

5th International Computer Programming Education Conference

ICPEC 2024, June 27–28, 2024, Lisbon, Portugal

Edited by

André L. Santos

Maria Pinto-Albuquerque



Editors

André L. Santos 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL, Portugal
Andre.Santos@iscte-iul.pt

Maria Pinto-Albuquerque 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL, Portugal
Maria.Albuquerque@iscte-iul.pt

ACM Classification 2012

Applied computing → Education; Applied computing → Interactive learning environments; Applied computing → Computer-assisted instruction

ISBN 978-3-95977-347-8

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/978-3-95977-347-8>.

Publication date

September, 2024

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <https://portal.dnb.de>.

License

This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): <https://creativecommons.org/licenses/by/4.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/OASlcs.ICPEC.2024.0

ISBN 978-3-95977-347-8

ISSN 1868-8969

<https://www.dagstuhl.de/oasics>

OASlcs – OpenAccess Series in Informatics

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

Editorial Board

- Daniel Cremers (TU München, Germany)
- Barbara Hammer (Universität Bielefeld, Germany)
- Marc Langheinrich (Università della Svizzera Italiana – Lugano, Switzerland)
- Dorothea Wagner (*Editor-in-Chief*, Karlsruher Institut für Technologie, Germany)

ISSN 1868-8969

<https://www.dagstuhl.de/oasics>

■ Contents

Preface	
<i>André L. Santos and Maria Pinto-Albuquerque</i>	0:vii
Program Committee	
.....	0:ix
List of Authors	
.....	0:xi

Invited Talk

Hedy: An Inclusive, Multi-Lingual, and Gradual Programming Language	
<i>Felienne Hermans</i>	1:1–1:1

Automated Assessment

A Domain-Specific Language for Dynamic White-Box Evaluation of Java Assignments	
<i>Afonso B. Caniço and André L. Santos</i>	2:1–2:13
Seven Years Later: Lessons Learned in Automated Assessment	
<i>Bruno Pereira Cipriano and Pedro Alves</i>	3:1–3:14
Adaptation of Automated Assessment System for Large Programming Courses	
<i>Marek Horváth, Tomáš Kormaník, and Jaroslav Porubán</i>	4:1–4:11

Teaching Approaches

Kumon-Inspired Approach to Teaching Programming Fundamentals	
<i>Ivone Amorim, Pedro Baltazar Vasconcelos, and João Pedro Pedroso</i>	5:1–5:13
An Experience with Adaptive Formative Assessment for Motivating Novices in Introductory Programming Learning	
<i>Jagadeeswaran Thangaraj, Monica Ward, and Fiona O’Riordan</i>	6:1–6:12
Promoting Deep Learning Through a Concept Map-Building Collaborative Activity in an Introductory Programming Course	
<i>João Paulo Barros</i>	7:1–7:12
Scientific Whispers: Mapping Innovative Pedagogies in STEAM and Programming Education	
<i>Margarida Antunes and António Trigo</i>	8:1–8:12
Teaching Programming Courses with Digital Educational Escape Rooms (DEER): A Conceptual Proposal Conducive to Learning by Trial and Error	
<i>Antonio Trigo and Margarida Antunes</i>	9:1–9:8
Educational Program Visualizations Using Synthetized Execution Information	
<i>Rodrigo Mourato and André L. Santos</i>	10:1–10:8

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque

OpenAccess Series in Informatics



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

Games and Gamification

Client-Side Gamification Engine for Enhanced Programming Learning <i>Ricardo Queirós, Robertas Damaševičius, Rytis Maskeliūnas, and Jakub Swacha ..</i>	11:1–11:12
Game Development: Enhancing Creativity and Independent Creation in University Course <i>Lenka Bubenkova and Emilia Pietrikova</i>	12:1–12:13
Learning Paths: A New Teaching Strategy with Gamification <i>Filipe Portela</i>	13:1–13:12
Code Review for CyberSecurity in the Industry: Insights from Gameplay Analytics <i>Andrei-Cristian Iosif, Ulrike Lechner, Maria Pinto-Albuquerque, and Tiago Espinha Gasiba</i>	14:1–14:11
Implementing a Digital Twin for a Robotic Platform to Support Large-Scale Coding Classes <i>Michael Heeney, Kelly Androutsopoulos, and Franco Raimondi</i>	15:1–15:12

Software Security

To Kill a Mocking Bug: Open Source Repo Mining of Security Patches for Programming Education <i>Andrei-Cristian Iosif, Tiago Espinha Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque</i>	16:1–16:12
Improving Industrial Cybersecurity Training: Insights into Code Reviews Using Eye-Tracking <i>Samuel Riegel Correia, Maria Pinto-Albuquerque, Tiago Espinha Gasiba, and Andrei-Cristian Iosif</i>	17:1–17:9

Generative Artificial Intelligence

Using ChatGPT During Implementation of Programs in Education <i>Norbert Baláz, Jaroslav Porubán, Marek Horváth, and Tomáš Kormaník</i>	18:1–18:9
Exercisify: An AI-Powered Statement Evaluator <i>Ricardo Queirós</i>	19:1–19:6
Use of Programming Aids in Undergraduate Courses <i>Ana Rita Peixoto, André Glória, José Luís Silva, Maria Pinto-Albuquerque, Tomás Brandão, and Luís Nunes</i>	20:1–20:9
Authoring Programming Exercises for Automated Assessment Assisted by Generative AI <i>Yannik Bauer, José Paulo Leal, and Ricardo Queirós</i>	21:1–21:8

■ Preface

ICPEC 2024, the 5th International Computer Programming Education Conference took place on 27 and 28 of June in Lisbon, Portugal. Continuing the already established tradition of a warm and collaborative community promoting the exchange of perspectives and advances in computer science education challenges, we attracted more internationally diverse participants than in previous editions.

We were honored to have Professor Felienne Hermans delivering the invited talk “Hedy: an Inclusive, Multi-lingual, and Gradual Programming Language.” Felienne opened the conference by highlighting the importance of promoting computational thinking and programming for everyone, regardless of gender, language, or culture. These matters were discussed in the context of developing Hedy (<https://hedy.org>), a programming environment that she developed over the recent years.

The conference featured twenty accepted and presented papers, comprising thirteen full papers and seven short papers (ongoing work). These papers were organized into five thematic sections in the context of programming education: Automated Assessment, Teaching Approaches, Games and Gamification, Software Security, and Generative Artificial Intelligence.

We thank the Program Committee members for their effort in reviewing the papers with care and quality, and all the participants for contributing to lively discussions.

Lisboa, 31 July 2024

André Leal Santos and Maria Pinto-Albuquerque



■ Program Committee

Alberto Simões, 2Ai Lab – IPCA

Anabela Gomes, Instituto Politécnico de Coimbra

André L. Santos, Iscte (Instituto Universitário de Lisboa)

António José Mendes, University of Coimbra

Bárbara Cleto, ESMAD/uniMAD

Bertil Marques, GILT/ISEP/IPP

Cristiana Araújo, University of Minho

Filipe Portela, University of Minho

Filomena Lopes, Universidade Portucalense

J. Angel Velasquez-Iturbide, Universidad Rey Juan Carlos

Jakub Swacha, University of Szczecin

José Paiva, University of Porto

José Paulo Leal, University of Porto

Leonel Morgado, INESC TEC / Universidade Aberta

Marco Temeperini, Sapienza University of Rome

Maria José Marcelino, University of Coimbra

Maria Pinto-Albuquerque, Iscte (Instituto Universitário de Lisboa)

Mário Pinto, ESMAD, Polytechnic of Porto

Martinha Piteira, IPS – ESTSetúbal

Paula Tavares, ISEP (Instituto Superior de Engenharia do Porto)

Pedro Baltazar Vasconcelos, University of Porto

Pedro Rangel Henriques, University of Minho

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

Rytis Maskeliunas, Kaunas University of Technology

Sergio Ibarri, University of Zaragoza

Teresa Terroso, ESMAD, Polytechnic of Porto


Tiago Gasiba, Siemens AG

Vítor Sá, Universidade Católica Portuguesa

■ List of Authors

Pedro Alves  (3)


COPELABS, Lusófona University,
Lisbon, Portugal

Ivone Amorim  (5)

PORTIC - Porto Research, Technology &
Innovation Center, Polytechnic of Porto (IPP),
Portugal

Kelly Androutsopoulos  (15)

Department of Computer Science,
Middlesex University, London, UK

Margarida Antunes  (8, 9)

Polytechnic University of Coimbra, Portugal

Afonso B. Caniço  (2)

Instituto Universitário de Lisboa (ISCTE-IUL),
Portugal

Norbert Baláž (18)

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

João Paulo Barros  (7)

Polytechnic Institute of Beja, Portugal;
Center of Technology and Systems
(UNINOVA-CTS) and Associated Lab of
Intelligent Systems (LASI), Caparica, Portugal

Yannik Bauer  (21)

DCC – FCUP, Porto, Portugal

Tomás Brandão  (20)

Instituto Universitário de Lisboa (ISCTE-IUL),
ISTAR, Portugal

Lenka Bubenkova (12)

Department of Computers and Informatics,
FEI TU of Košice, Slovakia

Bruno Pereira Cipriano  (3)

COPELABS, Lusófona University, Lisbon,
Portugal

Robertas Damaševičius  (11)

Department of Applied Informatics, Vytautas
Magnus University, Vilnius, Lithuania

Tiago Espinha Gasiba  (14, 16, 17)

Siemens AG, München, Germany

André Glória  (20)


Instituto Universitário de Lisboa (ISCTE-IUL),
Instituto de Telecomunicações, Portugal

Michael Heeney  (15)


Department of Computer Science,
Middlesex University, London, UK

Felienne Hermans  (1)

Vrije Universiteit Amsterdam, The Netherlands

Marek Horváth  (4, 18)

Department of Computers and Informatics,
FEI TU of Košice, Slovakia

Andrei-Cristian Iosif  (14, 16, 17)

Universität der Bundeswehr München, Germany;
Siemens AG, München, Germany

Tomáš Kormaník  (4, 18)

Department of Computers and Informatics,
FEI TU of Košice, Slovakia

José Paulo Leal  (21)

CRACS – INESC TEC, Porto, Portugal;
DCC – FCUP, Porto, Portugal

Ulrike Lechner  (14, 16)

Universität der Bundeswehr München, Germany

Rytis Maskeliūnas  (11)

Centre of Real Time Computer Systems,
Kaunas University of Technology, Lithuania

Rodrigo Mourato  (10)

Instituto Universitário de Lisboa (ISCTE-IUL),
Portugal

Luís Nunes  (20)

Instituto Universitário de Lisboa (ISCTE-IUL),
ISTAR, Portugal

Fiona O’Riordan  (6)


CCT College, Dublin, Ireland

João Pedro Pedroso  (5)

CMUP & Department of Computer Science,
Faculty of Sciences, University of Porto,
Portugal

Ana Rita Peixoto  (20)

Instituto Universitário de Lisboa (ISCTE-IUL),
ISTAR, Portugal

Emilia Pietrikova  (12)

Department of Computers and Informatics,
FEI TU of Košice, Slovakia

Maria Pinto-Albuquerque  (14, 16, 17, 20)

Instituto Universitário de Lisboa (ISCTE-IUL),
ISTAR, Portugal

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque


OpenAccess Series in Informatics




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

Filipe Portela  (13)


Algoritmi Centre, University of Minho,
Guimarães, Portugal

Jaroslav Porubän  (4, 18)

Department of Computers and Informatics,
FEI TU of Košice, Slovakia

Ricardo Queirós  (11, 19, 21)


School of Media Arts and Design & CRACS –
INESC TEC, Polytechnic of Porto, Portugal

Franco Raimondi  (15)

Gran Sasso Science Institute, L'Aquila, Italy

Samuel Riegel Correia  (17)

Instituto Universitário de Lisboa (ISCTE-IUL),
ISTA, Portugal

André L. Santos  (2, 10)

Instituto Universitário de Lisboa (ISCTE-IUL),
ISTAR-IUL, Portugal

José Luís Silva  (20)


ITI/LARSyS, Instituto Universitário de Lisboa
(ISCTE-IUL), Portugal

Jakub Swacha  (11)


Department of IT in Management,
University of Szczecin, Poland

Jagadeeswaran Thangaraj  (6)

School of Computing, Dublin City University,
Ireland

Antonio Trigo  (9)

Polytechnic University of Coimbra, Portugal;
CEOS.PP, ISCAP, Polytechnic of Porto,
Portugal

António Trigo  (8)

Polytechnic University of Coimbra, Portugal;
CEOS.PP, ISCAP, Polytechnic of Porto,
Portugal

Pedro Baltazar Vasconcelos  (5)

LIACC & Department of Computer Science,
Faculty of Sciences, University of Porto,
Portugal

Monica Ward  (6)

School of Computing, Dublin City University,
Ireland

Hedy: An Inclusive, Multi-Lingual, and Gradual Programming Language

Felienne Hermans   

Vrije Universiteit Amsterdam, The Netherlands

Abstract

Software is playing an increasing role in everyone's lives, and therefore it is important (and fun!) for kids to become creators in the digital world. However, existing programming languages are not necessarily designed for learnability, with cryptic error messages and a lack of easily accessible resources. In this talk, Felienne will outline what issues existing tools have, and how these issues disproportionately affect underrepresented minorities in programming including girls, kids with disabilities and non-English learners. She will then outline her story of inventing and creating Hedy, an inclusive, multi-lingual and gradual programming language for learners. Hedy is open source, runs in the browser, is free to use, and is available in 54 different languages (Including English, Spanish, Chinese, Arabic and Hindi). Hedy was launched in early 2020 and now serves about 500,000 monthly users.

2012 ACM Subject Classification Applied computing → Education; Social and professional topics → User characteristics

Keywords and phrases programming education, gradual programming, outreach, Hedy

Digital Object Identifier 10.4230/OASICS.ICPEEC.2024.1

Category Invited Talk



© Felienne Hermans;

licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).


Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 1; pp. 1:1–1:1

OpenAccess Series in Informatics




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

A Domain-Specific Language for Dynamic White-Box Evaluation of Java Assignments

Afonso B. Caniço ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

André L. Santos ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL, Portugal

Abstract

Programming exercises involving algorithms typically involve time and spatial constraints. Automated assessments for such implementations are often carried out in a black-box manner or through static analysis of the code, without considering the internal execution properties, which could lead to falsely positive evaluations of students' solutions. We present Witter, a domain-specific language for defining white-box test cases for the Java language. We evaluated programming assignment submissions from a Data Structures and Algorithms course against Witter's test cases to determine if our approach could offer additional insight regarding incomplete algorithmic behaviour requirements. We found that a significant amount of student solutions fail to meet the desired algorithmic behavior (approx. 21%), despite passing black-box tests. Hence, we conclude that white-box tests are useful to achieve a thorough automated evaluation of this kind of exercises.

2012 ACM Subject Classification Software and its engineering → Software testing and debugging; Software and its engineering → Domain specific languages; Social and professional topics → Student assessment

Keywords and phrases White-box assessment, student assessment, programming education

Digital Object Identifier 10.4230/OASISs.ICPEEC.2024.2

1 Introduction

Students of introductory-level programming courses, such as Algorithms and Data Structures, are expected to develop implementations that conform to specific algorithm behaviour to ensure the correct application of the algorithms under study. As a standard example, students might be tasked with implementing a specific sorting algorithm. Ideally, formative assessment of this kind of exercises should verify algorithmic behaviour – the essence of the subject – by measuring white-box aspects such as the number of operations executed or memory allocation. Automated constructive feedback that allows students to understand any possible mistakes and deepen their understanding is valuable [18, 13], saving time on human feedback and fostering autonomous learning.

While assessment tools providing feedback about the correctness of the outputs of a solution (i.e. black-box testing) are generally available, assessment tools that check internal algorithmic behavior are not (i.e. white-box testing) [9, 14, 12]. We believe that a technique for deeper evaluation of exercises could serve as the backbone for more elaborated automated assessment systems with richer feedback.

In this paper, we present an evolution of our previous work on Witter [3]¹, a library for white-box testing of Java code. We augmented the library with an internal domain-specific language (DSL) written in Kotlin, allowing instructors to define white-box test cases for Java source code that with stateful execution – a limitation of the initial approach. This kind of tests are appropriate to test data structures implemented with classes.

¹ <https://github.com/ambco-iscte/witter>



We tested the DSL against a set of real student programming assignment submissions from the Algorithms and Data Structures course offered at our institution. We observe that students can effectively be misled if only the outputs produced by their implementations are considered in their assessment, and thus conclude that programming assignments where the internal algorithmic behaviour is relevant could benefit from a tool providing execution information.

This paper proceeds as follows. Section 2 discusses related work on automated assessment. Section 3 presents background on our previous work on Witter. Section 4 describes our DSL for test specification. Section 5 describes the evaluation of our approach with student submissions. Section 6 discusses conclusions and outlines future work.

2 Related Work

Recent surveys [9, 14, 12, 5] show that research and development on automated white-box programming assessment systems focus primarily on well-known white-box assessment methods like static analysis or bytecode instrumentation, with our literature review yielding scarce mentions of assessment through the dynamic collection of details of a program's execution – the main novelty of our approach.

Programming languages or libraries such as AspectJ [10]², Javassist [4]³, ASM [2]⁴, and ByteBuddy⁵, require specialized knowledge on code instrumentation, allow existing systems to be augmented with deeper assessment functionalities, but are limited in that they do not allow for the collection of information regarding the code's execution out-of-the-box. The analyses have to be programmed and tailored to specific needs. Similarly, Valgrind⁶ is a code instrumentation toolkit that enables detection of memory management and threading bugs, along with program profiling. While these tools lay the groundwork upon which an automated assessment system may feature white-box analysis, we consider the requirement of specialised knowledge to be a negative factor when it comes to the tools' general accessibility. Additionally, significant analysis or instrumentation is required to accurately collect a relevant amount of information during a program's execution, negatively impacting the applicability of these tools.

Lizard⁷ is a multi-language static analysis tool supporting the Java language, focusing on determining the cyclomatic complexity of implemented functions, among other forms of static code analysis. While cyclomatic complexity is a useful tool for measuring and therefore managing the complexity of a program, it does not provide detailed algorithmic behaviour of a solution. Similar tools, like Cppcheck⁸ for C and C++, focus on bug detection through static code analysis, providing no dynamic analysis functionalities for a program's execution.

ConGu [6] is a runtime verification tool that enables the assessment of Java classes against formal algebraic specifications. Its main goal is to test abstract data types against function domain restrictions and algebraic conditions or axioms that functions must verify.

² <https://www.eclipse.org/aspectj>

³ <https://www.javassist.org>

⁴ <https://asm.ow2.io>

⁵ <https://bytebuddy.net/>

⁶ <https://valgrind.org/>

⁷ <https://github.com/terryyin/lizard>

⁸ <https://cppcheck.sourceforge.io/>

Jeed⁹ is a toolkit for Java and Kotlin in-memory execution with a focus on safety and performance. While Jeed's goals align with those of our proposed library by enabling code execution in a sandboxed environment providing access to code evaluation metrics, its assessment is focused on source code analysis rather than dynamic runtime events. Namely, Jeed supports linting, cyclomatic complexity analysis, and a listing of which language features are present in a given program.

Mooshak [11] is a programming assessment tool that checks whether a submitted program functions correctly. To this effect, Mooshak analyses the programs for their returned or printed outputs, and if any compilation or runtime errors were produced [17]. Mooshak's goal is broad, aiming to be a full online programming context judge for several programming languages [11, 17], and as such, enables third-party extensions to execute custom static and dynamic analysers, which might perform white-box analyses. This aligns with our goal of providing Witter as a library which is easily integrable into existing assessment systems.

JavAssess [8] is a Java library used to integrate deeper code analysis capabilities into existing automated assessment tools. This approach relies on combining traditional black-box unit testing with code instrumentation and meta-programming functionalities, yet does not offer a way to dynamically collect white-box execution metrics. Nonetheless, the goals of JavAssess are considerably similar to Witter's, aiming to be a library that can be integrated into existing automated assessment systems to facilitate a deeper analysis of student code, and the two libraries could work in tandem to provide a comprehensive assessment toolkit. AutoGrader [7] is a similar assessment library, leveraging on the meta-programming functionalities of the Java language along with typical unit testing for code assessment.

3 Background: Witter Library

In previous work we developed Witter, a library for specifying white-box test cases for Java source code [3]. The core functionality of Witter relies on Strudel¹⁰, a library providing an interpreter allowing clients to perform fine-grained observation of code execution events.

In the first version of Witter, test cases were defined by annotating reference solutions with header comments containing different directives that specified each test case and which runtime metrics should be considered. Figure 1 presents an example of using Witter to specify a test for the *insertion sort* algorithm. We can see two test cases and directives to measure the number of array reads/writes and to check if the desired side-effects are met. The white-box testing is based on comparing the measurements of the reference solution with those of a students' solution. For example, one may implement *selection sort* and end up with a sorted array, but the array read/writes will not match, implying that the implementation is not as intended.

While the initial version of Witter was suitable to evaluate algorithms that could be implemented as standalone methods (e.g., involving arrays), it lacked support for assessing stateful or object-oriented solutions. Furthermore, the limited Java support of Strudel at the time also constrained the scope of code solutions we are able to analyze. Meanwhile, we further developed Strudel to support a larger subset of Java's language features and therefore broaden the scope of programs that Witter is able to evaluate.

⁹ <https://github.com/cs125-illinois/jeed>

¹⁰ <https://github.com/andre-santos-pt/strudel>

```

/*
@Test({5, 4, 3, 2, 1})
@Test({3, 2, 5, 3, 1})
@CountArrayReads(2)
@CountArrayWrites(1)
@CheckSideEffects
*/
static void insertionSort(int[] a) {
    for (int i = 1; i < a.length; i++) {
        for (int j = i; j > 0; j--) {
            if (a[j] >= a[j - 1]) break;
            int tmp = a[i];
            a[i] = a[j];
            a[j] = tmp;
        }
    }
}

```

■ **Figure 1** Reference solution of *insertion sort* annotated with Witter tests (initial version).

■ **Table 1** Runtime metrics and corresponding values measured.

Metric	Usage	Measure
Loop Iterations	CountLoopIterations(<i>[margin]</i>)	Number of loop iterations
Array Reads	CountArrayReadAccesses(<i>[margin]</i>)	Number of array read accesses
Array Writes	CountArrayWriteAccesses(<i>[margin]</i>)	Number of array write accesses
Recursive Calls	CountRecursiveCalls(<i>[margin]</i>)	Number of recursive calls
Object Allocations	CheckObjectAllocations	Object allocations per type
Array Allocations	CheckArrayAllocations	Array allocations per type
Side Effects	CheckSideEffects	Side effects on arguments

4 Approach: DSL for White-Box Tests

In order to tackle the inherent difficulty of supporting the assessment of object-oriented implementations through annotated code solutions, we implemented an internal DSL in Kotlin providing a programmatic way for defining stateful test cases. We argue that the DSL requires less effort from instructors for defining test cases when compared to the required knowledge on specialised topics like code instrumentation or meta-programming, since there are only a few DSL directives to be used in a declarative, high-level style.

Figure 2 illustrates Witter’s DSL with a *test suite* for list data structures, containing two *test cases*. These test cases can be configured to use any number of white-box metrics either throughout the test or within a bounded scope (*using* directive). As in the initial version of Witter, evaluation metrics (summarised in Table 1) can be optionally instantiated with a *margin* parameter that specifies an acceptable deviation interval from the reference value, in order not to constrain students’ code to a single, rigid solution.

An object can be created using the *new* directive by passing the name of the class to instantiate followed by a list of arguments to one of the class constructors. References to the created objects can be stored using *ref*. Class methods can be invoked by using the *call* directive on a previously declared reference. A sequence of these directives defines a stateful test case.

```

val tests = TestSuite(referencePath = "path/reference/List.java") {
  Case("testContains") {
    // Create new object and store a reference to it
    val list = ref { new("List") }

    // Executed without white-box metrics (black-box only)
    list.call("size") // 0
    list.call("add", "hello")
    list.call("size") // 1
    list.call("add", "world")
    list.call("size") // 2

    using(CountLoopIterations() + CountArrayReadAccesses()) {
      // These calls compare loop iterations
      list.call("contains", "hello") // true
      list.call("contains", "algorithm") // false
    }
  }

  // All the calls within this case compare loop iterations
  Case(CountLoopIterations(), "testIsEmpty") {
    val list = ref { new("List") }
    list.call("isEmpty", expected = true)
    list.call("add", "hello")
    list.call("isEmpty", expected = false)
  }
}

```

■ **Figure 2** Witter’s DSL syntax. Example Test Suite for a list data structure comprising the operations *add*, *contains* and *isEmpty*.

The *call* directive is used by specifying the name of the method to be invoked and a list of arguments. We may use the “dot notation” to perform calls on instance methods given its reference (*ref.call(...)*). For every call, the return values of the evaluated method are compared to the reference solution, allowing for regular black-box testing. Additionally, if the optional *expected* argument is passed, Witter will assert that both the reference solution and the solution under evaluation produce the expected result. This verification allows educators to assert that their solution works as expected, preventing the accidental usage of a faulty reference solution as the ground truth for evaluating students’ implementations.

Consider an assignment where a student must implement a function for calculating the average of an array of double values. We wish to assess not only the correctness of the produced result, but also that of the algorithm behaviour by checking that the number of loop iterations matches that of the reference solution. Figure 3 illustrates an evaluation scenario for this assignment using Witter’s API, composed of: reference solution (3a), test suite (3b), a solution to check against the tests (3c), a snippet of using Witter as a library (3d), and the output of executing the tests (3e).

Educators design an exercise by providing a pair of artifacts consisting of a reference solution and a corresponding test suite. Different solutions to the exercise (submitted by students) may be checked against the test suite through Witter’s API. The example submission (Figure 3c) has a defect, given that the array iteration starts at index 1 (rather than 0). Witter runs the tests simultaneously for the reference solution and for the solution

2:6 A DSL for Dynamic White-Box Evaluation of Java Assignments

(a) Reference solution (Average.java).

```
static double average(double[] a) {
    double sum = 0.0;
    for (int i = 0; i < a.length; i++) sum += a[i];
    return sum / a.length;
}
```

(b) DSL test suite.

```
val tests = TestSuite("path/to/reference/Average.java") {
    Case(CountLoopIterations()) {
        call("average", listOf(1,2,3,4,5), expected = 3.0)
        call("average", listOf(0,2,3,5,7), expected = 3.4)
    }
}
```

(c) Solution under testing (Solution.java) – with a defect, starting at index 1.

```
static double average(double[] a) {
    double sum = 0.0;
    for (int i = 1; i < a.length; i++) sum += a[i];
    return sum / a.length;
}
```

(d) Invoking the execution of a test suite to a solution under evaluation.

```
val results: List<ITestResult> = tests.apply(
    subjectPath = "path/to/Solution.java"
)
results.forEach { println("$it\n") }
```

(e) Output of the test results.

```
[fail] average([1, 2, 3, 4, 5])
      Expected: 3.0
      Found: 2.8

[fail] average([1, 2, 3, 4, 5])
      Expected loop iterations: 5
      Found: 4

[pass] average([0, 2, 3, 5, 7])
      Expected: 3.4

[fail] average([0, 2, 3, 5, 7])
      Expected loop iterations: 5
      Found: 4
```

■ **Figure 3** Exercise evaluation scenario using Witter's API.

under testing, comparing the two to evaluate the latter's execution. In this case, a mismatch of iterations is detected, given that the solution under testing perform always performs one less iteration in contrast to the reference solution. Notice that in the output of the test results (Figure 3e) this is being reported as a failure, while it succeeds in one of the test inputs.

