 27 primary studies selected in the review, considering the time frame from 2016 to 2026 adopted in the search strategy. Although the defined period covers ten years, no eligible studies published in 2016 or 2026 were identified after applying the inclusion and exclusion criteria.

Regarding publication sources, there was a predominance of studies indexed in the ACM Digital Library, which accounted for 55.6% of the selected publications, followed by IEEE Xplore, with 37.0%. Web of Science represented 7.4% of the analyzed studies, indicating a higher concentration of research in the areas of Computing, Educational Technologies,

and Digital Learning Environments. As for temporal distribution, publication growth was observed over the years, especially from 2021 onward. The year 2025 recorded the highest volume, with **18.5%** of the sample. The years 2022 and 2024 each accounted for **14.8%**, while 2018, 2021, and 2023 each represented **11.1%**. The years 2017 and 2020 each accounted for **7.4%**, and 2019 had the lowest frequency, with **3.7%** of the selected articles.

4.1 Sources, Organization, and Good Practices in the Use of Digital Educational Resources (RQ1)

The results show that the use of Digital Educational Resources (DERs) in programming education involves different resource sources, technological platforms, and pedagogical strategies. Figure 2 synthesizes these elements through a conceptual model that shows how different resource sources, such as LMSs, MOOCs, programming environments, digital repositories, and artificial intelligence-based tools, are made available through platforms and educational environments that enable their access and use.

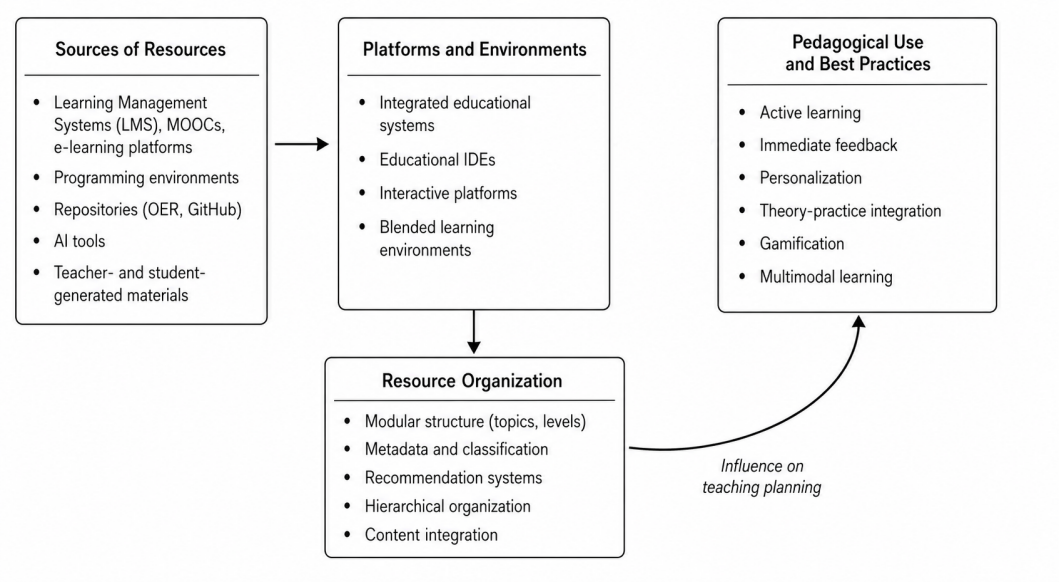


Figure 2 Conceptual model of the digital educational resource ecosystem in programming education, highlighting the relationship between resource sources, educational platforms, organization mechanisms, and their impact on pedagogical use.

The analyzed studies show that these resources are frequently used in integrated platforms, such as educational IDEs [22, 28], interactive environments, and automatic feedback systems [30, 20], supporting practices related to active learning, immediate feedback, teaching personalization, and integration between theory and practice.

Regarding resource sources, there is a predominance of digital teaching platforms, such as Learning Management Systems (LMSs) [1, 17, 30], e-learning environments [18, 3, 8], and MOOCs [6, 26, 13], as well as programming environments, digital repositories, such as OER and GitHub [12, 24, 20], and, more recently, artificial intelligence-based tools. The production of materials by teachers and students also stands out, indicating practices of adaptation and authorship within the educational context.

However, the results indicate that the pedagogical use of these resources depends directly on how materials are organized in these environments. Aspects such as modular structure, hierarchical organization, use of metadata, and recommendation systems [11, 29, 28] were identified as important elements for facilitating search, retrieval, reuse, and pedagogical integration of materials.

As shown in Figure 2, although platforms enable access to resources, the organization of materials has a direct influence on pedagogical practices and on the planning of teaching activities. This result suggests that the challenges faced by teachers are not only related to resource availability, but mainly to how these materials are structured and integrated into educational contexts.

4.2 Types of Digital Educational Resources (RQ2)

The diversity of digital educational resources identified in the selected primary studies ($N = 27$) reveals a scenario strongly oriented toward practice and interactivity in programming education. Since the articles could be classified into multiple categories simultaneously, the accumulated percentages reflect the coexistence of different media and formats in the educational ecosystems analyzed. Figure 3 presents the categorized distribution of the main types of resources.

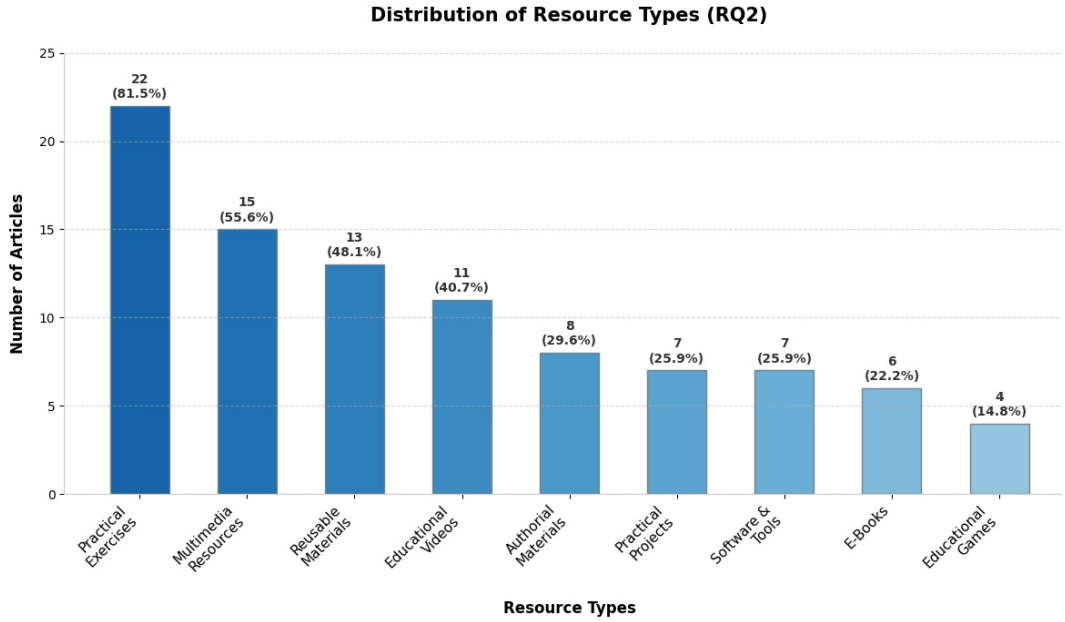


Figure 3 Distribution of resource types identified in the primary studies (N = 27, multiple categories per article permitted).

Among the most recurrent resources, practical exercises stand out, appearing in **81.5%** of the studies, highlighting the centrality of approaches based on problem solving and code implementation [12, 15]. This result reinforces the predominance of pedagogical strategies aligned with active learning, in which students take a more participatory role in the knowledge construction process.

Multimedia resources, which include structured texts, PDFs, images, and audio materials, appear in **55.6%** of the studies. These resources play an important complementary role, acting as theoretical support that precedes or accompanies programming activities [11, 23]. Similarly, reusable materials, such as structured courses, shared source code, and open repositories, were mapped in **48.1%** of the cases, indicating a strong tendency in the academic community to adapt and reuse pre-existing artifacts, especially those hosted on collaborative platforms such as GitHub [6, 28].

Educational videos are present in **40.7%** of the studies and are widely used to support hybrid models, synchronous moments of *live coding*, and asynchronous teaching strategies [8, 26]. Author-created materials, which include artifacts directly produced by teachers, such as *slides* and quizzes, appear in **29.6%** of the studies. This reflects teachers' efforts to personalize and align expository content and reading checks with the specific objectives of their classes [16].

Less frequently, but with similar pedagogical relevance, practical projects and the use of software and simulations were both identified in **25.9%** of the articles [9, 10]. These formats usually require greater student autonomy and are applied in contexts focused on real-world scenarios or the development of complex systems [30]. Finally, *e-books*, with **22.2%**, and educational games, with **14.8%**, complete the distribution, the latter being restricted to specific dynamics of gamification and the introduction of abstract concepts for beginners [2, 18, 14].

Overall, the data point to the consolidation of learning ecosystems based on media hybridization. Although this diversity of formats enriches the student experience, it also transfers to teachers the critical challenge of articulating and integrating these different resources into a coherent and manageable curriculum design.

4.3 Challenges in the Use of Digital Educational Resources (RQ3)

The difficulties reported in the literature show that the use of digital educational resources in programming education involves challenges that go beyond access to technologies, encompassing pedagogical, organizational, and technical aspects. Figure 4 presents the distribution of the main categories of difficulties identified.

Among the most recurrent challenges, those related to teacher training and traditional methods, as well as technological and tool limitations, stand out equally, both appearing in **37.0%** of the studies [2, 8, 20]. This result indicates, on the one hand, the need for continuous training to support the integration of technologies and practice-oriented approaches and, on the other hand, the obstacles imposed by complex platforms.

Next, difficulties associated with the quality and updating of materials were mapped in **33.3%** of the studies [16, 17]. This points to the rapid obsolescence of technological content and to the need for careful evaluation of the sources used. Additionally, the alignment between theory and practice and the adequacy of resources to students' level were reported in **29.6%** of the cases [5, 16], suggesting that resource effectiveness depends critically on contextualization and on the ability to manage the cognitive complexity of abstract concepts.

With the same frequency of **25.9%**, three interconnected bottlenecks emerge: resource organization and standardization, workload and time required for support, and student engagement and learning. This group of results reveals a challenging cycle for teachers: the possible lack of standardized repositories generates fragmentation [29], which substantially increases the time spent structuring classes and providing individual feedback [12], directly affecting the ability to mitigate low student engagement in the face of traditional methodologies.

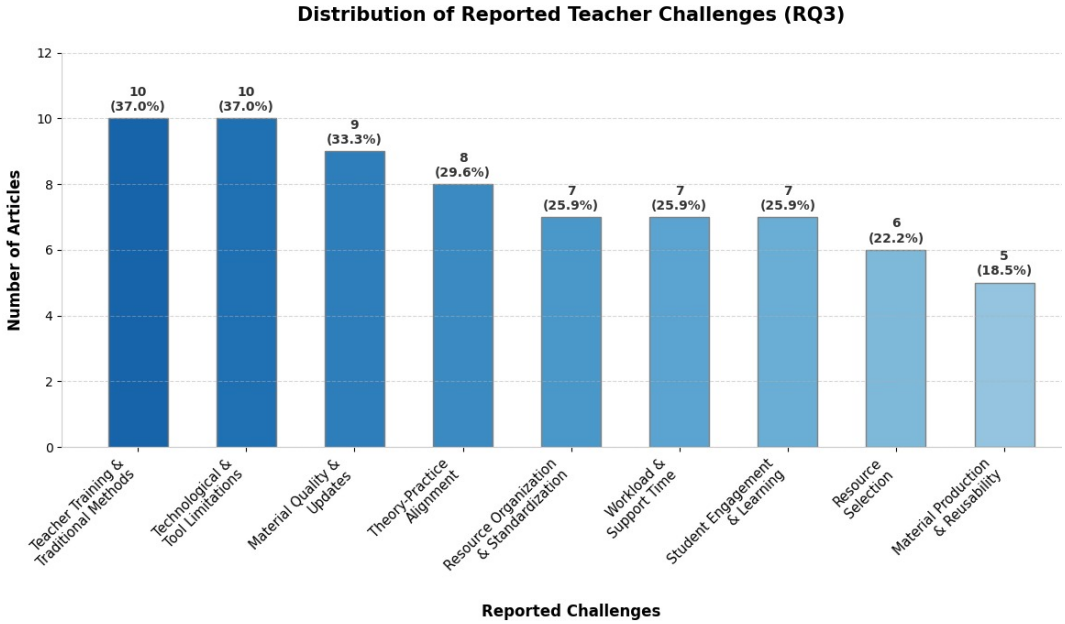


Figure 4 Distribution of difficulties reported in the analyzed studies.

Less frequent, but still limiting, are challenges related to resource selection, identified in **22.2%** of the studies, which expose bureaucracy and difficulties in searching government portals or open repositories [11]. Finally, problems related to the production and reuse of materials appear in **18.5%** of the studies, revealing barriers such as lack of modularity, which hinder the efficient sharing of artifacts among different authors and institutional contexts [28].

Overall, the results indicate that the main obstacles involve the organization and integration of resources into the teaching process.

5 Discussion

The analysis of the studies reveals a scenario marked less by the scarcity of digital educational resources and more by the complexity involved in their pedagogical use. Although there is a wide diversity of materials and platforms, the central challenge lies in selecting, organizing, and integrating these resources into teaching contexts.

This result is consistent with the literature on Digital Educational Resources (DERs), which indicates that their effectiveness depends not only on availability, but also on adequate description, organization, and pedagogical alignment. As discussed in the related work, the use of metadata is essential to support the retrieval and reuse of educational materials [11, 29, 28]. However, the findings of this review indicate that, in practice, significant limitations remain, especially those related to lack of standardization and difficulty in locating suitable resources.

Regarding the sources and forms of material availability (RQ1), the results show that resources are distributed across multiple platforms, such as virtual learning environments, open repositories, development environments, and artificial intelligence-based tools. Although this diversity expands pedagogical possibilities, it also contributes to the fragmentation of the teaching process, requiring teachers to act as integrators of different technologies. This scenario reinforces the need to consolidate structured mechanisms that support the organization and integration of these resources.

In relation to the types of resources used (RQ2), the predominance of practical and interactive activities confirms the emphasis, present in Computing education guidelines, on practice-based learning and problem solving. These guidelines advocate the integration of theory and practice as an essential element for the development of programming competencies. However, the results show that this integration remains a challenge in real teaching contexts, especially when resources are used in an isolated or disconnected manner.

The difficulties identified (RQ3) help explain this scenario. Issues related to teacher training, material quality, and the integration between theory and practice show that the use of digital resources is not only a technological issue, but also a pedagogical and organizational one. The recurrent need for continuous teacher training indicates a gap between the demands of programming education and the support available for teachers.

In addition, aspects such as teachers' workload and tool limitations reinforce that the adoption of digital resources can increase the complexity of teaching when adequate support is not available. This factor is directly related to resource fragmentation and the absence of organization standards, which makes reuse and adaptation more difficult.

In this context, the results of this review reinforce the need for more structured approaches to support the use of digital educational resources in programming education. Initiatives based on ontologies, structured repositories, and recommendation systems appear promising because they can contribute to improving the organization, retrieval, and pedagogical alignment of resources [11, 29, 28].

Thus, more than increasing the number of available materials, it is necessary to invest in strategies that enable their effective integration and use in the teaching and learning process.

6 Study Limitations

Although this review followed a structured protocol for searching, selecting, and analyzing studies, some limitations should be considered. The search was conducted only in specific databases and considered studies published between 2016 and February 2026, written in English or Portuguese. This may have restricted the scope of the results and introduced linguistic and geographical limitations.

In addition, the search strategy prioritized terms related to the pedagogical context of programming education, which may have excluded part of the technical literature on specialized tools used as digital educational resources. The categorization of resources, pedagogical practices, and reported difficulties also involved qualitative interpretation of the analyzed studies, which may introduce some subjectivity into the classification process.

To reduce possible biases, inclusion and exclusion criteria, quality assessment criteria, and data categorization procedures were defined in advance. Despite these limitations, the results provide a relevant overview of the use of digital educational resources in programming education and the challenges associated with teaching practice.

7 Conclusion

This systematic review showed that the main challenge in the use of digital educational resources in programming education is not centered on their availability, but on how these resources are organized and integrated into pedagogical practices.

Although the literature reveals a significant diversity of materials and approaches, their effective use still depends on factors such as teacher training, pedagogical alignment, and the ability to articulate different types of resources. In this context, teachers play a central role not only as users, but also as mediators and integrators of educational technologies.

The findings also indicate that the fragmentation of resources across multiple platforms, combined with the absence of standards for organization and description, makes retrieval and reuse more difficult, limiting their educational potential. This scenario reinforces the need for solutions that go beyond simply making materials available and that include mechanisms to support their structuring and pedagogical use.

In this sense, this systematic review is not limited to synthesizing the literature, but also provides a conceptual and empirical basis for the development of a repository of digital educational resources for programming education. The results obtained guide the definition of requirements for this repository, especially regarding the organization of materials, the use of metadata, and the need for recommendation mechanisms and support for teaching practice.

As its main contribution, this study highlights a gap between the availability of digital resources and their effective pedagogical application, indicating the need for approaches based on semantic organization. The use of ontologies, in this context, emerges as a promising strategy to structure, classify, and retrieve resources more efficiently.

As future work, the development and validation of an ontology-based repository is proposed, with the aim of supporting teachers in the selection and integration of digital educational resources and contributing to the reduction of complexity in programming education.

Thus, more than increasing the number of available resources, it is essential to advance in how these resources are organized and used, in order to enhance their impact on the teaching and learning process.

References

1 Mohammad Abolnejadian, Sharareh Alipour, and Kamyar Taeb. Leveraging chatgpt for adaptive learning through personalized prompt-based instruction: A cs1 education case study. In *Extended abstracts of the CHI conference on human factors in computing systems*, pages 1–8, 2024. doi:10.1145/3613905.3637148.

2 Abdulmalek Al-Gahmi, Yong Zhang, and Hugo Valle. Jupyter in the classroom: An experience report. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*, pages 425–431, 2022. doi:10.1145/3478431.3499379.

3 Rahadyan Fannani Arif, Harits Ar Rosyid, and Utomo Pujianto. Design and implementation of interactive coding with gamification for web programming subject for vocational high school students. In *2019 International Conference on Electrical, Electronics and Information Engineering (ICEEIE)*, volume 6, pages 177–182. IEEE, 2019.

4 Association for Computing Machinery. Computing curricula 2020: Paradigms for global computing education, 2020. Accessed on: Apr. 30, 2026. URL: https://www.acm.org/education/curricula-recommendations.

5 Lorena A Barba. Engineers code: reusable open learning modules for engineering computations. *Computing in Science & Engineering*, 22(4):26–35, 2020. doi:10.1109/MCSE.2020.2976002.

6 Anastasiia Birillo, Mariia Tigina, Zarina Kurbatova, Anna Potriasaeva, Ilya Vlasov, Valerii Ovchinnikov, and Igor Gerasimov. Bridging education and development: Ides as interactive learning platforms. In *Proceedings of the 1st ACM/IEEE Workshop on Integrated Development Environments*, pages 53–58, 2024. doi:10.1145/3643796.3648454.

7 Kevin Buffardi and Richert Wang. Integrating videos with programming practice. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 1*, pages 241–247, 2022. doi:10.1145/3502718.3524778.

8 Yige Chen and Bernardo Pereira Nunes. Stubents: Videos created by and for students, active learning resources in large and diverse computer science classrooms. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024)*, pages 653–659, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3649217.3653569.

9 Hugo Costelha and Carlos Neves. Technical database on robotics-based educational platforms for k-12 students. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 167–172. IEEE, 2018. doi:10.1109/ICARSC.2018.8374178.

10 Josivan Pereira Da Silva, Paulo Henrique Gonçalves Pimentel, Luciano Gonçalves Pimentel, and Ismar Frango Silveira. Pixel python rpg: repurposing an entertainment game to an open educational resource for computer programming fundamentals. In *2021 XVI Latin American Conference on Learning Technologies (LACLO)*, pages 326–333. IEEE, 2021.

11 Marcela Ribeiro de Oliveira, Israel Barreto Sant'Anna, Guilherme Scariot Ramos, Luis Carlos Erpen De Bona, Marcos Alexandre Castilho, Marcos Didonet Del Fabro, and Eduardo Todt. Open educational resources platform based on collective intelligence. In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pages 346–353. IEEE, 2018. doi:10.1109/CIC.2018.00053.

12 Hillary Fleenor and Hyrum D Carroll. Creating an oer collection of automatically scored practice exercises for computer science 1. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 1280–1280, 2020.

13 Elena Harris and Richert Wang. Codewit. us: A platform for diverse perspectives in coding. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*, pages 780–786, 2022.

14 Wahyu Nur Hidayat, Wiji Dwi Prasetyo, Ayu Najmatus Zahiroh, Esa Pramudheva Maydi Syahri, Mochammad Mu'iz Afdloly, and Rahajeng Kartika Sari. Development of gamification and live coding-based programming learning platform to foster learning motivation of vocational students. In *2023 International Conference on Electrical and Information Technology (IEIT)*, pages 299–304. IEEE, 2023.

15 Muntasir Hoq, Atharva Patil, Kamil Akhuseyinoglu, Peter Brusilovsky, and Bita Akram. An automated approach to recommending relevant worked examples for programming problems. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, pages 527–533, 2025. doi:10.1145/3641554.3701951.

16 Wenhui Hou and Xiaohan Yu. Exploring the blended teaching mode of python programming language course based on obe concepts. In *Proceedings of the 2nd Guangdong-Hong Kong-Macao Greater Bay Area Education Digitalization and Computer Science International Conference*, pages 434–439, 2025.

17 Xiaoyan Jian, Feilong Li, and Zhaohong Liu. A study on the application of aigc context in the development of digital resources for online courses. In *Proceedings of the 2024 International Conference on Artificial Intelligence and Teacher Education*, pages 136–141, 2024.