An automated assessment system may import Witter as a third-party library. In this example we are merely displaying the objects of the test results to the console, but these can be inspected for custom reporting, depending on the assessment system.

5 Evaluation

In order to evaluate the feasibility and usefulness of the approach, we carried out an experiment using the proposed DSL to evaluate student assignments.

5.1 Context

We collected a set of 2,389 student assignment submissions spanning two offerings of the Algorithms and Data Structures course taken by first year undergraduate students of Computer Science and Engineering and related bachelor's degrees. The submission process was independent from our approach, hence no constraints were posed regarding Witter's limitations. For this reason, some submissions could not be handled. We analyzed five distinct assignments with the following guidelines:

1. Implement three different classes, each solving the dynamic connectivity (union-find) problem using different approaches: Quick-Find, Quick-Union, and Weighted Quick-Union with Path Compression.
2. Implement a Queue data structure supporting the String data type. Use a circular resizing array implementation to store the data internally without needing to limit the queue's memory capacity *a priori*.
3. Implement an optimised version of the Insertion Sort algorithm which executes fewer array access operations by avoiding swap operations and instead shifting elements directly one position to the right as needed.
4. Implement a generic List data structure utilising a dynamic sequence of simply-linked nodes internally to store elements.
5. The implementation of the Heap Sort algorithm seen in the lectures assumes the array indices begin at 1. Modify the algorithm's implementation to support sorting arrays starting at index 0.

Each assignment was given to students in this order throughout the semester. The guidelines were translated from Portuguese, which included more API details that have been truncated for presentation in this paper.

Table 2 presents the number of valid submissions used for evaluating the DSL for each assignment. We consider a submission to be valid if the submitted file matches the name and extension indicated in an assignment's guidelines, and if Java can successfully compile the source code. The inconsistent number of submissions is explained by two factors: our observations of a growing rate of absenteeism as the semester progresses; and, every assignment being present in the two course offerings considered except assignment A3, which was only present in a single offering.

■ **Table 2** Number of student submissions used for evaluating Witter’s DSL.

Assignment	Description	Total	Valid
A1	Dynamic Connectivity	584	424 (72.6%)
A2	Resizing Array Queue of Strings	573	490 (85.5%)
A3	Improved Insertion Sort	212	120 (56.6%)
A4	Generic Linked List	550	476 (86.5%)
A5	Heap Sort	470	266 (56.6%)
		2389	1776 (74.3%)

5.2 Method

Figure 4 presents the test specifications for each assignment using Witter’s DSL. Each student submission was evaluated using Witter through the corresponding test specification, and the results of the evaluation were processed to find cases where the student’s implementation passed black-box tests but failed white-box tests. Given the nature of the chosen assignments, the evaluation focused on the metrics for counting loop iterations, array read and write access operations, allocated memory (for resizing array operations), and argument side effects (for checking whether sorting algorithms effectively sorted the input array).

In order to assess Witter’s suitability for large-scale assessment processes, we measure the average execution time for the evaluation of each submission. The execution took place in a laptop system with a 12-core 2.6GHz Intel i7-9750H CPU and 16GB of RAM. While a completely isolated simulation is impossible in a standard system, care was taken to minimise the impact of other operating system processes on the performance of Witter’s execution.

It is usually the case that computer engineering students have a general propensity to cheat in programming assignments [1]. We took this factor into consideration during Witter’s evaluation by running plagiarism analysis on all student submissions using JPlag¹¹, a plagiarism checking tool resistant to a broad range of obfuscation techniques and which provides easily-interpretable results of its analysis [15, 16]. While a connection between plagiarism checking and white-box assessment has not been observed, we include this analysis as a means to safeguard our evaluation against possible accidental biases stemming from students having plagiarised the same incorrect sources (e.g. copying from a fellow classmate who made mistakes).

5.3 Results

Table 3 summarizes the evaluation results. Witter successfully loaded a total of 1,526 student submissions from the 1,776 valid submissions described in Table 2, constituting approximately 86% of all valid submissions. For these, all the assignments had a black-box tests pass rate greater than 90%. However, the white-box tests failure rate ranged approximately between 9% and 45%, with assignment A4 exhibiting the largest failure rate.

Figure 5 presents the distribution of white-box metrics that produced failures in each assignment. Approximately 50% of failed assertions for assignment A1 were caused by an incorrect number of array write operations, with the majority of the remaining errors relating to an incorrect number of array read operations. Out of the considered metrics, assignments A2, A3, and A5 contained white-box errors relating to the number of loop iterations and array

¹¹<https://github.com/jplag/JPlag>

(a) Test specification for assignment A1.

```
Case(CountLoopIterations(1) + CountArrayWriteAccesses(1) +
CountArrayReadAccesses(1)) {
    val uf = ref { new("QuickFindUF", 100) }
    val connected = mutableListOf<Pair<Int, Int>>()
    (1 .. 100).forEach { _ ->
        val p = (100 * random()).toInt()
        val q = (100 * random()).toInt()
        uf.call("union", p, q)
        connected.add(Pair(p, q))
    }
    connected.forEach { uf.call("connected", it.first, it.second) }
}
```

(b) Test specification for assignment A2 (excerpt).

```
Case(CountLoopIterations(1), "testDequeue") {
    val queue = ref { new("Queue") }
    queue.call("enqueue", "witter")
    queue.call("enqueue", "is")
    queue.call("enqueue", "cool")
    queue.call("dequeue")
    queue.call("dequeue")
    queue.call("dequeue")
}
Case(CountLoopIterations(1) + CountMemoryUsage()) {
    val queue = ref { new("Queue") }
    (1..100).forEach { queue.call("enqueue", it.toString()) }
}
```

(c) Test specification for assignment A4 (excerpt).

```
Case(CountLoopIterations(2) + CheckObjectAllocations, "testSize") {
    val list = ref { new("List") }
    list.call("size")
    list.call("add", "hello")
    list.call("size")
    list.call("add", "world")
    list.call("size")
}
```

(d) Test specification for assignments A3 and A5.

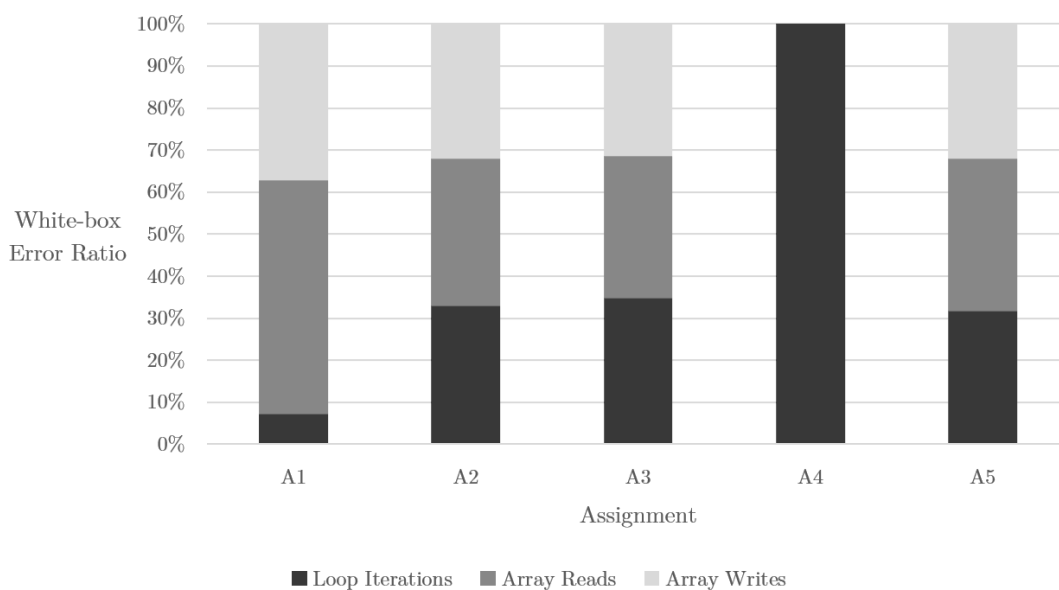
```
Case(CheckSideEffects + CountLoopIterations(1) +
CountArrayWriteAccesses(1) + CountArrayReadAccesses(1)) {
    call("sort", listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
    call("sort", listOf(7, 3, 2, 1, 5, 6, 10, 8, 9, 4))
    call("sort", listOf(7.32, 3.14, 2.14, 1.93, 5.99, 6.74, 10.21,
        8.84, 9.26, 4.56))
    call("sort", listOf("sorting", "algorithms", "are", "really",
        "very", "cool"))
    call("sort", listOf(10, 9, 8, 7, 6, 5, 4, 3, 2, 1))
}
```

■ **Figure 4** Test specifications for the evaluated assignments (A1-5).

2:10 A DSL for Dynamic White-Box Evaluation of Java Assignments

■ **Table 3** Number of black-box and white-box passes and failures for loaded submissions. Loaded submissions are given as a % of valid submissions. Submission pass/fails are given as a % of loaded submissions. Execution time is the average over all submissions of the corresponding assignment.

Assignment	Valid	Loaded	Blackbox Pass	Whitebox Fail	Avg. Time
A1	424	413 (97.4%)	407 (98.5%)	79 (19.1%)	850 ms
A2	490	433 (88.4%)	391 (90.3%)	98 (22.6%)	300 ms
A3	120	110 (91.7%)	107 (97.3%)	49 (44.5%)	20 ms
A4	476	347 (72.9%)	347 (100%)	32 (9.22%)	100 ms
A5	266	233 (83.8%)	221 (99.1%)	58 (26.0%)	30 ms
	1776	1526 (85.9%)	1473 (96.5%)	316 (20.7%)	



■ **Figure 5** Ratio of each white-box metric failure per assignment.

read and write operations, the proportion of which are approximately equal, each constituting approximately one third of each assignment's detected errors. Assignment A4 produced the simplest results, with all failed assertions relating to the number of loop iterations.

Finally, our suspicion regarding code similarity was confirmed by JPlag's analysis, with the average similarity between submissions ranging approximately from 20% to 64%, as summarised in Table 4.

6 Discussion

The results of our evaluation show that a considerable number of students produce faulty implementations when it comes to algorithmic behaviour, which would go unnoticed by an evaluation process focusing only on the results produced by the students' code. Even in the assignment which contained the least faulty implementations (A4) 32 students could be misled by a black-box-only assessment process, a number which we argue is not negligible. Furthermore, in assignment A3 (lowest submission similarity), which required arguably the most originality in developing an alternative version of a standard algorithm, we saw a 44.5% failure rate, which constitutes nearly half of all considered submissions. We can

■ **Table 4** Average submission code similarity per assignment.

Assignment	Average \pm Std. Dev.
A1	64.2 \pm 20.0%
A2	25.7 \pm 17.5%
A3	19.8 \pm 17.6%
A4	22.6 \pm 14.0%
A5	44.1 \pm 24.0%

thus conclude that the usage of an assessment tool which provides information about the execution of students' implementations is useful, not only from the perspective of instructors aiming at accurate grading, but also to prevent misleading students by informing them of the correctness of their implementation's results regardless of how those results were produced.

The current version of Witter successfully tackles the main limitations of our previous work [3], with the internal development of Strudel broadening the scope of supported language features and therefore that of supported assignments, coupled with the development of a DSL for test specification, allowing the specification of stateful tests for assignments using objects (e.g., for implementing data structures).

Our current work on Witter and Strudel allowed an acceptable coverage of standard Java constructs and language features, as seen by the overall 85.9% successful file loading rate seen during the evaluation phase. Nevertheless, further work should be carried out to extend the scope of supported functionalities and therefore enable the usage of more diverse assignments for a more detailed evaluation.

The average time taken to execute the evaluation of each submission reveals an acceptable performance for using Witter in educational contexts, with each submission taking from a few milliseconds to no more than one second to be evaluated. The execution time is dependent on the complexity of the specified tests, with longer or more comprehensive tests corresponding to a longer execution time.

Care should be taken in future work to tackle the limitations introduced by the characteristics of the dataset chosen for evaluation. For instance, taking Strudel's supported functionalities into account during the submission process could provide a larger usable dataset, enabling a more significant evaluation. Furthermore, a more unbiased evaluation could be carried out by utilising an external, publicly-available dataset of programming student submissions, guaranteeing the usage of code produced by students outside of our institution. Additionally, this could tackle possible issues of cheating or plagiarism, whose presence was made evident by our evaluation process and, as hypothesised, could skew the results by introducing a bias relating to students unknowingly plagiarising from incorrect sources. While an average code similarity between submissions of approximately 20%, as for assignment A3, can fall within reasonable expectations for an assignment of this scope, values of 44% or 61% average similarity, as seen for assignments A5 and A1, respectively, begin to raise concerns of considerable plagiarism and how it can affect the results of our analysis.

We continue to envision Witter as a tool not only aiming to be integrated into existing automated assessment systems as a way to extend the scope of their assessment functionalities, but also as courseware to be used by introductory programming students in a classroom environment, providing a learning process augmented through instant feedback on their attempts to solve programming exercises. To this effect, we envision the implementation of more high-level metrics, such as counting the number of array swap or move operations, which are standard when analysing programs from an algorithm complexity standpoint, and thus relevant for introductory programming classes focusing on algorithms.


While we have not yet been able to conduct a user study with programming students or instructors, this is likely to be our main focus in the future as the scope and stability of both Witter and Strudel’s functionalities increase. Namely, a study with introductory programming students is necessary to gauge whether Witter is useful for its envisioned context, and a study with programming instructors can offer insight into the effort required to develop assignments adopting Witter’s DSL. Finally, given a longer time frame, a study could be conducted in the context of an introductory programming course to analyse the long-term effect on the usage of Witter or similar assessment tools in students’ learning outcomes.

References

- 1 C.L. Aasheim, Paige Rutner, L. Li, and S.R. Williams. Plagiarism and programming: A survey of student attitudes. *Journal of Information Systems Education*, 23:297–314, January 2012.
- 2 Eric Bruneton, Romain Lenglet, and Thierry Coupaye. Asm: A code manipulation tool to implement adaptable systems. In *In Adaptable and extensible component systems*, 2002. URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.5769>.
- 3 Afonso B. Caniço and André L. Santos. Witter: A library for white-box testing of introductory programming algorithms. In *Proceedings of the 2023 ACM SIGPLAN International Symposium on SPLASH-E*, SPLASH-E 2023, pages 69–74, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3622780.3623650.
- 4 Shigeru Chiba. Load-time structural reflection in java. In Elisa Bertino, editor, *ECOOP 2000 — Object-Oriented Programming*, pages 313–336, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
- 5 Sébastien Combéfis. Automated code assessment for education: Review, classification and perspectives on techniques and tools. *Software*, 1(1):3–30, 2022. doi:10.3390/software1010002.
- 6 Pedro Crispim, Antónia Lopes, and Vasco T. Vasconcelos. Runtime verification for generic classes with congu2. In *Proceedings of the 13th Brazilian Conference on Formal Methods: Foundations and Applications*, SBMF’10, pages 33–48, Berlin, Heidelberg, 2010. Springer-Verlag.
- 7 Michael T. Helmick. Interface-based programming assignments and automatic grading of java programs. *SIGCSE Bull.*, 39(3):63–67, June 2007. doi:10.1145/1269900.1268805.
- 8 David Insa and Josep Silva. Automatic assessment of java code. *Computer Languages, Systems & Structures*, 53:59–72, 2018. doi:10.1016/j.cl.2018.01.004.
- 9 Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. A systematic literature review of automated feedback generation for programming exercises. *ACM Trans. Comput. Educ.*, 19(1), September 2018. doi:10.1145/3231711.
- 10 Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In Jørgen Lindskov Knudsen, editor, *ECOOP 2001 — Object-Oriented Programming*, pages 327–354, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- 11 José Paulo Leal and Fernando Silva. Mooshak: a web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003. doi:10.1002/spe.522.
- 12 Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaoqing Shi. Automated grading and feedback tools for programming education: A systematic review. *ACM Trans. Comput. Educ.*, December 2023. Just Accepted. doi:10.1145/3636515.
- 13 Samim Mirhosseini, Austin Z. Henley, and Chris Parnin. What is your biggest pain point? an investigation of cs instructor obstacles, workarounds, and desires. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, pages 291–297, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3545945.3569816.

- 14 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.
- 15 Timur Sağlam, Moritz Brödel, Larissa Schmid, and Sebastian Hahner. Detecting automatic software plagiarism via token sequence normalization. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ICSE '24*, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3597503.3639192.
- 16 Timur Sağlam, Sebastian Hahner, Larissa Schmid, and Erik Burger. Obfuscation-resilient software plagiarism detection with jplag. In *46th IEEE/ACM International Conference on Software Engineering: Companion Proceedings, ICSE-Companion*. Institute of Electrical and Electronics Engineers (IEEE), 2024. doi:10.1145/3639478.3643074.
- 17 Manuel Sánchez, Päivi Kinnunen, Cristóbal Flores, and J. Ángel Velázquez-Iturbide. Student perception and usage of an automated programming assessment tool. *Computers in Human Behavior*, 31:453–460, February 2014. doi:10.1016/j.chb.2013.04.001.
- 18 Anne Venables and Liz Haywood. Programming students need instant feedback! In *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20, ACE '03*, pages 267–272, AUS, 2003. Australian Computer Society, Inc.

Seven Years Later: Lessons Learned in Automated Assessment

Bruno Pereira Cipriano¹ ✉ 

COPELABS, Lusófona University, Lisbon, Portugal

Pedro Alves ✉ 

COPELABS, Lusófona University, Lisbon, Portugal

Abstract

Automatic assessment tools (AATs) are software systems used in teaching environments to automatically evaluate code written by students. We have been using such a system since 2017, in multiple courses and across multiple evaluation types. This paper presents a set of lessons learned from our experience of using said system. These recommendations should help other teachers and instructors who wish to use or already use AATs in creating assessments which give students useful feedback in terms of improving their work and reduce the likelihood of unfair evaluations.

2012 ACM Subject Classification Applied computing → Computer-assisted instruction

Keywords and phrases learning to program, automatic assessment tools, unit testing, feedback, large language models

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.3

Funding This research was funded by the Fundação para a Ciência e a Tecnologia under Grant No.: UIDB/04111/2020 (COPELABS).

1 Introduction

Automated assessment tools (AATs) are software tools used to automatically grade students' software solutions. These tools allow students to check if their code respects the assignment's requirements in terms of functional and/or code quality requirements, leading to increased student autonomy and reduced teacher workload [9, 13].

We have been using such a system since 2017 for multiple types of assessment, from small formative exercises to larger scale projects which have a large weight on the students' grades. Furthermore, we have employed automatic assessment in multiple courses, ranging from Introduction to Programming, Data-Structures and Algorithms, and Object-Oriented Programming (OOP).

Our team employs a common practice in terms of automatic assessment. The starting point of the process is a text-based description of the exercise(s) or project to solve. This description is sometimes complemented with auxiliary diagrams containing mathematical formulas, UML diagrams, and so on. Besides presenting the description, the assignment also informs students about any mandatory files, classes and/or functions, thus defining an Application Programming Interface (API). Finally, when students submit a solution², their code is validated by a set of teacher-defined unit tests. For example, here's the assignment text for a typical exercise used in introductory programming courses: *Implement the function `static long sum(int n1, int n2)` which should return the sum of its two arguments.* Listing 1 shows an unit test which could be used to validate this assignment.

¹ corresponding author

² Assignments allow multiple submissions, promoting that students improve their work and learning outcomes.



3:2 Seven Years Later: Lessons Learned in Automated Assessment

■ **Listing 1** An example *JUnit* test for an exercise where students must implement a function which sums two integers. The feedback messages are only displayed when the respective assertion fails.

```
1 @Test
2 public void testSum() {
3     String feedback = "returned the wrong value";
4     assertEquals(5, Main.sum(1, 4), "sum(1, 4) " + feedback);
5     assertEquals(10, Main.sum(8, 2), "sum(8, 2) " + feedback);
6 }
```

A more complex OOP exercise could be the following: (...) *The GameManager class must have a static List<Player> getPlayers() function which returns a list with the players that are participating in the current game.* Listing 2 presents an example of a unit test for validating that requirement.

■ **Listing 2** An example *JUnit* test for an OOP project where students must implement a board game. The messages are only displayed when the respective assertion fails.

```
1 @Test
2 public void testGetPlayers01() {
3     GameManager manager = new GameManager("game-board-with-2-players.txt");
4     List<Player> players = manager.getPlayers();
5     assertNotNull(players, "getPlayers() should not return null");
6     String feedback = "getPlayers() returned a list of the wrong size";
7     assertEquals(2, players.size(), feedback);
8 }
```

We opted for this unit-tests-based approach because it is a more robust way of testing than the alternative of using “output matching”. Also, it is closer to current industry practices, with the added benefit of familiarising students with the types of tools that they will more likely use in their future careers.

Our experience over these last 7 years showed us that certain testing practices or inadequate feedback can lead students to request extra support from the teaching staff, meaning that their autonomy is reduced and that valuable teacher time is consumed. Consequently, this undermines two key benefits of employing AATs: fostering student autonomy and optimizing teacher resource allocation. Furthermore, certain testing patterns can result in assessments which are too strict and even unfair, either by letting students receive better grades than they should – for example, by considering hardcoded solutions as correct – or by penalizing them too much – for example, by not informing students when their code fails due to minor formatting issues.

From our experience, we have derived a set of recommendations which we share with new teaching assistants who join our courses and have to prepare new auto-graded assignments.

In this paper, we share those lessons with the Computer Science Education (CSE) community. We believe this information might be useful to other teachers and instructors who wish to use start using or already use AATs.

Finally, the recent advent of Large Language Models (LLMs) – artificial intelligence tools with capacity to generate computer code from natural language descriptions – such as OpenAI’s GPT and Google’s Bard has generated some controversy in the CSE community, due to the risk of producing graduates who are weak in fundamental computing skills such as computational thinking and program design [11]. While LLMs pose certain risks, they also offer new opportunities for the CSE community. We explore how integrating AATs with LLMs can generate beneficial possibilities.

This paper makes the following contributions:

- Presents a list of recommendations for defining unit tests for automatic assessment, aimed at enhancing student autonomy and grading fairness;
- Presents adaptations of those recommendations for specific assessment types;
- Discusses how Large Language Models can play a role in the future development of AATs.

This paper is organized as follows: first, in Section 2, we review relevant papers related with automatic assessment, feedback and LLMs. In Section 3 we present 1) a categorization of assessment types typically found in computer programming courses, 2) some AAT concepts, and, 3) some Unit Testing concepts. Section 4, presents our main contribution: recommendations for test-case creation and feedback definition. Section 5, discusses the future of AATs in light of the recent advent of LLMs. In Section 6 we address some threats to the validity of this paper. Finally, in Section 7, we finish our paper by drawing some conclusions.

2 Related work

Automated Assessment. AATs have been used since at least 1960 [13]. These tools evaluate students' code solutions to determine if they satisfy the requirements of the assignment, giving students more autonomy and reducing teacher workload [13]. This is usually done by automating the execution of students' code in a restricted environment and validating its correctness by executing unit tests, performing output matching, or by other techniques [9].

Research on this area is varied, with many studies presenting tools, their advantages, limitations and impact on students' learning [5], amongst other topics [9, 13]. Some of these tools attempt to simulate professional software development work environments promoting that students work using Integrated Development Environments ([3, 2]) and even complementary tools such as Git (e.g. [8, 3]), while others introduce more virtual work environments which are usually implemented using web-based solutions (e.g. [4, 16]).

AAT cheating patterns. Research has also delved into students' ability to work around automatically graded assessments and get successful scores without performing the assignment as prescribed by the teachers. The authors of [10] analysed students' code submissions in 2 AAT-based courses, identifying and categorizing cheating attempts into 4 distinct patterns:

- **Overfitting:** Students' code is hardcoded to match expected output; ineffective beyond the specific assignment's tests. Overfitting corresponded to 63% of the cheating attempts.
- **Problem evasion:** Students' code fails to meet non-functional requirements (e.g. the use of recursion). It accounted for 30% of the cheating attempts.
- **Redirection:** Students call the reference solution from their own code. Redirection corresponded to 6% of the cheating attempts;
- **Injection:** Malicious code is injected to change the assignment's validation process. This pattern corresponded to 1% of the cheating attempts.

Some of these issues, particularly Redirection and Injection, are highly dependent on the AAT's architecture. For instance, AATs that isolate the reference solution from the students' code, such as [3], indirectly prevent Redirection. However, both Overfitting and Problem evasion are cheating patterns which do not directly exploit AATs' weaknesses.

3:4 Seven Years Later: Lessons Learned in Automated Assessment

Feedback. Feedback is considered a facilitator of learning and performance [15]. In a literature review focused on feedback, Shute defined formative feedback as *information communicated to the learner that is intended to modify his or her thinking or behaviour to improve learning* [15]. Formative feedback can be presented to the learner using information (e.g. verification of response accuracy, explanation of the correct answer, hints) and can be administered at various times during the learning process (e.g. immediately following an answer, after some time has elapsed) [15]. This research also offered guidelines for creating formative feedback. In a recent literature review focused on formative feedback for computer science students [17], the authors concluded that, to better assist novice programmers in learning programming, formative assessments must improve the presentation of error messages. Finally, in a study evaluating the use of GPT-3.5 for generating hints for problems detected by an AAT [14], students were divided into two groups – one receiving GPT-based hints and the other only receiving regular AAT feedback. The usefulness of the GPT hints was assessed using a 5-point Likert scale; 46% of students rated them highly (4 or 5), while 19% gave low ratings (1 or 2). The study found that students with access to GPT-based hints tended to use the AAT’s default feedback less. However, the findings also suggest some potential over-reliance on the GPT-generated feedback.

In summary, despite extensive research in this field, we are not aware of any published studies providing guidelines or recommendations for creating test functions/cases and determining the appropriate level of feedback to support programming students in their learning.

3 Definitions

3.1 Assessment types

Our experience with automated assessment has led us to conclude that different assessment (or evaluation) types require different approaches, due to their specificities (e.g. whether the evaluation is time-bound or not).

This section presents the 4 evaluation types for which we recommend that different levels of feedback are considered and provided. These definitions should help teachers bridge our evaluation types with their own, and thus be able to more effectively apply our lessons and recommendations.

In-class/homework exercises: These are usually small exercises in which students have to implement code to solve small problems (e.g. 20-100 LOC) using specific techniques (e.g. iteration, recursion, etc). These are usually formative, although they might have some contribution to the students’ grades. During said exercises, students can get support from both teachers as well as their colleagues.

Mini-test: Small exercises to be done individually and independently (e.g. without support). These exercises are time-bound (e.g. maximum one hour) and count towards the students’ final grade.

Project: Mid-to-large scale exercises which take weeks or even months to solve. These usually have a larger weight in the students’ final grades.

Defense: A proctored evaluation where students must make changes to their assignment’s code, in order to demonstrate knowledge of it. This is usually done for projects and other evaluations with significant impact in the students’ grades, in order for teachers to have more confidence that students actually authored the delivered code instead of copying or plagiarising. For group projects, it has the added benefit of adjusting the grades given to different group elements.

3.2 Test function types

Usually AATs inform students about their results. However, some AATs allow the definition of hidden tests, which are tests whose existence and result is not disclosed.

Public tests: Test functions whose results are reported to the student (e.g. correct or incorrect) who possibly also receives messages with extra feedback.

Private or hidden tests: Test functions which are executed without the student receiving any feedback. Hidden tests are a common strategy to counter the usage of overfitted solutions: teachers prepare a public test for a function and then prepare a private test for the same function, but with different test cases. Since students will not even know that the test exists, they will not be able to derive an overfitted solution that passes both the public and the hidden tests. Hidden tests also have the potential for better distinguishing between students' grades, since students who test their code locally with more cases than the ones made available in the public test function might end up also passing the hidden tests. However, this will probably depend on the cases which are actually been tested and whether they are corner and/or otherwise complex cases.

3.3 Unit Testing frameworks

Unit Testing frameworks typically supply programmers with assertions: functions which verify if a certain condition is being respected. Some notable examples are: “assertEquals()”, “assertTrue()”, and “assertFalse()”. They also typically have a “fail()” function, which can be used when the programmer wants to test a condition without using one of the default assertion functions.

These assertion functions usually receive an optional “message” parameter, which can be used to customize the message that is displayed when the assertion fails. In professional software development practice, this parameter is usually not very relevant, because programmers have access to the test code and can observe the assertion's implementation details. However, it gains importance when developing tests for AATs, since students will usually not have access to the assignment's test code.

Several Unit Testing frameworks exist for a variety of programming languages. The most well-known example for Java is *JUnit* [6]. Another example is Python's *unittest*³ library.

4 Lessons Learned: recommendations

This section presents our recommendations. The examples are given in Java, using *JUnit* 5, but they can easily be adapted to other languages and frameworks, such as Python.

4.1 Generic recommendations

4.1.1 Tackling problem evasion

In order to detect cases where students try to evade the prescribed problem (e.g. recursion), we recommend that static analysis tools are employed in the assessment process. An example tool which can be used to do this for Java code is the Checkstyle plugin⁴ which, among other functionality, allows teachers to define forbidden keywords (e.g. “for”, “while”) and signals their usage. This enables teachers to inspect and/or penalize solutions which use those keywords.

³ <https://docs.python.org/3/library/unittest.html>

⁴ <https://checkstyle.org/>

Another useful technique is Reflection [12]. This technique allows certain non-functional requirements, such as the need for a class to be abstract, to be validated using unit tests.

4.1.2 Test construction

Before we delve into our recommendations for test case construction, it should be noted that, in some scenarios, it might be necessary for the exercises to be changed in order for them to be properly evaluated in an automatic fashion using unit tests. This means that test functions influence API design, similarly to what happens in test driven development. In order to apply some of our recommendations, the tested functions should always receive at least one parameter, and the more parameters, the easier it will be to create rich test-cases. Consider this when evaluating our first 2 recommendations: they might seem somewhat contradictory, but they are achievable by adjusting the exercises' APIs.

At least 2 independent test functions for each API function

We recommend creating at least two independent test functions for each of the exercise's API. As an example, consider this exercise: *Implement a recursive function that compares two char arrays through an extra argument representing the initial position (assume a1 and a2 have equal length):* `static boolean equalArrays(char[] a1, char[] a2, int initialPos)`

You can break this problem down into different test functions, considering different input cases, such as 1) empty or null arrays, 2) initial position greater than the length of the array, 3) arrays with just one element, and, 4) arrays with 2 or more elements. By having multiple small test methods for each function, you allow gradual exercise completion with tests of increasing difficulty (an essential ingredient of gamification). Not only does this increase the student's motivation, but it also allows for more fine-grained grading (i.e. a student who is able to implement the base case but not the recursive case doesn't have zero). Also note that each of these test-methods can further help the student by having a name which hints at the case being tested (e.g. `testEqualArraysSingleElement()`).

At least 2 sub-cases in each test function

Each test should have at least 2 sub-cases for the same function, with different expected return types, in order to prevent hardcoded/overfitted solutions from being considered correct, even if only partially. This should be implemented by having 2 different assertions with different return values. The most obvious example for this appears when evaluating boolean functions: if the test function only has cases for when the expected value is "true" (as in Listing 3), then a solution which always returns "true" is going to pass that test.

■ **Listing 3** Test function which validates a boolean function using a single assertion. This test will pass if the student's `isOpen()` function returns an hardcoded "true".

```

1 @Test
2 public void testDoorOpening() {
3     Door door = new Door();
4     door.open();
5     assertTrue(door.isOpen(), "isOpen() incorrectly returned false");
6 }

```

On Feedback

Since we want to help students to solve the exercises, assertions without a specific message should be avoided (consistent with previous recommendations on feedback generation [15]). As such, the minimum feedback should, at least, indicate the name of the function returning the incorrect result (e.g. “getSalary() returned the wrong value” or, at least, “getSalary()”). This recommendation might be slightly relaxed if we are testing a function with a very specific return format which will make it obvious for the students which is the tested function. For example, when assessing an object’s “toString()” method, if its output format significantly differs from that of other objects, this approach may be appropriate.

However, just mentioning the function name might not be sufficiently helpful. For example, consider the assertion in Listing 4:

■ **Listing 4** In certain cases, just referring the function name is insufficient and might cause confusion.

```

1 List<String> movieTitles = Main.getTitles();
2 boolean found = movieTitles.contains("The Matrix");
3 assertTrue(found, "getTitles()");

```

If the title “The Matrix” is not included in the list, the respective assertion will simply yield “AssertionFailedError: getTitles() => Expected: true Actual: false”. This will let the student know that something is wrong with the function “getTitles()”, but, unless the problem is very obvious, debugging will be hard since due to lack of clues. Also, we have seen situations where this pattern has led some students to confusion, due to the expected/actual result indicating “true” and “false” when the “getTitles()” function is not supposed to return a boolean. It would be more helpful to let students know which value is missing so that they can direct their debugging efforts. Refer to Listing 5 for an alternative implementation which contains more actionable feedback:

■ **Listing 5** A test with more actionable feedback than the one in Listing 4.

```

1 List<String> movieTitles = Main.getTitles();
2 String title = "The Matrix";
3 boolean found = movieTitles.contains(title);
4 assertTrue(found, "getTitles() missing title: " + title);

```

Of course, giving students this extra piece of information could lead to partial or full overfitting, since the student could insert the “The Matrix” String in the returned list to circumvent their original error. In fact, students’ ability to produce overfitted solutions is highly dependent on the information which is available to them, both via the assignment’s description as well as via the employed test cases. To better understand this idea, consider the example unit test presented in Listing 1. That test can be easily bypassed using the solution shown in Listing 6:

■ **Listing 6** Overfitted solution that passes the test defined in Listing 1. It would be easy for a student to hardcode a solution for that test after seeing its feedback.

```

1 static long sum(int n1, int 2) {
2     if(n1 == 1) {
3         return 5; // first assertion indicates 1 + 4
4     }
5     else if(n1 == 8) {
6         return 10; // second assertion indicates 8 + 2
7     }
8     return 0;
9 }

```


As such, it is necessary to find a balance between the level of feedback which is given to students: on one hand, too much feedback might lead to overfitted solutions, since, if students know all expected input/output pairs, then they can prepare a function which returns the correct result but is not generic. On the other hand, too little feedback will leave students lost and unable to progress. This is particularly important when dealing with API functions such as *JUnit*'s “`assertTrue()`” and “`assertFalse()`”, which provide no clue other than “`AssertionFailedError: Expected: true Actual: false`”. This problem is made even worse in courses where the tests might be calling multiple functions, which commonly happens in OOP exercises. In those cases, such reduced feedback will likely make students wonder which of their functions has the problem.

However, with a varied number of test cases per test function (as previously recommended), it will be hard for students to implement an overfitted version which passes all the sub-cases. Furthermore, teachers can make testing functions which give a lot of detail in the first few cases, and little to no detail in the final cases ([15]). See Listing 7 for an example.

■ **Listing 7** Test function with a progressive reduction of feedback level.

```

1  @Test
2  public void testSumArray() {
3      String suffix = "returned an incorrect value.";
4
5      // feedback includes the arguments (vulnerable to overfitting)
6      int[] array1 = new int[]{1, 4};
7      assertEquals(5, Main.sum(array1), "sum({1, 4}) " + suffix);
8
9      // feedback includes the arguments (vulnerable to overfitting)
10     int[] array2 = new int[]{2, 2, 2};
11     assertEquals(6, Main.sum(array2), "sum({2, 2, 2}) " + suffix);
12
13     // feedback doesn't include the arguments, only a hint
14     String suffix2 = suffix + "Consider arrays w/ negative numbers.";
15     int[] array3 = new int[]{-1, -2, -3};
16     assertEquals(-6, Main.sum(array3), "sum(...) " + suffix2);
17
18     // feedback only includes the function name
19     int[] array4 = new int[]{1, 2, 3, 4, -1};
20     assertEquals(9, Main.sum(array4), "sum(...) " + suffix);
21 }

```

In some situations, besides the name of the function and some hints ([15]) with regards to the expected values, the minimum feedback should also indicate relevant particularities of what is being tested. For example, imagine that we are testing a function called “`int[] getStatistics()`” which returns an integer array where each element has a different semantics: the first element is the input array’s minimum value, while the second element is the input array’s maximum value. If we provide students with simply:

```
assertEquals(42, Main.getStatistics()[1], "getStatistics()");
```

debugging will be hard due to lack of information. It can also cause interpretation issues, since the student might be confused when seeing a message similar to “`getStatistics() expected: 42 but was: 99`” since “`getStatistics()`” returns an “`int[]`” and not a single “`int`”. In this example, it would be preferable to define the test’s feedback in order to let the student know which of the array’s positions has the wrong value, as follows:

```
assertEquals(42, Main.getStatistics()[1], "getStatistics()[1]");
```


This will give the student extra clues with regards to where to start analysing the bug.

Finally, the amount of feedback also depends on the exercise type. This will be further expanded later in this paper.

On hidden tests

Avoid using hidden tests to validate things such as String contents. It is easy for students to make minor mistakes, such as having an incorrect capitalization or spelling mistake. Validating such behaviours only in a hidden test might be too penalizing. Furthermore, although hidden tests can be an interesting tool, they can also be the source of problems: since their results are not visible to the students, eventual errors are likely to go unnoticed. An option to mitigate this issue is to use an AAT which is able to indicate how many students are passing each test [3], and, if no student passes a test or only a small minority of students pass it, then maybe the teaching staff should review it, since it might be incorrect.

Assert-early and often

Consider performing one or more assertions after each action in the test, instead of performing multiple assertions after executing a number of actions. This is particularly helpful when validating OOP assignments, where an object's state changes with each action, meaning that an error on the first action might cause problems in the following actions. If the first assert is done after multiple actions, the resulting feedback might induce the student in error. Listing 8 presents an example of this strategy.

■ **Listing 8** Example of the assert-early and often strategy. After each action, a number of assertions is performed, in order to point to the bug as early as possible.

```
1 Door door = new Door();
2
3 // perform an action
4 door.open();
5 // assert that the object's state is 'open' after the first action
6 assertTrue(door.isOpen(), "isOpen() incorrectly returned false");
7 // also validate the object's toString()
8 assertEquals("This door is open!", door.toString());
9
10 // perform another action
11 door.close();
12 // assert that the object's state is 'closed' after the second action
13 assertFalse(door.isOpen(), "isOpen() incorrectly returned true");
14 // also validate the object's toString()
15 assertEquals("This door is closed!", door.toString());
```

Null references

Be careful when creating test cases which might result in hard to debug errors such as “NullPointerException”. Consider using “assertNotNull()” to let students know where their code is returning unexpected null references. See Listing 9 for an example.

3:10 Seven Years Later: Lessons Learned in Automated Assessment

■ **Listing 9** Gracefully handling potential null references to provide useful feedback.

```
1 // unprotected approach, which will result in a hard to debug
2 // NullPointerException
3 Actor actor1 = Main.getActorByID(1000);
4 assertEquals("Keanu Reeves", actor1.getName(), "Actor getName()");
5
6 // protected approach, which will let the student know that
7 // an unexpected null reference was returned
8 Actor actor2 = Main.getActorByID(1000);
9 assertNotNull(actor2, "getObject(1000) returned null");
10 assertEquals("Keanu Reeves", actor2.getName(), "Actor getName()");
```

Test-function names

Although the industry norms and conventions for test code do not recommend prefixing test functions with “test_” nor having a number for each test, we believe that these two practices are actually helpful in educational contexts. Including said prefix has the advantage of facilitating the interpretation of the tests’ stack-traces (when available) since it makes it obvious which of the functions in the stack is the test. Furthermore, numbering the test makes it somewhat easier to communicate with the student.

The order of the tests matters

Even though, according to the best practices, the tests should be independent of each other, students tend to tackle each failed test by the order it shows up in the report provided by the AAT. As such, consider ordering the tests by difficulty level.

Progressive disclosure

Consider activating certain advanced tests only after students pass simpler tests. This is not directly accomplishable by *JUnit* (since it goes against the philosophy of independent tests) but it is easily implemented using static variables (see Listing 10). Important: Students should know that these tests exist and what is needed to “unlock” them (e.g. “This test will only be executed after you pass the tests based on the small input files”). This rule also interacts with the previous one: the feedback for more complex tests should appear after the feedback for the simpler tests.

■ **Listing 10** Progressive disclosure: this test is only executed after the student passes at least 4 simple tests.

```
1 @Test
2 public void advancedTest() {
3     if (SimpleTests.testsOK < 4) {
4         fail("This test will be unlocked after passing 4+ simple tests");
5     }
6     // ... actual test
7 }
```

Tests involving collections

Be careful with “OutOfBounds” runtime errors: check the length of the collection before accessing individual elements. However, don’t just check the total length of the collection upfront, since it will be difficult for students to understand the problem when they have insufficient or excessive data. Instead, first validate a few elements in order to allow students to debug missing values. See Listing 11 for examples of both approaches.

■ **Listing 11** Testing collections' contents: its better to validate a few examples before asserting the full list size.

```
1 List ids = Main.calculateIds();
2
3 // this is difficult to debug
4 assertEquals(347, ids.size(), "calculateIds() returned wrong number of elements");
5
6 // this is better
7 int size = ids.size();
8 assertTrue(size >= 2, "calculateIds() should have returned at least 2 elements, but
9   returned " + size);
10 assertEquals(3, ids[0], "calculateIds()[0]");
11 assertEquals(5, ids[1], "calculateIds()[1]");
12 assertTrue(size >= 5, "calculateIds() should have returned at least 5 elements");
13 assertEquals(6, ids[4], "calculateIds()[4]");
```

Include a timeout on each test

In order to detect infinite loops and/or implementations with inappropriate time complexity (i.e. big-O), teachers should consider defining timeouts for some or all of the test functions.

4.2 Specific recommendations per assessment type

As we've seen before, in order to allow students to act and solve a problem reported by the AAT, the respective tests should supply the student with some level of information. We argue that the ideal level of information will be different depending on the assessment type.

In-class/Homework exercises

Due to the formative goals of these exercises, we consider it important to provide students with feedback which allows them to fully master the material. As such, all test functions should be public and all, or at least the majority, of assertions should have significant feedback.

Mini-tests

In such assignments, we recommend that some feedback is given, but at least one of the asserts should give very little feedback, in order to reduce the likelihood of *overfitted* solutions. For example, at least the last assertion of each test function should not give any hints in terms of input and expected output. All test functions should be public, since it is important for students to have a real notion of how they are doing.

Projects

Use hidden tests to make it harder for *overfitted* solutions to get good grades. Define a subset of the public tests as “mandatory”. These tests will define the minimum mandatory functionality which a student must implement in order to get a passing grade, also promoting that the approved projects have a minimum level of quality. Furthermore, if you employ project defenses, having mandatory tests will aid in avoiding certain issues (refer to the next paragraph for details). Hidden tests should not be mandatory, as students are unaware of their existence and cannot take action to address or rectify any related issues.

Defenses

The rules presented for mini-tests also apply to project defenses, since both are time-bound evaluations where students tend to become stressed. However, a few additional recommendations make sense when defining tests for project defenses. If possible, build the defenses around the project’s “mandatory functionality” in order to prevent penalizing students whose projects have bugs that were not detected originally. If “mandatory” tests were not defined, teachers should avoid testing scenarios which were only tested in the project’s hidden tests. These two recommendations aim at reducing the chance of penalizing students whose projects have pre-existing bugs that might condition their performance in the defense. Furthermore, for similar reasons, we recommend using adapted versions of the project’s public tests in the defense’s tests (e.g. same input file, similar actions, and so on). Teachers should also be careful with interdependence between the test cases. For example, if the defense’s assignment asks students to change the “toString()” function, then it might be dangerous to use the “toString()” as a criteria to determine the correctness of other defense tests. Finally, common pitfalls should be identified and avoided. For example, in projects involving people’s names, some students only support two-named individuals and overlook those with multiple names, like “Samuel L. Jackson” versus “Samuel Jackson.” It’s important to recognize and steer clear of these issues in project defenses, as students often fail to address them despite detailed feedback.

4.3 Other recommendations

Sometimes students complain that a test is incorrectly failing, justifying their position with something along the lines of “in my computer, it works”. As such, we find it important to educate students on how their code will be tested.

One way of doing this is to teach them how to programmatically test their own code. Even so, some students might still struggle with transferring this skill to the interpretation of teachers’ tests, due to lack of testing experience and/or other issues. To help with this, consider giving students access to the code of one of the assignment’s tests, in order to allow them to reflect on how testing is being performed. This might have the added benefit of students making their own adaptations of the sample test, thus better testing their code.

Finally, we also suggest a strategy of promoting that students write their own tests when they ask for help. In response to help requests, we often prompt students to demonstrate a unit test that evaluates a specific aspect, akin to our AAT’s test. Although this usually requires some interactions until the student produces a test which mimics the actual problem, it has the advantage of forcing students to enter a testing-oriented mindset.

5 The Future of Automatic Assessment

Although LLMs have been received somewhat controversially by the CSE community [11], we expect CS educators to adopt these technologies in their teaching practices sooner or later. More concretely, we believe LLMs will be used to improve AATs and/or assessment practices [1].

Some possible research avenues are: 1) using LLMs to automate the generation of tests ([11]) that respect the recommendations presented in this paper; 2) integrating LLMs and AATs to simplify the process of providing hints for failed tests, similarly to [14], but with guardrail mechanisms to prevent LLM over-reliance; 3) employing LLMs to generate customized feedback that considers both the student’s code as well as the issues identified

by the AAT; 4) integrating LLMs into AAT's evaluation schemes ([7]) to detect issues such as overfitting and problem evasion; 5) replacing AATs with LLMs entirely. Note that developments in terms of improving and/or complementing AAT feedback (i.e. #2 and #3) might reduce the need for educators to follow recommendations such as the ones presented in this paper. With regards to #5, we find it unlikely, at least in the near future, since current state-of-the-art LLMs have limitations (namely, hallucinations⁵), and still require human supervision [7]. The need for said supervision would potentially negate one of the biggest advantages of automated assessment: reduction of teacher workload.

6 Limitations

The recommendations in this paper are based on our multi-year experience across various courses but have not been statistically validated. As such, it is possible that they are somewhat biased. To mitigate this issue, we have reviewed our recommendations in relation to earlier studies on feedback, such as [15], and confirmed their alignment with prior recommendations.

7 Conclusion

This paper presented a set of recommendations for the definition of unit tests to automatically verify students' work and provide students with formative feedback which allows them to learn and/or act on their mistakes. We believe that following these recommendations has helped us help our students in their learning processes. They also helped us avoid certain mistakes which would have otherwise consumed our time or penalized students excessively.

References

- 1 Bruno Pereira Cipriano. Towards the Integration of Large Language Models in an Object-Oriented Programming Course. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 2*, ITiCSE 2024, pages 832–833, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3649405.3659473.
- 2 Bruno Pereira Cipriano, Bernardo Baltazar, Nuno Fachada, Athanasios Vourvopoulos, and Pedro Alves. Bridging the Gap between Project-Oriented and Exercise-Oriented Automatic Assessment Tools. *Computers*, 13(7), 2024. doi:10.3390/computers13070162.
- 3 Bruno Pereira Cipriano, Nuno Fachada, and Pedro Alves. Drop project: An automatic assessment tool for programming assignments. *SoftwareX*, 18:101079, 2022.
- 4 Stephen H Edwards and Krishnan Panamalai Murali. CodeWorkout: Short Programming Exercises with Built-in Data Collection. In *Proceedings of the 2017 ACM conference on innovation and technology in computer science education*, pages 188–193, 2017.
- 5 Emma Enström, Gunnar Kreitz, Fredrik Niemelä, Pehr Söderman, and Viggo Kann. Five Years with Kattis — Using an Automated Assessment System in Teaching. In *2011 Frontiers in education conference (FIE)*, pages T3J–1. IEEE, 2011.
- 6 Boni Garcia. *Mastering Software Testing with JUnit 5: Comprehensive guide to develop high quality Java applications*. Packt Publishing Ltd, 2017.
- 7 Sklyer Grandel, Douglas C Schmidt, and Kevin Leach. Applying Large Language Models to Enhance the Assessment of Parallel Functional Programming Assignments. In *Proceedings of the 2024 International Workshop on Large Language Models for Code*, pages 1–9, 2024.

⁵ Occasions in which LLMs produce outputs which are factually incorrect.

- 8 Sarah Heckman and Jason King. Developing Software Engineering Skills using Real Tools for Automated Grading. In *Proceedings of the 49th ACM technical symposium on computer science education*, pages 794–799, 2018.
- 9 Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, pages 86–93, 2010.
- 10 Nane Kratzke. Smart Like a Fox: How Clever Students Trick Dumb Automated Programming Assignment Assessment Systems. In *CSEDU (2)*, pages 15–26, 2019.
- 11 Sam Lau and Philip Guo. From “Ban it till we understand it” to “Resistance is futile”: How university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot. In *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1*, pages 106–121, 2023.
- 12 Yue Li, Tian Tan, and Jingling Xue. Understanding and Analyzing Java Reflection. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 28(2):1–50, 2019.
- 13 José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. Automated assessment in computer science education: A state-of-the-art review. *ACM Transactions on Computing Education (TOCE)*, 22(3):1–40, 2022.
- 14 Maciej Pankiewicz and Ryan S Baker. Large Language Models (GPT) for automating feedback on programming assignments. *arXiv preprint*, 2023. [arXiv:2307.00150](https://arxiv.org/abs/2307.00150).
- 15 Valerie J Shute. Focus on Formative Feedback. *Review of educational research*, 78(1):153–189, 2008.
- 16 Igor Škorić, Tihomir Orehovački, and Marina Ivašić-Kos. Exploring the Acceptance of the Web-based Coding Tool in an Introductory Programming course: A pilot Study. In *Human Interaction, Emerging Technologies and Future Applications III: Proceedings of the 3rd International Conference on Human Interaction and Emerging Technologies: Future Applications (IHET 2020), August 27-29, 2020, Paris, France*, pages 42–48. Springer, 2021.
- 17 Jagadeeswaran Thangaraj, Monica Ward, and Fiona O’Riordan. A Systematic Review of Formative Assessment to Support Students Learning Computer Programming. In *4th International Computer Programming Education Conference (ICPEC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

Adaptation of Automated Assessment System for Large Programming Courses

Marek Horváth  

Department of Computers and Informatics, FEI TU of Košice, Slovakia

Tomáš Kormaník  

Department of Computers and Informatics, FEI TU of Košice, Slovakia

Jaroslav Porubän  

Department of Computers and Informatics, FEI TU of Košice, Slovakia

Abstract

This paper presents a new automated assessment system tailored for programming courses, addressing the challenge of evaluating a large number of students in extensive courses at the Technical University of Košice. The primary issue with current systems is their inability to handle massive course loads while ensuring objective evaluation and timely feedback. Our proposed system enhances the scalability of the assessment process, allowing for the simultaneous handling of a greater volume of assignments. It is designed to provide regular and systematic feedback to students, supporting their continuous learning and improvement. To ensure the objectivity of evaluations, the system utilizes a variety of unit test suites, selecting them randomly in each assessment to discourage students from hardcoding solutions. This approach not only supports fair and precise assessments but also significantly reduces the administrative burden on educators, enabling them to meet a wide range of educational demands.

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

Keywords and phrases Automated Assessment, Informatics Education, Programming Feedback Systems, Continuous Integration in Education, Code Quality Analysis, Educational Technology, Computer Science Education

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.4

Funding This work was supported by project VEGA No. 1/0630/22 “Lowering Programmers’ Cognitive Load Using Context-Dependent Dialogs”.

1 Introduction

The expansion of programming courses across educational institutions poses significant challenges in managing the assessment of a broad spectrum of students. Traditional manual grading methods are increasingly proving to be overwhelming due to their demanding nature and potential for human error and bias.

In response, we have developed an automated assessment system tailored to manage the complexities of modern programming education, which was heavily influenced by previous findings from development of similar system [22]. This system is structured to provide continuous, regular feedback, similar to the iterative processes seen in professional software development environments where code is continuously tested and refined. This approach not only helps in the gradual enhancement of coding skills but also meets the evolving standards of industry practices.

Security and privacy in automated systems are critical, as they manage sensitive student data and are exposed to potential breaches, including threats from malicious code submissions. Therefore, our system implements essential security protocols and modules [2] to protect against these risks while maintaining high operational standards.



© Marek Horváth, Tomáš Kormaník, and Jaroslav Porubän;
licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 4; pp. 4:1–4:11

OpenAccess Series in Informatics



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

This paper outlines the updates to our automated assessment system, now supporting a broader spectrum of programming languages and course requirements – from basic C programming to more advanced topics such as cybersecurity in Python, along with object-oriented programming in Java. We discuss the system’s scalability and flexibility, which are crucial for adapting to new testing methods and managing an increasing number of student submissions efficiently.