18 Diamantis Kintsakis and Maria Rangoussi. An early introduction to stem education: teaching computer programming principles to 5 th graders through an e-learning platform: a game-based approach. In *2017 IEEE Global Engineering Education Conference (EDUCON)*, pages 17–23. IEEE, 2017. doi:10.1109/EDUCON.2017.7942816.

19 Barbara Kitchenham and Stuart Charters. *Guidelines for performing systematic literature reviews in software engineering*. Citeseer, 2007.

20 Andrew Meads, Yu-Cheng Tu, and Gillian Dobbie. Moving a bootcamp-style computer science programme online: An experience report. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 785–791, 2023. doi:10.1145/3545945.3569814.

21 Ceres Germanna Braga Morais. *Ensino e aprendizagem de programação: estudo de caso no Ensino Superior*. Tese de doutoramento em ciências da educação, especialidade em tecnologia educativa, Universidade do Minho, Braga, Portugal, 2021.

22 Nektarios Moumoutzis, George Boukeas, Vassilis Vassilakis, Nikos Pappas, Chara Xanthaki, Ioannis Maragkoudakis, Antonios Deligiannakis, and Stavros Christodoulakis. Design, implementation and evaluation of a computer science teacher training programme for learning and teaching of python inside and outside school: Establishing and supporting code clubs to learn computer programming by self-contained examples. In *Interactive Mobile Communication, Technologies and Learning*, pages 575–586. Springer, 2017. doi:10.1007/978-3-319-75175-7_56.

23 İbrahim Halil Özdemır, Tarık Kişla, and Serdar Stefano Tito. Mobile programming with kuika: A course design. In *2023 32nd Annual Conference of the European Association for Education in Electrical and Information Engineering (EAEEIE)*, pages 1–6. IEEE, 2023.

24 James Paterson, Joshua Adams, Brian Hainey, and Sajid Nazir. A repository of resources and exemplars for the cloud curriculum. In *Proceedings of the 5th Conference on Computing Education Practice*, pages 9–12, 2021. doi:10.1145/3437914.3437971.

25 Filipe Portela. A Generative Artificial Intelligence Tool to Correct Programming Exercises. In Ricardo Queirós, Mário Pinto, Filipe Portela, and Alberto Simões, editors, *6th International Computer Programming Education Conference (ICPEC 2025)*, volume 133 of *Open Access Series in Informatics (OASIcs)*, pages 7:1–7:16, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2025.7.

26 Xunchao Qiu, Chunyue Zhang, and Libo Zhao. Preliminary study on the online teaching mode of" technical fundamentals of web front-end" based on spoc and wechat. In *Proceedings of the 7th International Conference on Distance Education and Learning*, pages 133–138, 2022. doi:10.1145/3543321.3543343.

27 Ana Paula Silva. SLR Protocol. Text (visited on 2026-07-13). URL: https://drive.google.com/open?id=1WMa233ZR-LRSCwsrNX7SMtLi9cgrP3-1&usp=drive_fs, doi:10.4230/artifacts.27126.

28 Sebastian Speiser. Composition of open educational resources through dynamic linking of modular components. In *Proceedings of the 2024 the 16th International Conference on Education Technology and Computers*, pages 255–260, 2024. doi:10.1145/3702163.3702423.

29 Thomas Staubitz, Ralf Teusner, and Christoph Meinel. Towards a repository for open autogradable programming exercises. In *2017 IEEE 6th International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, pages 66–73. IEEE, 2017. doi:10.1109/TALE.2017.8252306.

30 Yan Watequlis Syaifudin, Siti Rohani, Nobuo Funabiki, and Pramana Yoga Saputra. Blending android programming learning assistance system into online android programming course. In *2021 9th International Conference on Information and Education Technology (ICIET)*, pages 26–33. IEEE, 2021.

31 Nan Xie and Weimin Chen. Research on teaching reform of aigc empowering c language programming course. In *Proceedings of the 2024 2nd International Conference on Information Education and Artificial Intelligence*, pages 176–181, 2024.

The SOB Monitor: Supporting Competency-Based Assessment with Gamification

David Dorvekinger
Department of Computer Science, Middlesex University, London, UK

Michael Heeney
Department of Computer Science, Middlesex University, London, UK

Kelly Androutsopoulos
Department of Computer Science, Middlesex University, London, UK

Abstract

The SOB Monitor is a web-based platform that supports competency-based assessment through Student Observable Behaviours (SOBs). SOBs are defined as measurable units aligned with learning outcomes and are assessed at Threshold, Typical, and Excellent levels. The assessment strategy involves continuous assessment with multiple resubmission opportunities, allowing for real-time monitoring of student progress. It is adopted on several undergraduate Computer Science modules, including first-year Programming, Systems and Architecture, Foundations of Computing, and Design and Development of Applications.

In this paper, we describe assessment with SOBs, the new architecture of the SOB Monitor platform, its implementation and how it supports competency-based assessment in Computer Science undergraduate modules. Gamification elements including leaderboard and rankings have been introduced to enhance student motivation, engagement and self-regulated learning within pass/fail modules.

2012 ACM Subject Classification Applied computing → Interactive learning environments

Keywords and phrases Competency-based education, interactive learning environments, assessment, higher education, computer science education, gamification

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.13

Category Short Paper

1 Introduction

There are a number of challenges when designing an undergraduate first-year Computer Science module. One such challenge is the wide variety of student prior knowledge, particularly in programming [4], where some students have never coded before, while others have for years. This leads to difficulties in pacing and structuring of material to ensure that all students are equally engaged. It can also lead to difficulties in designing assessment that is fair across diverse student backgrounds, as this can influence performance and progression, e.g. assessments that seem reasonable for a beginner may be trivial for an experienced student. Moreover, in programming, a correct program does not mean that the student has a deep understanding of the concepts [13]. A student might produce a working program by trial-and-error, external help or reusing existing code, while another student might submit a partially working program but have a stronger conceptual understanding.

These problems are not unique to just programming modules, but can take different forms in other modules such as Systems and Architecture. A key difficulty is again what counts as understanding. Students may be able to memorise how components such as caches, pipelines or memory hierarchies work and reproduce this in an exam, but still lack a coherent mental model of how those components interact at runtime [12]. With the growing availability

of generative artificial intelligence (GenAI) tools, students are regularly using these for generating solutions, raising further concerns about academic integrity, over-reliance from students, misleading outputs and other topics such as bias and vulnerabilities [6]

We adopt a competency-based assessment approach for alleviating these problems, by prioritising the demonstration of learner competence through measurable knowledge, skills, and professional behaviours rather than progression based on time or content coverage. This approach supports responsible use of GenAI tools. Students can use GenAI tools as part of their learning process, but for assessment are required to articulate and defend their work, ensuring that understanding is assessed. Synthesising perspectives from both conceptual and applied research, this approach is characterized by clearly defined competencies aligned with stakeholder and societal needs, flexible learning pathways that accommodate individual pacing and prior experience, and an emphasis on authentic, performance-based assessment [8].

The SOB Monitor is a web-based platform that supports competency-based assessment of first-year Computer Science modules through Student Observable Behaviours (SOBs) [1], which represent fine-grained decompositions of the learning outcomes. Assessment of SOBs in the context of first-year programming education, along with the lessons learned, is described in [2]. The SOB Monitor enables continuous monitoring of student progression, allowing for early intervention. The first-year modules are not graded, but pass/fail, and maintaining student motivation beyond minimum expectations can be difficult. Some modules include several group-based assessment activities, which complicates the management and marking of SOBs.

Module creation and maintenance can be difficult, particularly when troubleshooting login failures and tracking errors across module instances. As the SOB Monitor has grown in popularity across the institution, supporting 28 modules in Computer Science, Engineering, and Mathematics has further amplified these issues.

This paper addresses these limitations with the re-development and extension of the SOB Monitor to enhance platform security, enable support for multiple modules, and address limitations in the existing project infrastructure. The new architecture is presented which consists of a dedicated web framework (Django) with new code infrastructure that improves the platform's ease of use across multiple modules. Gamification elements have been introduced, specifically leaderboard and rankings, to enhance student motivation and engagement, particularly with more advanced material. Features for supporting group activities with multiple student and SOB marking have also been introduced.

2 Related Work

Competency-based education is a framework for teaching and assessment that shifts the focus from time-based progression to mastery of demonstrable competencies [15]. This approach supports self-paced learning and accommodates variation in student progression. By design, competency-based education is well suited to cohorts with diverse ability levels, as it allows learners to progress according to their individual development rather a fixed pace [8].

A number of widely used Learning Management Systems (LMSs) support competency-based education through assessment, outcome-alignment, and progress-tracking features. Platforms such as Moodle LMS (`https://moodle.com/products/lms/`), Blackboard LMS (`https://www.blackboard.com/`), Canvas LMS (`https://www.instructure.com/canvas`), and OpenOLAT (`https://www.openolat.com/`) include mechanisms for representing competencies, learning outcomes, goals, or related progress indicators. However, these tools are not built around competency-based education as a core pedagogical model, this capability can be added as a plug-in or third party tool in some cases, but it takes away from the coherent feel and adds complexity.

Unlike LMS, Artemis[1] implements competency-based adaptive learning more explicitly and focuses on CS education and programming-heavy education. It supports competency-based education, including competency definitions, competency relations, mastery tracking, and personalized learning paths. The evidence used for competency tracking is generated primarily through student interaction with the platform and the competency is inferred from this generated data. While Artemis supports manual assessment and tutor feedback on submissions, live oral explanations, "show me" demonstrations, and dialogue based tutor-student interactions do not appear to be central evidence sources in the native competency-tracking model [16]. This limits the system's ability to capture forms of understanding best assessed through discussion, observation, or oral defense. As a result, it may not fully capture the competencies that require explanation, judgment, reasoning, or observed performance.

Gamification uses game elements, such as leaderboards and rankings, to increase engagement [7]. It creates an excellent synergy for competency-based learning, as these elements encourage progress and achievement. These game elements give students a richer experience and push them to pursue more than they would otherwise [10]. In some cases these elements can introduce negative effects. For example, the competitive nature of the leaderboard can cause anxiety for some students, as they may feel they are falling behind their peers which in some cases can reduce motivation, instead of enhancing it [9]. This is particularly relevant in scenarios where students are not progressing at the same pace. These effects can be addressed through proper explanation and support from the teaching team. In some situations this anxiety can overshadow the student's understanding of actual deadlines or expectations. In these cases the teaching team plays an important role of giving reassurance, additional context, and encouragement, helping students to stay motivated, and bringing them back to a clearer understanding of their progress.

Gamification elements can be used in some LMS either with built-in features or using plug-ins. For example, in Moodle, features such as ranking, leveling up, badges etc. can be added to enhance engagement and learning [5]. In [5], the authors note that adding too many gamification features could lead to distracting a student.

3 Assessment with Student Observable Behaviours (SOBs)

The assessment strategy is designed to support both constructive alignment [3] and competency-based learning [15]. Learning outcomes are first defined, and the teaching and assessment activities are then aligned with these outcomes. Students participate in relevant, student-centered learning activities, developing their skills through active engagement while tutors act as coaches rather than providing step-by-step instruction [14]. The emphasis is placed on students demonstrating their learning through practical application rather than solely through theoretical understanding. The strategy also incorporates continuous assessment, multiple resubmission opportunities, and real-time monitoring of student progress.

Student Observable Behaviours. (SOBs) [1] describe fine-grained decompositions of the learning outcomes and are categorised into three levels. The Threshold level defines the required SOBs that students must demonstrate to pass the module. Typical and Excellent levels are provided to extend beyond the minimum requirements, thereby avoiding the risk of stunting knowledge acquisition, as it can be an issue in certain competency-based educational settings [17]. This multi-level structure enables competencies to be described more precisely

[1] `https://artemis.tum.de/`

across levels of achievement, thereby mitigating the tendency of competency-based approaches to become overly discrete or fragmented. By incorporating multiple levels of performance, the model captures a broader range of knowledge and skill development. Expectations are specified more clearly at each stage, supporting transparent and consistent assessment practices that reduces ambiguity in what is required.

The programming module has 33 SOBs: 11 Threshold, 15 Typical and 7 Excellent. The deadline for completing SOBs is the last day of the teaching term (a term consists of 12 weeks) but they can be demonstrated multiple times, anytime, until then. We have attached some suggested dates to the SOBs of when we believe students will be able to complete them by. Once the SOBs pass this date and have not been marked, they become overdue. At the start of the module, students know of all SOBs and learning materials. The approach to SOBs varies slightly between modules. For example, in the programming module [2], new weekly lab activities mapped to SOBs are provided for Threshold SOBs that must be completed during class time. Once a student completes an activity, they present it to the tutor, who asks short questions to check understanding. If the tutor is satisfied, the SOB is recorded as achieved on the SOB Monitor. Feedback is given regardless of the outcome. For example, for the Threshold SOB *"Show and explain a simple program using a for loop"*, the activity for this SOB could be "Write a Racket function that takes a list of numbers and returns their sum. E.g., (func '(2 4 3)) returns 9. Make sure to test your function and explain how it works.". The student will present their solution to the tutor, who will ask questions like "how does the for-loop work?", "what happens if the list is empty?" etc. If the student's solution is correct (or with some minor changes that the student can determine through a discussion with the tutor) and shows the understanding of what has been done, the SOB will be marked. The student could also get this SOB if they implemented a for loop as part of a project or another SOB, i.e. Typical or Excellent. Some higher-level SOBs subsume lower-level ones. For example, the Typical SOB *"Write a function that uses vectors to solve a simple problem"*. If the student implements a function for this that uses a for loop, the student is able to get the for loop SOB as well.

4 The SOB Monitor

The SOB Monitor supports both students and teaching staff, with their own dedicated views. Students can track their own progress and understand how they compare within their cohort, while preserving the privacy of other students' data (see Figure 1). Leaderboard and ranking elements have been added in the student view. The rank is calculated using the number of SOBs a student has achieved, comparing that count to other active students and their achieved SOB count in the same cohort. Using this rank, the percentile is calculated by subtracting the student's rank from the total number of active students in the cohort, dividing this result by the total number of active students, and multiplying by 100. If multiple students have the same count of achieved SOBs, they will have the same rank.

Staff can monitor all enrolled students via a detailed table of SOB achievements by level (see Figure 2), access cohort performance reports, and manage SOBs by creating, editing, and annotating them with dates and notes. The staff interface includes a dashboard for summarising student progress against module expectations, an administration page for managing staff roles and permissions, a SOBs view for creating and editing SOB details and timings, and a monitoring view for tracking student progress and providing comments. As a new feature, students can be organised into groups for project work that can be jointly marked. In the assessment view, staff can observe SOBs, lock submissions in cases of plagiarism, or

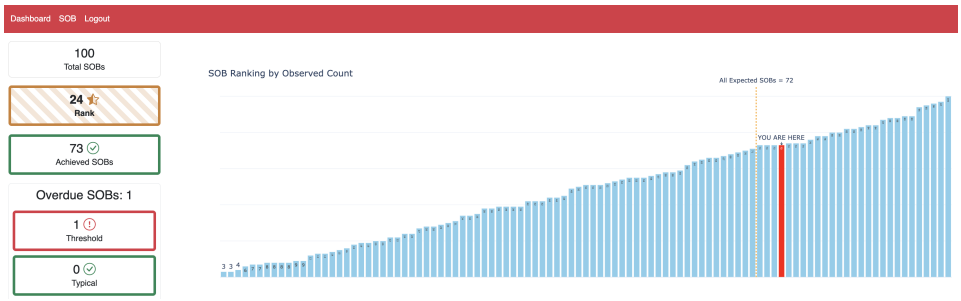


Figure 1 Student Dashboard for SOB Monitor, showing student individual ranking compared to other students, SOBs completed, overdue SOBs and total available.

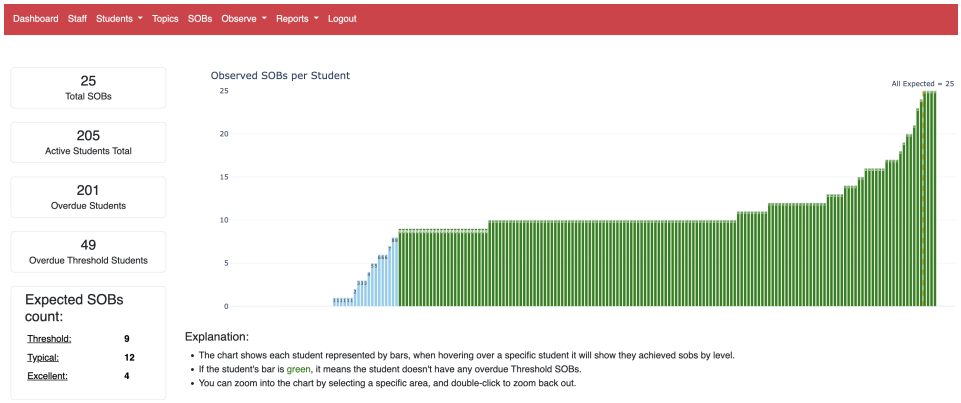


Figure 2 Staff dashboard showing each student as an individual bar, with an expectation marker and colour coding to indicate progress.

assign "in-progress" attempts to avoid duplication (i.e. showing the same SOB to another tutor). The reports view provides analytics on progression, pass rates, overdue SOBs, and other key metrics.

4.1 Architecture and Implementation

The architecture of the SOB Monitor is illustrated in Figure 3. Instead of extending the previous PHP codebase, the platform has been rebuilt around a Python based web framework, Django, which provides a clearer structure for request handling, application logic, database interaction, authentication, and page rendering. This creates a more consistent codebase and simplifies future development as the platform expands to support additional modules. The Django application is served through Gunicorn, with both components packaged together inside a Docker container, while Nginx and PostgreSQL remain installed directly on the host machine. The deployment infrastructure has also been redesigned through a Continuous Integration and Continuous Deployment pipeline that builds and reloads the Docker container from the latest GitLab version. This reduces manual deployment work and allows changes to be released more consistently.

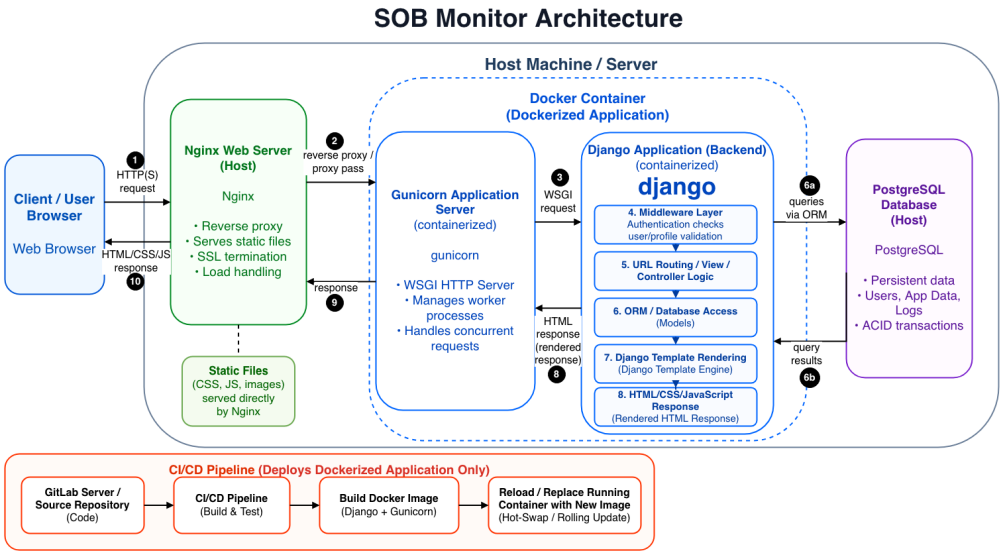


Figure 3 System Architecture of SOB Monitor as described above.

5 Observations

We have observed that students like being able to work at their own pace, deciding when they are ready to complete their SOBs, and fitting assessment around their other commitments, e.g. part-time work or caring responsibilities. The SOB Monitor allows students to view their progress regularly and help manage SOB completion. The suggested dates given with SOBs, help students to not fall too behind. We have observed students working harder around the suggested dates, or attending extra, non-timetabled sessions.

Since adding the leaderboard and rankings, students are discussing their progress actively and asking others how they are demonstrating work, if they can practice submissions together and asking for feedback when necessary. We have also noticed some healthy competition, where students are trying to be ranked number 1, many aiming to complete all possible SOBs. When a student passes Threshold and/or all SOBs, the leaderboard is animated with celebrations for the student. A drawback of the leaderboard is that some students who need more time to complete SOBs may feel rushed into showing work to tutors when it is not ready or not their own (e.g. generated by GenAI tools).