By detailing the system’s architecture and functionality, we aim to provide insights into how to build an automated assessment system that enhances educational practices, making them more effective and responsive to the needs of both educators and students.

2 Related work

Automated assessment systems enable constant feedback for students, enhancing their learning beyond traditional methods [29][23]. These systems provide iterative feedback that helps students progressively refine their coding skills. However, the effectiveness of this feedback depends on student engagement [18].

System Arena employs parallel evaluation using Docker containers, providing detailed, data-driven insights and ensuring thorough assessments by stopping evaluations if errors are detected [19].

Code quality is crucial, preparing students for professional settings. Academic assessments often focus primarily on problem-solving rather than coding practices. Automated tools now offer context-sensitive feedback to bridge this gap [17].

Automatic evaluation advantages include speed, fairness, and objectivity, significantly improving reliability [23]. For example, JUnit tests provide specific feedback, aiding students in identifying and correcting errors [11].

Grading techniques in automated systems use formulas like:

$$\text{Final score} = (0.4 \times \text{LOC}) + (0.3 \times \text{WMC}) + (0.3 \times \text{DIT})$$

to ensure fair evaluations [1]. Threshold-based methods set clear performance standards for improvements. Automated assessments must be transparent and well-managed, ensuring a reliable framework that can adapt to the demands of programming education [9] while maintaining reasonable amount of safety [28].

Existing Tools and Technologies Used in Automated Assessment

Version control systems like Git, GitHub, and GitLab are crucial in modern programming education, facilitating functions essential for collaborative learning [27]. GitHub’s management capabilities notably enhance learning outcomes and classroom efficiency [29]. Additionally, our system integrates GitLab to manage and retrieve student code submissions effectively. This integration ensures seamless communication between the assessment system and GitLab repositories, allowing for automatic synchronization and real-time feedback on student assignments.

Assessment Tools for Immediate Feedback

Advanced tools facilitate prompt and precise feedback to students, which we extensively tested during education in our department:

- **2TSW and PAAA, PCQL and DANTE:** These tools utilize both static and dynamic analysis to evaluate student submissions in real time [24, 26, 6, 7].

- **JavaBrat and Web-CAT:** Specifically designed to automate grading processes by analyzing both the structure and the output of code [14, 8].
- **JThreadSpy and BOSS:** These tools not only provide execution traces but also enhance the efficiency of online assignment submissions [10, 15].

These tools provide a solid foundation for effective programming education by offering feedback that is both specific and timely, thus helping students improve their coding skills. They also streamline the grading process, reducing the workload on instructors and allowing them to focus more on teaching rather than administrative tasks. Each of these tools has different approach in terms of evaluating results and similarities, which creates differences in their accuracy, performance and reliability.

3 Assessment Techniques

This section discusses the main methods used in our assessment system, explaining how static and dynamic analysis techniques help evaluate programming assignments. These methods improve the accuracy of assessments and address the needs of various programming environments.

Analysis Techniques

In programming assessment, Static Analysis and Dynamic Analysis are essential methods that work together to effectively evaluate code submissions [27]. Figure 1 shows a flowchart of these analysis techniques.

Dynamic Analysis

Dynamic analysis evaluates the actual running of programs to check their functionality against different test cases. It includes:

- Black-box testing – The internal structure of the program is unknown to the tester; the focus is solely on the inputs and outputs [3, 25, 12, 20].
- Grey-box testing – This method examines the results of each function within the program and combines these findings to assess overall performance.

Dynamic analysis is user-friendly, allowing testers to evaluate programs by comparing the actual outcomes with expected results. It can test applications without needing source code access. However, it poses risks such as security vulnerabilities, including buffer overflows that can crash servers. Dynamic analysis also relies solely on feedback from output matches and cannot assess non-compiling programs or verify compliance with specific coding instructions [16].

Static Analysis

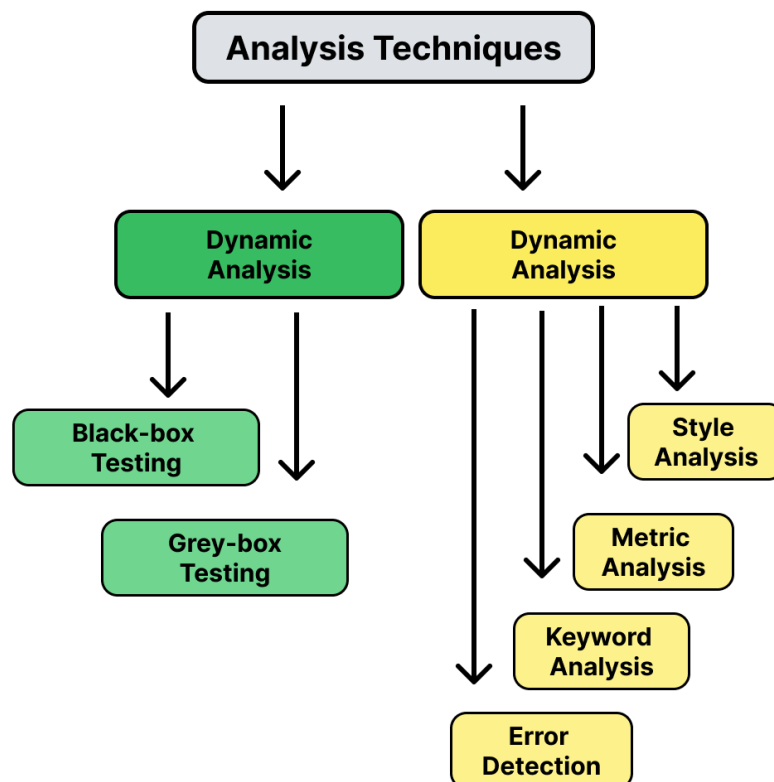
Static analysis looks at code without running it to find potential errors and check compliance with coding standards. It includes:

- Style analysis – Evaluates the readability of code using meaningful variable names, appropriate comments, and correct indentation [1].
- Error detection – Identifies errors that might not appear during compilation but could cause runtime issues, such as infinite loops or divisions by zero.
- Metric analysis – Assesses various aspects of the program to evaluate its complexity and reliability.

4:4 Adaptation of Automated Assessment System for Large Programming Courses

- Keyword analysis – Checks for the presence of specific keywords required by the assessment criteria.
- Structural analysis – Compares the structure of the assessed program against expected solutions to judge its correctness.

Static analysis is thorough, considering all potential execution paths to identify issues. It can evaluate code with compilation errors, enhancing the depth of assessment. However, its application to complex programs is challenging due to the variety of possible solutions, which can result in misjudgments, as is proven by previously conducted research [21].



■ **Figure 1** Flowchart illustrating analysis techniques in programming assessment.

4 Challenges and Enhancements in System Architecture for Programming Education

In the foundational course on the “Fundamentals of Programming and Algorithmization”, we currently serve approximately 1500 first-year students. Many of these students have no prior experience in programming, emphasizing the need for a system capable of providing regular feedback to facilitate learning. The course, which primarily teaches the basics of the C language, requires students to complete seven assignments throughout the semester. Given the volume of students and frequency of assignments, manually evaluating their work repeatedly is not feasible.

The complexity is further increased as these students are simultaneously enrolled in more advanced courses, such as “Object-Oriented Programming in Java”, which not only involve more complex assignments but also require significantly more system resources for evaluation.

Previously, our system relied on outdated and inflexible technologies such as SFTP for assignment submissions, which slowed down performance and hindered our ability to further improve this system. Multiple modules were affected by bad coding practices, and their versions were incompatible with currently desired technologies. Additionally, the absence of a user-friendly interface for instructors made it challenging to implement and manage new tests effectively.

To overcome these challenges, our plan includes upgrading our system to increase performance and extend its capabilities to additional programming courses such as “Data Structures”, “Operating Systems in C”, and “Cybersecurity in Python”. This enhancement will involve moving away from dependency on older systems and introducing a more interactive and manageable interface for teachers, enabling easier integration of new tests and better adaptation to evolving educational needs. This strategic upgrade is aimed at creating a more dynamic and responsive educational environment that supports both students and instructors more effectively.

In order to facilitate the development of new versions of our system, we often create either a closed group of student testers or we test results on all students while not interfering with evaluation processes in a specified subject. When the effectiveness of our system is proven and necessary issues and bugs are addressed, the system can be slowly pushed into production.

The architecture of our assessment system has been updated to better handle the increasing needs of educational environments today. This updated architecture includes high availability, automatic scaling, continuous integration and continuous deployment. Figure 2 shows how the different modules of the system interact. We’ve named each module with a callsign inspired by Roman history to make communication within our development team clearer. This distribution of systems into services is quite new since it allows us to distribute system loads across multiple physical machines. Each node in our physical machine cluster will be labeled with its specifications, and services will have their requirements defined in their manifests. This allows us to distribute tasks that require more computing power to higher-end devices or tasks that utilize CUDA to machines that are fitted with compatible GPUs.

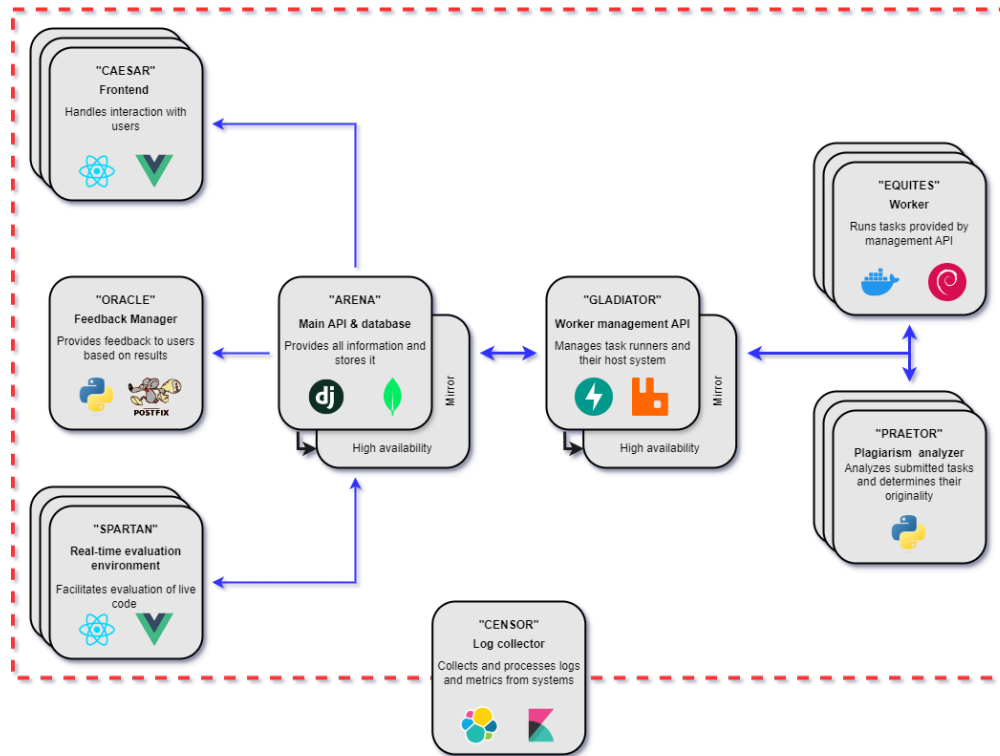
Caesar

The **Caesar** module serves as the web application and is the primary interface for both students and teachers. It displays test results and feedback designed to help students correct their solutions. The feedback includes points awarded for various test types such as structure, static code analysis checks, error handling, and edge cases. This detailed feedback illustrates the differences between expected outputs and student submissions, aiding in identifying specific mistakes (Figure 3). For teachers, Caesar also offers functionalities to create new assignments and manage scoring, ensuring the system can handle peak access times efficiently.

Oracle

The **Oracle** module employs Python to perform data analytics on the testing and evaluation processes. It gathers statistics such as peak usage times, average success rates, code issues in individual sub-tasks and most common programming malpractices across different parts of assignments [5]. This data helps reevaluate assignment complexity and provides anonymized advice to students on effective problem-solving strategies based on the performance of the most successful peers.

4:6 Adaptation of Automated Assessment System for Large Programming Courses



■ **Figure 2** A simplified diagram of the assessment evaluation system architecture.

```

STDOUT PRODUCED EXPECTED DIFF
Welcome to the game, Hangman!
I am thinking of a word that is 7 letters long.
-----
You have 8 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
- Please guess a letter: Good guess: _ e _ _ _ _ e
-----
- You have 8 guesses left.
- Available letters: abcdefghijklmnopqrstuvwxyz
- Please guess a letter: Oops! That letter is not in my word: _ e _ _ _ _ e
?
+ Please guess a letter: Oops! That letter is not in my word: _ _ _ _ _ _
?
-----
You have 7 guesses left.
- Available letters: bcd fghijklmnopqrstuvwxyz
?
+ Available letters: abcdefghijklmnopqrstuvwxyz
?
+ +
- Please guess a letter: Good guess: r e _ _ _ _ e
  
```

■ **Figure 3** Interface of Caesar showing detailed test feedback.

Spartan

Spartan is a complementary web application to Caesar but focuses on real-time evaluation. It provides a web-based code editor where students can write and submit code snippets for immediate feedback during class sessions. This module is particularly useful for simulating pressure-filled exam conditions and monitoring for unethical practices utilizing IP address detection and analysis of select, copy, and paste usage. We plan to further expand the features of this module in the future.

Praetor

Praetor is dedicated to plagiarism detection, employing tools like JPlag, Moss, and Sherlock, along with custom extensions for analyzing lines of code, regex patterns, and variable usage. This module is critical for maintaining academic integrity and is being developed to include a new web interface that allows teachers to actively monitor and review the plagiarism evaluation process. Previously developed experimental versions of this tool [13] have proven its effectiveness and usability while also providing valuable information for its development.

Additional Components

The **Arena** module acts as the central hub, using the Django framework and a Mongo database to facilitate communication between all modules via a RESTful API. **Gladiator** manages task distribution and system checks, utilizing FastAPI and RabbitMQ for efficient operation. **Equites**, the worker module, runs on Debian Linux and handles Docker-contained assessment processes, ensuring scalability and security. The **Censor** module, supported by Elastic and Kibana, tracks metrics and system health, aiding in predictive maintenance and machine learning applications.

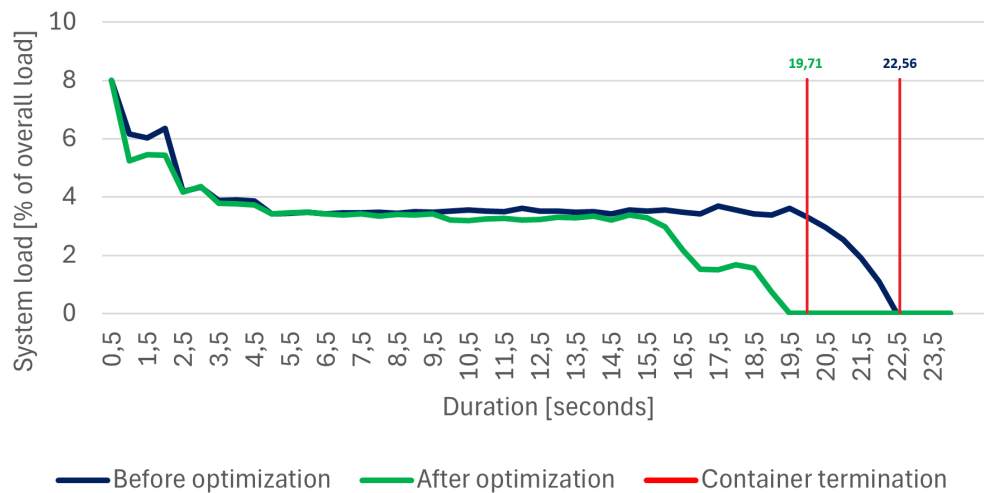
This updated system architecture is designed as a robust, scalable, and secure assessment environment that adapts to the needs of both students and educators, enhancing the educational experience through technology.

5 Enhancements in Assessment Methodologies

Our recent changes and improvements have significantly enhanced the accuracy of evaluations, accelerated the assessment processes and reduced the number of false positives in plagiarism detection. On the contrary, these enhancements have led to increased consumption of system resources.

The most notable improvement has been in the infrastructure of the system. Critical updates can now be tested, organized, and deployed to production in an average of two minutes (time until the pushed update is reflected on the side of users). Optimization of used base images, elimination of unnecessary nested virtualization and cleanup of host systems have reduced resource consumption by 21% (based on normalized average statistics collected over a 30 day period).

Code quality of our system has also improved, making the code more readable and easier to maintain. Implementation of multiple linting tools and coding standards has decreased the amount of LOC by approximately 14%. We have adopted a policy for capturing all possible data processed, which could be utilized to train language models or perform predictions. During testing, we noted a visible reduction in system load (Fig. 4), particularly when adjusting the compression algorithm at the beginning and end of tests. The reduction in load during test runs is modest but could be more apparent in larger projects that use asynchronous calls and multithreading more extensively than our test samples.



■ **Figure 4** Comparison of container metrics based on an average of 20 runs.

We initially experimented with artificially generated scenarios for assignments. After analysis of the results, we have chosen to move away from this approach as it often produced scenarios that were too generic or oddly phrased. Qualified staff now handles the design and evaluation of each test scenario. While this approach may not be groundbreaking, it preserves the human element, addresses ethical considerations, and avoids the pitfalls associated with fully automated systems. Previously created web services [4] were already tailored for such use cases and were easily adapted to suit our needs.

The current architecture readily accommodates the evaluation of commonly used programming languages. The standard method of testing via result value comparison proved insufficiently detailed since, in some courses, not only the output of programs is considered in evaluation. A multitude of courses with more specific topics, such as “Web Technologies” or “Intelligent Systems” (a machine learning-oriented subject), require evaluation of the usability, reliability, or performance of the provided solution.

Typically, we evaluate submissions in C, Java, Python, JavaScript, and Shell. Evaluating other languages is possible but requires additional time to design appropriate tests. We plan to standardize the evaluation of Assembly source code, particularly for courses specifically focused on this language. Besides standard testing, we have also integrated basic semantic analysis of source code and are looking to expand this analysis to more effectively determine if the code was written by a human. Evaluation of non-programming assignments is possible, however we are currently focusing mainly on mentioned programming languages.

6 Future Directions

As we continue to develop our automated assessment system for programming assignments, our primary objectives include enhancing system scalability, improving usability, and expanding the range of supported programming languages. We are also exploring the integration of machine learning techniques to refine the feedback provided to students. These advancements will include analyzing code quality, identifying common programming errors, and offering customized feedback to help students enhance their coding proficiency. Major changes are also considered, for example, a change in the programming language used for heavy computing loads; Rust is a strong candidate due to its excellent performance.

Another significant area of development is expanding the system's capability to assess assignments written in various programming languages. This will not only broaden the system's applicability but also ensure its adaptability to diverse educational requirements. To further support this goal, we will incorporate advanced static analysis tools. These tools will help provide clearer feedback on code quality, helping students grasp the details of efficient coding practices. By analyzing the structural and syntactic elements of code submissions, these tools will improve the learning experience by identifying specific areas for improvement.

Finally, we plan to test the system in a production environment to evaluate its stability and performance under real-world conditions. This will include thorough usability tests with educators to ensure that the system is not only robust but also user-friendly, allowing teachers to modify assessments according to their pedagogical needs.

These enhancements are aimed at creating a more effective and responsive educational tool that supports the continuous development of both students and educators in the field of programming.

7 Conclusion

In this paper, we have presented updates to an automated assessment system designed to support programming education effectively. The system has been adjusted to manage a broader array of courses, including basics of programming, object-oriented programming, and more complex areas like cybersecurity.

Our discussions highlighted the practical applications of static and dynamic analysis in improving the accuracy of student assessments and simplifying the grading process. We have also outlined how enhancements to the system's architecture help it handle increasing student numbers and a diversity of programming courses more efficiently.

As we continue to refine this system, we are focused on making incremental improvements that support the day-to-day needs of educators and students. By sharing our experiences and the specific updates we have made, we hope to provide useful insights that can assist others in developing or enhancing their own automated assessment systems [5]. This straightforward approach aims to ensure that the system not only meets current educational demands but is also prepared to adapt to future changes in the programming education landscape.


References

- 1 Burcu Alper, Selma Nazlioglu, and Hurevren Kilic. Ace-pe: An automated code evaluation software tool for programming education. In *2023 11th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–5, 2023. doi:10.1109/ISDFS58141.2023.10131776.
- 2 Anton Balaz, Norbert Adam, Emilia Pietrikova, and Branislav Mados. Modsecurity idmef module. In *2018 IEEE 16th World Symposium on Applied Machine Intelligence and Informatics (SAMI 2018): Dedicated to the Memory of Pioneer of Robotics Antal (Tony) K. Bejczy*, pages 43–48. IEEE, 2018. IEEE 16th World Symposium on Applied Machine Intelligence and Informatics (SAMI) Dedicated to the Memory of Pioneer of Robotics Antal (Tony) K. Bejczy, Kosice, SLOVAKIA, FEB 07-10, 2018.
- 3 S. Benford, E. Burke, E. Foxley, N. Gutteridge, and A. M. Zin. Experiences with the ceilidh system. In *Proceedings of the International Conference in Computer Based Learning in Science*, 1993.
- 4 M. Binas. Identifying web services for automatic assessments of programming assignments. In *12th IEEE International Conference on emerging E-learning Technologies and Applications (ICETA 2014)*, pages 45–50. IEEE, 2014. 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA), Slovakia, Dec 04-05, 2014.

- 5 Miroslav Biñas and Emília Pietriková. Impact of virtual assistant on programming novices' performance, behavior and motivation. *Acta Electrotechnica et Informatica*, 22(1):30–36, 2022. doi:10.2478/aei-2022-0005.
- 6 Skanda V. C, S. S. Prasad, and G. R. Dheemanth. Assessment of quality of program based on static analysis. In *IEEE Transactions on Learning Technologies*, 2019. 2019 IEEE. doi:10.1109/T4E.2019.00072.
- 7 P. Duch and T. Jowrki. Dante, automated assessment of programming assignments. In *IEEE Transactions on Learning Technologies*, 2018. 2018 IEEE.
- 8 S. H. Edwards and M. A. Perez-Quinones. Web-cat: automatically grading programming assignments. In *Proc. Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pages 328–328, 2008.
- 9 M. Fabijanic, G. Dambic, B. Skracic, and M. Kolaric. Automatic evaluation of student software solutions in a virtualized environment. In *2023 46th MIPRO ICT and Electronics Convention (MIPRO)*, pages 642–647, 2023. doi:10.23919/MIPRO57284.2023.10159927.
- 10 Xiang Fu, Kai Qian, Lixin Tao, and J. Liu. Apogee – automated project grading and instant feedback system for web based computing. In *SIGCSE'08, March 12–15, 2008, Portland, Oregon, USA*, 2008. Copyright 2008 ACM.
- 11 Sebastian Geiss, Tim Jentzsch, Nils Wild, and Christian Plewnia. Automatic programming assessment system for a computer science bridge course - an experience report. In *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, pages 527–536, 2022. doi:10.1109/APSEC57359.2022.00074.
- 12 J. B. Hext and J. W. Winings. An automatic grading scheme for simple programming exercises. *Commun. ACM*, 12(5):272–275, May 1969.
- 13 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, 2024. doi:10.1109/SAMI60510.2024.10432863.
- 14 S. Imam and V. Sarkar. Habanero-java library: a java 8 framework for multicore programming. In *Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java Platform Virtual Machines, Languages, and Tools*, pages 75–86, 2014. ACM DL.
- 15 Mike Joy, Nathan Griffiths, and Russell Boyatt. The boss online submission and assessment system. *J. Educ. Resour. Comput.*, 5(3):Article 2, September 2005.
- 16 Jan Juhar and Liberios Vokorokos. Separation of concerns and concern granularity in source code. In V Novitzka, S Korecko, and A Szakal, editors, *2015 IEEE 13th International Scientific Conference on Informatics*, pages 139–144. i'15; SSAKI KPI; KPI; Technicka University – Vkosiciach; ISVTS; IEEE, 2015. IEEE 13th International Scientific Conference on Informatics, Poprad, Slovakia, Nov 18-20, 2015.
- 17 Oscar Karnalim and Simon. Promoting code quality via automated feedback on student submissions. In *2021 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, 2021. doi:10.1109/FIE49875.2021.9637193.
- 18 Christian Kaufmann, Joao Pavão, and Harald Wahl. Is there a need for automated code review to be used in teaching? : From the perspective of students. In *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, 2022. doi:10.23919/CISTI54924.2022.9820030.
- 19 Matej Madeja, Miroslav Biñas, and Lukáš Prokein. Continuous analysis of assignment evaluation results from automated testing platform in iterative-style programming courses. In *2019 17th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 486–492, 2019. doi:10.1109/ICETA48886.2019.9040122.
- 20 Urs Von Matt. Cassandra: the automatic grading system. *SIGCUE Outlook*, 22(1):26–40, January 1994.

- 21 Emilia Pietrikova and Sergej Chodarev. Profile-driven source code exploration. In M Ganzha, L Maciaszek, and M Paprzycki, editors, *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems*, volume 5 of *ACSIS-Annals of Computer Science and Information Systems*, pages 929–934. IEEE Comp Soc; Polish Informat Proc Soc; IEEE Reg 8; IEEE Poland Sect Comp Soc Chapter; IEEE Poland Gdansk Sect Comp Soc Chapter; IEEE CIS Poland Sect Chapter; ACM Special Interest Grp Applied Comp; ACM Lodz Chapter; European Alliance Innovat; Polish Acad Sci, Comm Comp Sci; Polish Operat & Syst Res Soc; Eastern Cluster ICT Poland; Mazovia Cluster ICT, 2015. 3rd International Conference on Innovative Network Systems and Applications (iNetSApp) held in conjunction with Federated Conference on Computer Science and Information Systems (FedCSIS), Technical Univ Lodz, Lodz, Poland, SEP 13-16, 2015. doi:10.15439/2015F238.
- 22 Emilia Pietrikova, Jan Juhar, and Jana Stastna. Towards automated assessment in game-creative programming courses. In *2015 13th International Conference on emerging E-learning Technologies and Applications (ICETA)*, pages 307–312. The Amer Chamber of Commerce in the Slovak Republic; Elfa; TU; IEEE; Stu Fiit; Sanet; CTF atm; PPP; It Asociacia Slovenska; It News; EurActiv; PC revue; Education.sk; Infoware, 2015. 13th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, NOV 26-27, 2015.
- 23 Adam Pinter and Sandor Szenasi. Automatic analysis and evaluation of student source codes. In *2020 IEEE 20th International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 000161–000166, 2020. doi:10.1109/CINTI51262.2020.9305819.
- 24 G. Polito and M. Temperini. 2tsw: Automated assessment of computer programming assignments, in a gamified web-based system. In *IEEE Transactions on Learning Technologies*, 2019. 2019 IEEE.
- 25 Kenneth A. Reek. The TRY system – or how to avoid testing student programs. In *Proceedings of the twentieth SIGCSE technical symposium on Computer science education, SIGCSE '89*, pages 112–116, New York, NY, USA, 1989. ACM.
- 26 Shao Tianyi, Kuang Yulin, Huang Yihong, and Quan Yujuan. Paaa: An implementation of programming assignments automatic assessing system. In *ICDEL 2019, May 24–27, 2019, Shanghai, China*, 2019. 2019 Association for Computing Machinery.
- 27 Erika Baksane Varga and Antal Kristof Fekete. Applications for automatic c code assessment. In *2023 24th International Carpathian Control Conference (ICCC)*, pages 21–26, 2023. doi:10.1109/ICCC57093.2023.10178987.
- 28 Liberios Vokorokos, Anton Balaz, and Branislav Mados. Application security through sandbox virtualization. *Acta Polytechnica Hungarica*, 12(1):83–101, 2015.
- 29 Soundous Zougari, Mariam Tanana, and Abdelouahid Lyhyaoui. Towards an automatic assessment system in introductory programming courses. In *2016 International Conference on Electrical and Information Technologies (ICEIT)*, pages 496–499, 2016. doi:10.1109/EITech.2016.7519649.

Kumon-Inspired Approach to Teaching Programming Fundamentals

Ivone Amorim  

PORTIC – Porto Research, Technology & Innovation Center
Polytechnic of Porto (IPP), Portugal

Pedro Baltazar Vasconcelos  

LIACC & Department of Computer Science
Faculty of Sciences, University of Porto, Portugal

João Pedro Pedroso  

CMUP & Department of Computer Science
Faculty of Sciences, University of Porto, Portugal

Abstract

Integration of introductory programming into higher education programs beyond computer science has led to an increase in the failure and drop out rates of programming courses. In this context, programming instructors have explored new methodologies by introducing dynamic elements in the teaching-learning process, such as automatic code evaluation systems and gamification. Even though these methods have shown to be successful in improving students' engagement, they do not address all the existing problems and new strategies should be explored. In this work, we propose a new approach that combines the strengths of the Kumon method for personalized learning and progressive skill acquisition with the ability of online judge systems to provide automated assessment and immediate feedback. This approach has been used in teaching *Programming I* to students in several bachelor degrees and led to a 10% increase in exam approval rates compared to the baseline editions in which our Kumon-inspired methodology was not implemented.

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

Keywords and phrases Programming teaching, Programming education, Kumon method, Progressive learning, Online judge system

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.5

Funding *Ivone Amorim*: Partially supported by CMUP, which is financed by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the project with reference UIDB/00144/2020.

Pedro Baltazar Vasconcelos: Partially supported by: Base Funding – UIDB/00027/2020 of the Artificial Intelligence and Computer Science Laboratory – LIACC – funded by national funds through the FCT/MCTES (PIDDAC).

João Pedro Pedroso: Partially supported by CMUP, which is financed by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the project with reference UIDB/00144/2020.

1 Introduction

Over the past decade, information technology has suffered a remarkable growth, with its evolution and application having significant impacts on our society, namely in education. Today, for graduates to easily integrate the labour market they need not only to acquire specific skills, but also to develop the agility to rapidly acquire new knowledge and address new challenges with creativity and critical thinking [12].

In the past, programming skills were associated mostly with computer science and engineering fields. However, today, Programming is seen as a fundamental area for the development of characteristics and skills such as creativity, problem-solving, persistence,



© Ivone Amorim, Pedro Baltazar Vasconcelos, and João Pedro Pedroso;
licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 5; pp. 5:1–5:13

OpenAccess Series in Informatics



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

logical and critical thinking [17, 7]. Additionally, professionals from different areas can benefit from the ability to write codes for different applications. For instance, in biology or chemistry, analysing large datasets is often necessary, and this task can be facilitated by programming skills.

As a result, many diverse fields of knowledge now incorporate programming courses into their curricula [7]. Therefore, several students with different backgrounds and characteristics are involved in introductory programming [12]. For example, an instructor may have students in the same class who have no prior knowledge about programming, as well as students who have already learned different programming languages. This leads to difficulties in aligning classes with the students' interests and motivations. Other challenges include the lack of resources, such as computational labs, for practical classes. On the students' side, the already well-known difficulties they face are exacerbated by the different backgrounds students may have, which may lead them not to perceive programming as an important skill or competence. According to Gomes and Mendes [8], some of the main difficulties students may encounter while learning programming include inadequate teaching approaches, which sometimes prioritize theoretical concepts over improving students' problem-solving abilities; a lack of problem-solving skills necessary to understand the logic behind programming; the use of inappropriate self-learning methods to enhance academic programming success; and psychological factors.

Consequently, significant failure and dropout rates, as well as a lack of motivation, have been observed in programming courses [4, 16]. This has raised concerns among programming instructors, leading them to develop and utilize novel teaching and learning approaches to enhance students' learning experiences and ultimately increase success rates. The most used strategies include the introduction of dynamic elements in learning strategies such as automatic code evaluation systems [18, 11], namely online judge systems, gamification [15, 14], and systems that use visual representations of algorithms, such as Python Tutor¹. These strategies have shown to reduce some of the challenges instructors and students face when teaching and learning programming, respectively. However, they still do not offer a complete solution to address all the existing challenges.

In this work, we explore a new approach that combines the strengths of the Kumon method [21] for personalized learning and progressive skill acquisition with the ability of online judge systems to provide automated assessment and immediate feedback.

The rest of this paper is organized as follows. Section 2 provides an explanation on some key concepts regarding online judge systems and the Kumon method, and discusses some related work. Section 3 presents the course to which our methodology was designed, details the online judge system used and explains our Kumon-inspired approach. Section 4 presents the case study methodology employed to assess the effectiveness of our approach. Section 5 presents and discusses the results obtained. Finally, Section 6 presents the conclusions of our work and proposes future directions to further explore and validate this approach.

2 Background and Related Work

2.1 Online judge systems

The assessment of students by instructors can be done for several reasons: to provide feedback to students on their learning path, to assess previous knowledge in a specific subject, to evaluate teaching methodologies, to identify at-risk students, among other reasons [5].

¹ <https://pythontutor.com/>

However, due to the increasing number of students with diverse backgrounds that instructors have to assess and support in introductory programming courses, a wide range of assessments is required. This demand may lead to instructors being overloaded with work and reduces the time they have for other tasks. Online judges are automated assessment tools [2] designed for the reliable evaluation of algorithm source code submitted by users. These tools may be a crucial help for instructors in these assessment processes. Online judges are now popular in various applications, including in programming contests and education. The use of online judges in the classical educational system has advantages for both instructors and students. For instructors, they allow the assessment of students' assignments automatically, increasing assessment accuracy and significantly reducing the time needed for evaluation. Consequently, many more exercises can be prepared and assigned to students [24]. On the other hand, students receive almost instant feedback on their answers, which motivates them to perform more exercises, promotes active learning by helping them to easily understand the main difficulties they should address, and fosters independence.

There are several factors that may limit the accessibility of existing online judges for use in introductory programming courses. According to Asuncion et al. [2], one of the problems is that most existing online judges were not designed for introductory programming classes. Therefore, the exercises they provide may not align with the curricular units of these courses. For example, the well-known online judge Codeforces² was primarily designed for programming contests. Consequently, some of their problems considered “easy” may not be suitable for introductory programming courses, as they may require concepts typically covered in a data structures and algorithms course rather than in introductory programming classes. Additionally, the same authors claim that creating and uploading new exercises for many online judges is not easy, if possible at all.

Fortunately, several reviews of existing online judge systems and their application in education have been conducted over the years, helping instructors in finding the right tool for their purpose [1, 9]. Recently, Wasik et al. [24] provided a comprehensive review of the state of the art for these systems. They classified them according to their principal objectives into categories which include “Development platforms.” This refers to systems that are often provided as open source projects or binary archives that can be downloaded and installed locally, providing full administrative privileges to the user. They can be used to host a programming competition or a course using the user's own infrastructure. Moreover, most can be adapted to user needs and integrated with external services. Well known platforms to prepare and perform programming contests are DOMjudge³, Mooshak⁴, and SIO2⁵. There are other development platforms which are dedicated to support the educational process, such as CloudCoder [19], BOSS [10], and Web-CAT [6]. Unfortunately, some of these latter systems are not open-source, and others have been updated for the last time more than 15 years ago. In this work, we use an online judge system named Codex that falls into the category of “Development platform”. This system was developed at the Faculty of Sciences of the University of Porto, and its source-code is publicly available through the GitHub platform⁶. Further details on this system are presented in Section 3.2.

² <https://codeforces.com/>

³ <https://www.domjudge.org/>

⁴ <https://mooshak.dcc.fc.up.pt/>

⁵ <https://github.com/sio2project>

⁶ <https://github.com/pbv/codex>

2.2 Kumon-inspired methods and potential benefits

The Kumon Method is a pedagogical method that emerged in 1954 in Osaka, Japan, by the hand of Toru Kumon, a Mathematics teacher of a high school, who developed this to help his son’s mathematical education. This method has been extended to other subjects, such as “Reading” and “English as a foreign language”. Initially popularized in Japan and subsequently in the United States of America [21], the method has gained global recognition and is now implemented in learning centres all around the world, mainly for teaching Mathematics and English⁷.

The main objective of the Kumon Method is to maximize the learning potential of each individual. To achieve this, a series of habits and abilities have to be acquired, namely:

- Self-learning; the students learn how to learn by themselves, without depending on another person.
- Study habits; students are encouraged to divide the study effort into smaller, more regular steps.
- To foster concentration: if students are not able to focus on a specific task, it will be difficult for them to learn effectively.
- Self-confidence, that allows students to face any educational challenge.
- Motivation to learn, to perceive learning as something enjoyable that will help them to grow as persons.

Several research studies have assessed the effectiveness of this method, primarily focusing on improving mathematical skills [13, 22]. However, there is no known study on the application of this method in teaching programming. Although some works propose similar strategies, such as intensive training [3], none have directly applied the principles of the Kumon method to foster the development of the aforementioned habits and abilities in programming students.

3 Progressive Method for Teaching Programming

In this section, we begin by introducing the course for which our Kumon-inspired approach was designed. Following that, we provide an overview of Codex, the online judge system utilized in our methodology. Lastly, we present our novel progressive method for teaching programming fundamentals.

3.1 The Course – Programming I

Programming I is a first year course for several bachelor degrees at the Faculty of Sciences from the University of Porto. It is mandatory for the bachelor degrees of Agricultural Engineering, Engineering Physics, Geospatial Engineering, Bioinformatics, and Physics. Besides, it is an optional course for the bachelor degrees of Biology, Geology, and Chemistry.

The main objectives of this course are: Get acquainted with personal computers in the GNU/Linux operating system and their usage; Learn how to write computer programs using Python and execute them in a terminal; Acquire competence in the implementation of simple algorithms; Acquire good code structuring and programming style; Learn some basic data structures and algorithms; Get acquainted with program debugging and testing.

The main learning outcomes and competences expected from students who successfully perform this course are: understanding the role of programming for solving problems in their degree, acquaintance with the basic components of a recent programming language, ability

⁷ <https://kao.kumonglobal.com/our-global-network/>

to write programs that allow accomplishing useful goals, and confidence in the usage of the Python language and its standard library. The students have 2 hours per week of face-to-face theoretical classes and 2 hours per week of laboratory classes, for 14 weeks, resulting in a total of 56 contact hours.

3.2 The online judge system: Codex

Codex⁸ is a web system for setting up programming exercises with automatic assessment, which is intended for learning environments, similar to a judge system. It was developed and is currently being used at the Faculty of Sciences of the University of Porto for introductory courses on Python, Haskell and C programming [23]. Its main features are:

Simple exercise authoring. Exercise descriptions are written in a human-readable Markdown format that can easily be copied, mailed, kept in a version repository, compared for changes, etc.

Allows assessing program units. Exercises can assess single functions, classes or methods as well as complete programs.

Provides automatic feedback. Rather than report just an accept/reject result, Codex can report the failed examples to students.

Multiple types of exercises. Besides code testing, Codex also allows multiple-choice and fill-in questionnaires.

In Codex, student's submissions are classified as follows:

- *CompileError*: rejected attempt due to a compile-time error or warning; for an interpreted language such as Python, this is typically a syntax error;
- *RuntimeError*, *TimeLimitExceeded* or *MemoryLimitExceeded*: the execution was aborted due to a runtime error or resource exhaustion;
- *WrongAnswer*: testing failed in at least one case;
- *Accepted*: all tests passed.

For the *Programming I* course we used input-output and unit testing with the *Doctest* Python library [20]. The *Doctest* files were generated semi-automatically and comprise a large number of test cases (typically, about one thousand); this not only allows testing the student's attempts more thoroughly, but also prevents over-fitting solutions to a small number test cases.

For beginner exercises that are typically not computationally intensive, the turn-around for student feedback is quick (typically 2-4 seconds).

Another important advantage of Codex is that it allows instructors to get real-time insights regarding the performance of students in laboratory assignments. Therefore, it helps to identify students who are struggling early on, allowing for timely intervention and support.

3.3 Our Kumon-based approach

Our approach for teaching and assessing programming fundamentals in Python is inspired in the Kumon method that emphasizes self-learning through repetitive practice of progressively challenging exercises. Instructors have developed three different types of assignments that allow students to progressively develop their Python skills and which are fully aligned with the course syllabus. The way these exercises are provided to students and the methodology

⁸ <https://github.com/pbv/codex>

5:6 Kumon-Inspired Approach to Teaching Programming Fundamentals

employed for course evaluation were designed to encourage self-learning, build the habit of studying, increase self-confidence, and ultimately motivate students to learn. These are all key objectives of the Kumon method. Additionally, our approach allows students to have some control over their own progress through individualized learning, and immediate feedback.

The three types of assignments developed are the following:

- **Theoretical assignments:** Following every theoretical class, students were given theoretical assignments in the form of quizzes to encourage them to actively participate in those classes and solidify their understanding of the presented concepts. These assignments were intended to be completed outside the classroom within a limited time frame. Students were allowed to consult any resources to help them in providing correct answers.
- **Worksheet assignment:** These are weekly worksheet assignments with problems provided to students approximately a week before the corresponding laboratory class. Students are encouraged to attempt to solve these problems outside the classroom, and any doubts can be clarified with instructors during laboratory sessions. Its main goal is to encourage frequent study and self-learning among students.
- **Laboratory assignments:** These assignments are sets of coding problems made available to students through the Codex system and are fundamental elements of our approach. A total of 10 assignments, each corresponding to a different level of difficulty, are provided over a 14-week period. Students can only access these assignments in the classroom and must successfully complete the current level to unlock the next one. The success in one level means that the student was able to submit a correct answer to all questions in one of the three possible attempts. If a student is not able to submit a correct answer in any of the three attempts, he or she has to attempt another problem set of the same level in the following label class. Hence, in a given lab class, different students may be attempting problems at distinct levels according to their own progress. This strategy was designed to ensure that students only proceed a new level when the knowledge from the previous one has been assimilated. Additionally, students must complete at least the first five levels to qualify for the final exam. The number of questions in these weekly assignments varies between 6 and 8 depending on the topics assessed.

The assessment strategy for determining students' final grades was also designed to align with the principles of Kumon's method. This involved considering grades obtained from both the Theoretical and Laboratory assignments as key components contributing to the final grade. Additionally, a final exam covering all the course topics is conducted in a codex-based environment under the same conditions of laboratory assignments: maximum of three attempts per exercise and no access to external resources.

The final grade is determined using the following formula:

$$\text{Final Grade} = 0.2t + 0.2l + 0.6e, \quad (1)$$

where t represents the grade obtained from theoretical assignments, l denotes the grade from laboratory assignments, and e denotes the grade from the final exam.

4 Case Study Methodology

To evaluate the effectiveness of our novel methodology, we compared the learning results of students enrolled in the *Programming I* course, as outlined in Section 3.1, over a period of four school years. Our Kumon-inspired approach was implemented in two of these years, while in the other two years, we did not utilize this method. Below, we outline the teaching methodology employed in each school year considered in our study.

- **2019:** In this school year, the students's had three different types of assignments: weakly laboratory assignments (different from the ones described in Section 3.3), intermediate test and final exam. The final grade was computed using

$$\text{Final Grade} = 0.2l + 0.2i + 0.6e,$$

where l , i , e denote the grade from laboratory, intermediate test, and final exam assignments, respectively. Students were considered eligible to take the final exam only if at least one of the following conditions was met: having a non-null grade in the intermediate test or correctly answering at least 50% of the laboratory assignments. For a student to be approved, they needed to achieve a grade of at least 40% on the final exam. The laboratory assignments were provided through Codex, with each assignment consisting of only one question and without including levels and progressive learning. The intermediate test and the final exam were also conducted through Codex.

- **2020:** This was the first school year in which our Kumon-inspired approach was implemented, and the final grade was determined using Equation (1). The students were eligible to go to the final exam only if they had answered correctly to at least 50% of the laboratory assignments. For a student to be approved, it had to have a grade on the final exam not less than 45%. It is important to note that in this year our Kumon-inspired approach was still being adjusted.
- **2021:** In this year, our Kumon-inspired approach was also utilized, albeit with more stringent conditions for a student to qualify for the final exam. More specifically, a student was eligible for the final examination only if they had successfully completed at least 50% of both theoretical and laboratory assignments. To be approved, a student needed to achieve a grade of at least 50% on the final exam.
- **2023:** In the school year 2023, technical issues resulting from renovation works in the Computer Science building caused difficulties with the network infrastructure. Despite being provided with worksheets and laboratory assignments through Codex, as described in Section 3.3, the students did not have the appropriate conditions to ensure fair assessment process using our Kumon-inspired approach. As a result, we decided to assess students only through a final examination, which was conducted in Codex. For a student to be approved, they had to have an exam grade of at least 50%.

The school years 2019 and 2023 serve as the baseline against which we compare the effectiveness of our Kumon-inspired approach in the other years (2020 and 2021). In the school year 2022, the team of professors in charge of teaching *Programming I* was different, as well as the methodological approach applied. Therefore, this year was not considered in our study.

It is important to add that in the 2021 and 2023 school years, students were allowed to seek help from their more advanced peers in laboratory classes. However, students assisting their colleagues were prohibited from writing code for them. In 2019 and 2020, students were not incentivized to seek help from other students.

A total of 937 students from different bachelor's degrees were part of our study, with most of them coming from Physics, Engineering Physics, and Geospatial Engineering.

5 Results and Discussion

To analyse the potential impact of our Kumon-inspired approach on teaching and learning Programming fundamentals, we began by calculating statistics on the number of approved students. More specifically, we computed the percentage of students who obtained approval

5:8 Kumon-Inspired Approach to Teaching Programming Fundamentals

among both those enrolled and those effectively assessed. Table 1 provides an overview of the number of students enrolled in the course, the number of students assessed, and the grade statistics by school year.

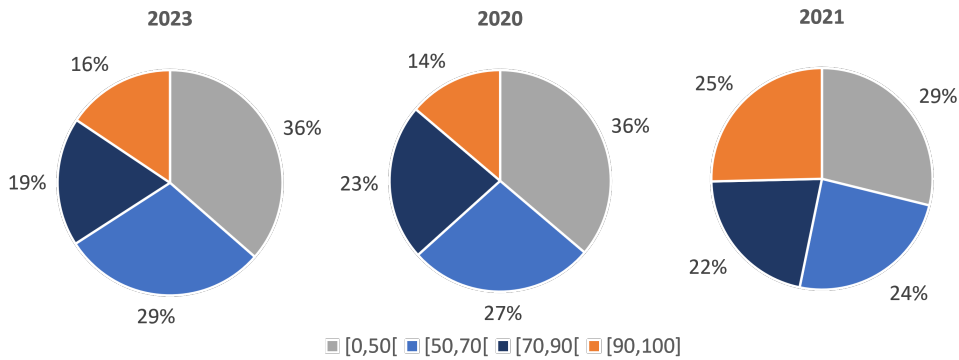
■ **Table 1** Overview of student enrolment, assessment, and grades statistics by school year.

	Baseline		Kumon approach	
	2019	2023	2020	2021
N° of enrolled students	223	207	259	248
N° of students assessed	159	173	210	201
N° of approved students	92	110	134	143
Grades mean	62.52	51.45	52.95	59.95
% approved/enrolled	41.26	53.14	51.74	57.67
% approved/assessed	57.86	63.58	63.81	71.14

In Table 1, we can observe that 2021 was the school year with the highest percentage of approved students among those assessed, with an approval rate of around 71%. Moreover, this year was the one with the highest percentage of approved students among the enrolled ones, reaching approximately 58%. When comparing the results of the school year 2021 with the baseline years, it becomes clear that the percentage of approved students is significantly higher when the Kumon-inspired approach is applied. Furthermore, when comparing this year with 2020, a much higher approval rate is observed. These results suggest a positive correlation between the grades obtained and the level of engagement required from students. For instance, in 2021, students had to complete at least half of both their theoretical and laboratory assignments, which led to better final grades compared to 2020, where they only needed to complete half of their laboratory assignments. It is important noting that these results may also have been affected by the fact that in 2021, struggling students were allowed to get assistance from more advanced peers, as detailed in Section 4. Interestingly, comparing the results between 2020 and 2023, we notice a slight overall improvement in the latter year. This suggests that allowing students to request help from their peers has a positive impact on their learning achievements. Another interesting point to observe in Table 1 is that years in which the Kumon-inspired approach was (even partially) applied had a higher rate of assessed students among those enrolled.

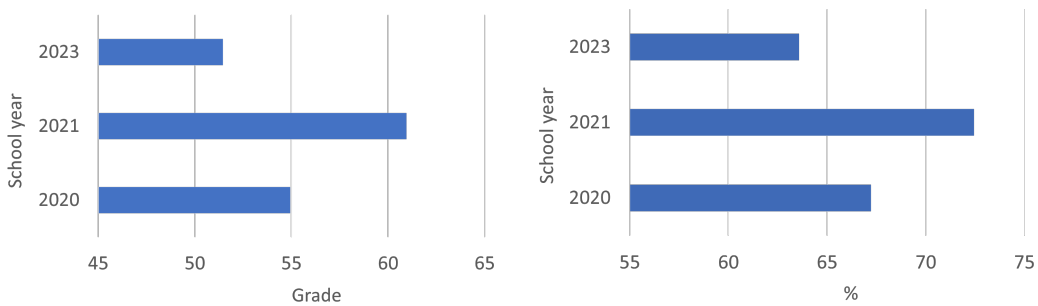
To evaluate the potential impact of our Kumon-inspired methodology on students' performance, we conducted an additional analysis. We divided the grades into four intervals: $[0, 50[$ (indicating insufficient performance), $[50, 70[$ (indicating satisfactory performance), $[70, 90[$ (indicating good performance), and $[90, 100]$ (indicating excellent performance), and calculated the percentage of students falling within each interval. Figure 1 displays the results obtained by school year. The grading system in 2019 was different from the rest of the years, so we could not include it in this analysis.

As can be observed from Figure 1, the 2021 school year was the one that has the highest percentage of students with an excellent performance. More specifically, 25% of students scored within the interval $[90, 100]$, compared to 14% in 2020 and 16% in 2023. In terms of students with insufficient performance, 2021 recorded the lowest percentage (29%), further suggesting a significant improvement in overall student performance due to our approach. When comparing 2020 and 2023, both school years showed the same percentage of students with insufficient performance. However, in 2023, a higher percentage of students demonstrated performance above the level of sufficiency. This strengthens the hypothesis that the peer assistance provided during lab classes positively impacts students' performance.



■ **Figure 1** Percentage of students with final grades within each grade interval and grouped by school year.

Finally, to assess the impact of our progressive learning methodology on the knowledge acquired by students during the course, we analysed their exam grades. Since the exam covers all taught subjects and is conducted at the end of the course, the grades obtained provide a good indicator of the knowledge acquired and skills developed. As such, we computed the average exam grades among students with a lab level of 5 or higher during the 2020 and 2021 school years. We also calculated the percentage of those students that scored 50% or higher on their exams. The results obtained are shown in Figure 2. For the school year 2023, the exam grade corresponds to the course grade, since no theoretical or laboratory assignments were requested.



(a) Exam grade average.

(b) Percentage of students with an exam grade ≥ 50 .

■ **Figure 2** Overview of exam grade statistics for students which achieved level of 5 or higher in the 2020 and 2021 school years, and for all assessed students in 2023.

The statistics presented in Figure 2 demonstrate the impact of implementing the Kumon-inspired progressive learning approach on exam grades, particularly among students with a lab level of 5 or higher during the 2020 and 2021 school years. Those who had access to this approach showed notably higher exam grades compared to those who did not. Moreover, among these students, those who were requested to complete at least 50% of the theoretical assignments saw even more significant improvements, as evidenced by the data from the 2021 school year. More specifically, in 2021, students with a lab level of 5 or higher achieved an average exam grade of 61. Approximately 75% of these students attained an exam grade of 50% or higher, indicating a high success rate for those who followed this approach. In

contrast, when the progressive learning approach was not applied, students achieved an average exam grade of 51%, with 64% of them reaching an exam grade of 50% or higher. This translates to an increase of more than 10% in exam approval rates between the school year when the Kumon-inspired methodology was employed and the theoretical assignments were required, and the year when students did not have access to the progressive learning method. These findings strongly suggest that our new methodology enhances the teaching-learning process and improves students performance in final examinations. When comparing the exam grades between the 2020 and 2023 school years, it becomes evident how the implementation of progressive learning positively influenced student performance in final examinations. Notably, in 2020, students were required to complete the first five laboratory assignments successfully, whereas no such prerequisite was imposed in 2023. This observation substantiates the positive impact of the progressive learning approach on students' achievement.

5.1 Threats to validity

Testing and validating new teaching methodologies in a school environment has its own challenges, due mainly to ethical considerations, since all students should be given the same opportunities and conditions to learn but also because of all external factors which can not be controlled and may impact the students' success (e.g. socio-economic context).

Considering this, we have identified the following main factors that may have had an impact in the results obtained:

- **COVID-19 pandemic:** During the school years in which the Kumon-inspired approach was applied (2020 and 2021), some lockdowns were imposed in Portugal. However, Programming I is a course taught in the first semester, which was not directly affected by those lockdowns. Moreover, in those schools years, the theoretical classes of Programming I were delivered online synchronously, and the laboratory classes were conducted face-to-face. This allowed to fully implement our Kumon-based approach despite the imposed restrictions. Therefore, the results obtained are considered robust, and we believe they were not significantly affected by external pandemic-related factors.
- **Socio-economic context:** It is well known that the socio-economic background can significantly impact students' success. In the specific case of our Kumon-inspired approach, students are required to actively participate in their learning process by practising laboratory assignments outside of classrooms and completing theoretical assignments outside of class as well. Students from different socio-economic backgrounds may have varying levels of access to resources such as computers and internet connectivity, which can negatively impact their learning outcomes. However, at our faculty, students have 24-hour access to computer laboratories and can also work anywhere within the faculty premises using their personal computers with free internet connection. Although this does not guarantee equal conditions for all students, it does help to minimize disparities. Additionally, our study benefited from the participation of a total of 937 students, which enhances its validity and reliability.
- **Variability in exams difficulty levels:** The difference in the levels of difficulty among exams across different school years may have influenced the outcomes observed in our study. Specifically, we acknowledge that exams administered after 2020 were generally more challenging compared to those in 2019. However, this reinforces the advantages of our Kumon-inspired approach, as students were able to achieve better overall results despite having more difficult exams.