To assess the effect of the gamification elements, we compared three Programming cohorts: the September 2024 cohort, which used the SOB Monitor without leaderboard and ranking features, and the January 2025 and September 2025 cohorts, which used the gamified version. SOB completion was normalised by cohort size to account for differences in enrolment. The chart in Figure 4 shows the percentage of each cohort that completed each Programming SOB, with SOB numbers 1–33 shown on the x-axis. These SOBs are organised by level: SOBs 1–11 are Threshold, SOBs 12–26 are Typical, and SOBs 27–33 are Excellent. Across the 33 Programming SOBs, the cohorts using the gamified SOB Monitor showed a broadly comparable or stronger pattern of completion, particularly across the Threshold SOBs, where a high proportion of students completed the required behaviours. Completion of Typical and Excellent SOBs remained lower overall, which is expected because these levels extend beyond the minimum pass requirement. However, the presence of visible rankings and progress comparison appears to support continued engagement beyond the Threshold level for at least

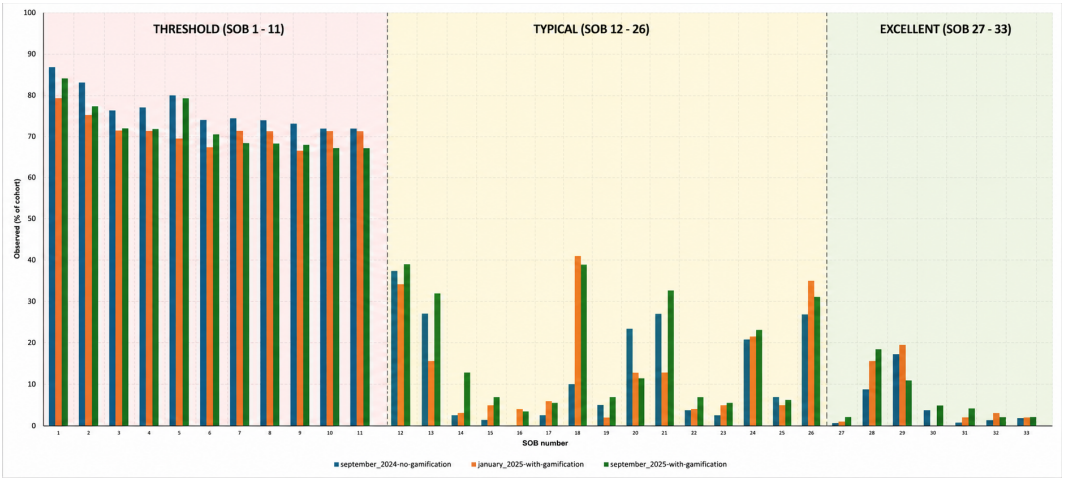


Figure 4 Normalised Programming SOB completion by cohort. SOBs 1–11 are Threshold, SOBs 12–26 are Typical, and SOBs 27–33 are Excellent.

some students. Some Typical SOBs were attempted by very few students in the September 2024 cohort, including SOBs 14, 16, and 18. These results should not be interpreted as causal evidence, as the cohorts differ by intake and timing, but they provide useful early evidence that gamification may contribute positively to student engagement with competency-based assessment.

In Programming, new Threshold SOB questions are introduced in each class and completed in-session by students who have engaged with the exercises, preventing reliance on SOBs as a substitute for learning. In Design and Development of Applications, students work in groups on projects, with SOBs assessed using the SOB Monitor's group assessment feature, making marking more efficient. The SOB Monitor scales effectively to large cohorts, such as the Foundations of Computing module (300+ students), where a fixed set of SOB activities is released for students to complete independently and present when ready for tutor review. In Systems and Architecture, students typically work in pairs to complete SOBs using methods such as physical circuits, digital twins [11], or discussion. Progression through workbook material is required beforehand, and demonstrated understanding in discussion may be sufficient for awarding a SOB. Across all modules, tutors confirm attainment through observation and targeted questioning, with SOBs recorded in the Monitor.

6 Conclusion and Future Work

We have presented the SOB Monitor platform that supports competency-based assessment using SOBs within, but not limited to Computer Science. This new implementation enhances platform security, enables support for multiple modules, and includes new features for supporting the assessment of group work. In addition, gamification elements have been implemented to enhance student engagement and motivation. Initial observations have been promising. We plan to conduct further studies to evaluate the effectiveness of the gamification elements that were introduced. We also plan to extend the SOB Monitor so students can submit code directly and run it against tutor-defined test cases. The resulting submissions and progress data, such as the number of test cases passed, could help track student development over time and support tutors in improving teaching materials.

References

1 K. Androutsopoulos, N. Gorogiannis, M. Loomes, M. Margolis, G. Primiero, F. Raimondi, P. Varsani, N. Weldin, and A. Zivanovic. A racket-based robot to teach first-year computer science. In *7th European Lisp Symposium*, volume 54, pages 54–61, Paris, France, May 2014.

2 K Androutsopoulos, M Heeney, S Smith, and M A Ticar. Lessons from adopting a competency-based assessment approach for an introductory programming module. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 2*, pages 743–744, 2025.

3 J. B. Biggs and C. Tang. *Teaching for quality learning at university.* Open University Press., Buckingham, England, 4th ed. edition, 2011.

4 N C. C. Brown, S Sentance, T Crick, and S Humphreys. Restart: The resurgence of computer science in uk schools. *ACM Trans. Comput. Educ.*, 14(2), 2014. doi:10.1145/2602484.

5 Claudia de Armas de Armas, Isaac Guillermo Gonzales Vizcarra, Douglas Lima Dantas, Sergio Takeo Kofuji, and Antonio Carlos Seabra. Analysis of gamification elements in the virtual learning environment context. In *2019 IEEE World Conference on Engineering Education (EDUNINE)*, pages 1–5, 2019. doi:10.1109/EDUNINE.2019.8875851.

6 P Denny, J Prather, B A Becker, J Finnie-Ansley, A Hellas, J Leinonen, A Luxton-Reilly, B N Reeves, E A Santos, and S Sarsa. Computing education in the era of generative ai. *Communications of the ACM*, 67(2):56–67, 2024. doi:10.1145/3624720.

7 S Deterding, D Dixon, R Khaled, and L Nacke. From game design elements to gamefulness: Defining "gamification". In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '11, pages 9–15, New York, NY, USA, 2011. Association for Computing Machinery.

8 J Frank, L Snell, O ten Cate, E Holmboe, C Carraccio, S Swing, P Harris, N Glasgow, Dr. C Campbell, D Dath, R Harden, W Iobst, D Long, R Mungroo, D Richardson, J Sherbino, I Silver, S Taber, M Talbot, and K Harris. Competency-based medical education: Theory to practice. *Medical teacher*, 32:638–45, August 2010. doi:10.3109/0142159X.2010.501190.

9 Michael D. H and Jesse F. Assessing the effects of gamification in the classroom: A longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance. *Computers & Education*, 80:152–161, 2015. doi:10.1016/j.compedu.2014.08.019.

10 Juho Hamari, Jonna Koivisto, and Harri Sarsa. Does gamification work? – A literature review of empirical studies on gamification. In *2014 47th Hawaii International Conference on System Sciences*, pages 3025–3034, 2014. doi:10.1109/HICSS.2014.377.

11 M Heeney, K Androutsopoulos, and F Raimondi. Implementing a digital twin for a robotic platform to support large-scale coding classes. In *5th International Computer Programming Education Conference*, 2024. doi:10.4230/OASIcs.ICPEC.2024.15.

12 G L. Herman, M C. Loui, and C Zilles. Students' misconceptions about medium-scale integrated circuits. *IEEE Transactions on Education*, 54(4):637–645, 2011. doi:10.1109/TE.2011.2104361.

13 R Lister, E S. Adams, S Fitzgerald, W Fone, J Hamer, M Lindholm, R McCartney, J E Moström, K Sanders, O Seppälä, B Simon, and L Thomas. A multi-national study of reading and tracing skills in novice programmers. In *ITiCSE: Innovation and Technology in Computer Science Education*, pages 119–150, New York, NY, USA, 2004. ACM. doi:10.1145/1044550.1041673.

14 Seymour Papert and Idit Harel. *Constructionism.* Ablex Publishing Corporation, Norwood, NJ, 1991.

15 Joze Rugelj, Sven Knockaert, Roel Van Steenberghe, Luk Schoofs, Janne Salonen, Kari Björn, Jose L. Marzo, and Carlos Vaz de Carvalho. Competence based joint study program on advanced networking technologies for blended learning. In *Proceedings of the 9th International Conference on Information Technology Based Higher Education and Training.* IEEE Press, 2010.

16 Maximilian Sölch, Moritz Aberle, and Stephan Krusche. Integrating competency-based education in interactive learning systems. *arXiv preprint arXiv:2309.12343*, 2023. doi:10.48550/arXiv.2309.12343.

17 L Wheelahan. How competency-based training locks the working class out of powerful knowledge: A modified bernsteinian analysis. *British Journal of Sociology of Education*, 28:637–651, September 2007. doi:10.1080/01425690701505540.

5W2H as a Flexible Analytical Framework for Investigating Programming Teaching and Learning Across Different Methodological Designs

Gabryella Rodrigues
Federal Institute of Pará (IFPA), Brazil

Ceres Morais
Graduate Program in Computer Science (UERN/UFERSA), State University of Rio Grande do Norte (UERN), Federal University of the Semi-Arid Region (UFERSA), Brazil

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

Abstract

Teaching and learning programming in Higher Education is a complex process involving cognitive, pedagogical, contextual, and metacognitive dimensions. Traditional analyses focused only on final learning outcomes, such as code correctness, performance, or pass/fail rates, may not fully capture how students construct knowledge, face difficulties, mobilize strategies, and interact with different learning contexts. This article discusses the use of the 5W2H framework as a flexible analytical tool in two doctoral studies on programming teaching and learning. The first study used 5W2H to organize and triangulate data from a mixed-methods single case study on the teaching and learning of programming in a Higher Education course. The second study articulated 5W2H with the spiral model and Bloom's revised taxonomy to analyze a qualitative multiple case study focused on students' algorithm construction process. Through a comparative analysis, the article shows that 5W2H can operate at different analytical scales: as a broad structure for mapping subjects, contexts, methodologies, difficulties, and learning factors, and as a process-oriented framework for interpreting students' actions during algorithmic problem solving. The findings suggest that 5W2H supports the organization, categorization, triangulation, and interpretation of complex educational data. Therefore, the framework offers a useful methodological lens for research in programming education, helping researchers understand the relationships between subjects, contexts, strategies, difficulties, and cognitive processes involved in teaching and learning programming.

2012 ACM Subject Classification Social and professional topics → Computing education; Social and professional topics → Computer science education

Keywords and phrases 5W2H, programming education, programming learning, analytical framework, Higher Education

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.14

Funding The authors acknowledge the financial support provided by the Graduate Program of the State University of Rio Grande do Norte (UERN) for this submission.

1 Introduction

The teaching and learning of programming are part of the curricular guidelines of Computing and related courses [15]. However, this process remains challenging for both teachers and students. Teaching programming requires strategies to monitor, guide, assess, and support students with different profiles and levels of prior knowledge [5]. Learning to program, in turn, requires constant practice and the development of skills such as problem solving, logical reasoning, abstraction, interpretation, and algorithmic thinking [3, 17, 9, 14]. When these dimensions are not adequately supported, students may experience demotivation, frustration, failure, or dropout [13].

Because of this complexity, the analysis of programming education cannot be restricted to final learning outcomes, such as grades, code correctness, performance, or pass/fail rates. These indicators are relevant, but they do not fully explain how students understand problems, mobilize prior knowledge, construct algorithms, deal with errors, adapt strategies, and interact with different learning contexts. Therefore, research in this field requires analytical structures capable of organizing multiple sources of evidence and relating cognitive, pedagogical, contextual, and metacognitive dimensions.

In this context, the 5W2H framework offers a structure for organizing and interpreting data through seven guiding parameters: "What", "Who", "Where", "When", "Why", "How", and "How Much" [6, 8, 7, 10]. Although traditionally associated with planning and management, 5W2H can also be adapted as an analytical framework, especially when the phenomenon under investigation involves multiple actors, contexts, actions, reasons, strategies, and impacts.

This article discusses the use of 5W2H in two doctoral studies conducted in the field of Educational Sciences, both focused on programming teaching and/or learning in Higher Education. The first study [11] aimed to understand how the process of teaching and learning programming occurs in Higher Education, seeking to identify methodologies, tools, learning factors, required knowledge and skills, difficulties faced by teachers and students, and the consequences of these difficulties.

The second study [16] aimed to understand how students in an introductory programming course assimilate and use programming logic concepts in the construction of algorithms. Its analytical focus was the process through which students assimilate, accommodate, and equilibrate new programming knowledge, in dialogue with Piaget's genetic epistemology [12].

Although the two studies adopted different methodological designs, both used 5W2H to organize, categorize, triangulate, and interpret data. In the first study, the framework served as a broad structure for triangulating evidence from documents, questionnaires, and interviews. In the second, it was articulated with the spiral model and Bloom's revised taxonomy to support a more process-oriented analysis of students' actions during algorithm construction.

Despite previous uses of 5W2H as a planning and management tool, its potential as an analytical framework in educational research, particularly in programming education, remains comparatively less discussed. This article contributes to this gap by examining how 5W2H can be methodologically adapted to support different research designs and analytical scales in studies on programming teaching and learning.

Based on this context, this article is guided by the following research question: how can the 5W2H framework support the organization, triangulation, and interpretation of data in doctoral studies on programming teaching and learning developed with different methodological designs? To answer this question, the article presents and compares the methodological adaptation of 5W2H in the two studies, highlighting its role in organizing data sources, supporting evidence triangulation, and interpreting contextual, pedagogical, cognitive, and metacognitive dimensions.

The remainder of this article is organized as follows. First, the 5W2H framework is presented as an analytical tool for educational research. Next, the methodological approach of this article is described. The following section explains how 5W2H was applied in the two doctoral studies. Then, a comparative analysis discusses similarities, differences, analytical scales, and methodological functions of the framework. Finally, the discussion, limitations, and conclusion sections address the contributions, challenges, and potential of 5W2H as a methodological resource for research on programming teaching and learning.

2 5W2H as an Analytical Tool

The 5W2H framework is traditionally recognized as a management and planning tool used to structure ideas, projects, tasks, actions, or problems in a clear and systematic way. Its logic consists of answering seven guiding questions: What, Who, Where, When, Why, How, and How Much. These questions make it possible to examine different aspects of a problem in an articulated manner, avoiding analyses restricted to a single dimension of the phenomenon under investigation [7].

Although 5W2H is widely used in areas such as business, strategy, planning, quality management, and process organization, its structure can be adapted to different analytical contexts. Previous studies have reported its application in areas such as gamification, programming language development, software requirements reuse, product design, safety analysis, and programming education [6, 8, 20, 2, 4, 10]. In Educational Sciences, it has also been explored mainly as a tool for planning, organizing, and managing pedagogical or institutional actions. For example, Veiga et al. [19] used SWOT and 5W2H to support collaborative pedagogical planning aimed at promoting equity in Distance Education. Other applications have associated the framework with the design and evaluation of gamified educational systems [6].

However, many of these applications approach 5W2H primarily as an operational tool for planning, intervention design, or management. Its use as an analytical framework for organizing, categorizing, triangulating, and interpreting empirical data in educational research remains less discussed. This distinction is central to the present article: here, 5W2H is not treated as a prescriptive action plan, but as a methodological structure for analyzing complex educational phenomena, particularly in studies on programming teaching and learning.

The analytical potential of 5W2H lies in its ability to decompose a complex situation into interrogative dimensions. The What parameter identifies the action, problem, object, or phenomenon under analysis; Who directs attention to the subjects involved; Where situates the phenomenon in a context or institutional setting; When introduces the temporal dimension; Why explores reasons, causes, or difficulties; How describes processes, strategies, methods, or procedures; and How Much observes intensity, quantity, frequency, effort, cost, recurrence, relevance, or impact.

In educational research, the How Much parameter requires particular adaptation. Unlike its traditional use in management contexts, where it is often associated with cost or measurable resources, in educational studies it may refer to the perceived intensity of difficulties, recurrence of behaviors, frequency of strategies, relevance attributed by participants, or impact of a given condition on teaching and learning. Thus, the framework can be adapted without reducing educational phenomena to exclusively quantitative dimensions.

In research on programming teaching and learning, this multidimensional structure is particularly relevant because the phenomenon involves technical, cognitive, pedagogical, contextual, social, and metacognitive dimensions. Programming is not limited to writing code or achieving final performance indicators; it involves subjects who teach and learn, institutional contexts, moments in the educational pathway, conceptual difficulties, problem-solving strategies, forms of mediation, and different levels of effort or impact. From this perspective, 5W2H can function as an analytical lens that connects subjects, contexts, actions, reasons, strategies, and impacts, providing the basis for the comparative discussion developed in this article.

3 Methodological Approach

This article adopts a comparative and methodological-reflective approach. It does not aim to reanalyze the empirical data produced in the two doctoral studies, but to examine how the 5W2H framework was used as an analytical structure in each study. Therefore, the focus of the analysis is not on the empirical results of the studies themselves, but on the methodological role played by 5W2H in the organization, categorization, triangulation, and interpretation of data.

The corpus of analysis consists of two doctoral studies conducted in the field of Educational Sciences, in the specialty of Educational Technology, both focused on the teaching and/or learning of programming in Higher Education. The first study adopted a mixed-methods single case study design, while the second adopted a qualitative multiple case study design. These studies were selected because they share a common object of investigation, programming teaching and learning, but differ in their methodological designs, analytical scales, data sources, and ways of applying the 5W2H framework.

To make the methodological basis of the comparison explicit, Table 1 summarizes the main characteristics of the two doctoral studies.

Table 1 Methodological characteristics of the two doctoral studies.

Aspect	Study 1	Study 2
Research focus	Teaching and learning programming in Higher Education	Programming learning and algorithm construction in introductory programming
Research design	Mixed-methods single case study	Qualitative multiple case study
Context	Higher Education course in Informatics in the Brazilian context	Introductory programming contexts in Higher Education, including formal and non-formal contexts
Participants	Students and teachers from a Higher Education course in the field of Informatics	Students participating in introductory programming learning contexts
Data sources	Documents, questionnaires, and interviews with students and teachers	Semi-structured interviews, clinical interviews, and digital and paper-based records
Analysis procedures	Content analysis and interpretation, followed by categorization according to the theoretical framework and 5W2H parameters	Content analysis supported by NVivo, articulated with the spiral model, Bloom's revised taxonomy, and 5W2H
Validation and triangulation	Triangulation of multiple sources of evidence: documents, questionnaires, and interviews	Triangulation of interviews, students' explanations, algorithmic productions, and digital/paper records
Role of 5W2H	Broad analytical matrix for organizing and triangulating evidence about subjects, contexts, difficulties, factors, methodologies, tools, and consequences	Process-oriented analytical structure for interpreting students' actions, strategies, difficulties, effort, and reflections during algorithm construction

Source: Prepared by the authors.

In addition to the methodological descriptions of the two doctoral studies, the analysis also considered reflective accounts produced by the principal investigators regarding the role, advantages, challenges, and limitations of using 5W2H in their respective studies. These reflections were used to support the interpretation of how the framework operated in each study and to identify the conditions under which 5W2H contributed to data organization, triangulation, and analytical interpretation.

The comparative analysis was organized around five dimensions: the object of analysis of each study; the methodological design adopted; the function attributed to the 5W2H framework; the analytical scale at which the framework was applied; and the reflective assessment of the framework by the principal investigators. This procedure made it possible to identify similarities and differences between the two applications of 5W2H and to discuss its potential as a flexible analytical framework for research in programming education.

Thus, the methodological contribution of this article lies in examining how 5W2H can be adapted from its traditional use as a planning and management tool to operate as an analytical framework capable of supporting different stages of educational research, including the organization of evidence, the triangulation of data sources, and the interpretation of complex teaching and learning processes.

4 Application of the 5W2H Framework in the Doctoral Studies

This section describes how the 5W2H framework was applied in each doctoral study, highlighting its analytical function, the data sources involved, and its contribution to the organization and interpretation of the investigated phenomenon.

4.1 Application of 5W2H in Study 1

Study 1, entitled *Teaching and Learning Programming: A Case Study in Higher Education*, aimed to understand how the process of teaching and learning programming occurs in Higher Education. To this end, it sought to inventory methodologies and tools, characterize factors that influence learning, explain the knowledge and skills required to learn programming, and identify difficulties faced by teachers and students, as well as the consequences of these difficulties.

The research adopted a mixed-methods approach and was developed through a single case study in a Higher Education course in the field of Informatics in the Brazilian context. Data were collected through documents, questionnaires, and interviews with students and teachers. After collection, the data were subjected to content analysis and interpretation. Subsequently, triangulation of the sources of evidence was carried out, categorized in light of the theoretical framework and according to the parameters of the 5W2H framework.

In this study, 5W2H was used to support the categorization of units of record obtained from different data sources. After the development of conceptual maps related to the research objectives, it was possible to refine and characterize the parameters used to analyze the learning process. These parameters were organized according to 5W2H and served as the basis for analyzing students' and teachers' perceptions of the investigated process.

In this study, the framework assumed a broad organizational function for the phenomenon. It made it possible to structure information according to the seven 5W2H parameters: What, by identifying factors, competencies, knowledge, and skills required for learning programming; Who, by considering the participants involved in the process; Where, by situating the contexts in which learning occurs; When, by observing moments in the educational pathway; Why, by examining reasons associated with difficulties; How, by analyzing the ways in which teachers

and students act; and How Much, by considering the intensity or impact of the factors analyzed. Thus, 5W2H supported data triangulation and the construction of an integrated reading of programming teaching and learning, as shown in Figure 1.

Figure 1 Broad organizational function of 5W2H in Study 1.

4.2 Application of 5W2H in Study 2

Study 2, entitled *Programming Learning in Higher Education: A Multiple Case Study on Cognitive and Contextual Factors*, focused on understanding how students in an introductory programming course assimilate and use programming logic concepts in the construction of algorithms. The study adopted a qualitative approach and used a multiple case study methodology.

To achieve this understanding, meetings were held with students using semi-structured interviews, clinical interviews, and digital and paper-based records as data collection instruments. The data were analyzed through content analysis and organized using the spiral software development model [18], Bloom's revised taxonomy [1], and the 5W2H framework. The spiral model structured the activity-solving process into phases, from goal definition to evaluation of the constructed solution; Bloom's taxonomy related these phases to the types of knowledge mobilized by students; and 5W2H provided a systematic structure for organizing and interpreting students' actions throughout the coding process, from the initial understanding of the problem to the final tests, as shown in Figure 2.