6 Conclusion

This study demonstrates the potential of combining the strengths of the Kumon method with the automatic assessment abilities of online judge systems. Our approach encompasses three different types of assignments aligned with the principles of Kumon. Theoretical assignments encourage active participation in theoretical classes and solidify newly acquired knowledge. Worksheet assignments promote frequent study and foster self-learning skills. Finally, laboratory assignments provide progressive learning opportunities to increase self-confidence. An online judge system facilitates the implementation of our strategy, providing automated assessment and immediate feedback. Together, these strategies enable the achievement of most of the objectives of the Kumon method, fostering a successful teaching-learning process.

In addition to outlining the methodology, results from school years with our approach implemented showed significant improvements in student performance and engagement. More specifically, the 2021 school year stands out as the most successful, with the highest percentage of approved students among the ones assessed and enrolled. This success can be attributed to our progressive approach and the requirements set forth by our methodology, including the completion of theoretical and laboratory assignments, which encouraged active participation and self-learning. Furthermore, analysis of students' performances revealed a notable increase in the percentage of students achieving excellent grades in the 2021 school year compared to other years. Conversely, the percentage of students with insufficient performance decreased significantly, indicating an overall improvement in student outcomes. Moreover, our analysis of exam grades suggests a significant impact of our progressive learning methodology on students' performance. Students who participated in the Kumon-inspired approach exhibited higher exam grades.

In the future, it would be interesting to investigate students' perceptions of this new methodology, for example, through interviews. Students who have experienced more traditional programming approaches would provide valuable insights in this context. Their feedback may help to further understand the effectiveness of our methodology and guide the design of new approaches. It would also be important to assess the developed skills, habits, and abilities, particularly in the areas of self-learning, self-regulation, study habits, self-confidence, and motivation levels. Additionally, studying the impact of peer-assisted learning could help to understand collaborative learning dynamics and their influence on academic achievement. Conducting such research could help to refine educational practices and promote effective learning environments for programming fundamentals courses.


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 Aldrich Ellis Asuncion, Brian Christopher Guadalupe, and Gerard Francis Ortega. The abc workbook: Adapting online judge systems for introductory programming classes. In *Proceedings of the 30th International Conference on Computers in Education*, volume 2, pages 395–400. IEEE, 2022. URL: https://icce2022.apsce.net/uploads/P2_W05_052.pdf.
- 3 Yoram Bosse, David Redmiles, and Marco A. Gerosa. Pedagogical content for professors of introductory programming courses. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, pages 429–435, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3304221.3319776.
- 4 Chin-Soon Cheah. Factors contributing to the difficulties in teaching and learning of computer programming: A literature review. *Contemporary Educational Technology*, 2020.

- 5 Rodrigo Duran, Jan-Mikael Rybicki, Juha Sorva, and Arto Hellas. Exploring the value of student self-evaluation in introductory programming. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, ICER '19, pages 121–130, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3291279.3339407.
- 6 Stephen H. Edwards and Manuel A. Perez-Quinones. Web-cat: automatically grading programming assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '08, page 328, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1384271.1384371.
- 7 José Figueiredo and Francisco José García-Peñalvo. Strategies to increase success in learning programming. *2022 International Symposium on Computers in Education (SIIE)*, pages 1–6, 2022. URL: <https://api.semanticscholar.org/CorpusID:254911096>.
- 8 Anabela Gomes and Antonio Mendes. Learning to program - difficulties and solutions. In *Proceedings of the International Conference on Engineering Education – ICEE 2007*, pages 283–287, January 2007.
- 9 Petri Ihanntola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 86–93, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1930464.1930480.
- 10 Mike Joy, Nathan Griffiths, and Russell Boyatt. The boss online submission and assessment system. *J. Educ. Resour. Comput.*, 5(3):2–es, September 2005. doi:10.1145/1163405.1163407.
- 11 Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. Towards a systematic review of automated feedback generation for programming exercises. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, pages 41–46, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2899415.2899422.
- 12 Alain Kabo Mbiada, Bassey Isong, Francis Lugayizi, and Adnan Abu-Mahfouz. Introductory computer programming teaching and learning approaches: Review. In *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pages 1–8, 2022. doi:10.1109/ICECET55527.2022.9873427.
- 13 L. Orcos, R. M. Hernández-Carrera, M. J. Espigares, and Á. Alberto Magreñán. The kumon method: Its importance in the improvement on the teaching and learning of mathematics from the first levels of early childhood and primary education. *Mathematics*, 7(1), 2019. doi:10.3390/math7010109.
- 14 José Carlos Paiva, José Paulo Leal, and Ricardo Queirós. Authoring game-based programming challenges to improve students' motivation. In Michael E. Auer and Thrasylvoulos Tsiatsos, editors, *The Challenges of the Digital Transformation in Education*, pages 602–613, Cham, 2020. Springer International Publishing.
- 15 Mário Pinto and Teresa Terroso. Learning Computer Programming: A Gamified Approach. In Alberto Simões and João Carlos Silva, editors, *Third International Computer Programming Education Conference (ICPEC 2022)*, volume 102 of *Open Access Series in Informatics (OASIS)*, pages 11:1–11:8, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIS. ICPEC.2022.11.
- 16 Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.*, 18(1), October 2017. doi:10.1145/3077618.
- 17 Atajan Rovshenov and Firat Sarsar. Research trends in programming education: A systematic review of the articles published between 2012-2020. *Journal of Educational Technology and Online Learning*, 6(1):48–81, 2023. doi:10.31681/jeto1.1201010.
- 18 Ján Skalka, Martin Drlík, and Juraj Obonya. Automated assessment in learning and teaching programming languages using virtual learning environment. In *2019 IEEE Global Engineering Education Conference (EDUCON)*, pages 689–697, 2019. doi:10.1109/EDUCON.2019.8725127.

- 19 Jaime Spacco, Paul Denny, Brad Richards, David Babcock, David Hovemeyer, James Moscola, and Robert Duvall. Analyzing student work patterns using programming exercise data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE '15*, pages 18–23, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2676723.2677297.
- 20 Python standard library. Doctest library documentation. URL: <https://docs.python.org/3/library/doctest.html>.
- 21 Nancy Ukai. The kumon approach to teaching and learning. *Journal of Japanese Studies*, 20:87, 1994. URL: <https://api.semanticscholar.org/CorpusID:150129343>.
- 22 Usmedi, Amelia Agita, and Ergusni. The effect of application kumon learning method in learning mathematics of ability troubleshooting mathematics of students. *Journal of Physics: Conference Series*, 1429(1):012005, 2020. doi:10.1088/1742-6596/1429/1/012005.
- 23 Pedro Vasconcelos and Rita P. Ribeiro. Using Property-Based Testing to Generate Feedback for C Programming Exercises. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *Open Access Series in Informatics (OASICs)*, pages 28:1–28:10, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASICs.ICPEC.2020.28.
- 24 Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. A survey on online judge systems and their applications. *ACM Comput. Surv.*, 51(1), January 2018. doi:10.1145/3143560.

An Experience with Adaptive Formative Assessment for Motivating Novices in Introductory Programming Learning

Jagadeeswaran Thangaraj ✉ 

School of Computing, Dublin City University, Ireland

Monica Ward ✉ 

School of Computing, Dublin City University, Ireland

Fiona O’Riordan ✉ 

CCT College, Dublin, Ireland

Abstract

This study presents empirical research that uses adaptive formative assessment framework in addition to traditional lectures to motivate novice students in an introductory programming course. The primary goal of this work is to provide guidance for the creation of adaptive formative assessments in Python programming language to inspire novice students. The experiment is based on lessons learned from the literature and pedagogical theories that support learning through assessment and scaffolding. This study investigates how the experiment helped the novices, whether it increased their confidence, whether it assisted in identifying and correcting common errors, and whether it covered the material on learning modular programming components. It reports on extensive survey results of over 265 attempts of 90 students taking CS1 (introductory programming) that included five quizzes covering fundamental concepts. The students responded favorably to the experiment, and results are also included.

2012 ACM Subject Classification Applied computing → Education; Social and professional topics → Computing education; Social and professional topics → Student assessment

Keywords and phrases Assessment and feedback, Computer programming, CS1, Formative assessment, Introductory programming, Novice students

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.6

Funding I want to thank the research committee of School of Computing at Dublin City University (DCU) for funding this research project.

1 Introduction

Any course in a higher education institution worldwide that is concerned with software development requires programming modules. By introducing syntax and semantics, these modules aim to impart fundamental knowledge of programming languages. These modules are crucial for students to feel confident in their study in Computer Science (CS). Students who are learning their introductory programming must also become familiar with the often-hard syntax of the language, numerous data types and its operations, the effects of various statements on variables, and control flow. Novice programmers are those taking their first computer programming courses or those with no prior programming experience. E.g. First year computer science degree students, second level students such as junior or senior cycle years. Independent components of programming will increase the difficulties of novices in learning programming [23]. There are a number of activities introduced to motivate novice students in programming modules in addition to traditional lecture and practical sessions. The recommended pedagogical activities are pair programming, peer instructions, live coding, collaborative learning and assessment and feedback systems [7, 30].



© Jagadeeswaran Thangaraj, Monica Ward, and Fiona O’Riordan;
licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 6; pp. 6:1–6:12

OpenAccess Series in Informatics



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

6:2 An Experience with Adaptive Formative Assessment

Every lecture in the classroom is given by a lecturer and includes exposition lectures, live coding to solve practical problems in real time and practical exercises using programming languages in lab sessions. The traditional lecture session occupied the majority of the practice. Also, students felt frustrated that they had to repeatedly go over concepts in lab session from they had already learned in lecture [4]. Furthermore, it does not help students to grasp fundamental ideas. Conversely, they found the live coding and practical portion especially enjoyable since it allowed the students to witness the theories and concepts being used to build a program [25]. However, novice students are unable to interpret program code and have a lack of understanding of individual elements of programming due to their lack of programming experience [24]. The assessment system of programming helps in this scenario as it is an essential component of education that promotes learning [26]. Formative assessment is one of the approaches for effective programming learning that aims to increase student understanding and learning by providing feedback on students' submissions [23]. Taking this into consideration, this research created a formative assessment framework that aims at increasing students' confidence by familiarising novice students with independent components of programming. As an expansion of our earlier research [32], this paper explores the development of a formative assessment framework for this purpose, describes the experiment, and offers insights into its performance and future directions.

2 Exploring Formative Assessments' Potential to Improve Programming Learning

One strategy for efficient programming learning is formative assessment including feedback [31]. Formative feedback is information provided to a learner with the intention of altering their behavior or way of thinking to improve learning [27]. Formative feedback gives teachers the chance to personalize their responses, motivate student interest, gather data on each student's progress, encourage reflective thought, and encourage self-directed learning [33, 21]. Therefore, formative assessment refers to routine, interactive evaluations of student development and comprehension that are used in educational settings to identify learning needs and adapt training [15].

2.1 Enhancing Programming skills through Learning from Errors

When writing programs, both novice and experienced programmers make errors – just not to the same extent. Different kinds of errors demotivate novices to engage in learning to program. Error detection and correction play a central role in programming learning: it is an essential cognitive and pedagogical component of learning to program [20]. Programming errors can appear in a variety of forms such as syntax and logical errors [6, 3, 2]. A program error that breaks language conventions and stops execution is known as a syntax error [3]. When a syntax error occurs in a compiled language, the compiler will usually produce an error message that points to the incorrect program lines. Most of the time, program compilers provide information on the kind and location of syntax errors. Compiler error messages can also be used to identify and fix syntax and semantic errors during program compilation or execution. However, for novice programmers, these error messages can occasionally be difficult to understand and make them confused [5]. Therefore, some research suggested improving error messages to make them easier to interpret and understand [10, 8]. They demonstrated that these enhancement messages were more useful than compiler messages for figuring out common syntax and semantic errors [5, 8].

In addition, students suffer more with logical errors than syntax errors [3]. There are three possible explanations for these logical mistakes: Algorithm, misunderstanding, and false information [12]. Another factor, called “Programmer misconception”, contributes to the misunderstanding of control flow in logic that led to the erroneous code [18]. As a result, the code can deliver inaccurate results. It is also quite challenging to identify and correct logical errors. Various studies have identified common logical errors that people make frequently and offered suggestions for how to avoid them [3, 12]. Thus, helping students understand their programming errors is another effective technique to teach them programming, as mistakes are an incredible means of instruction for programming courses [14, 36]. Therefore, our goal is to familiarise students with these types of errors ahead to make it easier for them to understand. It enables students to comprehend the common code errors they make as well as compiler error messages.

2.2 Enhancing Formative assessment with adaptive strategy

Assessments that are customized to each student individually based on their responses to previous test items are referred to as adaptive assessments [22]. Students who complete adaptive assessments have their ability level determined by the answers they provide; the most illuminating problems are then shown to the students to measure their abilities more accurately [13]. When taking an adaptive assessment, every student will experience it differently from another [34]. In a traditional assessment, every student works on the same set of tasks with varying degrees of difficulty. Students can work on the predetermined tasks in any sequence because they are predetermined. On the other hand, in an adaptive assessment, every student works on a customised set of tasks since the questions are chosen by an algorithm that considers the answers that each student has previously provided. Therefore, it varies from traditional assessment in that each participant is asked a separate set of questions rather than all the same ones [34]. As a result, students work on different questions that take different amounts of time to complete [13]. The system chooses the next test from a pool of available tests based on the the students’ performance [35].

2.3 Encouraging Learning and Proficiency in programming through Adaptive Formative assessment

The process of learning programming involves starting from beginning and using a completion method, such as changing or finishing program code [4]. When it comes to cognitive skills, programming is more advanced than rote memorisation; to complete programming activities, students must grasp particular concepts and understand how to apply them together [35]. Self-efficacy of students can be used to adaptively generate assessments that are customised to each student in terms of question difficulty, assessment length, and question types (e.g., multiple choice, fill-in-the-blank, or short response) [35]. An answer key and a number of choices make up the two components of a multiple-choice question (MCQ) [1]. A query or a remark is typically made by the stem. To proceed with the stem, the learner must choose the best or accurate option. MCQs are insufficient for evaluating a student’s coding proficiency in programming modules since they do not encourage learners to write their own code, even when they are at an advanced level. The ability to retain programming concepts and increase engagement, however, can be useful. The assessment length and question difficulty can also be modified. Using adaptive assessment, which organizes a collection of questions into three cognitive levels according to complexity (easy, moderate, and difficult), adaptive formative assessment evaluates students’ knowledge in programming courses [9]. Consequently, rather than supplying resources that are universally applicable, adaptive assessment systems could customize instruction and evaluation by considering each student’s uniqueness.

6:4 An Experience with Adaptive Formative Assessment

■ **Table 1** An example question with answer choices and accompanying feedback.

Question	Choices	Feedback
What will be the output of following code? <code>var = "computer" print(var[5::1])</code>	computer	Incorrect! It prints a range of items starting from index 5 with step 1. [ter]
	ter	Correct! <code>print(var[5 :: 1])</code> - it prints a range of items starting from index 5 with step 1. [ter]
	compu	Incorrect! if the index is "0" <code>print(var[0::1])</code> , then it prints compu.
	u	Incorrect! if the index is "5" <code>print(var[5])</code> , then it prints u.

2.4 Research Questions

This study intends to investigate the scaffolding and support that formative assessment quizzes might provide for students learning to program. Using the adaptive formative assessment quizzes, this study will particularly investigate the following research questions.

RQ-1: Does formative assessment help to build self-confidence in novice programmers in learning basic concepts of programming?

RQ-2: Does formative assessment help to understand and correct the errors in order to improve their programming skills?

RQ-3: Does formative assessment help novices effectively learn the independent components of programming concepts?

3 Development of Adaptive Formative Assessment Framework

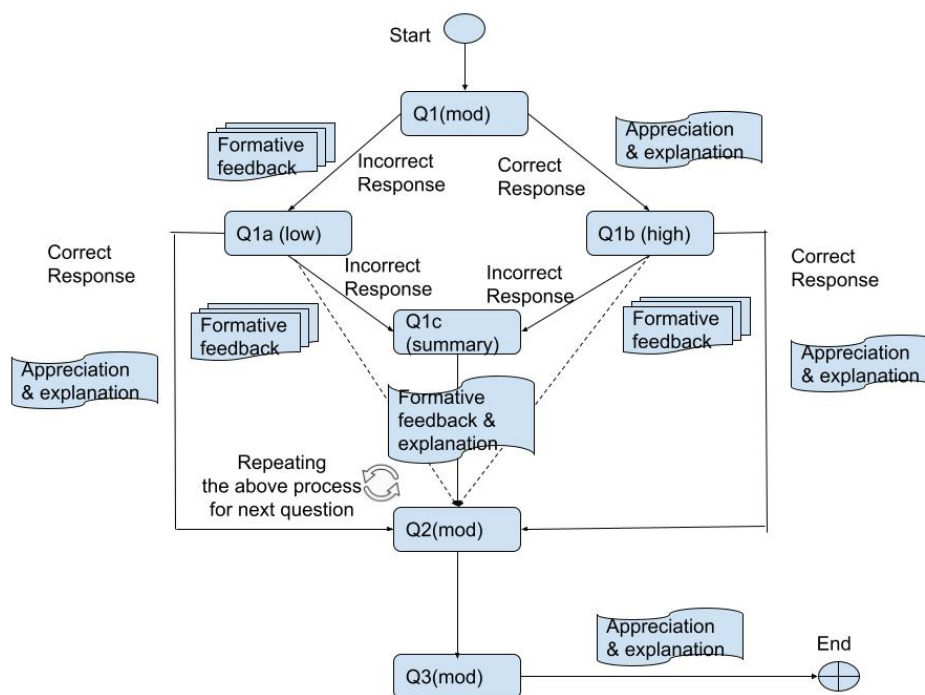
This study has led us to create formative assessment quizzes to introduce common programming errors. These quizzes are an excellent method to increase student confidence and introduce more frequent errors while programming. We have a list of questions with various answers in these quizzes. As they offer feedback for each selection, these quizzes assist in fostering their learning. While the wrong answer feedback helps them locate the appropriate response, the right answer feedback acknowledges their responses. Students can learn from their incorrect responses and determine the correct response. As a result, it is a system that progresses and aids in their ability to learn from mistakes. This study examines if the formative assessment increases participants' confidence in their capacity to understand the fundamental ideas behind programming. They assist students in comprehending the common code errors they make as well as compiler error messages.

3.1 Quiz Implementation

Google Forms is utilized to implement the quizzes because, according to [11], it can be an effective tool for formative assessment and for promoting active learning. Furthermore, integration with all learning management systems is possible. Each quiz aims to educate students about common code errors they made when studying the assigned topics. A sample question is shown in Table 1. Feedback will be given to students for each potential response, which will help them better comprehend the errors and help them to understand easily as shown in Table 1. Feedback are customised messages that are similar to enhanced error messages [5]. Every incorrect reaction offers advice on how to respond. Students can select the best alternative based on the feedback.

3.2 Adaptive Approach

Difficulty levels have been added to learning objects in the model. These learning objects could be topics, questions, a variety of errors. The goals relate to questions with varying degrees of difficulty. Difficulties in programming are classified as Bloom’s taxonomy of programming [29]. In this model, we classified a list of questions in three cognitive levels based on the complexity (like easy, moderate, and difficult) [17]. Here easy questions assess the basic concepts, moderate questions assess comprehensive knowledge and difficult questions do the applications of the knowledge [34]. If a student successfully responds to a moderate question on this assessment, the subsequent question is hard. If not, the easy questions will be asked as indicated in Figure 1. It goes on until the system forecasts the competency level of the students [28]. A sample classification is described as Table 2.



■ **Figure 1** Adaptive approach.

■ **Table 2** Summary of print statement question in adaptive model.

Purpose	Difficulty low	Difficulty moderate	Difficulty high	Summary
String notation and character traverse	var = 'Amazon' print(var[4])	var = 'Computer' print(var[5 :: 1])	var = 'Python' print(var[4 :: -1])	var = 'James Bond' print(var[3]) print(var[5 :: 1]) print(var[5 :: -1])

3.3 Questions Development

Python is an established programming language that is utilized in introductory programming courses due to its convenience and syntactical simplicity [16]. Variables, operators, conditionals, loops, and functions were all covered in the introductory programming course. We created quizzes corresponding with these topics that familiarise modular parts of programming. Each question intends to introduce some of the most common errors made by novices [3, 36]. Details of the questions are shown in Table 3. Quiz topics included syntax errors, logical errors, and common misconceptions among novice students [2, 12, 18]. Every quiz has at least five questions of a moderate difficulty. Depending on the responses, the question moves automatically from a moderate level to a high level or low level as shown in Figure 1. It goes to a summary question that describes the relevant concept to provide proficiency level if they are unable to respond to all of the questions. With the help of the lecturer, we conducted these quizzes periodically during teaching sessions to build novice's confidence as well as to capture their barriers in programming.

■ **Table 3** Summary of questions of each quiz.

Quiz no	Topic	Modular parts	Objectives
1	Print & operators	Print statement Arithmetic operators Assignment operator	Familiarising syntax errors in print statement Familiarising syntax errors in arithmetic expressions Familiarising syntax errors in assignment
2	Variables, input & operations	Input function Operators & precedence Higher arithmetic operator	Familiarising variables and type conversion when read a value from keyboard Familiarising operators & precedence in expressions Familiarising higher precedence arithmetic operators (% , // , **)
3	If/Else statements	The if/elif/else structure with comparative operators Operators & precedence Errors in If/else	Familiarising if/else process & comparative operators Familiarising comparative expression using AND, OR, NOT Familiarising syntax error & indentation error in assignment
4	While loop	While loop process Introducing multiple while loops & infinitive executions Mixed loops	Introducing while loop elements, process before or after increment or decrement Familiarising multiple loops and syntax errors & Non-terminating loops Familiarising multiple loops (while & if/else) & Syntax or logical errors
5	Strings & Functions	String representation Functions Global & Local variables in function	Familiarising different string array representations Familiarising values passing to parameters & syntax errors Familiarising logical errors in assignment of global or local variables

4 Research Methodological paradigm

This research combines both qualitative and quantitative elements. This instance is constrained by the CS1 module during the academic year 2023-24, and different student cohorts of first year undergraduate Computer Science students. In quantitative research, the interven-

tion technique is used to address reliability and validity. A brief, voluntary, anonymous survey is used in conjunction with formative assessment to gain insight of students perceptions on how they perceived and experienced it. It also includes qualitative elements and makes use of a range of data sources and data collection methods. It was conducted via an anonymous “Google forms” questionnaire.

4.1 The Population

The University’s Introductory programming modules provided the data for this investigation. The data includes 267 students’ programming quiz attempts that they submitted at the end of each quiz session. Some students attempted many surveys, in relation to the total number of quiz attempts. These participants were from non-CS major course.

4.2 Data collection strategies

Data was gathered using a survey consisting of both closed-ended and open-ended questions to obtain both qualitative and quantitative data [19]. In addition to traditional teaching and practical sessions, regular quizzes were provided during the study periods. This gave the chance to take quizzes and consider what they had learned. For this study, a short optional and anonymous survey was employed to get an idea of how learners viewed and experienced the quizzes for introductory programming at the end of each quiz. At the end of each quiz, we conducted a survey about how it effectively helped them to learn programming. The respondents were questioned about how they felt about formative assessment quizzes of each programming topic. Open-ended questions for qualitative data and closed-ended Likert scale questions for quantitative data were both used in the survey form. The Likert scale had five possible scores: strongly disagree, disagree, neutral, agree, and strongly agree. The surveys provide both quantitative and qualitative information about the effects of this intervention and perceptions about the use of formative assessment in learning programming. The student questionnaires and their reflective writing assignments provide the qualitative data. Every piece of qualitative data is anonymous.

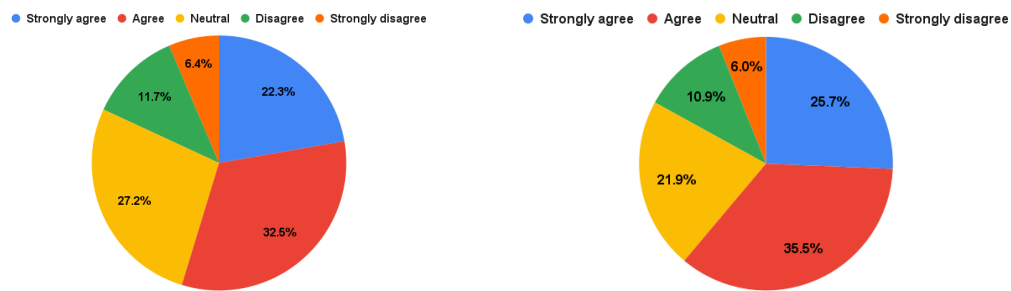
5 Results & Discussion

In order to collect more thorough data for analysis, this study gave students quizzes at regular intervals. Each quiz contained survey questions in addition to other quiz questions. Following an intervention survey, all analysis was completed.

■ **Table 4** Post Survey Likert Questions 1 and 2 (N=265).

Question	Strongly Dis-agree:1	Disagree:2	Neutral:3	Agree:4	Strongly Agree:5	Mean	SD
Do these quizzes increase your self-confidence in learning programming?	17	31	72	86	59	3.52	1.14
Do these quizzes help to understand and correct errors in Python?	16	29	58	94	68	3.64	1.15

6:8 An Experience with Adaptive Formative Assessment



■ **Figure 2** Overall feedback on self-confidence. ■ **Figure 3** Overall feedback on understanding & correcting errors.

5.1 RQ-1: Increasing self-confidence

To answer the RQ-1, it included a Likert question, “Do these quizzes increase your self-confidence in learning programming?”, at the end of the quizzes. The responses ranged from ‘Strongly disagree’ to ‘Strongly agree’. The whole survey data results are presented in Table 4. It offers a thorough understanding of the students’ feelings regarding their level of self-confidence in handling these quizzes. Responses for ‘Strongly agree’ and ‘Agree’ ($\approx 55\%$) were higher than ‘Disagree’ part ($\approx 18\%$) as shown in Figure 2. As a result, this study discovered that the adaptive formative assessment quizzes helped them increase their self-confidence.

5.2 RQ-2: Understand & Correct the errors

This study asked, “Do these quizzes help to understand and correct errors in Python?”. The responses ranged from ‘Strongly disagree’ to ‘Strongly agree’. The whole survey data results are presented in Table 4. Huge responses ($\approx 61\%$) were received as ‘Agree’ as a result as in Figure 3. Their responses for these two questions show a clear shift from their post-quizzes average of 3.52 to an average of 3.64 (with less volatility) as shown in Table 4. These results indicate that the adaptive formative assessment quizzes helped them comprehend the common programming errors.

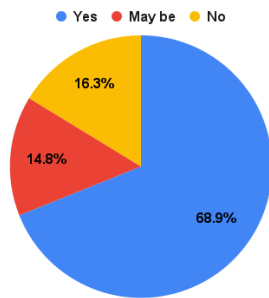
5.3 RQ-3: Students’ perception on learning modular parts

This study asked, “Do these quizzes help to better understand basic concepts of Python language?”. The responses were ‘Yes’, ‘May be’ and ‘No’. The result is presented in Table 5. Huge responses ($\approx 68\%$) were received as ‘Yes’ as a result as shown in Figure 4. Based on the surveys, it demonstrates that the novice students believe the quizzes assisted in grasping fundamental programming principles and significantly increased students’ confidence in learning programming after attending. This study also asked, “Is the feedback you receive for each question helpful in finding the correct answers and understanding errors?”. The responses were ‘Yes’, ‘May be’ and ‘No’. These responses were also highly positive as shown in Table 5.

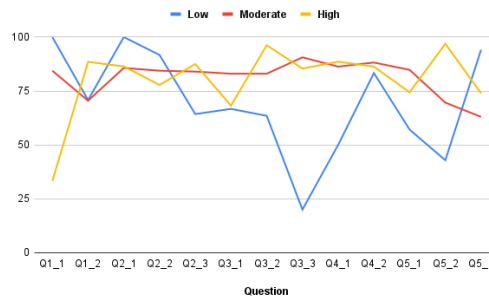
Quantitative data alone does not provide the full picture of the learning experience. Finding out what students think and feel about formative assessment as a computer programming learning activity is critical. According to sentimental analysis, they delighted in gaining knowledge by taking quizzes in various models and they also valued these quizzes for various reasons as stated in the comments below.

■ **Table 5** Post Survey Likert Question 3 (N=267).

Question	No: 1	May be: 2	Yes: 3	Mean	SD
Do these quizzes help to better understand basic concepts of Python language?	40	45	182	2.53	0.74
Is the feedback you receive for each question helpful in finding the correct answers and understanding errors?	43	39	185	2.52	0.76



■ **Figure 4** Overall feedback on understanding basic concepts.



■ **Figure 5** The correct answer frequency (in %) of difficult levels during the quiz attempts.

...It introduced me to new elements of python...made me realise what i didn't know...was good to refresh my brain...very helpful exercises...I think they are much better than the way the lectures are being taught...It helped to recall... Maybe do the quizzes in the lectures to fully understand what is being taught...

Responses for understanding modular concepts ($M = 2.53$, $SD = 0.74$) out of four questions were given a higher weight than other responses with a small effect size as shown in Table 5. It is evident from their feedback that these quizzes enabled them to review their programming expertise and make necessary revisions. Some students claimed that taking the quizzes had taught them some new material.

5.4 Low vs Moderate vs High – Difficulty levels

The frequency of correct responses to questions at varying degrees of difficulty is shown by the outcome chart that we have plotted in Figure 5. It demonstrates that the majority ($\approx 81\%$) of students correctly answer questions at a moderate difficulty level. On the other hand, students who attempt questions with low difficulty levels tend to give less ($\approx 70\%$) correct answers. Additionally, students who tackle highly difficult questions give more ($\approx 80\%$) correct answers. Eventually, students get more detailed feedback to their summary questions, which helps them understand the modular parts of programming better. Moreover, it demonstrates that adaptive formative assessments helped them understand the basic concepts better.

6 Conclusion and Future Work

The experience presented in this paper describes the use of adaptive formative assessment in introductory programming to help novice students overcome the challenges of learning to program, particularly in identifying and understanding more frequent errors. The results

are clearly confirming that this approach helped novice students become more confident in programming and dispel some of the misconceptions surrounding it. As a conclusion, we argue that adaptive formative assessment quizzes motivate students to evaluate and learn from their mistakes, which in turn encourages them to learn computer programming. It can be a viable teaching and learning tool for computer programming. The reflection makes it easier to comprehend the basic concepts. The students claim that learning through formative assessment quizzes has improved their comprehension and increased their confidence in learning programming. They also claim to be satisfied and indicate that they would repeat this teaching technique again, as evidenced by their high level of loyalty. This study will further investigate whether the adaptive formative assessment quizzes could help novice students learn more effectively in other programming languages. Consequently, we intend to incorporate this technique into other languages like Irish, Portuguese and Spanish.

References

- 1 Pedro Henriques Abreu, Daniel Castro Silva, and Anabela Gomes. Multiple-choice questions in programming courses: Can we use them and are students motivated by them? *ACM Transactions on Computing Education (TOCE)*, 19(1):1–16, 2018.
- 2 Alireza Ahadi, Raymond Lister, Shahil Lal, and Arto Hellas. Learning programming, syntax errors and institution-specific factors. In *Proceedings of the 20th Australasian Computing Education Conference, ACE '18*, pages 90–96, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3160489.3160490.
- 3 Nabeel Alzahrani and Frank Vahid. Common logic errors for programming learners: A three-decade literature survey. In *2021 ASEE Virtual Annual Conference Content Access*, Virtual Conference, July 2021. ASEE Conferences. URL: <https://peer.asee.org/36814>.
- 4 Zahra Atiq and Michael Loui. A qualitative study of emotions experienced by first-year engineering students during programming tasks. *ACM Transactions on Computing Education*, 22, March 2022. doi:10.1145/3507696.
- 5 Brett Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. Effective compiler error message enhancement for novice programming students. *Computer Science Education*, pages 1–28, September 2016. doi:10.1080/08993408.2016.1225464.
- 6 Anat Ben-Yaacov and Arnon Hershkovitz. Types of errors in block programming: Driven by learner, learning environment. *Journal of Educational Computing Research*, 61(1):178–207, 2023. doi:10.1177/07356331221102312.
- 7 Neil C. C. Brown and Greg Wilson. Ten quick tips for teaching programming. *PLOS Computational Biology*, 14(4):1–8, April 2018. doi:10.1371/journal.pcbi.1006023.
- 8 Tessa Charles and Carl Gwilliam. The effect of automated error message feedback on undergraduate physics students learning python: Reducing anxiety and building confidence. *Journal for STEM Education Research*, 6(2):326–357, August 2023. doi:10.1007/s41979-022-00084-4.
- 9 Dimitra Chatzopoulou and Anastasios Economides. Adaptive assessment of student’s knowledge in programming courses. *J. Comp. Assisted Learning*, 26:258–269, August 2010. doi:10.1111/j.1365-2729.2010.00363.x.
- 10 Paul Denny, James Prather, Brett A. Becker, Catherine Mooney, John Homer, Zachary C Albrecht, and Garrett B. Powell. On designing programming error messages for novices: Readability and its constituent factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, CHI '21*, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3411764.3445696.
- 11 Mireille Djenno, Glenda M. Insua, and Annie Pho. From paper to pixels: using google forms for collaboration and assessment. *Library Hi Tech News*, 32(4):9–13, 2022. doi:10.1108/LHTN-12-2014-0105.

- 12 Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. Common logic errors made by novice programmers. In *Proceedings of the 20th Australasian Computing Education Conference, ACE '18*, pages 83–89, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3160489.3160493.
- 13 Takayuki Goto, Kei Kano, and Takayuki Shiose. Students’ acceptance on computer-adaptive testing for achievement assessment in japanese elementary and secondary school. *Frontiers in Education*, 8, 2023. doi:10.3389/feduc.2023.1107341.
- 14 Heather J. Hoffman and Angelo F. Elmi. Do students learn more from erroneous code? exploring student performance and satisfaction in an error-free versus an error-full sas® programming environment. *Journal of Statistics and Data Science Education*, 29(3):228–240, 2021. doi:10.1080/26939169.2021.1967229.
- 15 Seyed M. Ismail, D. R. Rahul, Indrajit Patra, and Ehsan Rezvani. Formative vs. summative assessment: impacts on academic motivation, attitude toward learning, test anxiety, and self-regulation skill. *Language Testing in Asia*, 12(1):40, 2022. doi:10.1186/s40468-022-00191-4.
- 16 Fionnuala Johnson, Stephen McQuistin, and John O’Donnell. Analysis of student misconceptions using python as an introductory programming language. In *Proceedings of the 4th Conference on Computing Education Practice, CEP '20*, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3372356.3372360.
- 17 Fatima Ezzahraa Louhab, Ayoub Bahnasse, and Mohamed Talea. Towards an adaptive formative assessment in context-aware mobile learning. *Procedia Computer Science*, 135:441–448, 2018. The 3rd International Conference on Computer Science and Computational Intelligence (ICCS CI 2018) : Empowering Smart Technology in Digital Era for a Better Life. doi:10.1016/j.procs.2018.08.195.
- 18 Davin Mccall and Michael Kölling. Meaningful categorisation of novice programmer errors. In *Proceedings - Frontiers in Education Conference, FIE*, volume 2015, October 2014. doi:10.1109/FIE.2014.7044420.
- 19 Donna Mertens. *Research and Evaluation in Education and Psychology: Integrating Diversity with Quantitative, Qualitative, and Mixed Methods 5th edition*. SAGE Publications, Inc, June 2019.
- 20 Anastasia Misirli and Vassilis Komis. Computational thinking in early childhood education: The impact of programming a tangible robot on developing debugging knowledge. *Early Childhood Research Quarterly*, 65:139–158, 2023. doi:10.1016/j.ecresq.2023.05.014.
- 21 Gunilla Näsström, Catarina Andersson, Carina Granberg, Torulf Palm, and Björn Palmberg. Changes in student motivation and teacher decision making when implementing a formative assessment practice. *Frontiers in Education*, 6, 2021. doi:10.3389/feduc.2021.616216.
- 22 Elena Papanastasiou. *Adaptive Assessment*, pages 1–2. Springer Netherlands, Dordrecht, 2021. doi:10.1007/978-94-007-6165-0_3-4.
- 23 Yizhou Qian and James Lehman. Students’ misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.*, 18(1), October 2017. doi:10.1145/3077618.
- 24 Sjaak Smetsers Renske Weeda and Erik Barendsen. Unraveling novices’ code composition difficulties. *Computer Science Education*, 0(0):1–28, 2023. doi:10.1080/08993408.2023.2169067.
- 25 Anders Schlichtkrull. An experience with and reflections on live coding with active learning. In *International Computer Programming Education Conference*, 2023. URL: <https://api.semanticscholar.org/CorpusID:260777639>.
- 26 Nicole Shanley, Florence Martin, Nicole Collins, Manuel Perez-Quinones, Lynn Ahlgrim-Delzell, David Pugalee, and Ellen Hart. Teaching programming online: Design, facilitation and assessment strategies and recommendations for high school teachers. *TechTrends*, 66, April 2022. doi:10.1007/s11528-022-00724-x.
- 27 Valerie J. Shute. Focus on formative feedback. *Review of Educational Research*, 78(1):153–189, 2008. doi:10.3102/0034654307313795.

6:12 An Experience with Adaptive Formative Assessment

- 28 E'loria Simon-Campbell and Julia Phelan. Effectiveness of an adaptive quizzing system as a self-regulated study tool to improve nursing students' learning. *International Journal of Nursing & Clinical Practices*, 5, August 2018. doi:10.15344/2394-4978/2018/290.
- 29 Sonia Sobral. Bloom's taxonomy to improve teaching-learning in introduction to programming. *International Journal of Information and Education Technology*, 11:148–153, March 2021. doi:10.18178/ijiet.2021.11.3.1504.
- 30 Sonia Sobral. *Strategies on Teaching Introducing to Programming in Higher Education*, pages 133–150. Springer, Cham, March 2021. doi:10.1007/978-3-030-72660-7_14.
- 31 Qing Sun, Ji Wu, Wenge Rong, and Wenbo Liu. Formative assessment of programming language learning based on peer code review: Implementation and experience report. *Tsinghua Science and Technology*, 24:423–434, August 2019. doi:10.26599/TST.2018.9010109.
- 32 Jagadeeswaran Thangaraj, Monica Ward, and Fiona O'Riordan. The impact of using formative assessment in introductory programming on teaching and learning. *10th International Conference on Higher Education Advances (HEAd'24)*, Valencia, June 2024.
- 33 Fabienne S. Van der Kleij and Lenore Adie. Formative assessment and feedback using information technology. *Second Handbook of Information Technology in Primary and Secondary*, pages 601–615, 2018. doi:10.1007/978-3-319-71054-9.
- 34 Jill-Jënn Vie, Fabrice Popineau, Éric Bruillard, and Yolaine Bourda. *A Review of Recent Advances in Adaptive Assessment*, volume 94, pages 113–142. Springer International Publishing, February 2017. doi:10.1007/978-3-319-52977-6_4.
- 35 Albert C.M. Yang, Brendan Flanagan, and Hiroaki Ogata. Adaptive formative assessment system based on computerized adaptive testing and the learning memory cycle for personalized learning. *Computers and Education: Artificial Intelligence*, 3:100104, 2022. doi:10.1016/j.caeai.2022.100104.
- 36 Zihe Zhou, Shijuan Wang, and Yizhou Qian. Learning from errors: Exploring the effectiveness of enhanced error messages in learning to program. *Frontiers in Psychology*, 12, 2021. doi:10.3389/fpsyg.2021.768962.

Promoting Deep Learning Through a Concept Map-Building Collaborative Activity in an Introductory Programming Course

João Paulo Barros ✉ 

Polytechnic Institute of Beja, Portugal

Center of Technology and Systems (UNINOVA-CTS) and Associated Lab of Intelligent Systems (LASI), Caparica, Portugal

Abstract

Programming courses focus heavily on problem-solving and coding practice. However, students also face numerous interrelated concepts that should be given more attention to foster more effective and comprehensive learning. Often, students only get an incomplete knowledge of those concepts and their relations as no adequate reflection is promoted or even seen as necessary. The result is a superficial surface learning about essential programming concepts and their relations. This experience report presents a learning activity to promote deep learning of concepts and their relations. The activity challenges students to specify relations between concepts. Students search definitions for a given set of concepts and define relations between those concepts in textual form. To that end, they use a freely available tool that produces a graph from textual descriptions. This tool dramatically simplifies and speeds up the creation of readable graphical representations. Although many different courses can take advantage of the presented activity, we present the activity's application to an introductory object-oriented programming course. We also present and discuss the student's feedback, which was highly positive. In the end, we provide recommendations, including possible variations. These can help educators to effectively foster active learning of concepts and their relations in their classrooms.

2012 ACM Subject Classification Social and professional topics → Computing education

Keywords and phrases active-learning, ontologies, concepts, concept maps, learning activity, object-oriented programming, oop, pedagogy, education

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.7

Funding *João Paulo Barros*: This research was funded by Portuguese Agency FCT – Fundação para a Ciência e a Tecnologia no âmbito da Unidade de Investigação CTS – Centro de Tecnologia e Sistemas/UNINOVA/FCT/NOVA, com a referência UIDB/00066/2020.

1 Introduction

Programming courses emphasize practical problem-solving projects and coding exercises. At the same time, students are exposed to and put to practical use a wide variety of interrelated concepts. However, these concepts often need additional and adequate emphasis to counter the risk of a shallow understanding of them and their mutual relations. One way is to promote active reflection and discussion about those concepts and their connection in class. This experience report presents a learning activity that fosters conceptual thinking to counter surface learning and promote deep learning about concepts identified as necessary for the course's intended learning outcomes (e.g., [12]). To that end, the teacher identifies a set of concepts and challenges the students to create their concept maps by establishing significant relations between those concepts in a (graphical) concept map. Here, the concept map is seen as a simplified ontology, as it also forces students to investigate and learn the meaning of the concepts to describe their mutual relations, thus achieving a simultaneously specific and holistic understanding of them. However, as formal descriptions of concepts and their



© João Paulo Barros;

licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 7; pp. 7:1–7:12

OpenAccess Series in Informatics



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

mutual relations, ontologies are often exhaustive, hence quite complex, and around a specific area. Therefore, the respective reference tools are also complex (e.g., [21]) and do not provide a quick and straightforward way to build even simple ontologies. Here, we propose a learning activity using simple textual language to describe relations between concepts, from which a graphical representation is automatically generated. This is very significant for the activity's effectiveness, as we have observed that this fact generates a subtle but effective wow effect that promotes engagement and motivation around a topic that can easily be seen as uninteresting, unimportant, or both.

The tool that allows the graphical representation is freely and readily available on the web with no need to install software locally: it can be used on any machine where a browser is available, including mobile devices [6, 4, 5]. The activity was applied in an introductory course on object-oriented programming, a subject matter where students struggle with numerous complex and strongly interconnected concepts. Although the tool's nature could probably make it more attractive to computer science students, its ease of use suggests that it can be suitable for any topic, even outside computer science.

In the following section, we give the background and motivation for the key concepts, namely ontologies, concept maps, and their relation to the reported activity. Section 3 presents related work, and Section 4 the context where the activity was applied. Section 5 details how the activity was conducted and includes some lessons from our experience applying similar activities. Section 6 presents and discusses the observed outcomes based on in-class observation and a student survey. Finally, Section 7 provides several recommendations allowing possible activity variants and concludes.

2 Background and Motivation

An ontology is a concept with its origins in Philosophy, where it was “coined in 1613, independently, by two philosophers, Rudolf Göckel (Goclenius) in his *Lexicon philosophicum* and Jacob Lorhard (Lorhardus) in his *Theatrum philosophicum*” [23]. It seeks a definitive and exhaustive classification of entities, including the types of relations that tie them.

From an engineering perspective, ontologies are design artifacts used to describe knowledge and support knowledge-sharing activities. In this sense, their use in several areas became a trend, as testified by the many articles citing seminal work (e.g., [7] with more than thirteen thousand citations). Ontologies gained such widespread popularity that they are now defined in an international standard for vocabulary in systems and software engineering: “ontology. (1) logical structure of the terms used to describe a domain of knowledge, including both the definitions of the applicable terms and their relationships (ISO/IEC/IEEE 24765i:2020)” [11].

The above definition includes the need to define the applicable terms. Hence, in brief, an ontology becomes a set of defined terms and their (defined) relationships.

The term “ontology” is not as common in the education literature as “concept maps,” “concept mapping,” or even “knowledge map”. In 2006, Nesbit and Adosope [18] estimated that “(...) more than 500 peer-reviewed articles, most published since 1997, have made substantial reference to the educational application of concept or knowledge maps.” This may be related to the genesis of “concept mapping,” as it has emerged “inside” education research, namely to study changes in students understanding of science concepts along twelve years of schooling [19]. However, concept maps, although not so formally defined and discussed, can be seen as a simple way to create ontologies where the “concepts” are the “terms” and the “map” establishes the “relationships” among “concepts.” In our reported activity, the used “concept maps” are more straightforward and shorter because the definitions for the “concepts” are not mandatory. However, students must still know enough about each concept to create meaningful relations among them.

More significant for our work is the fact that “conceptual mapping” is an effective learning activity: the already cited meta-analysis by Nesbit and Adosope [18] concluded that “(...) in comparison with activities such as reading text passages, attending lectures, and participating in class discussions, concept mapping activities are more effective for attaining knowledge retention and transfer.”

By promoting deep learning of concepts, our reported activity can easily allow for the identification of students’ misconceptions, a common topic in the literature (e.g., [14],[13],[22]) and provide a basis for a new form of “misconception-driven-feedback” [8]. In [14], formal interviews were used to identify misconceptions; in [13], the objective was to validate a methodology for building a concept inventory dedicated to OOP; in [22], multiple-choice-based questions were used to identify misconceptions. The activity we report in this paper allows a quick, in-class, collaborative identification of misconceptions in any domain or subdomain with the added anecdotal benefit of directly promoting learning and being more engaging than a traditional assessment element such as a multiple-choice or oral exam. Additionally, the reported activity also provides a tool for assessing and identifying incomplete and inconsistent student mental models, another significant problem (e.g., [17]).

While promoting critical and conceptual thinking, the created concept maps also offer an index and a structured map of the concepts students have to know and apply. Hence, they provide a “structural view” (concepts and relations) that complements the “behavioural view” (coding), where students spend most of their time.

Finally, students creating relations between concepts can also be seen as a direct application of constructivism, a theory of learning that claims that students construct knowledge rather than merely receive and store knowledge transmitted by the teacher[2].

3 Related Work

In a literature survey on the use of ontologies in education, the authors identify numerous uses, but primarily for describing learning domains [24]. Surprisingly, the authors go to the point of asserting that “While developing ontologies, one must use a programming language.” Thus, they somehow impose the use of computer support for creating ontologies. They identify several types of ontologies and the methodologies used to define them. The authors also present an overview of existing systems that use ontologies in the education domain. In that context, it is easy to find papers proposing ontologies for one or more domains in the education area. In the area of programming, one focus is the languages themselves (e.g., [15]).

Also, there are numerous cases of using concept mapping and concept maps as part of learning activities (e.g., [1]). However, it is important to stress that *concept map* is a term nowadays usually associated with the work by Novak in the 70s (e.g., [25]) for a specific method to represent knowledge by relating concepts. More specifically, “concepts are arranged hierarchically with the most general, most inclusive concept at the top, and the most specific, least general concepts toward the bottom.” (in [20]). Novak and Cañas claim that “Concept mapping was invented in 1972”. However, it seems clear that concept mapping can be seen as part of what an ontology is, as presented in the previous section: “classification of entities including also the types of relations that tie those .” In fact, we also used the name “concept map” to better explain to students what they were going to create: they map concepts to other concepts to build a simple ontology. We also alert them that, in the literature, the name “concept map” is frequently used for a specific way to map concepts.

Unlike previous works, our approach is based on a freely available tool that automatically generates graph layouts from textual descriptions. This automation streamlines the creation of concept maps, increasing student engagement and motivation. Additionally, automatic graph layout generation from text is a compelling topic in itself, particularly for computer

7:4 Promoting Deep Learning Through a Concept Map-Building Collaborative Activity

science students, some of whom have expressed interest in applying the tool across courses. We provide detailed instructions and variations for the activity, enabling teachers to adapt it to various contexts.

4 Context

The reported activity was applied in 2022/2023 in an introduction to object-oriented programming course. The course is offered in the second semester of a three-year computer science bachelor's degree program at a small university with around three thousand students in total, of which around two hundred are in the computer science program. It is a second course on programming as the students already have a first course in the first semester. However, students may have failed the first course and still try to do the second, as allowed. The present article reports the activity conducted with the students in the 2022/2023 course edition. This activity already integrated some lessons learned in previous applications of similar activities in 2021/2022. These lessons are presented in Subsection 5.2.

5 Concept map creation activity

This section describes the activity as applied in the 2022/2023 course edition. As already stated, this incorporates lessons learned in previous applications in 2021/2022, which were briefly reported elsewhere [*anonymized*]. This first subsection should provide enough information to allow educators to reproduce the activity in their courses. The second subsection presents lessons learned from the 2021/2022 edition that motivated some changes in the 2022/2023 edition.

5.1 Application

The reported activity was designed and applied as an in-class face-to-face activity. Nevertheless, with minimal modifications, it can surely be applied in synchronous remote classes and, with minor changes, even as an asynchronous remote activity. The activity steps were the following:

1. The motivation for the activity is explained to the students, namely the importance of learning the concepts and their relations;
2. The GraphViz online tool [6, 4, 5] is briefly presented by explaining the given base description and how it quickly and automatically generates the respective graph; the set of concepts related to previously studied contents are presented as nodes in the graph; the syntax to define relations between concepts is also exemplified;
3. Each student is given 20 minutes to create a graph relating concepts that, for the sake of simplicity, we call a “concept map”;
4. Each student gets together with another colleague to create an improved concept map; another 20 minutes are given;
5. Each pair of students gets together with another pair to create an even better concept map; each group is given an additional 20 minutes to create an improved concept map;
6. Each group of four publishes, in a shared document, the resulting concept map for all to see;
7. The teacher presents and discusses with the class one or more of the shared concept maps.

In step (1), the teacher notes that students have already used and applied numerous concepts that should be better understood. To that end, they must search for and relate the definitions, thus creating their own concept map. In this way, they should better understand

those concepts and their mutual relations. Regarding step (2), the teacher presents a slide with the link and the respective QR code to the start graph in the online tool (see Fig. 1). The graph layout automatically generated is presented in the center of the slide. A giant QR code (omitted in the figure) is presented to facilitate students accessing the start graph in the online web tool. The concepts to be related are already presented in the context of the tool to be used, more specifically in the listing on the left side of the slide in Fig. 1. For readability purposes, the listing is also presented after the figure. In this example, the specified graph is directed (**digraph**) (arcs are “one side arrows”). Nodes are shaped like ellipses. Those nodes are the concepts the students should relate: **value**, **variable**, **constant**, ..., **operator**. Then, four examples of arcs are also presented. As the syntax is straightforward, for each new relation, students only have to copy and paste one of the lines 22 to 25 and change the node names and the parameter **label** to assign the intended meaning to the relation.

What are we going to do?

Start from the graph in <https://linkTostartGraph> and add the relations you find adequate between the concepts already there.

The figure shows a slide with three main components. On the left is a code editor window with a dark background and light text. It contains a list of concepts (lines 4-21) and a set of relations (lines 22-25). The relations are: `class -> object [label = "source_of"]`, `parameter -> class [label = "has_a"]`, `value -> value [label = "uses"]`, and `procedure -> function [label = "is_a"]`. In the center is a graph with several nodes (represented as ellipses) and directed edges (arcs) connecting them. On the right is a large square QR code with the text "QR Code" in the center.

■ **Figure 1** Introductory slide.

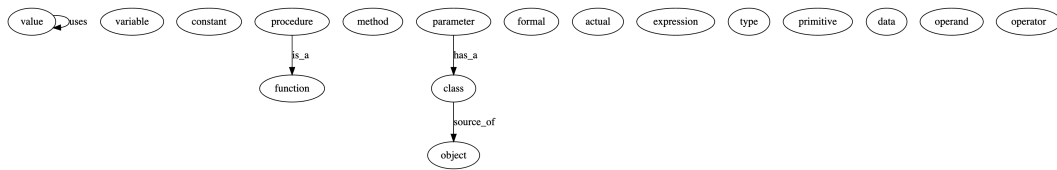
```

1  digraph G {
2  { node [shape=ellipse]
3  value
4  variable
5  constant
6  function
7  procedure
8  method
9  class
10 object
11 parameter
12 formal parameter
13 actual parameter
14 expression
15 type
16 primitive type
17 data type
18 operand
19 operator
20 }
21 // to correct and complete
22 class -> object [label = "source_of"]
23 parameter -> class [label = "has_a"]
24 value -> value [label = "uses"]
25 procedure -> function [label = "is_a"]
26 }

```

7:6 Promoting Deep Learning Through a Concept Map-Building Collaborative Activity

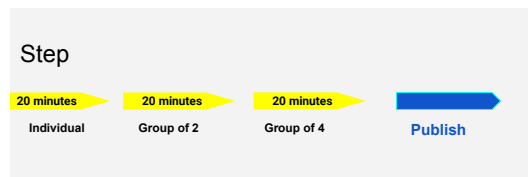
Fig. 2 presents the graph layout generated from the description. It shows the concepts as graph nodes (ellipses) and the four relations as directed arcs.