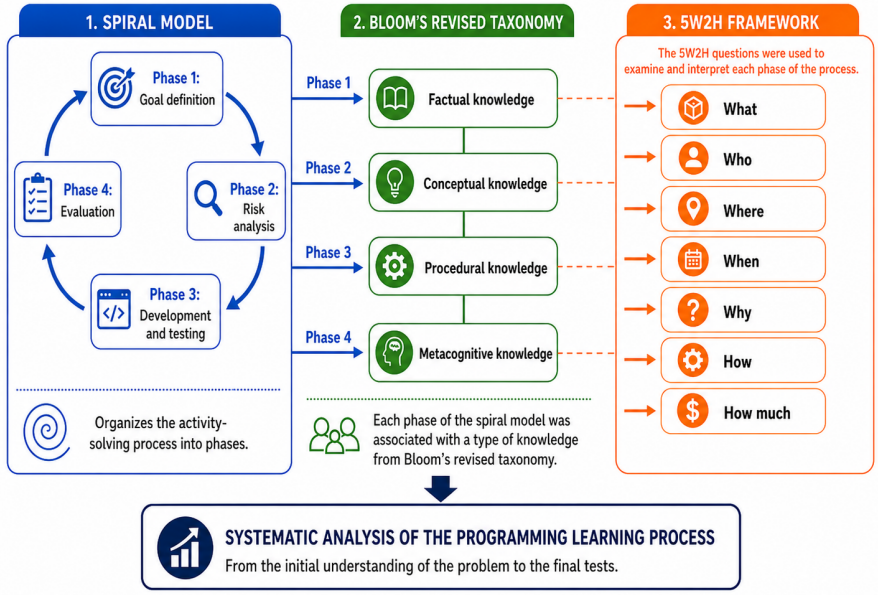


Figure 2 Articulation between the Spiral Model, Bloom's Revised Taxonomy and 5W2H.

5 Comparative Analysis of the Applications of 5W2H

The comparison between the two studies shows that the 5W2H framework was used in both as an instrument for organizing and interpreting data on programming teaching and/or learning in Higher Education. However, its role differed according to the object of analysis, methodological design, data sources, and analytical scale of each study.

The first difference concerns the object of analysis. Study 1 analyzed programming teaching and learning in a broad way, considering methodologies, tools, learning factors, required knowledge and skills, difficulties faced by teachers and students, and the consequences of these difficulties. Study 2 focused more specifically on programming learning, seeking to understand how introductory programming students assimilate and use programming logic concepts in the construction of algorithms.

The second difference lies in the methodological design. Study 1 adopted a mixed-methods single case study, using documents, questionnaires, and interviews with students and teachers from a Higher Education course in Informatics in the Brazilian context. Study 2 adopted a qualitative multiple case study, using semi-structured interviews, clinical interviews, and digital and paper-based records with students from formal and non-formal programming learning contexts.

These methodological differences influenced how 5W2H was applied. In Study 1, the framework organized a broader reading of the phenomenon, allowing data to be categorized according to parameters capable of encompassing subjects, contexts, methodologies, tools, learning factors, difficulties, and impacts. In Study 2, 5W2H was used in a more process-oriented analysis, aimed at understanding how students moved through different moments of learning: before solving the activities, during algorithm construction, and in the outcome/reflection after solving them.

The third difference refers to the level of granularity of the analysis. In Study 1, 5W2H supported a macro-level analysis of programming teaching and learning as an institutional, pedagogical, and educational phenomenon. In Study 2, the framework was combined with the spiral model and Bloom's revised taxonomy to support a more fine-grained analysis of students' actions during algorithm construction, including goal definition, risk analysis, development and testing, and evaluation.

Despite these differences, both studies converge in recognizing programming teaching and learning as complex processes composed of multiple interdependent factors. In both cases, 5W2H helped avoid a fragmented reading of the phenomenon by organizing information about subjects, contexts, actions, reasons, strategies, and impacts. This convergence shows the methodological flexibility of the framework: in one study, it functioned as a broad matrix for categorizing and triangulating data; in the other, it was integrated with complementary models to analyze specific learning processes.

Table 2 summarizes the main differences between the two uses of 5W2H, focusing on analytical scale, categorization, triangulation, articulation with other frameworks, and methodological contribution.

Table 2 Comparative analysis of the use of 5W2H in the two studies.

Compared aspect	**Study 1**	**Study 2**
Analytical scale	Macro-level analysis of the teaching and learning process in a Higher Education course	Micro/process-oriented analysis of students' algorithm construction
Main function of 5W2H	Broad matrix for organizing and triangulating evidence from different sources	Analytical structure for interpreting students' actions across stages of problem solving
Focus of categorization	Factors, participants, contexts, difficulties, methodologies, tools, and consequences related to programming teaching and learning	Students' understanding, planning, coding, testing, correction, effort, and reflection during algorithm construction
Use in triangulation	Direct triangulation of documents, questionnaires, and interviews through the 5W2H parameters	Triangulation of interviews, students' explanations, and digital/paper records, articulated with other analytical models
Articulation with other frameworks	Used mainly in relation to the theoretical framework and content analysis categories	Combined with the spiral model, Bloom's revised taxonomy, and Piaget's genetic epistemology
Methodological contribution	Supports an integrated reading of programming teaching and learning as an institutional, pedagogical, and contextual phenomenon	Supports a fine-grained interpretation of cognitive and metacognitive processes involved in algorithm construction

Source: Prepared by the authors.

To illustrate how 5W2H was operationalized in practice, the two studies provide different examples of analytical use. In Study 1, students' reports on difficulties with programming logic, abstraction, problem interpretation, and basic programming concepts were categorized under What as learning difficulties. The same evidence was interpreted under Why when

the analysis focused on possible causes, such as insufficient prior knowledge or abstraction barriers, and under How Much when the focus was on perceived effects, such as demotivation, failure, or dropout. Evidence from documents, questionnaires, and interviews was also associated with Who, Where, When, and How, allowing the study to relate participants, institutional context, curricular organization, and teaching strategies in the triangulation of teachers' and students' perceptions.

In Study 2, 5W2H was operationalized at a more process-oriented level. Students' actions during problem reading, solution planning, coding, testing, and correction were associated with What, When, and How, making it possible to identify what was being solved, when each action occurred in the phases of the spiral model, and how students proceeded. Their explanations, uncertainties, revisions, and reflections were associated with Why and How Much, supporting the interpretation of reasons for decisions, difficulties faced, effort involved, recurrence of corrections, and evidence of conceptual or metacognitive adjustment.

These examples show that 5W2H did not operate merely as a descriptive checklist. In both studies, empirical evidence was associated with one or more parameters and then interpreted in relation to the research objectives and theoretical frameworks. This illustrates how the framework supported the movement from data organization to analytical interpretation.

Thus, the comparative analysis allows us to state that 5W2H is not limited to a planning tool or an initial organizational instrument. In both studies, it assumed an analytical function, supporting the categorization, triangulation, and interpretation of data. This function makes the framework relevant for research on programming teaching and learning, especially when the aim is to understand complex phenomena involving subjects, contexts, strategies, difficulties, and cognitive processes.

6 Discussion: 5W2H as an Analytical Framework for Research on Programming Teaching and Learning

The main contribution of this article lies in showing that 5W2H can be methodologically adapted from its traditional role as a planning and management tool to function as an analytical framework in programming education research. This contribution does not consist of proposing a new learning theory, a new pedagogical model, or a new technique for content analysis. Rather, its contribution lies in its use as an integrative analytical matrix capable of organizing and relating different dimensions of complex educational phenomena.

Several consolidated frameworks already contribute to the analysis of programming teaching and learning. Bloom's revised taxonomy, for example, supports the identification of cognitive processes and types of knowledge mobilized by students [1]. The spiral model helps structure phases of problem solving, development, testing, and evaluation [18]. Content analysis provides procedures for coding, categorizing, and interpreting empirical material. Learning theories, such as Piaget's genetic epistemology, contribute to explaining how knowledge may be constructed, reorganized, and stabilized throughout the learning process [12].

The role of 5W2H is different. It does not replace these frameworks; instead, it provides an interrogative structure that helps connect them. By asking What, Who, Where, When, Why, How, and How Much, the framework allows researchers to relate the object of analysis, the participants involved, the context, the temporal organization of the process, the reasons for difficulties or decisions, the strategies adopted, and the intensity, recurrence, or impact of the observed phenomena.

In Study 1, this adaptation supported the organization and triangulation of heterogeneous sources of evidence related to the teaching and learning of programming. In Study 2, it supported a more process-oriented analysis of students' actions during algorithm construction, especially when articulated with the spiral model and Bloom's revised taxonomy. Thus, 5W2H contributes not by explaining learning on its own, but by helping researchers structure the relationships among empirical evidence, theoretical categories, participants, contexts, actions, reasons, strategies, and impacts.

The comparison between the two studies also suggests that 5W2H performed three complementary analytical functions. First, it had an organizational function by structuring heterogeneous data from documents, questionnaires, interviews, records, and students' productions. Second, it had an integrative function by connecting data sources, analytical categories, and theoretical references, especially in the triangulation process in Study 1 and in the articulation with the spiral model and Bloom's revised taxonomy in Study 2. Third, it had an interpretive function by allowing difficulties, strategies, contexts, actions, and impacts to be analyzed in relation to one another rather than as isolated elements. These functions help clarify the methodological contribution of 5W2H as an analytical layer for programming education research.

Its originality therefore lies in its methodological function as a flexible and integrative analytical layer, capable of supporting different research designs and analytical scales in programming education.

6.1 Reflections from the Principal Investigators

From the perspective of the principal investigator of Study 1, the main contribution of 5W2H was its ability to organize heterogeneous data sources within a coherent analytical structure. Since the study involved documents, questionnaires, and interviews with both teachers and students, the framework helped relate different types of evidence to the same analytical dimensions. This facilitated triangulation and supported a broader understanding of the teaching and learning process. However, one challenge observed was the need to adapt some parameters, especially How Much, since educational phenomena are not always expressed through direct numerical indicators or measurable costs.

From the perspective of the principal investigator of Study 2, 5W2H was particularly useful because it allowed the analysis of students' actions during algorithm construction without isolating these actions from their context, motivations, difficulties, and strategies. When articulated with the spiral model and Bloom's revised taxonomy, the framework helped organize evidence about how students understood problems, planned solutions, tested hypotheses, corrected errors, and reflected on their own learning. Nevertheless, this use also required careful articulation with other theoretical models, since 5W2H alone does not explain cognitive development or learning processes.

These reflections indicate that the value of 5W2H in educational research does not lie in mechanically applying its seven questions. Its analytical strength depends on adapting the framework to the nature of the phenomenon under investigation, the research objectives, the data sources, and the theoretical assumptions of each study. In this sense, 5W2H should be understood as an organizing and interpretive layer rather than as a self-sufficient analytical method.

Overall, the two studies show that 5W2H can support research on programming teaching and learning by helping researchers move between broader contextual dimensions and more specific learning trajectories. From a macro perspective, it can organize institutional,

pedagogical, and curricular aspects of programming education. From a micro perspective, it can support the analysis of students' actions, decisions, difficulties, strategies, and reflections during algorithm construction.

This flexibility is precisely what makes the framework relevant as a methodological resource for programming education research.

7 Limitations and Challenges of Using 5W2H

Although the 5W2H framework proved useful for organizing, triangulating, and interpreting data in the two studies analyzed, its use also presents limitations and challenges. First, 5W2H is not a learning theory, a pedagogical model, or a data analysis method by itself. Therefore, it should not be used as a substitute for theoretical frameworks or systematic analytical procedures. Its value depends on its articulation with research objectives, theoretical assumptions, data sources, and methods of analysis.

Second, the framework may lead to superficial or merely descriptive categorization if applied mechanically. Simply assigning empirical evidence to the questions What, Who, Where, When, Why, How, and How Much does not guarantee analytical depth. The researcher must interpret the relationships among these dimensions and connect them to the research problem. Without this interpretive movement, 5W2H risks becoming only a checklist.

Third, some parameters may overlap during the categorization process. For example, a student's difficulty in understanding an algorithmic problem may be related to What, when the focus is on the difficulty itself; to Why, when the focus is on its causes; and to How, when the focus is on the strategies used to overcome it. This overlap does not invalidate the framework, but it requires clear operational definitions and consistent coding decisions.

Fourth, the How Much parameter requires particular adaptation in educational research. In management contexts, this dimension is often associated with cost, quantity, time, or measurable resources. In educational studies, however, it may refer to perceived intensity, recurrence, relevance, effort, or impact. This adaptation increases the flexibility of 5W2H, but also requires methodological caution to avoid imprecise or arbitrary interpretations.

Finally, 5W2H does not explain learning processes on its own. In Study 2, for example, its analytical role depended on its articulation with the spiral model, Bloom's revised taxonomy, and Piaget's genetic epistemology. Similarly, in Study 1, its contribution depended on its integration with content analysis and triangulation procedures. Therefore, the framework should be understood as an organizing and integrative analytical layer, not as a self-sufficient explanatory model.

These limitations indicate that the use of 5W2H in educational research requires careful methodological planning. Its strength lies in helping researchers organize complex phenomena and relate different analytical dimensions, but its effectiveness depends on how rigorously the parameters are defined, applied, and interpreted.

8 Conclusion

This article presented and discussed the methodological adaptation of the 5W2H framework in two doctoral studies on programming teaching and learning in Higher Education. Although the studies adopted different methodological designs, both used 5W2H to organize, categorize, triangulate, and interpret data related to complex educational phenomena.

The comparative analysis showed that 5W2H can operate at different analytical scales. In Study 1, it functioned as a broad matrix for organizing and triangulating heterogeneous sources of evidence related to teaching and learning programming, including documents,

questionnaires, and interviews with students and teachers. In Study 2, it supported a more process-oriented analysis of students' actions during algorithm construction, especially when articulated with the spiral model and Bloom's revised taxonomy.

The main contribution of this article lies in showing that 5W2H can be used not only as a planning or management tool, but also as an integrative analytical framework in educational research. Its value is not in replacing consolidated theoretical or methodological frameworks, but in helping researchers connect subjects, contexts, actions, reasons, strategies, and impacts across different data sources and levels of analysis.

The examples of operationalization presented in this article indicate that 5W2H can support the movement from empirical evidence to analytical interpretation. By associating data with the parameters What, Who, Where, When, Why, How, and How Much, the framework helps researchers avoid fragmented readings of programming teaching and learning and favors a more systematic understanding of the relationships among the dimensions involved.

However, the use of 5W2H requires methodological caution. The framework does not explain learning processes by itself and should not be applied mechanically as a checklist. Its analytical strength depends on clear operational definitions, careful categorization procedures, triangulation strategies, and articulation with theoretical models and data analysis methods.

As future work, further studies may apply 5W2H in other programming education contexts, including different institutions, teaching modalities, educational levels, student profiles, and programming courses. Such studies may contribute to refining the framework, testing its analytical robustness, and expanding its use as a methodological resource in Computing Education research.

References

1 Lorin W. Anderson and David R. Krathwohl. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Addison Wesley Longman, 2001.

2 M. B. do Prado, J. da S. F. Junior, J. F. Raphanhin, and M. D. Â. M. Sarreta. Determinação e gestão de causas raízes de falhas e proposta de melhoria por meio do 5w2h no setor de atendimento de uma pizzaria em de minas gerais. *Brazilian Journal of Business*, 3(4):3295–3305, 2021.

3 A. de J. Gomes. *Dificuldades de aprendizagem de programação de computadores: Contributos para a sua compreensão e resolução*. PhD thesis, Universidade de Coimbra, 2010. URL: `https://hdl.handle.net/10316/14586`.

4 L. Heng, Z. Longfu, and Y. Qian. Research on safety control of refined oil depot based on big data technology. In *Proceedings of the 2019 2nd International Conference on Data Science and Information Technology*, pages 249–254, 2019. `doi:10.1145/3352411.3352450`.

5 S. Kawaguchi, T. Nishikawa, Y. Sato, R. Onuma, H. Nakayama, S. Nakamura, and Y. Miyadera. Development of a training data creation support environment for estimating programming learning situations. In *2019 18th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–6, 2019. `doi:10.1109/ITHET46829.2019.8937339`.

6 A. C. T. Klock, I. Gasparini, and M. S. Pimenta. 5w2h framework: A guide to design, develop and evaluate the user-centered gamification. In *Proceedings of the 15th Brazilian Symposium on Human Factors in Computing Systems*, pages 1–10, 2016.

7 Manuel Laguna and Johan Marklund. *Business process modeling, simulation and design*. Chapman and Hall/CRC, 2018.

8 J. C. S. Leite, Y. Yu, L. Liu, S. Eric, and J. Mylopoulos. Quality-based software reuse. In *International Conference on Advanced Information Systems Engineering*, pages 535–550, 2005.

9 R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bulletin*, 36(4):119–150, 2004. doi:10.1145/1041624.1041673.

10 C. G. Morais, J. N. de FL Araújo, and R. W. de Lima. Utilizando o framework 5w2h para compreender o processo de ensino e aprendizagem de programação. In *Anais do XXXIV Simpósio Brasileiro de Informática na Educação*, pages 222–233, 2023.

11 Ceres Germanna Braga Morais. *Ensino e aprendizagem de programação: estudo de caso no Ensino Superior*. Tese de doutoramento em ciências da educação, especialidade em tecnologia educativa, Universidade do Minho, Braga, Portugal, 2021.

12 Jean Piaget. *Genetic epistemology*. Columbia University Press, 1970.

13 S. S. Prietch and T. A. Pazeto. Mapeamento de cursos de licenciatura em computação seguido de proposta de padronização de matriz curricular. In *XVIII Workshop de Educação em Computação (WEI 2010), Anais do XXX Congresso da Sociedade Brasileira de Computação-CSBC*, pages 921–930, 2010.

14 A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003. doi:10.1076/CSED.13.2.137.14200.

15 S. N. M. Rum and M. A. Ismail. Metocognitive support accelerates computer assisted learning for novice programmers. *Journal of Educational Technology & Society*, 20(3):170–181, 2017. URL: https://www.j-ets.net/ETS/journals/20_3/14.pdf.

16 Gabryella Rocha Rodrigues da Silva. *A aprendizagem de programação no Ensino Superior: um estudo multicasos sobre fatores cognitivos e contextuais*. Tese de doutoramento em ciências da educação, especialidade em tecnologia educativa, Universidade do Minho, Braga, Portugal, 2025.

17 Elliot Soloway and James C. Spohrer. *Studying the novice programmer*. L. Erlbaum Associates, 1988.

18 Ian Sommerville. *Engenharia de software*. Pearson Brasil, 10 edition, 2019.

19 Susana da Veiga, Caroline Alves dos Santos, Júlia Rodrigues Estéfano, Kátia Celina da Silva Richetto, and Willian José Ferreira. Práticas colaborativas com swot e 5w2h para a promoção da equidade na educação a distância. *EaD em Foco*, 15(2):e2581, 2025. doi:10.18264/eadf.v15i2.2581.

20 K. S. Ventura and A. B. V. Suquisaqui. Aplicação de ferramentas swot e 5w2h para análise de consórcios intermunicipais de resíduos sólidos urbanos. *Ambiente Construído*, 20:333–349, 2019.

Architecture and Design Study with Analytical Validation of a Mixed Reality Software System for Pilot Training with Real-Time Visual Scene Substitution

Serhii D. Prykhodchenko
Department of Computer systems' software, Dnipro University of Technology, Ukraine

Oksana Yu. Prykhodchenko
Department of Economics and Economic Cybernetics, Dnipro University of Technology, Ukraine

Andrii A. Martynenko
Department of Computer systems' software, Dnipro University of Technology, Ukraine

Dmytro Babets
Department of Applied Mathematics, Dnipro University of Technology, Ukraine

Marcin Paszkuta
merQlab, Department of Computer Graphics, Vision and Digital Systems,
Silesian University of Technology, Gliwice, Poland

Dariusz Myszor
Department of Algorithmics and Software, Faculty of Automatic Control,
Electronics and Computer Science, Silesian University of Technology, Gliwice, Poland

Przemysław Skurowski
merQlab, Department of Computer Graphics, Vision and Digital Systems,
Silesian University of Technology, Gliwice, Poland

Krzysztof Cyran
merQlab, Department of Computer Graphics, Vision and Digital Systems,
Silesian University of Technology, Gliwice, Poland

Abstract

This paper presents the design and architectural formalization of a mixed reality (MR) system for pilot training that supports selective, region-level visual scene substitution. The proposed approach enables context-aware replacement of semantically meaningful regions within the pilot's egocentric view – such as cockpit displays and external windows – while preserving direct physical interaction with real cockpit controls and the pilot's hands.

The core contribution is a modular real-time architecture that integrates semantic perception, structured scene representation, and GPU-accelerated mask-based compositing into a unified dataflow pipeline. The system is designed with explicit latency-aware modelling to support analytical evaluation of performance under strict real-time constraints (targeting below 20–30 ms).

A functional prototype was implemented and evaluated under different hardware configurations. Experimental results demonstrate that the GPU-accelerated version running on an NVIDIA RTX 3060 achieves an average end-to-end latency of 26.3 ms (approximately 38 fps) with improved temporal stability. These results confirm the practical feasibility of the proposed architecture under controlled conditions.

Adopting a design-science perspective, this work combines analytical modelling, architectural design, and preliminary prototype validation. The study addresses the need for stronger technical grounding in XR-based aviation training research prior to large-scale human-subject evaluation. The presented approach provides a reproducible foundation for the development of hybrid training systems that combine the sensorimotor fidelity of physical cockpits with the instructional flexibility of real-time visual substitution.

7th International Computer Programming Education Conference (ICPEC 2026).
Editors: Filipe Portela, Luís Matos, and Tiago Guimarães; Article No. 15; pp. 15:1–15:11

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Computer systems organization → Real-time systems; Human-centered computing → Mixed / augmented reality; Computing methodologies → Computer vision; Computing methodologies → Image segmentation; Computer systems organization → Embedded and cyber-physical systems; Software and its engineering → Data flow architectures

Keywords and phrases Mixed Reality, Pilot Training, Visual Scene Substitution, Region-level Compositing, Real-time Architecture

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.15

Supplementary Material *Software*: `https://github.com/prykhodchenkosd/MRPilotRealtimeTest` archived at `swh:1:dir:c6e42056d4c3aa7bb17f93baecb3230bb20d1110`

1 Introduction

The training of commercial pilots requires a high degree of perceptual fidelity, sensorimotor realism, and the ability to safely rehearse a wide range of operational scenarios, including rare and safety-critical events. Traditional full-flight simulators provide high realism but are expensive, immobile, and difficult to reconfigure quickly. Virtual reality (VR) systems offer excellent scenario control, yet they typically eliminate direct tactile interaction with physical cockpit controls, which is important for developing accurate procedural memory [4, 10, 6]. Augmented reality (AR) approaches preserve the real environment but are generally limited to additive overlays and cannot meaningfully modify existing visual elements or instrument behaviour [2, 13].

Mixed reality (MR) systems have emerged as a promising direction by combining physical cockpit interaction with virtual content. However, most current MR implementations focus primarily on replacing the external visual scene visible through the cockpit windows, while leaving internal instrumentation largely unchanged [7, 12, 20, 5]. Although video see-through and passthrough-based systems provide greater potential for visual manipulation, they have rarely been used for structured, region-selective substitution of semantically meaningful areas inside the cockpit.

This work addresses the above limitations by proposing a modular real-time architecture for selective, region-level visual scene substitution. The approach aims to enable context-aware replacement of specific regions within the pilot's egocentric view – such as cockpit displays and external windows – while maintaining full physical interaction with the real controls and the pilot's hands.

To clarify the positioning of the proposed system, Table 1 compares the main immersive training paradigms.

Table 1 Structural comparison of immersive training paradigms for pilot training.

System Type	Physical Cockpit	Scene Modification	Region-Level Control	Dynamic Reconfiguration
VR	✗	Full	✗	✓
AR	✓	Additive only	✗	Limited
MR (classic)	✓	External only	✗	Limited
Proposed	✓	**Selective**	✓	✓

The problem addressed in this work can be formalised as the real-time synthesis of a perceptually coherent mixed visual stream under strict latency and sensorimotor constraints. At each discrete time step t, the system receives a multimodal input tuple (I_t, P_t, G_t), where

$I_t \in \mathbb{R}^{H\times W\times 3}$ denotes the captured egocentric RGB image, $P_t \in SE(3)$ the six-degree-of-freedom head pose, and $G_t \in \mathbb{R}^3$ the gaze direction. The objective is to produce an output image

$$O_t = f(I_t, P_t, G_t, \theta_t),$$

where θ_t denotes adaptive control parameters. The transformation operates over a dynamically inferred set of semantically meaningful regions with associated masks $M_t^k(x) \in [0, 1]$, such that the composited output satisfies

$$O_t(x) = (1 - M_t(x))I_t(x) + M_t(x)V_t(x),$$

with $M_t(x) = \sum_k M_t^k(x)$ (normalized to $[0, 1]$) and V_t the rendered virtual content. The system is designed to satisfy end-to-end latency $T_{\text{total}} \leq T_{\text{frame}}$ (targeting below 20–30 ms) while ensuring temporal stability and geometric alignment between real and virtual elements.

The main contributions of this work are as follows:

- A formally defined approach to region-level visual substitution for mixed reality training systems, enabling selective replacement of semantically meaningful regions.
- A modular real-time architecture with explicit separation of perception, scene representation, and rendering layers, following a dataflow-driven design.
- A GPU-centric pipeline for low-latency semantic segmentation and mask-based compositing.
- An analytically grounded performance model for evaluating latency and concurrency in the pipeline.
- A prototype implementation and preliminary experimental evaluation under different hardware configurations.
- A design-science framework that combines analytical validation with a pathway toward future empirical studies.

This study aims to contribute a technically grounded foundation for hybrid mixed reality systems in pilot training.

2 Related Work

Immersive technologies have been increasingly explored to enhance pilot training, yet most solutions still fail to simultaneously provide high sensorimotor fidelity, fine-grained control over the visual environment, and real-time semantic adaptability. This section critically reviews Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR) approaches, with particular emphasis on their limitations regarding region-level visual substitution inside a physical cockpit.

2.1 Virtual Reality in Pilot and Drone Training

VR-based simulators offer excellent scenario control and have shown benefits in procedural learning and cost reduction [6, 10, 4, 16, 18]. Systems have been developed for both manned aircraft [4, 16] and drone pilot training [1]. Systematic reviews confirm that while VR improves immersion and allows training of rare events, the complete detachment from physical cockpit hardware often weakens the transfer of procedural and muscle memory [18, 10]. In contrast to the proposed architecture, VR operates at a full-scene level and cannot preserve the tangible interaction with real controls and instruments.

2.2 Augmented Reality for Instructional Support

AR solutions enhance the real cockpit with overlaid digital information, demonstrating improvements in situational awareness, procedural performance, and cognitive load reduction [2, 13, 17]. However, because AR is limited to additive overlays, it cannot replace or modify existing visual elements such as instrument displays or the external view through windows. This fundamental constraint restricts its use for dynamic failure injection and realistic scenario adaptation [13, 11]. The proposed region-level substitution paradigm overcomes this limitation by enabling replacement rather than mere augmentation of semantically meaningful regions.

2.3 Mixed Reality and Video See-Through Systems

Mixed Reality systems that integrate physical cockpits with virtual content represent the most relevant prior work. Several recent implementations replace the external visual scene while preserving physical controls [7, 12, 20, 21]. Ernst et al. [7] developed a video see-through MR flight simulator, while Kimura et al. [12] and Zintl et al. [20, 21] focused on instructor operator stations and motion-based MR simulators for advanced air mobility (eVTOL). Other studies explored MR applications in general aviation [11] and multi-user training scenarios [9].

Despite these contributions, the majority of existing MR systems remain limited in scope: they primarily substitute only the external view and leave cockpit instrumentation unchanged. Video see-through approaches [7, 19] enable deeper visual manipulation but are rarely combined with semantic understanding for selective, context-aware region substitution. Most implementations lack a modular architecture that explicitly separates perception, structured scene representation, and GPU-accelerated compositing. Consequently, instructors cannot dynamically modify display content or inject instrument failures without physical modifications to the cockpit.

Human factors research further highlights challenges such as latency, registration errors, cybersickness, and limited instructor control in current MR aviation simulators [8, 5, 14, 15]. Systematic reviews of XR in pilot training [4, 18, 16] consistently emphasise the need for better integration of real and virtual elements while maintaining physical fidelity – a gap the present work directly addresses.

2.4 Positioning of the Proposed Architecture

The proposed system advances beyond prior work by introducing a dedicated real-time architecture for *semantically grounded, region-level visual substitution.* Unlike full-scene VR solutions [6, 18], additive AR systems [2, 13], and most existing MR implementations that focus on external view replacement [7, 20, 12], this architecture enables selective replacement of both external windows and internal cockpit displays while preserving full physical interaction with controls and the pilot's hands.

Unlike prior VR-, AR-, and MR-based training systems, the proposed architecture combines semantically controlled region-level substitution, manipulation of both external and internal cockpit elements, and an explicit latency-aware analytical execution model within a unified real-time pipeline. Existing MR approaches primarily focus on replacement of the external visual scene, while cockpit instrumentation typically remains physically unchanged and semantically unmanaged.

By combining real-time semantic segmentation, structured scene representation, GPU-accelerated mask-based compositing, and an explicit latency-aware dataflow model, the presented design offers greater instructional flexibility and finer visual control than previous

Table 2 Comparison of the proposed architecture with representative immersive pilot training systems.

Work	External View	Internal Displays	Semantic Regions	Real-time	Analytical Model
VR simulators [4, 16, 1, 6]	✓	✗	✗	✓	✗
AR cockpit systems [2, 13, 19, 17]	✓	✓	✗	✓	✗
MR view substitution & blending [7, 12, 20, 21]	✓	✓	✗partial	✓	✗partial
General MR/AR aviation training [11, 9]	✓	✓partial	✗	✓	✗
Proposed architecture	✓	✓	✓	✓	✓

systems. Furthermore, the integration of analytical performance modelling with prototype validation distinguishes this contribution from predominantly conceptual or purely empirical studies in the field [5, 8, 3].

3 System Architecture

The proposed system implements a real-time mixed reality pipeline for selective region-level visual substitution in a pilot training environment. The architecture is designed as a modular dataflow system that jointly processes visual perception, scene representation, and GPU-based rendering under strict latency constraints.

Let the input at time t be defined as a multimodal tuple:

$$F_t = (I_t, P_t, G_t), \tag{1}$$

where I_t is the captured RGB frame, $P_t \in SE(3)$ is the head pose, and G_t is the gaze vector.

The system is formulated as a sequential dataflow pipeline:

$$F_t \to C_t \to M_t \to S_t \to \tilde{S}_t \to V_t \to O_t, \tag{2}$$

where C_t denotes capture, M_t semantic perception output, S_t structured scene representation, $\tilde{S}_t$ substitution-aware representation, V_t rendered virtual content, and O_t the final composited output. Compositing is included within the output stage O_t.

We distinguish between two latency models: the ideal sequential latency and the execution-aware latency under pipeline parallelism.

The ideal end-to-end latency is defined as:

$$T_{total} = T_{cap} + T_{seg} + T_{scene} + T_{subst} + T_{rend} + T_{disp}. \tag{3}$$

Here:

- T_{cap} – image acquisition latency,
- T_{seg} – semantic segmentation latency,
- T_{scene} – structured scene construction,
- T_{subst} – substitution reasoning and mask fusion,
- T_{rend} – GPU rendering of virtual content,
- T_{disp} – compositing, scan-out, and XR display synchronization.

Due to asynchronous execution and GPU/CPU overlap, the effective latency is defined as:

$$T_{eff} = \max(T_{seg}, T_{rend}) + T_{subst} + T_{disp}. \tag{4}$$

This formulation reflects the critical-path behavior of the system under parallel execution.

Real-time constraints require:

$$T_{eff} \leq 33\,ms. \tag{5}$$

Temporal stability is evaluated over a sliding window of N frames as:

$$\sigma_T = \mathrm{Var}(\{T_{total}^{(i)}\}_{i=1}^{N}), \tag{6}$$

capturing system jitter over time.

The substitution module transforms semantic perception into render-ready control signals. It performs:

- semantic-to-render mapping,
- mask fusion and normalization,
- overlap resolution between regions,
- gaze- and context-aware weighting.

Semantic perception output is defined as a structured set of region masks:

$$M_t = \{R_t^1, R_t^2, \ldots, R_t^k\}, \tag{7}$$

where each R_t^i is a binary or probabilistic region mask corresponding to a detected semantic class or instance.

The substitution-aware representation is defined as a constructive mapping:

$$\tilde{S}_t = \mathcal{F}(M_t, S_t, G_t), \tag{8}$$

where $\mathcal{F}$ is a rule-based or learned fusion operator integrating spatial, semantic, and gaze-driven constraints.

The final compositing operation is defined as:

$$O_t(x) = (1 - \alpha_t(x))I_t(x) + \alpha_t(x)V_t(x), \tag{9}$$

with blending mask $\alpha_t(x)$ constrained to semantically valid regions:

$$\mathrm{supp}(\alpha_t) \subseteq \bigcup_i R_t^i. \tag{10}$$

The system operates under a hybrid execution model:

- GPU: segmentation, rendering, compositing,
- CPU: control logic and scene management,
- XR runtime: tracking and display synchronization.

The architecture enables pipeline overlap between perception and rendering stages, improving throughput under real-time constraints.

The architecture separates perception, representation, substitution, rendering, and compositing into modular layers, enabling real-time performance while supporting semantically controlled region-level visual manipulation of the egocentric field of view.

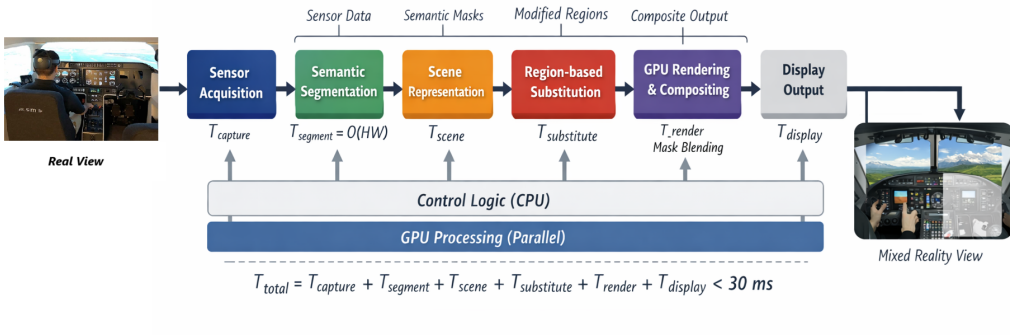


Figure 1 Latency-aware dataflow architecture of the proposed mixed reality substitution system.

4 Evaluation and Experimental Results

To address the limitations of purely conceptual validation frequently observed in mixed reality training research, the proposed architecture was evaluated using a reproducible real-time prototype implementation developed in Python with GPU-accelerated rendering and semantic segmentation. The complete benchmarking framework, including automated latency profiling and execution logging, is publicly available at `https://github.com/prykhodchenkosd/MRPilotRealtimeTest`. The evaluation focuses on validating the practical feasibility of region-level visual substitution under real-time constraints and on quantifying the impact of hardware acceleration on temporal stability and overall system responsiveness.

Three execution configurations were evaluated: a lightweight baseline pipeline without computationally intensive semantic inference, a CPU-based YOLO segmentation pipeline, and a GPU-accelerated YOLO pipeline executed on an NVIDIA RTX 3060 GPU. All configurations used the same rendering and compositing architecture to ensure direct comparability. Measurements were collected under continuous streaming conditions using frame-wise profiling of the capture, segmentation, rendering, and compositing stages. For each stage, both the mean execution time and the standard deviation were recorded to characterize not only throughput but also temporal stability, which is critical for perceptual consistency in immersive mixed reality systems.

The baseline configuration demonstrates the minimum processing overhead achievable by the proposed architecture under simplified semantic conditions. In this mode, the system achieves an average total latency of 27.92 ms and an average frame rate of 36.96 FPS. The segmentation stage contributes only 3.94 ms on average, while compositing remains the dominant computational stage due to image blending and synchronization overhead. Despite the absence of full semantic processing, this configuration establishes a lower-bound reference for the architectural pipeline itself and confirms that the rendering and compositing framework can operate within interactive real-time constraints.

The CPU-based YOLO configuration introduces substantially higher computational cost and temporal instability. Semantic segmentation becomes the dominant bottleneck, with an average latency of 30.43 ms and a large standard deviation of 48.29 ms. This variability propagates through the execution pipeline and results in a total average latency of 64.79 ms with a high temporal variance of 70.52 ms. The corresponding frame rate decreases to 25.36 FPS, approaching the lower acceptable boundary for interactive mixed reality applications. The obtained measurements indicate that CPU-only execution is insufficient for stable region-level visual substitution in semantically driven mixed reality scenarios, particularly under continuous streaming conditions where latency spikes negatively affect temporal coherence.

In contrast, the GPU-accelerated configuration executed on an NVIDIA RTX 3060 demonstrates a substantial reduction in both latency and temporal variance across all stages of the pipeline. Segmentation latency decreases to 6.85 ms with low variability, while GPU-based compositing reduces compositing overhead to 4.12 ms. The overall end-to-end latency decreases to 26.30 ms, satisfying the target real-time constraint defined for immersive mixed reality interaction. The achieved frame rate of 38.12 FPS confirms stable interactive operation, while the significantly reduced standard deviation of total latency (3.85 ms) indicates reduced latency variability and smoother temporal behavior. These results confirm that GPU acceleration not only improves throughput but also substantially stabilizes frame delivery, which is essential for perceptual coherence and user comfort in mixed reality environments.

Segmentation quality was evaluated through systematic visual inspection of the output frames. The system produced stable and well-defined region masks for both windows and cockpit displays under controlled conditions, enabling coherent real-time visual substitution. Although a formal quantitative assessment using annotated datasets (e.g., mean IoU across diverse lighting conditions) is planned for future iterations, the current perception pipeline already satisfies the architectural and performance requirements of the proposed mixed reality system.

Across all evaluated configurations, the results therefore validate the feasibility of the proposed architecture as a real-time mixed reality substitution system capable of semantically controlled region-level manipulation while preserving interactive responsiveness.

Table 3 Performance evaluation of the proposed mixed reality pipeline under different execution configurations.

Configuration	Capture (ms)	Segmentation (ms)	Rendering (ms)	Compositing (ms)	Total (ms)	FPS
Simple Test	10.22 ± 5.53	3.94 ± 1.05	3.33 ± 0.60	10.43 ± 0.99	27.92 ± 5.50	36.96 ± 6.09
YOLO on CPU	1.23 ± 1.70	30.43 ± 48.29	7.27 ± 1.02	17.52 ± 1.46	64.79 ± 70.52	15.4 ± 7.43
YOLO on GPU (RTX 3060)	1.21 ± 1.84	6.85 ± 1.12	6.42 ± 0.78	4.12 ± 0.65	26.30 ± 3.85	38.12 ± 4.71

5 Limitations and Discussion

Although the proposed architecture and prototype implementation demonstrate the feasibility of real-time region-level visual scene substitution, several limitations should be acknowledged. First, the current evaluation was conducted using a simplified desktop-based prototype with a standard RGB camera and static virtual content rather than within a realistic cockpit simulator or immersive head-mounted display environment. As a result, the reported performance metrics, including the average end-to-end latency of 26.3 ms on an NVIDIA RTX 3060 GPU, reflect controlled laboratory conditions and may not directly generalize to operational training scenarios involving rapid head motion, complex illumination conditions, and dynamic cockpit interactions.

Second, the perception pipeline currently relies on relatively lightweight colour-based and YOLO-based semantic segmentation models. While these models are sufficient for validating the architectural feasibility of real-time region-level substitution, their robustness

under realistic cockpit conditions has not yet been systematically evaluated. In particular, illumination variability, reflections from cockpit surfaces, motion blur, and partial occlusions caused by the pilot's hands or body may affect segmentation stability and temporal consistency. Improving robustness and domain adaptation of semantic perception therefore remains an important direction for future research.

Third, the present study primarily focuses on architectural design, analytical modelling, and preliminary technical validation. The proposed system has not yet been evaluated through controlled user studies involving pilots or flight instructors. Consequently, the work does not currently provide empirical evidence regarding training effectiveness, skill transfer, cognitive workload, situational awareness, or instructor usability. These aspects require dedicated human-subject experiments and represent a critical next step toward practical deployment.

From a methodological perspective, pixel-level segmentation accuracy metrics such as Intersection over Union (IoU) are not reported in this study, as the current prototype is not built upon a fully annotated ground-truth dataset. Instead, the focus is placed on real-time system performance, architectural feasibility, and temporal characteristics of the pipeline. A dataset-driven evaluation with formally defined ground truth annotations is left for future work, where more comprehensive benchmarking of perception accuracy will be required.

The proposed approach also inherits several limitations commonly associated with video see-through mixed reality systems. These include reduced visual fidelity compared to optical see-through displays, increased computational and energy requirements, and perceptual issues such as the vergence–accommodation conflict. Furthermore, although the GPU-accelerated implementation achieves interactive real-time performance, maintaining consistently low latency on untethered or resource-constrained hardware platforms remains a significant engineering challenge.

Despite these limitations, the presented work establishes a reproducible and analytically grounded foundation for semantically controlled region-level substitution in mixed reality pilot training. The modular architecture, explicit latency-aware design, and publicly available prototype framework provide a basis for further experimental development and comparative evaluation. Future work will focus on integration with immersive head-mounted displays and realistic cockpit environments, development of more robust domain-adapted perception models, incorporation of advanced occlusion and depth handling, and comprehensive user studies aimed at assessing training transfer, usability, and instructional effectiveness.

6 Conclusion and Future Work

This paper presented the architectural design, formalization, and preliminary technical validation of a mixed reality system for pilot training based on selective region-level visual scene substitution. The proposed approach enables context-aware replacement of semantically meaningful regions, such as cockpit displays and external windows, while preserving direct interaction with physical cockpit controls and the pilot's hands.

A modular latency-aware architecture was developed that integrates semantic perception, structured scene representation, and GPU-accelerated mask-based compositing within a real-time dataflow pipeline. The work additionally introduced an analytical execution model describing the relationship between segmentation, rendering, compositing, and overall end-to-end latency under parallel execution constraints.

A functional prototype implementation was evaluated under multiple execution configurations. Experimental results indicate that GPU acceleration substantially improves both throughput and temporal stability compared to CPU-only execution. In the evaluated

configuration using an NVIDIA RTX 3060 GPU, the system achieved an average end-to-end latency of 26.3 ms with an average frame rate of approximately 38 fps, satisfying the targeted real-time interaction requirements under controlled laboratory conditions.

The presented results should be interpreted as preliminary technical validation of the architectural feasibility of the proposed approach rather than a complete assessment of training effectiveness. The current prototype was evaluated using a desktop-based setup with a standard RGB camera and static virtual content. Formal user studies involving pilots or flight instructors, as well as quantitative perception benchmarks using annotated datasets, were outside the scope of the present work.

Nevertheless, the study demonstrates that semantically controlled region-level substitution can be implemented within interactive real-time constraints using a modular GPU-oriented mixed reality pipeline. The proposed architecture and publicly available prototype framework provide a reproducible technical basis for further research on hybrid mixed reality training systems.

Future work will focus on integration with immersive video see-through head-mounted displays and realistic cockpit mock-ups, improvement of segmentation robustness under challenging lighting and motion conditions, incorporation of depth-aware occlusion handling, and comprehensive experimental evaluation with pilots and instructors. Additional directions include quantitative segmentation benchmarking, comparative studies against existing XR training paradigms, and exploration of collaborative multi-user training scenarios.

References

1 G. Albeaino, R. Eiris, M. Gheisari, and R. Issa. Dronesim: a VR-based flight training simulator for drone-mediated building inspections. *Construction Innovation*, 2021. doi:10.1108/ci-03-2021-0049.

2 D. Arjoni, I. De Souza Rehder, J. Figueira, and E. Villani. Augmented reality for training formation flights: An analysis of human factors. *Heliyon*, 9, 2023. doi:10.1016/j.heliyon.2023.e14181.

3 R. Bödding, S. Schriek, and G. Maier. A systematic review and meta-analysis of mixed reality in vocational education and training: examining behavioral, cognitive, and affective training outcomes and possible moderators. *Virtual Reality*, 29, 2025. doi:10.1007/s10055-025-01118-z.

4 J. Cross, C. Boag-Hodgson, T. Ryley, T. Mavin, and L. Potter. Using extended reality in flight simulators: A literature review. *IEEE Transactions on Visualization and Computer Graphics*, 29:3961–3975, 2022. doi:10.1109/tvcg.2022.3173921.

5 J. Cross and T. Ryley. Assessing evidence-based training in a collaborative virtual reality flight simulator. *The Aeronautical Journal*, 2024. doi:10.1017/aer.2024.113.

6 P. Dymora, B. Kowal, M. Mazurek, and S. Romana. The effects of virtual reality technology application in the aircraft pilot training process. *IOP Conference Series: Materials Science and Engineering*, 1024, 2021. doi:10.1088/1757-899x/1024/1/012099.

7 J. Ernst, T. Laudien, and S. Schmerwitz. Implementation of a mixed-reality flight simulator: blending real and virtual with a video-see-through head-mounted display. In *Proceedings of SPIE*, volume 12538, pages 125380R–125380R–10, 2023. doi:10.1117/12.2664848.

8 S. Fussell, S. Rebensky, and S. Mcgee. Human factors challenges for extended reality aviation training simulation. In *AHFE International*, 2025. doi:10.54941/ahfe1006386.

9 Y. Go, H. Kang, J. Lee, M. Yu, and S. Choi. Multi-user drone flight training in mixed reality. *Electronics*, 2021. doi:10.3390/electronics10202521.

10 R. Guthridge and V. Clinton-Lisell. Evaluating the efficacy of virtual reality (vr) training devices for pilot training. *Journal of Aviation Technology and Engineering*, 2023. doi:10.7771/2159-6670.1286.

11 Christopher Katins, Sebastian S. Feger, and Thomas Kosch. Exploring mixed reality in general aviation to support pilot workload. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI EA '23, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3544549.3585742.

12 S. Kimura, M. Zintl, C. Hammann, and F. Holzapfel. Simulator-based mixed reality evtol pilot training: The instructor operator station. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 2024. doi:10.1145/3613904.3642060.

13 B. Moesl, H. Schaffernak, W. Vorraber, R. Braunstingl, and I. Koglbauer. Performance, emotion, presence: Investigation of an augmented reality-supported concept for flight training. *Applied Sciences*, 2023. doi:10.3390/app132011346.

14 A. Perez, A. Singh, J. Mitchell, and P. Swadling. Strategies to manage human factors in mixed reality pilot training: A survey. arXiv preprint arXiv:2507.15526, 2025. doi:10.48550/arxiv.2507.15526.

15 S. Rizvi, U. Rehman, S. Cao, and B. Moncion. Exploring technology acceptance of flight simulation training devices and augmented reality in general aviation pilot training. *Scientific Reports*, 15, 2025. doi:10.1038/s41598-025-85448-7.

16 G. Ross and A. Gilbey. Extended reality (xr) flight simulators as an adjunct to traditional flight training methods: a scoping review. *CEAS Aeronautical Journal*, 14:799–815, 2023. doi:10.1007/s13272-023-00688-5.

17 H. Schaffernak, B. Moesl, W. Vorraber, M. Holy, E. Herzog, R. Novak, and I. Koglbauer. Novel mixed reality use cases for pilot training. *Education Sciences*, 2022. doi:10.3390/educsci12050345.

18 A. Somerville, K. Joiner, T. Lynar, and G. Wild. Applications of extended reality in pilot flight simulator training: a systematic review with meta-analysis. *Visual Computing for Industry, Biomedicine, and Art*, 8, 2025. doi:10.1186/s42492-025-00206-w.

19 J. Wallace, Z. Hu, and D. Carroll. Augmented reality for immersive and tactile flight simulation. *IEEE Aerospace and Electronic Systems Magazine*, 35:6–14, 2020. doi:10.1109/maes.2020.3002000.

20 M. Zintl, S. Kimura, and F. Holzapfel. A mixed reality research flight simulator for advanced air mobility vehicles. *AIAA AVIATION FORUM AND ASCEND 2024*, 2024. doi:10.2514/6.2024-4651.

21 M. Zintl, S. Kimura, and F. Holzapfel. Development and human-centered evaluation: A motion-based mixed-reality flight simulator. *Journal of Aerospace Information Systems*, 2026. doi:10.2514/1.i011738.

Catching What the Borrow Checker Can't: Towards Serious Game-Based Security Training for Rust Developers

Frederico d'Abreu
Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Tiago Espinha Gasiba
Siemens AG, Munich, Germany

Sathwik Amburi
Siemens AG, Munich, Germany

Maria Pinto-Albuquerque
ISTAR, Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Abstract

Cybersecurity incidents cause significant financial damage to organizations, making secure software development a critical industry priority. Although Rust offers strong memory safety guarantees, developers can still introduce security vulnerabilities through improper coding practices. In this paper, we present the preliminary design of a serious game for secure Rust coding training in an industrial setting. We investigate real-world Rust vulnerabilities, map (five of) them to CWEs, and propose challenge scenarios alongside a structured evaluation protocol for validation by industry security experts. Our design incorporates defense-oriented coding tasks, a three-level AI-powered hint system, and automated backend evaluation, grounded in established pedagogical principles such as situated learning and cognitive load management. This work provides a foundation for future implementation and empirical evaluation of learning outcomes with professional software developers.

2012 ACM Subject Classification Security and privacy → Software and application security; Security and privacy → Software security engineering; Social and professional topics → Computing education; Social and professional topics → Computing education programs

Keywords and phrases Rust, Secure Coding Training, Secure Software Development, Serious Games, Gamification, Industry

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.16

Category Short Paper

Funding This work was partially supported by national funds through Fundação para a Ciência e Tecnologia, I.P. (FCT), ISTAR-Iscte Pluriannual Project UID/04466/2025 (`https://doi.org/10.54499/UID/04466/2025`).

Acknowledgements The authors would like to thank the industrial partners who supported this research.

1 Introduction

Cybersecurity incidents represent one of the most significant threats to modern organizations, frequently resulting in substantial financial losses and operational disruptions. In response, industry has established standards such as IEC 62443 that mandate secure development practices throughout the software lifecycle [7]. Despite advances in automated detection, developers remain responsible for writing secure code, and their security awareness directly impacts system resilience [13]. This human factor makes developer education a critical component of any security strategy. Moreover, recent research suggests that reliance on generative AI tools reduces developers' critical thinking effort [10], reinforcing the need for training that actively engages practitioners in reasoning about code security.

Serious games-interactive, game-based learning experiences-offer a promising approach to raising security awareness among developers [8], with challenge-based formats, such as Capture-the-Flag competitions, effectively engaging developers in industrial settings [8, 15].

Rust is a systems programming language whose ownership model, borrow checker, and type system eliminate entire classes of vulnerabilities common in C and C++ [9, 2]. However, developers can still introduce security issues through misuse of `unsafe` blocks, logic errors, improper input validation, or flawed concurrency patterns [16].

In this work, we focus on the human factor in secure Rust development by designing a serious game that trains industrial developers in Rust-specific secure coding. Our contributions are: (1) the design of secure coding challenges based on real-world Rust vulnerabilities mapped to CWEs, and (2) a structured evaluation protocol for validating the challenge design with industry security experts.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes our methodology. Section 4 presents the challenge design and planned evaluation protocol. Section 5 discusses design considerations, limitations, and future work. Section 6 concludes the paper.

2 Related Work

Prior work relevant to this paper spans three main areas: secure coding education and training needs, serious games and challenge design for technical learning, and Rust-specific software security.

Research on secure coding training shows that developers benefit from combining conceptual material with practical application [12]. Studies confirm that security knowledge directly affects software quality and that developer preparedness remains a concern [13, 17]. Furthermore, growing reliance on generative AI in development workflows has been shown to reduce cognitive effort in critical thinking tasks [10], highlighting the importance of training that requires active problem-solving. Large-scale surveys confirm that developers recognize gaps in their security knowledge [7], motivating the adoption of hands-on training approaches.

Serious games have been studied extensively as vehicles for effective learning, with systematic reviews identifying clear objectives, immediate feedback, difficulty progression, narrative context, and interactivity as important factors [11]. Prior work emphasizes aligning game design with professional tasks [4], supports game-based approaches for cybersecurity education [1], and validates structured, defense-oriented exercises in industrial settings [8]. Hybrid work environments have motivated flexible deployment of serious games [15], and using real-world vulnerabilities as a basis for exercises enhances engagement and perceived relevance [3].

Rust provides strong safety guarantees through its type system and ownership model, yet important security risks remain. Ecosystem-level analyses show that vulnerabilities in Rust crates are not uncommon, including logic errors, improper input validation, cryptographic misuse, and memory safety violations [16, 2]. These findings indicate that secure Rust development depends on developer understanding beyond what the compiler can enforce.

Despite the growing body of work on secure coding education, serious games, and Rust security, there is a lack of defense-based, interactive serious game challenges designed specifically for Rust secure coding training in industrial settings. The present work addresses this gap by transforming real-world Rust vulnerabilities into gamified challenges for professional developers [5, 6].

3 Methodology

Our methodology builds upon previous work on secure coding challenge design for industry [8, 4] and extends it to the Rust programming language. The overall approach consists of three main phases: (1) investigation of Rust security risks from literature and practice, (2) design of challenge scenarios and game structure, and (3) design of an evaluation questionnaire for industrial security experts.

In the first phase, we reviewed literature on security risks in the Rust ecosystem, particularly ecosystem-level analyses of vulnerabilities in Rust crates [16], complemented by the authors' practical experience with Rust in industrial contexts. For each vulnerability pattern, we assessed suitability for a challenge format based on: (a) relevance to typical Rust development tasks, (b) whether it can be reasonably simplified without losing educational value, and (c) amenability to a defense-oriented format where the developer must fix or prevent the issue.

In the second phase, based on the identified vulnerability patterns, we developed five challenge scenarios covering a range of vulnerability types and difficulty levels. For the storyline format, challenges are contextualized within a fictional company setting where the developer assumes the role of a security engineer receiving tickets describing codebase issues. Each ticket corresponds to a real vulnerability pattern, providing narrative motivation while maintaining technical authenticity.

In a third phase, to validate the approach, we plan to engage industry security experts to evaluate the proposed scenarios on criteria including technical accuracy, relevance, educational value, and engagement potential. The methodology follows an iterative approach: after initial evaluation, the design will be adjusted based on feedback, and subsequent reviews will assess the refined challenges. In this paper, we present our intended evaluation criteria, based on our experience in designing serious games, and teaching secure software development in an industrial context.

4 Results

In the current section we present the results obtained throughout the three phases, as described in the methodology section.

4.1 Rust Security Risks

From the vulnerability categories identified by Zheng et al. [16] and our industrial experience, we selected four CWEs that met all three selection criteria: CWE-190 (integer overflow), as arithmetic errors in Rust's `unsafe` contexts can bypass default overflow checks; CWE-125/CWE-787 (out-of-bounds read/write), representing memory safety violations common in sandbox and FFI code; CWE-20 (improper input validation), a pervasive issue even in safe Rust; and CWE-703 (improper handling of exceptional conditions), reflecting error handling patterns that lead to denial-of-service or undefined states. Table 1 presents the resulting five challenges.

4.2 Design of the Challenges

Based on the identified vulnerability patterns, we design interactive challenges with the following characteristics. Drawing on our experience and success designing serious games, we adopt a defense-oriented, code-entry format. The developer is presented with a vulnerable

Table 1 Mapping of challenges to CWEs, types, and difficulty levels.

ID	Challenge	Type	Difficulty	CWE(s)
C1	Integer Overflow	Individual	Easy	CWE-190
C2	Sandbox Breach	Individual	Medium	CWE-125 | CWE-787
C3	Input Validation	Storyline	Easy	CWE-20
C4	Improper Handling	Storyline	Easy	CWE-703
C5	Combined Vulnerabilities	Storyline	Medium	CWE-20 | CWE-703

Rust code snippet and must modify it to eliminate the vulnerability while preserving functionality. Challenges are delivered through Visual Studio Code (VSC), and submissions are evaluated by a backend pipeline combining unit tests and static analysis (e.g., `cargo clippy`) to verify correctness and security properties. Figure 1 illustrates the platform architecture.

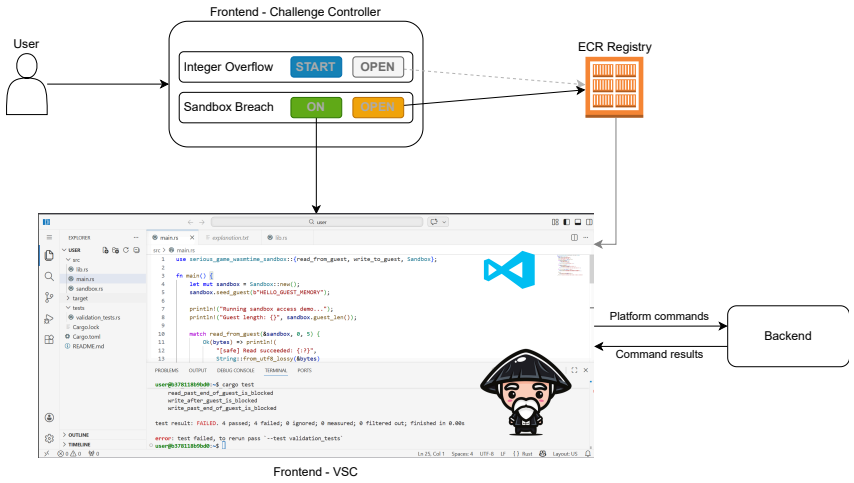


Figure 1 Architecture of the platform where challenges are deployed.

To illustrate, Listing 1 shows a simplified excerpt from challenge C2 (CWE-125/CWE-787), where the bounds check is insufficient-it verifies only the requested length without validating that the computed range stays within guest memory. Listing 2 shows the corrected version using safe bounds validation.

Listing 1 Vulnerable code: insufficient bounds check allows out-of-bounds read.

```
pub fn read_from_guest(
    sandbox: &Sandbox,
    offset32: u32,
    len: usize,
) -> Result<Vec<u8>, AccessError> {
    if len == 0 {
        return Err(AccessError::InvalidLength);
    }
    // Only checks length, not the full range
    if len > sandbox.guest_len() {
        return Err(AccessError::OutOfBounds);
    }
```

```
    let computed = sandbox.guest_base()
        .wrapping_add(offset32 as usize);
    unsafe {
        let ptr = sandbox.host_memory()
            .as_ptr().add(computed);
        let slice =
            std::slice::from_raw_parts(ptr, len);
        Ok(slice.to_vec())
    }
}
```

Listing 2 Corrected code: validates full range and avoids unsafe access.

```
pub fn read_from_guest(
    sandbox: &Sandbox,
    offset32: u32,
    len: usize,
) -> Result<Vec<u8>, AccessError> {
    if len == 0 {
        return Err(AccessError::InvalidLength);
    }
    let offset = offset32 as usize;
    let end = offset.checked_add(len)
        .ok_or(AccessError::OutOfBounds)?;
    if end > sandbox.guest_len() {
        return Err(AccessError::OutOfBounds);
    }
    Ok(sandbox.guest_slice()[offset..end].to_vec())
}
```

Since multiple valid fixes may exist for a given vulnerability, the backend evaluation relies on purpose-built unit tests and static analysis rather than exact output matching, allowing flexible acceptance of different correct solutions.

Based on previous work on scaffolding in secure coding challenges [8], we design an AI-powered hint system that dynamically generates personalized guidance at three progressive levels:

Level 1 – Conceptual: A general nudge towards the relevant secure coding principle, without referencing specific code.

Level 2 – Directional: Narrows focus to the relevant code area, suggesting what aspect to reconsider.

Level 3 – Explicit: A concrete indication of the fix, referencing the relevant Rust API or pattern, stopping short of the complete solution.

To illustrate, for challenge C2 (Sandbox Breach), the three levels might be:

L1: "Think about what happens when you trust a length value without considering where the access actually starts."

L2: "The bounds check might not be enough – consider whether the offset plays a role in the validity of the access."

L3: "You need to validate the full range (offset + length) against the buffer size, and watch out for arithmetic overflow when computing it."

Table 2 Expert evaluation dimensions.

Challenge Framing	Dimension	Statement for Evaluation
Individual	CVE Fidelity	The challenge adequately represents the associated CVE
	Standalone Clarity	The challenge can be understood without external context
	Real-world Representativeness	The challenge represents a real-world scenario
Storyline	Narrative Coherence	The storyline provides a coherent and believable context
	Inter-challenge Progression	The challenge fits logically within the storyline progression
	Contextual Added Value	The narrative context adds value to the learning experience
Both	Difficulty Appropriateness	The difficulty level is appropriate for the target audience
	CWE Coverage Adequacy	The challenges cover a sufficient range of vulnerability types
	Hint Quality	The hint system is helpful without revealing the solution
	Industry Relevance	The challenge reflects realistic industry scenarios
	Learning Potential	The challenge has potential to improve secure coding skills
	Engagement Potential	The challenge is engaging and motivating to solve

The implementation and validation of AI-generated hints constitutes a next step in this research.

A persistent "goal summary" command is always available, allowing the developer to revisit challenge objectives at any point. Upon submission, the developer receives immediate feedback on whether the solution passes security and functionality checks. After successful completion, a technical explanation is provided, including the canonical solution and why the original code was vulnerable. Unit tests are visible, ensuring transparency in evaluation criteria.

Challenges are organized into two difficulty levels: *easy* challenges target single, well-known vulnerabilities in isolation. In contrast, *medium* challenges introduce moderate complexity and may combine up to two vulnerability types in a single exercise. This progression reflects cognitive load management principles, presenting learners with incremental complexity. Similarly, the narrative framing of storyline challenges grounds tasks in authentic workplace scenarios (situated learning), while the scaffolded hint system reduces extraneous cognitive load by guiding attention progressively [14].

4.3 Expert Evaluation

To validate the challenge design, we define a structured evaluation protocol to be conducted with industry security experts from a large multinational technology company, targeting experts with experience in software security and familiarity with Rust development. This evaluation protocol is designed based on our experience in the design and evaluation of serious games for programming education in an industrial context.

Table 2 presents the designed evaluation framework, assessed on a 5-point Likert scale. This framework is used to evaluate the design of each challenge over different dimensions, grouped by three framings: *individual*, *storyline*, and *both*. Note that the CWE Coverage Adequacy dimension is assessed at the challenge set level rather than per challenge.

The dimensions capture both format-specific qualities (e.g., fidelity to source CVEs for individual challenges, narrative coherence for storyline challenges) and cross-cutting concerns (e.g., industry relevance, hint quality). The expected answers are on a 5-point Likert scale from 1 to 5, with 1 representing strong disagreement and 5 representing strong agreement with the statement.

5 Discussion

A fundamental premise of this work is that industrial developers have already mastered programming fundamentals, allowing the challenge design to focus exclusively on security and present sophisticated vulnerability patterns without requiring foundational instruction

on language mechanics. Previous work in academic settings [5] has shown that students often struggle with the language itself before engaging with security content; our work targets a population where this barrier does not exist.

The design process revealed several practical considerations. First, grounding challenges in real vulnerability patterns (via CWE mappings and adaptation to CVE) provides a concrete link between training and production concerns, which we expect to increase perceived relevance. Second, the distinction between storyline and individual formats emerged as a meaningful design axis: storyline challenges offer narrative context and progressive complexity, while individual challenges offer focused exercises with clear traceability to real incidents. Whether one format leads to better learning outcomes remains an open empirical question.

Relative to prior work on serious games for secure coding in C/C++ [4, 8], this work extends prior approaches in two ways: (1) challenge design tailored to Rust's ecosystem, targeting vulnerabilities that escape the compiler's static guarantees; and (2) the proposed use of an AI-powered hint system for personalized, context-sensitive guidance rather than static pre-authored hints.

Finally, we note that the challenge selection was informed by a simple study and the authors' experience which may introduce a certain bias. The current set of challenges covers only two difficulty levels, limiting the assessment of skills. Further, the evaluation protocol has not yet been executed; its results will inform iterative refinement of the challenge design.

In future work, we will conduct expert evaluation as described in Section 4.3 and refine the challenges. Subsequently, both storyline and individual formats will be deployed on the platform described in Figure 1, and we plan to compare their effectiveness through a developer survey measuring learning outcomes, engagement, and perceived relevance.

6 Conclusion

This paper presented the preliminary design of a serious game for secure Rust coding training in industrial settings. We mapped Rust vulnerability patterns to CWEs and developed challenge scenarios grounded in real-world issues. The design incorporates defense-oriented tasks, an AI-powered three-level hint system, automated toolchain-based evaluation, and progressive difficulty. Additionally, we defined a structured evaluation protocol for validation by industry security experts.

This work provides a foundation for expert evaluation, iterative refinement, and subsequent empirical comparison of both challenge formats, storyline-based and individual, with developer surveys constituting a later phase of this research.

References

1 Christopher J Berisford, Leighton Blackburn, Jennifer M Ollett, Thomas B Tonner, Chun San Hugh Yuen, Ryan Walton, and Olakunle Olayinka. Can gamification help to teach cybersecurity? In *2022 20th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–9, 2022. doi:10.1109/ITHET56107.2022.10031716.

2 Mohan Cui, Shuran Sun, Hui Xu, and Yangfan Zhou. Is unsafe an achilles' heel? a comprehensive study of safety requirements in unsafe rust programming. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3597503.3639136.

3 Denise Daniels, Joon Suk Lee, Hui Chen, and Kostadin Damevski. Utilizing real-world software vulnerabilities to enhance secure programming education. In *2024 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, 2024. doi:10.1109/FIE61694.2024.10893584.

4 Tiago Espinha Gasiba, Kristian Beckers, Santiago Suppan, and Filip Rezabek. On the requirements for serious games geared towards software developers in the industry. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 286–296, 2019. doi:10.1109/RE.2019.00038.

5 Tiago Espinha Gasiba, Andrei-Cristian Iosif, Santiago Suppan, Ulrike Lechner, and Maria Pinto-Albuquerque. Reflections on training next-gen industry workforce on secure software development. In *Proceedings of the 5th European Conference on Software Engineering Education*, ECSEE '23, pages 1–10, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3593663.3593665.

6 Tiago Espinha Gasiba and Ulrike Lechner. Raising secure coding awareness for software developers in the industry. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 141–143, 2019. doi:10.1109/REW.2019.00030.

7 Tiago Espinha Gasiba, Ulrike Lechner, Maria Pinto-Albuquerque, and Daniel Méndez. Is secure coding education in the industry needed? An investigation through a large scale survey. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 241–252, 2021. doi:10.1109/ICSE-SEET52601.2021.00034.

8 Tiago Gasiba, Ulrike Lechner, Maria Pinto-Albuquerque, and Alae Zouitni. Design of secure coding challenges for cybersecurity education in the industry. In Martin Shepperd, Fernando Brito e Abreu, Alberto Rodrigues da Silva, and Ricardo Pérez-Castillo, editors, *Quality of Information and Communications Technology*, pages 223–237, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-58793-2_18.

9 Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. Safe systems programming in rust. *Communications of the ACM*, 64(4):144–152, 2021. doi:10.1145/3418295.

10 Hao-Ping (Hank) Lee, Advait Sarkar, Lev Tankelevitch, Ian Drosos, Sean Rintel, Richard Banks, and Nicholas Wilson. The impact of generative AI on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers. In *CHI 2025*, April 2025. URL: https://dl.acm.org/doi/full/10.1145/3706598.3713778.

11 Werner Siegfried Ravyse, A. Seugnet Blignaut, Verona Leendertz, and Alex Woolner. Success factors for serious games to enhance learning: a systematic review. *Virtual Real.*, 21(1):31–58, March 2017. doi:10.1007/s10055-016-0298-4.

12 Ita Ryan, Utz Roedig, and Klaas-Jan Stol. Training developers to code securely: Theory and practice. In *Proceedings of the 2024 ACM/IEEE 4th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and 2024 IEEE/ACM Second International Workshop on Software Vulnerability*, EnCyCriS/SVM '24, pages 37–44, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3643662.3643956.

13 Ita Ryan, Utz Roedig, and Klaas-Jan Stol. Software security is your highest priority. Do your developers know that? *IEEE Security & Privacy*, PP:2–10, January 2026. doi:10.1109/MSEC.2026.3670641.

14 John Sweller, Jeroen J. G. van Merriënboer, and Fred Paas. Cognitive architecture and instructional design: 20 years later. *Educational Psychology Review*, 31(2):261–292, 2019. doi:10.1007/s10648-019-09465-5.

15 Tiange Zhao, Tiago Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. Thriving in the era of hybrid work: Raising cybersecurity awareness using serious games in industry trainings. *Journal of Systems and Software*, 210:111946, 2024. doi:10.1016/j.jss.2023.111946.

16 Xiaoye Zheng, Zhiyuan Wan, Yun Zhang, Rui Chang, and David Lo. A closer look at the security risks in the rust ecosystem. *ACM Trans. Softw. Eng. Methodol.*, 33(2), December 2023. doi:10.1145/3624738.

17 Shuqi Zuo, Qi Yang, Tianyi Gao, Zarin Tasnim Progga, and Jinjin Shen. How security coding knowledge impact software quality: An empirical study of stack overflow security issues. In *2025 25th International Conference on Software Quality, Reliability and Security (QRS)*, pages 727–738, 2025. doi:10.1109/QRS65678.2025.00076.

Bridging Generative AI and Gamification in Moodle: Design and Evaluation of GamiBot

José Carlos Paiva
Pedagogical Innovation Center, Polytechnic of Porto, Portugal

Ricardo Queirós
CRACS, INESC TEC & ESMAD, Polytechnic of Porto, Portugal

Raquel Simões de Almeida
CIR, ESS, Polytechnic of Porto, Portugal

Teresa Terroso
ID+, ESMAD, Polytechnic of Porto, Portugal

Mário Pinto
ID+, ESMAD, Polytechnic of Porto, Portugal

Abstract

While Learning Management Systems (LMS) like Moodle are ubiquitous in higher education, they often lack dynamic, personalized academic support. We present GamiBot, an intelligent pedagogical agent natively integrated into Moodle to bridge this gap. GamiBot combines Large Language Model (LLM) assistance, Retrieval-Augmented Generation (RAG) grounded in course materials, adaptive quizzes, and a structured gamification backend. To evaluate its effectiveness, we conducted a two-week pilot study across two higher education institutions. Results from telemetry (773 interactions) and student surveys demonstrated high system usability (74.6% positive sentiment) and strong chatbot performance (69.4%). Students heavily favored active self-regulation, dedicating 53% of interactions to generating adaptive quizzes rather than passive content summaries. Conversely, gamification elements received mixed feedback (31.7%), underscoring the need for technical refinement and high customizability. Ultimately, GamiBot suggests that integrating generative AI directly into existing LMS workflows can provide reliable, context-aware formative support that aligns with self-regulated learning strategies.

2012 ACM Subject Classification Applied computing → Education; Applied computing → E-learning; Human-centered computing → Interactive systems and tools; Computing methodologies → Artificial intelligence; Human-centered computing → HCI design and evaluation methods; Human-centered computing → Collaborative and social computing systems and tools

Keywords and phrases Generative AI, Educational chatbot, Pedagogical agent, Gamification, Moodle, Retrieval-augmented generation, Self-regulated learning

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.17

Supplementary Material
Text (Project Website): `https://inov-norte.github.io/gamibot-docs` [18]

Funding *Ricardo Queirós*: This work is funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the support UID/50014/2025 (`https://doi.org/10.54499/UID/50014/2025`).
Raquel Simões de Almeida: This work has been supported by Fundação para a Ciência e Tecnologia (FCT) through R&D Units funding (UID/5210/2025) `https://doi.org/10.54499/UID/05210/2025`.
Teresa Terroso: Funded by FCT – Fundação para a Ciência e a Tecnologia (UID/04057/2025, `https://doi.org/10.54499/UID/04057/2025`).

1 Introduction

Student engagement and self-regulated learning represent persistent challenges in higher education, particularly within technology-mediated environments. Learning Management Systems (LMS) such as Moodle have become cornerstones of institutional education, providing a centralized space for course materials, assessments, and communication [8]. Yet, despite their widespread adoption, these platforms often fall short in offering dynamic, personalized academic support. Students navigating complex course content must typically rely on static resources, delayed instructor feedback, or external tools that operate in isolation from their learning environment. This fragmentation between the learning environment and support tools creates friction that can undermine both motivation and academic performance [16].

In parallel, two prominent approaches have emerged to address engagement and support in digital learning: gamification and artificial intelligence (AI)-powered pedagogical agents. Gamification – the application of game design elements such as points, badges, and leaderboards to non-game contexts – has demonstrated potential for increasing student motivation and participation [10]. Meanwhile, AI chatbots and conversational agents have shown promise in providing on-demand tutoring, content clarification, and formative assessment at scale [4]. However, these two paradigms are rarely combined in a meaningful, integrated manner, and even more rarely deployed as native components of institutional LMS platforms. Most existing solutions remain either standalone applications or loosely coupled external tools, leaving a critical gap between the capabilities of modern generative AI and the institutional workflows in which students and teachers actually operate.

This paper addresses that gap by presenting GamiBot, an intelligent, gamified pedagogical chatbot designed for deep, native integration within Moodle. GamiBot combines large language model (LLM)-based assistance with retrieval-augmented generation (RAG) over course-specific materials, adaptive quiz generation, and a structured gamification layer – all delivered through a unified conversational interface embedded directly within the LMS. Rather than requiring students to switch between platforms or teachers to maintain separate tools, GamiBot situates AI-powered support within the institutional context where learning already occurs, keeping interactions anchored to official course content and pedagogical workflows. The project documentation and public website are available at `https://inov-norte.github.io/gamibot-docs`.

The system was architected around three interdependent objectives: delivering contextually grounded academic support through intelligent material summarization and instant answers to questions; enabling formative self-assessment through dynamically adaptive quizzes; and sustaining learner engagement through a coherent, progression-driven gamification framework. To evaluate whether these objectives translate into real pedagogical value, we conducted an empirical study involving both students and teachers. Specifically, this paper seeks to answer the following research questions:

- **RQ1**: How do students perceive GamiBot in terms of usefulness, engagement, and ease of use?
- **RQ2**: How do teachers perceive its pedagogical value and classroom applicability?
- **RQ3**: How do students' interaction patterns (telemetry) reflect self-regulated learning behaviors?

The remainder of this paper is organized as follows. Section 2 reviews related work on gamification in education, AI pedagogical agents, and the LMS integration gap. Section 3 details the design and implementation of GamiBot, covering its architecture, pedagogical

features, gamification mechanics, and user interface. Section 4 presents the evaluation methodology and results. Section 5 discusses the findings in relation to the research questions, and Section 6 concludes with implications and directions for future work.

2 Related Work

The development of GamiBot draws on four interconnected bodies of literature: the application of gamification in educational contexts, the design and evaluation of AI-powered chatbots and pedagogical agents, emerging work on systems that combine both paradigms, and the persistent challenge of integrating these solutions into institutional learning management systems.

2.1 Gamification in Education

Gamification – broadly defined as the use of game design elements in non-game contexts [5] – has been extensively studied as a mechanism for enhancing student motivation, engagement, and persistence in educational settings. Early empirical work demonstrated that the introduction of points, badges, and leaderboards into classroom activities could positively influence participation and task completion rates [9]. Subsequent research, however, introduced important nuances: the effectiveness of gamification is neither universal nor unconditional, and depends significantly on the meaningfulness of the reward structures, the alignment between game mechanics and learning objectives, and individual differences in learner profiles such as achievement orientation and intrinsic motivation [15, 22]. Studies have consistently warned against superficial implementations – so-called "pointsification" – in which rewards are decoupled from substantive academic actions, resulting in short-term compliance rather than deep engagement [17]. More recent work has advocated for adaptive and personalized gamification designs that respond to individual learner behavior and progression over time [12], a principle that informs the tiered badge system and event-driven reward mechanics adopted in GamiBot.

2.2 AI Chatbots and Pedagogical Agents

The use of conversational agents in educational settings has a long history, originating in intelligent tutoring systems (ITS) that modeled learner knowledge states and adapted instructional sequences accordingly [1, 21]. The advent of large language models has substantially expanded the capabilities and accessibility of such systems. Contemporary LLM-based educational chatbots are capable of handling open-ended natural language queries, generating explanations at varying levels of abstraction, and producing formative assessment items on demand, without the need for laboriously hand-crafted knowledge bases [11]. Retrieval-augmented generation (RAG) architectures have further strengthened these systems by grounding model outputs in domain-specific or course-specific document collections, reducing hallucination and increasing the contextual relevance of responses [14]. Empirical evaluations of LLM-based pedagogical agents have reported positive effects on students' perceived usefulness, learning efficiency, and willingness to engage with course content [11, 2], though concerns regarding accuracy, over-reliance, and the displacement of critical thinking have also been raised [7]. GamiBot builds on this tradition by combining RAG-grounded assistance with adaptive quiz generation, ensuring that AI-generated content remains tethered to institutionally curated materials rather than the unconstrained parametric knowledge of the underlying model.

2.3 Gamified Educational Agents Combining AI and Gamification

Despite the independent maturity of both gamification research and AI-driven pedagogical agents, systems that meaningfully integrate both paradigms remain relatively scarce in the literature. Several studies have explored the sequential or additive combination of chatbots with gamified elements – for instance, embedding chatbot interactions within gamified course shells or awarding points for chatbot usage – but these approaches typically treat the two components as independent modules rather than as a coherent, mutually reinforcing design [19]. A smaller set of works has proposed more tightly coupled architectures in which conversational agents adapt their behavior based on learner progression data, or in which gamification rewards are triggered dynamically by the nature and quality of the agent interaction [13]. The evidence base for such integrated systems remains limited, and existing prototypes have rarely been evaluated empirically in authentic classroom settings. GamiBot contributes to this emerging line of work by implementing a dedicated gamification backend that responds to semantically meaningful pedagogical actions, including question asking, summarization requests, and quiz engagement, thereby linking motivational incentives directly to the learning behaviors the system is designed to encourage.

2.4 The LMS Integration Gap

A recurring limitation in the educational technology literature concerns the difficulty of embedding AI and gamification tools within the institutional platforms that teachers and students already use on a daily basis. Learning management systems such as Moodle, Canvas, and Blackboard serve as the primary digital infrastructure for course delivery in higher education, yet they offer limited native support for generative AI or sophisticated gamification mechanics [20]. As a consequence, most AI-powered educational tools are deployed as standalone web applications or third-party plugins that require students to leave the LMS environment to access support, disrupting the continuity of the learning experience and reducing the likelihood of adoption [23]. Similarly, gamification in LMS contexts has often been implemented through external platforms such as Kahoot or Classcraft, which operate independently of the course content and activity data available within the LMS [6]. This fragmentation has practical consequences: teachers must maintain parallel systems, students must context-switch between tools, and the behavioral and performance data generated by AI or gamification interactions remains disconnected from the institutional record. GamiBot is specifically designed to close this integration gap by functioning as a native Moodle extension, co-locating AI assistance, adaptive assessment, and gamification feedback within the same interface through which students already access their course materials and activities.

3 GamiBot

To address the documented gap between generative AI capabilities and institutional learning management systems, we developed GamiBot: an intelligent, gamified pedagogical agent designed for deep integration within Moodle. This section details the technical and pedagogical foundations of the proposed system. First, subsection 3.1 outlines the high-level system architecture, explaining the orchestration of its core components and the flow of data during user interactions. Next, subsection 3.2 describes the AI-driven pedagogical features, specifically focusing on real-time assistance and adaptive quiz generation. Subsection 3.3 details the gamification design and its underlying mechanics. Finally, subsection 3.4 presents the user interface, explaining how progression mechanics are embedded directly into the conversational workflow to foster engagement and self-regulated learning.

3.1 System Overview

GamiBot is a modular agent-based ecosystem designed to extend Moodle with context-aware generative AI support and embedded gamification mechanisms. In contrast with standalone educational chatbots, the system is conceived as an LMS-native solution, enabling students to access tutoring, summarization, and formative assessment directly inside the course environment rather than through external platforms. This design choice is pedagogically relevant because it keeps support interactions close to the learning materials, the ongoing activities, and the institutional workflows already used by teachers and students.

At the architectural level, as illustrated in Figure 1, GamiBot is organized into four interconnected layers:

Client Layer: composed of the Moodle LMS and the GamiBot plugins, this layer provides the unified chat and gamification interface. It serves as the primary entry point for both students and professors, mediating all platform integrations.

Orchestration Layer: implemented via LangFlow, this layer is responsible for the coordination of pedagogical agents. It receives incoming requests (via webhooks and APIs) and orchestrates the workflows required to formulate a response.

Intelligence Layer: this layer executes the core cognitive tasks. It includes the Large Language Model (LLM) for text generation and the Qdrant vector database, which handles semantic search over course resources.

Gamification Layer: powered by GamiAPI, this layer manages learner progression, receiving user actions from the Client Layer to independently calculate and store rewards, points, and badges.

This separation of concerns allows the system to combine pedagogical flexibility with technical maintainability. Moodle remains the primary learning environment, LangFlow coordinates intelligent behavior, Qdrant and the LLM support content generation, and GamiAPI handles gamification logic independently of the conversational flow.

The Intelligence Layer is powered by OpenAI's `gpt-oss-20b`, a 20-billion parameter open-weights large language model. The selection of an openly available model is particularly advantageous in an educational context, as it allows us (and other institutions) to self-host the infrastructure. This ensures that sensitive student interactions and proprietary course materials remain secure and compliant with institutional data privacy regulations. This model serves as the underlying reasoning engine for both the conversational assistant and the adaptive quiz-generation workflow, operating over prompts enriched with retrieved course context.

A key feature of GamiBot architecture is the continuous alignment between course content and AI support. As instructors build or update a Moodle course, educational materials such as PDFs, slide decks, textual resources, and other uploaded assets are processed and indexed in Qdrant through vector embeddings. As a result, agents within the Orchestration Layer can later retrieve semantically relevant fragments of official course materials without requiring teachers to manually author question–answer pairs or scripted tutoring rules. This mechanism ensures that the system remains synchronized with the evolving course and that its outputs are anchored in institutionally curated content rather than relying exclusively on the parametric knowledge of the underlying LLM.

Operationally, when a student submits a question or requests a quiz through the interface, the request is passed by the Client Layer to LangFlow together with relevant contextual metadata. The active agent in the Orchestration Layer then determines the appropriate pedagogical action, queries Qdrant for semantic search when necessary, composes an enriched

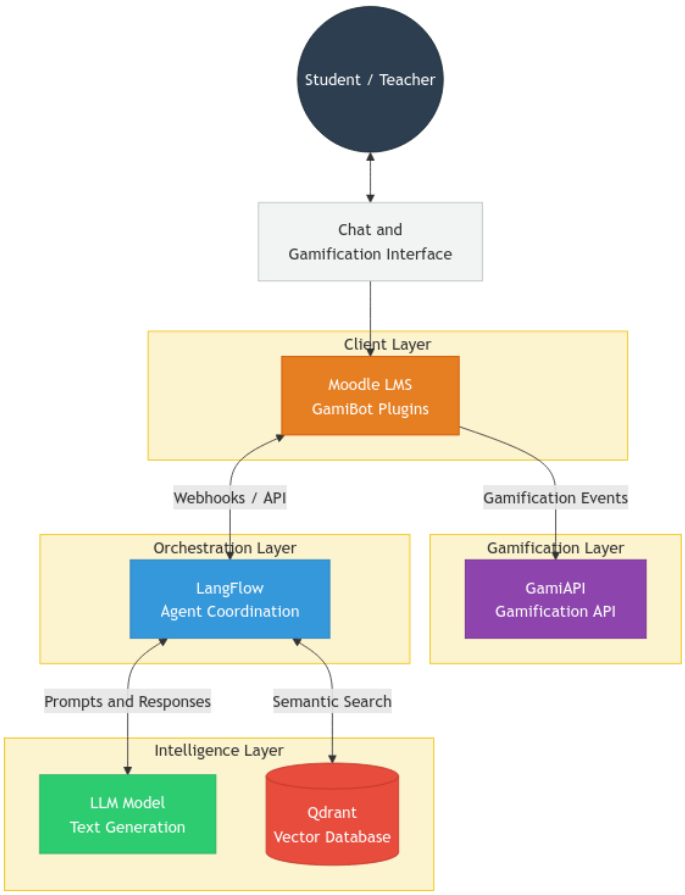


Figure 1 High-level architecture of the GamiBot ecosystem, illustrating the data flow and interactions across the Client, Orchestration, Intelligence, and Gamification layers.

prompt, and forwards it to the LLM in the Intelligence Layer. In parallel, the Client Layer reports the corresponding user action to GamiAPI in the Gamification Layer, which updates the learner's state and returns the refreshed profile to Moodle. The student therefore receives both the AI-generated educational output and the updated progress indicators within a single integrated interface.

3.2 Pedagogical Features

GamiBot provides context-aware support through two core AI-driven features: real-time assistance and adaptive quiz generation. Central to both workflows is the `gpt-oss-20b` large language model. By injecting explicit pedagogical instructions alongside semantically relevant course excerpts retrieved from Qdrant into the model's context window, these features position the system not merely as a conversational interface, but as a pedagogical assistant capable of responding to immediate learning needs while encouraging active engagement.

The first feature, real-time assistance and summarization, supports students during their interaction with Moodle resources. Learners may ask questions about a topic, request clarification about a specific concept, or ask for the summary of a section they are studying. In these cases, an assistance agent retrieves relevant content from Qdrant and uses it to

construct a grounded prompt for the `gpt-oss-20b` model. The resulting response is strictly anchored to the institutional curriculum, which increases contextual relevance and reduces the risk of generic or decontextualized explanations.

The second feature, adaptive quiz generation, supports formative assessment and knowledge consolidation. Rather than relying on a fixed bank of manually authored questions, GamiBot uses a dedicated quiz-generation agent that invokes Qdrant during its reasoning process. When a quiz is requested, the agent retrieves topic-specific content segments. The system prompt then constrains the `gpt-oss-20b` model to generate formative assessment items strictly aligned with this retrieved evidence. This dynamic generation minimizes hallucinations and allows the quizzes to remain sensitive to the course context while significantly reducing authoring overhead for instructors.

An important aspect of this feature is that the quiz interaction does not end with answer submission. After the student responds, the system produces a structured performance summary that includes the submitted answers, the expected correct answers, explanatory feedback, and targeted suggestions for further study. In pedagogical terms, this transforms quiz generation from a simple testing mechanism into a feedback-oriented formative loop. The value of the interaction therefore lies not only in whether the student answered correctly, but also in the interpretive layer that helps them understand mistakes and identify areas requiring further attention.

Combined, these two features support a cycle of inquiry, feedback, and reinforcement. Students can begin with a question, receive grounded assistance, test their understanding through an adaptive quiz, and then review a personalized explanation of their performance. This sequence is particularly compatible with self-regulated learning processes, as it encourages learners to identify knowledge gaps, seek targeted support, and act on feedback within the same digital environment.

3.3 Gamification Design

Beyond its AI capabilities, GamiBot incorporates a dedicated gamification layer intended to transform learning interactions into visible and motivating progress. Rather than treating gamification as a static overlay, the system links rewards to meaningful pedagogical actions performed within the conversational workflow. Through interaction with GamiBot, students accumulate points for actions such as asking questions, requesting summaries, or engaging with quizzes; earn badges associated with distinct forms of participation; and progress through topics, reflecting their continued engagement over the duration of the course.

To provide granular and continuous recognition, badges in GamiBot are structured around a three-level progression system. This multi-tier approach ensures that students remain motivated even after unlocking an initial reward, preventing the sudden drop-off in engagement often associated with one-time achievements. Examples of this badge progression include:

- The **Educational Material Analyst** badge, which recognizes active doubt clarification. A student unlocks Level 1 after asking 3 context-related questions, upgrades to Level 2 at 10 questions, and reaches Level 3 mastery at 20 questions.
- The **Rough Diamond** badge, which rewards persistence and the willingness to learn from mistakes. Level 1 is awarded when a student passes multiple quizzes after initially submitting incorrect answers, advancing to Levels 2 and 3 as they continually polish their understanding through trial, error, and reflection.
- The **Quiz Master** badge, which acknowledges consistent engagement with formative assessment. The first tier is awarded after completing 3 quizzes, scaling up to higher tiers as the learner routinely tests their knowledge across different course topics.

Furthermore, the gamification design includes a quiz life mechanic to reinforce purposeful engagement. Each quiz session begins with a limited number of lives (typically three), and incorrect answers consume one life. By making attempts finite, this mechanic discourages random guessing and encourages students to reflect carefully before responding or to revisit the AI's explanatory support when uncertain.

All of these mechanics are managed by GamiAPI, which operates as a separate backend service responsible for tracking user state, validating events, and returning updated progression data to Moodle. This architectural design has two distinct advantages. First, it keeps gamification rules configurable and reusable across courses or deployments without tightly coupling them to the conversational logic. Second, it allows the platform to record richer activity traces, including point accumulation rates, badge tier upgrades, and life consumption metrics, which can later support both learner self-monitoring and empirical analysis of student behavior.

3.4 User Interface

The user interface is intentionally chat-centric, designed to deliver motivational feedback synchronously with AI-driven academic support. From the student's perspective, GamiBot appears as an integrated conversational component inside Moodle, combining AI responses, interactive assessments, and gamification feedback within the same interaction space. This co-location is important as it ensures that pedagogical help and motivational feedback are not experienced as separate systems, but as logical parts of a single continuous learning interaction.

As illustrated in Figure 2, the interface is divided into distinct contextual views that guide the learner through the pedagogical loop:

- **Welcome and Action Menu**: upon opening GamiBot, the student is greeted with quick-action buttons to generate quizzes, request summaries, clarify doubts, or view their progress. This minimizes the cognitive load required to initiate a learning session.
- **Conversational Assistance**: the chat window allows for natural language queries. The AI's responses explicitly reference the course materials (e.g., citing specific documents) to ensure the student knows the information is grounded in the official curriculum.
- **Adaptive Quizzes and Feedback**: when a student engages in a quiz, the chat transforms into a testing interface. The header displays the remaining quiz lives (represented as hearts), enforcing the finite attempt mechanic. Immediate feedback is provided after each question, offering detailed justifications for both correct and incorrect answers. At the end of the quiz, the system automatically generates a personalized study guide highlighting the concepts the student struggled with, areas for improvement, and tips.
- **Progress Dashboard**: the progress area exposes detailed usage statistics (e.g., points, earned badges, time spent, and number of quizzes completed). It visually displays the tiered badges the student has unlocked (such as the "Rough Diamond" badge) and highlights specific topics where the student has struggled, enabling them to monitor their engagement and identify areas where further study may be needed.

Overall, GamiBot combines Moodle integration, agent orchestration, retrieval-augmented generation, and gamification into a cohesive educational ecosystem. Its contribution lies not only in adding AI features to an LMS, but in structuring these features around grounded support, adaptive formative assessment, and visible progression, thereby creating a more interactive and pedagogically expressive digital learning environment.

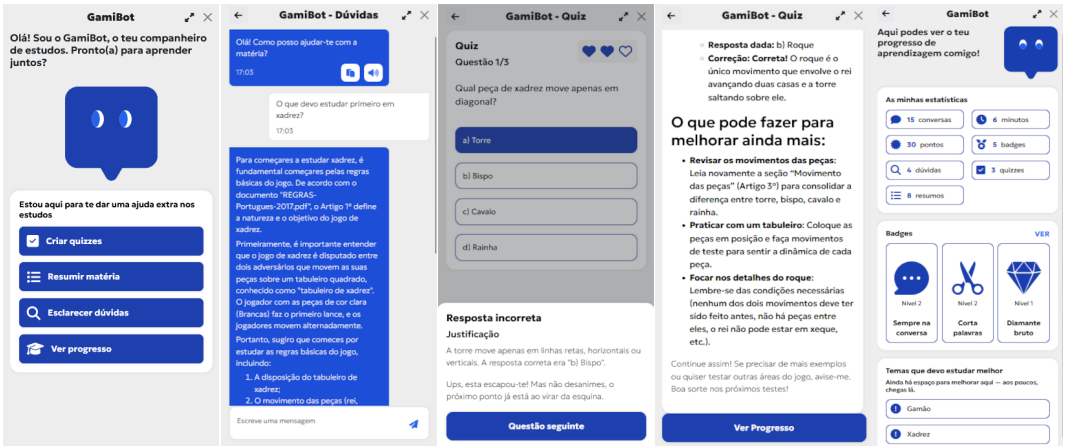


Figure 2 Screenshots of the GamiBot interface illustrating its core functionalities: quick-action menus, retrieval-augmented doubt clarification, adaptive quizzes with a finite-life mechanic, immediate formative feedback, and a comprehensive gamification dashboard that tracks student progress and achievements.

4 Evaluation

To assess the usability, pedagogical value, and engagement impact of GamiBot in real educational contexts, we conducted an exploratory pilot study across two higher education institutions within the Polytechnic Institute of Porto (IPP). This section details the methodology used to collect data and presents the empirical results of the intervention.

4.1 Context and Methodology

GamiBot was deployed directly into the Moodle environments of the School of Health (ESS IPP) and the School of Media Arts and Design (ESMAD IPP). To test the system's versatility across different knowledge domains and academic levels, it was activated in four distinct curricular units:

- **ESS IPP:** Master's in Occupational Therapy (*Neuroscience of Mental Illness*) and Master's in Digital Transformation of Health and Rehabilitation Services (*Serious Games and Gamification in Health and Rehabilitation*).
- **ESMAD IPP:** Bachelor's in Web Information Systems and Technologies (*Object-Oriented Programming*, 1st year; and *Web Programming II*, 2nd year).

The target population comprised 164 students (127 from ESMAD and 37 from ESS), resulting in a self-selected sample of 32 participants (19.5% response rate) for the survey evaluation. Participation was entirely voluntary, with the system positioned as a supplementary study tool rather than a mandatory course requirement.

Data were collected using a mixed-methods approach over a two-week pilot period. First, we extracted quantitative telemetry from the GamiBot orchestration layer to measure actual usage, tracking the volume of AI-generated quizzes, clarified doubts, and generated summaries. Second, at the end of the pilot period, students were invited to complete an anonymous, non-mandatory online questionnaire. The survey utilized a 5-point Likert scale (ranging from "Strongly Disagree" to "Strongly Agree") to evaluate five primary dimensions:

- **Usability:** assessed using items adapted from the System Usability Scale (SUS).
- **Chatbot Experience:** evaluating the clarity, contextual relevance, and accuracy of the AI's natural language responses.
- **Gamification:** measuring the motivational impact of points, badges, and progression levels.
- **Pedagogical Impact:** assessing perceived improvements in content comprehension and learner autonomy.
- **Intention to Use:** based on constructs from the Technology Acceptance Model (TAM), measuring the likelihood of future adoption and peer recommendation.

Open-ended questions were also included at the end of the survey to capture qualitative feedback on what students liked most about the system and what specific areas required improvement.

During this pilot, all AI-generated quizzes, doubt clarifications, and summaries were powered by the `gpt-oss-20b` model. Due to initial local hardware constraints, the model was deployed as a managed online endpoint via Microsoft Azure AI Foundry for the duration of the experiment. However, because `gpt-oss-20b` is an open-weights model, the architecture remains fully compatible with on-premise self-hosting for future, permanent institutional deployments.

4.2 System Usage and Interaction Patterns

Log data from the pilot revealed active, voluntary engagement from the student body across both institutions. Overall, the system processed a total of 773 distinct pedagogical interactions.

At ESMAD, students generated 361 quizzes, requested 142 course summaries, and asked 49 specific questions (doubts). At ESS, despite the smaller cohort, engagement remained high relative to class size, with 48 generated quizzes, 45 summaries, and 28 clarified doubts. Notably, across both institutions, the generation of adaptive quizzes was by far the most heavily utilized feature, accounting for approximately 53% of all interactions. This suggests a strong student preference for active, formative assessment over passive content summarization when using an AI assistant.

4.3 Student Perspectives on Usability and Pedagogy

A total of 32 students responded to the voluntary evaluation questionnaire. Of these, the vast majority (81%) reported using GamiBot at least once, with nearly half (47%) using it multiple times during the short pilot.

The student evaluation across the five primary dimensions yielded very encouraging results, particularly regarding the core system functionalities and the pedagogical value of the agent. Figure 3 summarizes the breakdown of Negative, Neutral, and Positive responses across all 29 survey items, properly accounting for reverse-coded statements in the usability scale.

Students reported exceptionally high levels of usability. Across the ten items adapted from the System Usability Scale (SUS), the overall positive sentiment was 74.6%. When asked specifically if the system was easy to use, 83.4% agreed or strongly agreed, highlighting the success of integrating the tool directly into the Moodle interface.

In terms of the Chatbot Experience, 69.4% of the comprehensive responses were positive. Specifically, 73.4% agreed or strongly agreed that GamiBot's answers were clear and understandable, and 70% agreed that the system responded adequately to the specific context of their discipline, validating the effectiveness of the Qdrant-based retrieval-augmented generation (RAG) approach.

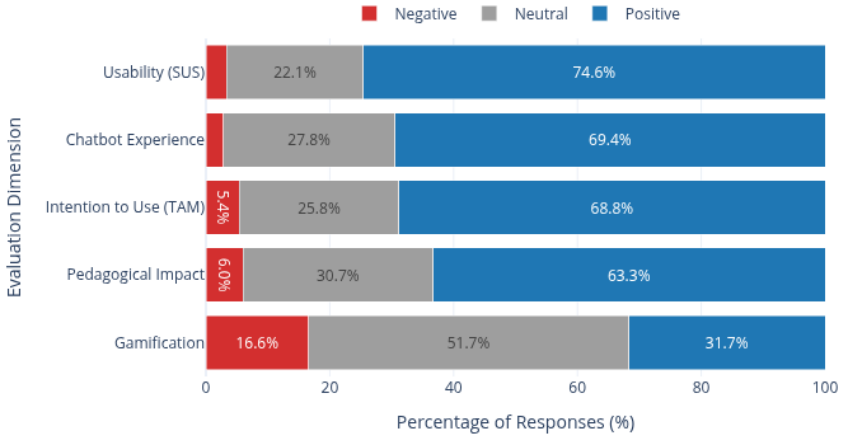


Figure 3 Student evaluation of GamiBot across five primary dimensions, showing the percentage of Negative (Disagree/Strongly Disagree), Neutral, and Positive (Agree/Strongly Agree) responses (n=32).

Regarding Pedagogical Impact, the overall positive sentiment reached 63.3%. Individually, 66.7% of respondents considered GamiBot a useful tool within the context of their curricular unit, and 56.7% agreed or strongly agreed that it improved their autonomy in learning. This sentiment was echoed in the qualitative open-ended feedback, where one student noted: *"[It is] highly intuitive, extremely easy to use; the responses are precise and well contextualized with the material provided in Moodle."*

Furthermore, the Intention to Use dimension achieved a 68.8% positive evaluation. Looking forward, 70% of the respondents stated they intend to continue using GamiBot, and nearly 60% said they would recommend the tool to their colleagues.

4.4 Perspectives on Gamification Mechanics

While the AI and usability aspects were highly rated, feedback on the gamification elements was more mixed. Across the five items measuring this dimension, the overall positive sentiment was 31.7%. When asked specifically if the experience points motivated them to continue using the system, 37.9% agreed or strongly agreed, but a significant portion (48.3%) remained neutral ("Neither agree nor disagree"), and 13.7% disagreed. Similar distributions were observed regarding the impact of badges on making the learning experience more interesting.

Qualitative feedback provided crucial context for these mixed results. Several students reported technical glitches with the progression interface during the pilot, stating, *"The gamification functionality was not working properly [at times], which is why I answered 'strongly disagree'."*. Additionally, students expressed a desire for greater control over the gamified elements, such as being able to explicitly choose the number of questions in a quiz session rather than relying on a fixed length set by teachers.

Despite these interface bugs, the overall reception of the core tool was highly positive, establishing a strong foundation for future iterative improvements to the gamified components.

5 Discussion

Evaluating GamiBot across multiple academic disciplines provided valuable insights into the integration of AI-driven, gamified pedagogical agents within LMSs. Addressing our first research question (RQ1) regarding student perceptions, the system was highly successful

in terms of ease of use and usefulness. The seamless integration directly into the Moodle interface eliminated the friction of navigating to third-party platforms, which was reflected in the 74.6% positive sentiment in the Usability (SUS) dimension, with 83.4% of students explicitly agreeing that the system was easy to use. In terms of usefulness, the Chatbot Experience dimension received 69.4% positive feedback. Students praised the clarity and contextual relevance of the AI's responses, confirming that the RAG architecture successfully grounded the language model in course-specific materials. This reliable performance built student trust and drove a high Intention to Use (68.8% positive).

However, student perception of engagement through gamification yielded mixed results (31.7% positive sentiment). While the core AI features effectively engaged students, the specific gamified elements did not achieve their intended motivational impact for the majority. Qualitative feedback revealed this was largely due to technical glitches in the progression interface and a lack of user control over gamified tasks, such as fixed quiz lengths. This dichotomy suggests that while students highly value reliable AI assistance, gamification in higher education must transcend "pointsification" and ensure high technical stability to avoid becoming a source of cognitive friction rather than an engaging motivator.

Regarding the second research question (RQ2) on pedagogical value and classroom applicability, the system demonstrated high versatility by functioning effectively across different knowledge domains. It adapted seamlessly to both technical, hard-skill courses and theoretical, health-science courses. Because the RAG pipeline utilizes existing Moodle documents, teachers were not required to manually train the AI, drastically lowering the barrier to classroom adoption. Furthermore, by autonomously processing 773 pedagogical interactions over the short pilot, including answering 77 specific student doubts and generating 409 targeted quizzes, GamiBot effectively acted as a 24/7 supplementary teaching assistant. This capacity to handle high-volume, low-complexity queries suggests strong pedagogical value, allowing educators to dedicate more classroom time to complex problem-solving and critical discussion.

Addressing the third research question (RQ3), the empirical data provides compelling evidence of GamiBot's positive impact on participation, self-regulation, and learning support. Participation was strong, with 81% of surveyed students interacting with the bot voluntarily and 47% returning to use it multiple times. Crucially, the telemetry revealed a distinct trend toward self-regulated learning: approximately 53% of all system interactions were requests for adaptive quizzes, far outpacing passive requests such as course summaries or direct Q&A. This behavioral pattern is consistent with the formative assessment framework proposed by Black and Wiliam [3], particularly the strategy of activating students as owners of their own learning. By prioritizing generative testing over passive review, learners used the agent to close the gap between their current understanding and the intended learning goals, thereby transforming the LMS into a responsive environment for self-regulated growth. This interpretation is further supported by the survey results on learning support, where 63.3% of responses in the Pedagogical Impact dimension were positive, and 56.7% of students explicitly agreed that the system improved their learning autonomy.

While these initial results are promising, this pilot study has limitations. The sample size for the evaluation questionnaire (n=32) was relatively small, and the pilot duration was brief. Additionally, while the usage data strongly implies benefits for educators, future studies must include formal qualitative and quantitative evaluations directly from the teachers to fully validate their perceptions of the system's classroom applicability.

6 Conclusion and Future Work

This study presented GamiBot, an educational agent that bridges the gap between generative AI, gamification, and institutional LMSs. The pilot deployment demonstrated that embedding a RAG-powered LLM directly within Moodle provides highly accessible and contextually accurate academic support without forcing users to leave their learning environment. Telemetry and survey data revealed that students heavily favor active self-assessment, frequently utilizing the system to generate adaptive quizzes. This underscores GamiBot's significant pedagogical value in promoting learner autonomy and self-regulated learning.

Conversely, the mixed reception of the gamification elements highlighted a critical design lesson: to be effective, motivational mechanics must be technically robust and highly adaptable to individual preferences. Future work will focus on stabilizing the gamification engine and granting students greater control over their learning parameters, such as dynamically adjusting quiz lengths and difficulty levels. Additionally, we plan to conduct longitudinal studies with larger cohorts to measure GamiBot's impact on objective academic performance, alongside formal evaluations to capture the long-term benefits for educators managing the platform.

References

1 John R Anderson, C Franklin Boyle, and Brian J Reiser. Intelligent tutoring systems. *Science*, 228(4698):456–462, 1985.

2 David Baidoo-Anu and Leticia Owusu Ansah. Education in the era of generative artificial intelligence (ChatGPT): Understanding the potential benefits and challenges. *Journal of AI*, 7(1):52–62, 2023.

3 Paul Black and Dylan Wiliam. Developing the theory of formative assessment. *Educational Assessment, Evaluation and Accountability(formerly: Journal of Personnel Evaluation in Education)*, 21(1):5–31, February 2009. doi:10.1007/s11092-008-9068-5.

4 Narius Farhad Davar, M. Ali Akber Dewan, and Xiaokun Zhang. Ai chatbots in education: Challenges and opportunities. *Information*, 16(3), 2025. doi:10.3390/info16030235.

5 Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: defining gamification. In *Proceedings of the 15th International Academic MindTrek Conference*, pages 9–15. ACM, 2011. doi:10.1145/2181037.2181040.

6 Darina Dicheva, Christo Dichev, Gennady Agre, and Gabriela Angelova. Gamification in education: A systematic mapping study. *Journal of Educational Technology & Society*, 18(3):75–88, 2015. URL: https://www.j-ets.net/ETS/journals/18_3/6.pdf.

7 Yogesh K Dwivedi, Nir Kshetri, Laurie Hughes, Emma Louise Slade, Anand Jeyaraj, Arpan Kumar Kar, Abdullah M Baabdullah, Alex Koohang, Vishnupriya Raghavan, Manju Ahuja, et al. Opinion paper: "so what if chatgpt wrote it?" multidisciplinary perspectives on opportunities, challenges and implications of generative conversational ai for research, practice and policy. *International Journal of Information Management*, 71:102642, 2023. doi:10.1016/J.IJINFOMGT.2023.102642.

8 Sithara H. P. W. Gamage, Jennifer R. Ayres, and Monica B. Behrend. A systematic review on trends in using moodle for teaching and learning. *International Journal of STEM Education*, 9(1):9, January 2022. doi:10.1186/s40594-021-00323-x.

9 Juho Hamari, Jonna Koivisto, and Harri Sarsa. Does gamification work? a literature review of empirical studies on gamification. In *Proceedings of the 47th Hawaii International Conference on System Sciences (HICSS-47)*, pages 3025–3034. IEEE, 2014. doi:10.1109/HICSS.2014.377.

10 Carlos J. Hellín, Francisco Calles-Esteban, Adrián Valledor, Josefa Gómez, Salvador Otón-Tortosa, and Abdelhamid Tayebi. Enhancing student motivation and engagement through a gamified learning environment. *Sustainability*, 15(19), 2023. doi:10.3390/su151914119.

11 Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103:102274, 2023.

12 Ana Carolina Tomé Klock, Isabela Gasparini, Marcelo Soares Pimenta, and Juho Hamari. Adaptive gamification in education: A literature review of adaptive frameworks, applications and instruments. *International Journal of Human-Computer Studies*, 141:102496, 2020.

13 Lela Labadze, Maya Grigolia, and Lasha Machaidze. Role of ai chatbots in education: Systematic literature review. *International Journal of Educational Technology in Higher Education*, 20(1):56, 2023.

14 Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

15 Elisa D Mekler, Florian Brühlmann, Alexandre N Tuch, and Klaus Opwis. Towards understanding the effects of individual gamification elements on intrinsic motivation and performance. *Computers in Human Behavior*, 71:525–534, 2017. doi:10.1016/J.CHB.2015.08.048.

16 Negar Monazam-Tabrizi, Yusuf Kurt, and William il Kuk Kang. Navigating learning disruptions: The role of digital learning platforms in student motivation, feedback and emotion. *Computers & Education*, 246:105534, 2026. doi:10.1016/j.compedu.2025.105534.

17 Scott Nicholson. A recipe for meaningful gamification. In *Gamification in Education and Business*, pages 1–20. Springer, 2015.

18 José Paiva and Ricardo Queirós. Gamibot. Text (visited on 2026-07-13). URL: https://inov-norte.github.io/gamibot-docs, doi:10.4230/artifacts.27132.

19 Jorge Quiroga Pérez, Llúcia Daiño, Caterina Pinya, and Joan J Muntaner-Guasp. Chatbots and conversational agents in education: A bibliometric analysis using scopus database. *Interactive Learning Environments*, 31(7):4500–4515, 2020.

20 Ahmed Tlili, Boulus Shehata, Michael Agyemang Adarkwah, Aras Bozkurt, Daniel T Hickey, Ronghuai Huang, and Boulus Agyemang. If history repeats itself, how will ai reshape education? a literature review of the state of research on chatgpt in education. *Smart Learning Environments*, 10(1):32, 2023.

21 Kurt VanLehn. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221, 2011.

22 Maarten Vansteenkiste, Joke Simons, Willy Lens, Kennon M Sheldon, and Edward L Deci. Motivating learning, performance, and persistence: the synergistic effects of intrinsic goal content and autonomy-supportive context. *Journal of Personality and Social Psychology*, 87(2):246–260, 2004.

23 Olaf Zawacki-Richter, Victoria I Marín, Melissa Bond, and Franziska Gouverneur. Systematic review of research on artificial intelligence applications in higher education – where are the educators? *International Journal of Educational Technology in Higher Education*, 16(1):39, 2019.