■ **Figure 2** Start graph layout.

The concepts are nodes in the *dot* description language, and, in the end, a set of four relations is presented. These are guaranteed to be syntactically correct but not semantically, as students are asked to correct and complete them. For example, in line 27 of the presented list, a procedure is (wrongly) defined as being a function. The teacher exemplifies how students can quickly create new relations by copying and pasting one of the relations in the last lines of the listing and changing the names. Next, the teacher follows the link and exemplifies some changes to the graph description so that students can see the immediate generation of the graph layout.

In steps (3), (4), and (5), the 1-2-4-all method [16] is applied to guide students in producing their concept maps: first individually, then in pairs, and finally in groups of four, students create graphs that relate the given concepts, thus creating their concept maps that are then shared with all. From our experience, it is essential to briefly explain the four method stages, with the help of a slide (see Fig. 3), and also that students know, all the time, the stage they are in (individual, pairs, or groups of four). For that reason, one slide with a countdown timer is used for each stage. Fig. 4 shows the slide for the first stage (the individual work one). It also includes a link to the shared document and the start graph.



■ **Figure 3** Slide for presenting the 1-2-4-all method to students.

Individually create a concept map that relates the concepts in **linkToStartGraph** (moodle). You have 20 minutes to make your map

Where to create it
<https://linkToStartGraph>

QR Code

Feedback in shared document
<https://linkToSharedDocument>

QR Code

■ **Figure 4** First stage slide.

Next, in step (6), each group shares (anonymously if they prefer) the result with the class in an online document. This document remains accessible as the result of the class activity and can be consulted any time after, in, or outside class.

Finally, in step (7), the teacher chooses one or more concept maps and discusses their contents with the class.

5.2 Lessons learned and practicalities

Preliminary versions of the presented activity were applied in the 2021/2022 academic year, together with a closely related activity: the presentation of a ready-made concept map. From those applications, some observations were made, and some lessons were learned:

- With no rule for collaboration, most students were quite individualistic, and collaboration was minimal;
- One student complaint they had too much time alone to create the concept map;
- Besides the tool's simplicity, some students asked for a brief explanation of its functioning;
- Some students asked for an exemplary concept map made by the teacher to be studied after the activity.

The activity here reported already incorporated the following strategies to answer the above points: regarding (1) and (2), students now start individually but then proceed in pairs and groups of four; regarding (3), the teacher gave a brief explanation about how to use the tool; regarding (4), the teacher provided an exemplary concept map after the activity. The following section presents the results of the reported activity as applied in the academic year 2022/2023.

6 Observed outcomes

This section presents the results as students' perceptions. These were collected in two ways: (1) in class, along the activity, students were randomly and informally asked about their opinions, and the teacher observed their performance; (2) a post-class questionnaire was applied to all 70 students in the activity and 42 students responded.

6.1 In-class observation

No significant difficulty was observed throughout the class, as no student felt impeded from proceeding. There were just a few questions about what was allowed as relations, primarily due to a lack of attention to the initial explanation and because some students wanted to create additional types of relations, e.g., using bidirectional arcs. Most students were very engaged, and one reason was the wow factor associated with observing the automatic graph layout generation after each change in the textual description.

Students vocalized a few opinions along the activity execution that we list next:

- For learning, it is also useful to compare the created concept maps with other concept maps at the end of the activity; only the creation would be insufficient;
- To see an exemplary solution would be very useful;
- One previous solution is not so useful, but it can be helpful to better understand what the objective is.

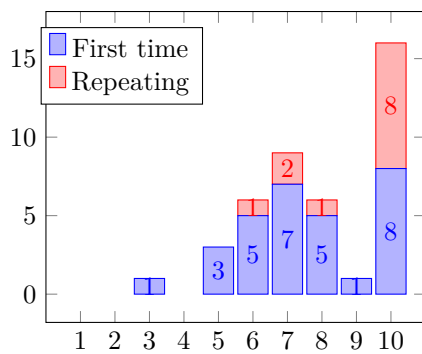
As students were given the possibility to create new relations (new labels in arcs), most did it. At the end of the class, the teacher's perception was that all students had found the activity very positive and helpful.

6.2 Student Survey

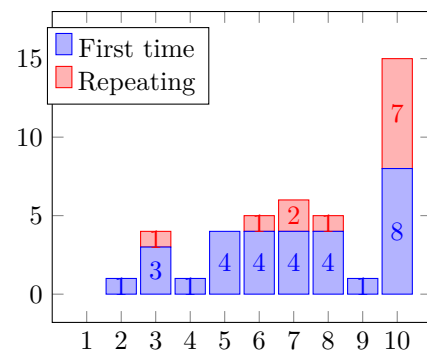
After the classes, all 70 participating students were invited to fill out a short questionnaire. We got 42 answers: 30 were first-time students, and 12 were repeating students. In an online questionnaire, they were asked to grade the following four assertions on a scale from 1 (totally disagree) to 10 (totally agree):

1. “It is useful to see and discuss a concept map in class.”;
2. “It is helpful to have concept maps for studying outside class.”;
3. “Concept maps with other sets of programming concepts can help understand them better.”;
4. “The concept map was asked to be done in three sequential steps: individually, in pairs, and groups of four. That method is something to repeat.”.

The survey results are presented as stacked bar charts in figures 5a, 5b, 6a, and 6b.

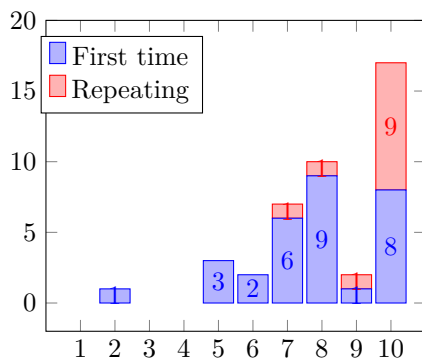


(a) Results for the assertion “It is useful to see and discuss a concept map in class.” 1-totally disagree to 10-totally agree.

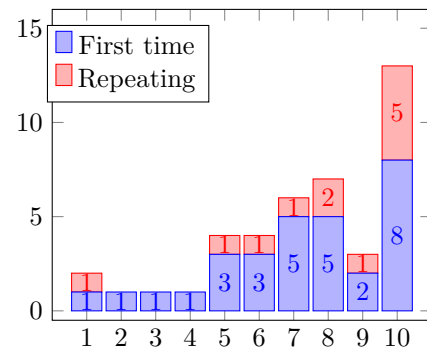


(b) Results for the assertion “It is helpful to have concept maps for studying outside class.” 1-totally disagree to 10-totally agree.

■ **Figure 5** Grading usefulness and helpfulness.



(a) Results for the assertion “Concept maps with other sets of programming concepts can help understand them better.” 1-totally disagree to 10-totally agree.



(b) Results for the assertion “The concept map was asked to be done in three sequential steps: individually, in pairs, and in groups of four. That method is something to repeat.” 1-totally disagree to 10-totally agree.

■ **Figure 6** Other concepts and process.

In all four, the most frequent answer was “totally agree.” Also, there is a clear agreement with all four assertions. Students’ opinions varied more regarding the usefulness outside class but still had a very high level of agreement: only 10 in 42 graded it between 1 and 5.

Regarding the fourth assertion, about the applied method, two students answered “totally disagree”: as made clear in their open answers, the first-time student stated a preference for an unrestricted number of students in the groups; the repeating student mentioned a timetable overlap with another class, a clearly unrelated motif.

In the same survey, students were also given the (optional) opportunity to leave some free comments about the activity. Next, we list the comments:

- “In my opinion, all the results obtained should be discussed to take the best possible advantage.”;
- “I think it’s a very good method, it helps a lot those who do not feel very comfortable in the area to become more interested.”;
- “Do more activities in class to encourage students to interact with each other.”;
- “I think we should have more similar classes, we really lack communication in the classroom, and group classes bring a different dynamic to the classes and to our own way of learning.”;
- “It would be important for the teacher to show a map made by himself in order to be able to compare the maps developed by the students.”;
- “It’s great. We should do it more often.”.

The main suggestion by students was for the teacher to make available an exemplary solution so that they could compare it with their own. This is probably due, at least in part, to the fact that this is not an exercise that they can “run” and check if it is correct, as they are used to when programming.

Next, we present some main recommendations, most of which are optional additional steps in the activity application.

7 Limitations

The current study has several limitations that should be acknowledged. First, the sample size was relatively small, with only 30 first-time and 12 repeating students participating in the study. These numbers may limit the generalization of the results, as the sample may not be representative of the broader student population. Additionally, a long-term follow-up assessment is needed to determine the sustained impact of the proposed activity on students’ learning outcomes. Future research should replicate this study with a more extensive and diverse sample and incorporate long-term follow-up assessments to evaluate the enduring effects of the activity.

8 Recommendations and Conclusions

The reported activity was applied in two 100-minute classes. However, with more class time or reducing the time for each stage, several additional sub-activities can be conducted to boost the activity objective. Here, we list the ones we have identified as potentially interesting. Some result from additional reflection after applying the reported activity and listening to students’ opinions. Others were identified initially but left out due to timing constraints. In any case, the addition of one or more of the presented sub-activities will probably promote the activity objective. Time seems to be the only significant restriction to their application. Next, we present the identified sub-activities grouped in three contexts: (1) before the class where the activity will take place; (2) during the activity itself; (3) after the presented in-class activity, but still in class; (4) as autonomous work, after the class where the activity took place.

7:10 Promoting Deep Learning Through a Concept Map-Building Collaborative Activity

1. Before the class where the activity will take place

Videos about concepts. Make available one or more videos about the concepts and reward comments students make to those videos or to the colleague's comments. A platform like VideoAnt [9, 3], or a simple online forum for this specific task can support this ;

Videos about GraphViz online tool. If more sophisticated relations are going to be asked, then one or more tutorial videos about the GraphViz online tool and the dot language may also be made available before the in-class activity;

Initial quiz. Answering some questions about the concepts before the activity can be useful for the teacher as a diagnostic assessment and for the students as a formative assessment;

Individual stage before class. The individually created concept map can be done as a pre-class exercise, leaving class time for group work and thus increasing collaboration time;

Exemplary concept maps. To better clarify the intended activity object and objective, one or more exemplary concept maps relating different but well-known sets of concepts can be given in advance;

2. During the activity itself

Add concepts' definitions. Strictly speaking, each ontology should include the concepts' definitions; here, we assumed students searched those definitions so that they could relate them, but we did not mandate them to include those definitions; however, this can be easily achieved by adding, in the dot description, links to webpages containing the student's definition of each concept;

Closed set of relations. Instead of letting students create new relation types, students may be asked only to use the given set of relations types; this has the advantage of allowing an easier measuring of outcomes, which can also simplify comparability and grading; in fact, the use of distinct and closed sets of relations and concepts can provide a useful formative assessment tool based on the observation and recording of students doubts, difficulties, and achievements;

3. After the presented in-class activity, but still in class

Public presentation. After the groups publish their ontology in the shared space, a speaker from each group can be asked to present the group's work;

Comments on the presentations After each presentation, all the students and the teacher can provide comments and discuss the merits and pitfalls of the presented ontology, thus providing an additional opportunity for clarification and discussion;

Improvement after presentations. The groups can be allowed to improve their concept maps based on the received comments from all students and the teacher; the comments can be oral or written, and their quality graded by the teacher or students as a reward and incentive;

End of activity quiz. Especially if an initial quiz was done, an end-of-activity quiz can be useful for the teacher and for the students as a formative assessment to answer some questions about the concepts and compare the result with those in the initial quiz;

Discussion about an exemplary concept map The teacher presents and discusses a concept map of his own with the students.

4. As autonomous work, after the class where the activity took place

Commenting a concept map. After the synchronous class, the teacher makes available a concept map of his/her authorship for the trainees to comment on; this can be supported by a specific tool (e.g., [10]) or by an online forum.

In the reported experience, the presented activity was perceived as valuable by the students and the teacher as a way to actively foster the deep learning of concepts and their relations in an introductory object-oriented programming course. Using a free online tool to automatically generate graph layouts significantly accelerated and simplified the creation of the graphical representation and promoted engagement. Its simplicity should allow the use of the activity in many other courses, including outside computer science.

References

- 1 Olusola O. Adesope and John C. Nesbit. *A Systematic Review of Research on Collaborative Learning with Concept Maps*, pages 238–255. Handbook of Research on Collaborative Learning Using Concept Mapping. IGI Global, Hershey, PA, USA, 2010. doi:10.4018/978-1-59904-992-2.ch012.
- 2 Mordechai Ben-Ari. Constructivism in computer science education. In *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '98, pages 257–261, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/273133.274308.
- 3 Digital Education and Innovation, College of Education and Human Development, University of Minnesota. Videoant, 2023. Available at <https://ant.umn.edu/>, accessed on 2023/08/15.
- 4 John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz— open source graph drawing tools. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing*, pages 483–484, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
- 5 Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software: Practice and Experience*, 30(11):1203–1233, 2000. doi:10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.0.CO;2-N.
- 6 Graphviz online, 2022. Available at <https://dreampuf.github.io/GraphvizOnline/>, accessed on 2023/08/17.
- 7 Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5):907–928, 1995. doi:10.1006/ijhc.1995.1081.
- 8 Luke Gusukuma, Austin Cory Bart, Dennis Kafura, and Jeremy Ernst. Misconception-driven feedback: Results from an experimental study. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ICER '18, pages 160–168, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3230977.3231002.
- 9 Bradford Hosack. Videoant: Extending online video annotation beyond content delivery. *TechTrends*, 54(3):45–49, May 2010. doi:10.1007/s11528-010-0402-7.
- 10 Hypothesis, Inc. Hypothesis, 2023. Available at <https://web.hypothes.is/>, accessed on 2023/08/15.
- 11 ISO/IEC/IEEE 2017. Systems and software engineering – Vocabulary. Standard ISO/IEC/IEEE 24765, International Organization for Standardization, Geneva, CH, 2017. URL: https://standards.iso.org/ittf/PubliclyAvailableStandards/c071952_ISO_IEC_IEEE_24765_2017.zip.
- 12 Gregor Kennedy John Biggs, Catherine Tang. *Teaching for Quality Learning at University*. McGraw Hill, 5 edition, 2022.
- 13 Henry Julie, Dumas Bruno, Heymans Patrick, and Leclercq Tony. Object-oriented programming: Diagnosis understanding by identifying and describing novice perceptions. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, 2020. doi:10.1109/FIE44824.2020.9273990.
- 14 Lisa C. Kaczmarczyk, Elizabeth R. Petrick, J. Philip East, and Geoffrey L. Herman. Identifying student misconceptions of programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 107–111, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1734263.1734299.



7:12 Promoting Deep Learning Through a Concept Map-Building Collaborative Activity

- 15 Ming-Che Lee, Ding Yen Ye, and Tzone I Wang. Java learning object ontology. In *Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05)*, pages 538–542, 2005. doi:10.1109/ICALT.2005.185.
- 16 Henri Lipmanowicz and Keith McCandless. 1-2-4-all – liberating structures including and unleashing everyone, 2023. Available at <https://www.liberatingstructures.com/1-1-2-4-all/>, accessed on 2023/08/15.
- 17 Syeda Fatema Mazumder. Investigating the role of explanative diagrams as a representation of notional machine on a novice programmer’s mental model. In *Proceedings of the 17th ACM Conference on International Computing Education Research, ICER 2021*, pages 409–410, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3446871.3469775.
- 18 John C. Nesbit and Olusola O. Adesope. Learning with concept and knowledge maps: A meta-analysis. *Review of Educational Research*, 76(3):413–448, 2006. doi:10.3102/00346543076003413.
- 19 Joseph D. Novak. Concept mapping: A useful tool for science education. *Journal of Research in Science Teaching*, 27(10):937–949, 1990. doi:10.1002/tea.3660271003.
- 20 Joseph D Novak and Alberto J Cañas. The origins of the concept mapping tool and the continuing evolution of the tool. *Information Visualization*, 5(3):175–184, 2006. doi:10.1057/palgrave.ivs.9500126.
- 21 Protégé, 2020. Available at <https://protege.stanford.edu/>, accessed on 2023/08/17.
- 22 Vijayalakshmi Ramasamy, Mourya Reddy Narasareddygari, Gursimran S. Walia, Andrew A. Allen, Debra M. Duke, James D. Kiper, and Debra Lee Davis. A multi-institutional analysis of cs1 students’ common misconceptions of key programming concepts. In Maria Virvou, George A. Tsihrintzis, Nikolaos G. Bourbakis, and Lakhmi C. Jain, editors, *Handbook on Artificial Intelligence-Empowered Applied Software Engineering: VOL.2: Smart Software Applications in Cyber-Physical Systems*, pages 127–144. Springer International Publishing, Cham, 2022. doi:10.1007/978-3-031-07650-3_8.
- 23 Barry Smith. *Ontology*, chapter 11, pages 153–166. John Wiley & Sons, Ltd, 2004. doi:10.1002/9780470757017.ch11.
- 24 Kristian Stancin, Patrizia Poscic, and Danijela Jaksic. Ontologies in education – state of the art. *Education and Information Technologies*, 25(6):5301–5320, November 2020. doi:10.1007/s10639-020-10226-z.
- 25 Ling Xu, Ming-Wen Tong, Bin Li, Jiang Meng, and Chen-Yao Fan. Application of concept map in the study of computational thinking training. In *2019 14th International Conference on Computer Science & Education (ICCSE)*, pages 454–459, 2019. doi:10.1109/ICCSE.2019.8845505.

Scientific Whispers: Mapping Innovative Pedagogies in STEAM and Programming Education

Margarida Antunes  

Polytechnic University of Coimbra, Portugal

António Trigo¹  

Polytechnic University of Coimbra, Portugal

CEOS.PP, ISCAP, Polytechnic of Porto, Portugal

Abstract

Traditional education, especially in STEAM and programming, faces several challenges in capturing the attention and stimulating the new student generation. These challenges are exacerbated by rigid teaching methods and reflect a global difficulty in the educational sector, primarily stemming from the disconnect between traditional teaching and the contemporary needs of the modern world. This article presents a systematic literature review with a mapping study to explore innovative approaches currently employed in teaching, specifically focusing on STEAM and programming education. The conclusions reached make a significant contribution to the field of education, and the mapping conducted has identified the teaching methodologies most researched and investigated by the scientific community. This research also presents a classifying proposal for those methodologies, considering their characteristics and weighing up three dimensions: resources, implementation and receptiveness. As a final reflection, some emerging methodologies were identified that are believed to have great potential to be used for STEAM and programming education.

2012 ACM Subject Classification Applied computing → Education; Software and its engineering → General programming languages

Keywords and phrases Education, STEAM Education, Programming Education, Teaching Methodologies, Innovative Approaches, Mapping, Systematic Literature Review

Digital Object Identifier 10.4230/OASICS.ICPEEC.2024.8

Acknowledgements A special acknowledgment is extended to the reviewers, for their diligent analysis, which has greatly contributed to enhancing not only this work but also the more comprehensive project currently under development.

1 Introduction

Traditional education has experienced several challenges over the years. With the massification of new technologies and the transition to the digital age, there have been many challenges for the education sector to capture the attention and motivate new generations, who are increasingly in need of digital contact and simplification of long and time-consuming procedures. The majority of today's university students belong to generation Z [3, 16], a group strongly characterized by the presence of the internet and who have a greater command of technology than previous generations, with whom they are closely linked [14]. They are fans of practicality, independent and self-taught, digital natives [14], always connected, a hyper-cognitive generation with the ability to experience several realities at the same time, although their attention span is very short as they are used to get quick answers. In the academic environment, these students are looking for an experience as similar as their

¹ Corresponding author



personal environment, with high interactivity and fluidity of processes, combined with new technologies. They prefer to learn in a more autonomous and flexible way, where they are allowed to experiment without there always being an associated punitive nature [19, 13].

The overall difficulty experienced in teaching is mostly related to the disconnection between traditional teaching and the needs of the modern world. In recent decades, there have been few real changes in teaching methodologies and approaches, which remain deeply rooted in the educational system, which has not been able to evolve at the same vertiginous speed as today's world and, with it, students. Consequently, the stagnation of teaching methodologies has caused strong demotivation and hindered the full learning of 21st century students. Often based on unidirectional methods and the memorization of information [4], traditional teaching thus faces the challenge of combining with new technologies to compete for the attention of the younger generations, by seeking to simplify procedures and create more dynamic contexts adapted to the needs of students in the modern world [7].

The new technologies have proved to be the greatest allies in the renewal of teaching, particularly in the STEAM fields (acronym for Science, Technology, Engineering, Arts and Mathematics), by enabling the creation of differentiated and innovative teaching strategies. They provide greater personalization, flexibility and adaptation to the individual needs of students. With digital platforms and educational applications, teaching can be more flexible and will allow students to progress at their own pace and have access to additional resources to help them deepen their knowledge [9].

Programming teaching is considered by Caspersen and Bennedsen [2] to be one of the seven great challenges of computer education. There are various theories around the factors considered essential in effective education, such as providing feedback, organizing tasks by level of complexity, creating personalized content [5] or offering real-time student support [20]. Over time, there has been a perception among students that the methodologies used in traditional teaching were useless and boring, and teachers have tried to introduce new pedagogical approaches to reverse this impression. Despite these initiatives, students maintain a general feeling of demotivation [5].

In view of the challenges faced by traditional programming teaching, the main aim of this work is to carry out a Systematic Literature Review (SLR), mapping recent studies, to identify innovative methodologies and pedagogical approaches that have been recommended, analyzed and developed to improve programming teaching and that allow students to be truly involved in learning and overcome the real teaching difficulties.

This article is organized into four sections. The first section introduces the topic and defines the study's objectives. The second section outlines the research methodology, including the formulation of the research question, the chosen research strategy, and the results of the literature review, which maps both quantitative studies and catalogued data to provide an overview of the current state of the art and scientific contributions related to innovative educational methodologies or approaches. The third section analyzes the results and includes a proposal for classifying the most researched methodologies based on their characteristics. Finally, the fourth section presents the study's conclusions and discusses future work, including recommendations for further research.

2 Methodology

In line with the general objective described and the motivations associated with the development of this research, a Systematic Literature Review (SLR) was conducted to identify and evaluate scientific production relevant to the topic under analysis and to synthesize existing

research, providing essential information for further studies. This work follows the guidelines of Kitchenham [10], which emphasize the importance and advantages of SLRs and establish a clear, easily reproducible, and highly efficient procedure for conducting them [11, 21].

Research question

It is essential to understand the distinction and interconnection between different levels of pedagogical instruments: educational approaches, teaching methodologies, techniques and tools, and technological applications [6]. Grasping these interconnected concepts is crucial for developing pedagogical practices that are both theoretically sound and practically effective. However, for this research, given the common use of the term “methodology” in educational contexts (where it encompasses any instrument related to pedagogical practices), the terms “methodologies” and “approaches” are employed broadly, without differentiation between the various pedagogical instruments.

The following research question (RQ) was formulated for this research:

RQ: What innovative methodologies and pedagogical approaches have been identified, tested and developed to improve programming education?

Search strategy

Considering the scale of the subject under analysis, a different approach to defining the search string was considered. A strategy of identifying relevant or related keywords was chosen through a bibliometric analysis in VOSviewer. This analysis facilitated the study of the word network associated with the main terms and allowed for the refinement of search filters in a more rigorous way. Therefore, a broader search was conducted in the Scopus database, with the following string:

TITLE-ABS-KEY ((programming OR stem OR steam) AND (education OR teach* OR learn* OR classroom OR school) AND (methodology))*

The file containing the 10744 results was exported, including information on the abstracts and keywords. Words with at least 35 occurrences were considered in VOSviewer. Upon analyzing the word network obtained, it became evident that additional relevant or related words needed to be included, while certain terms like “covid” and “data mining” needed to be excluded, since they largely hidden the real results that were intended to be achieved.

The base search string was readjusted as follows:

TITLE-ABS-KEY ((new OR novel OR innovat*) PRE/0 (method* OR approach*) AND (programming OR stem OR steam) AND (education* OR teach* OR learn* OR classroom OR school OR pedagog*))*

This search string includes terms such as “method”, “methodology”, or “approach” because these words are commonly used to refer to methodologies. Similarly, terms like “education”, “teaching”, “learning”, “classroom”, “school”, or “pedagogy” were included, as any of these can denote learning environments. The search targeted programming or STEAM disciplines. Most terms in the search string carry the wildcard operator to encompass diverse variations of each term (e.g., “education” or “educational” for the term “education*”). Regarding the word “classroom”, it was decided to use the complete word instead of the broader term (“class*”), as the latter returned too many articles related to programming classes. The proximity operator ‘PRE/0’ was employed between the terms associated with “innovative” and those associated with “methodology” to ensure these words appeared sequentially and in this exact order.

8:4 Mapping Innovative Pedagogies in STEAM and Programming Education

The research exclusion criteria were directly applied to the terms intended for exclusion, and the following condition was added to the base string:

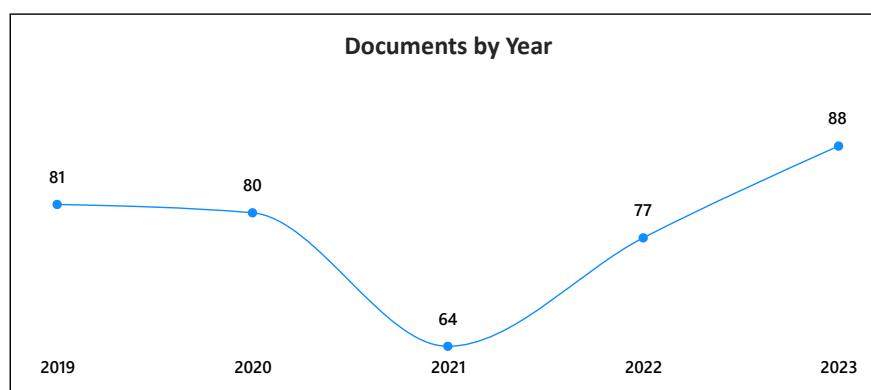
AND NOT ("covid-19" OR "data-mining" OR "neural network" OR "machine-learning" OR "deep-learning" OR reinforce* OR algorithm* OR genetic OR clustering OR classification OR optimization* OR graph* OR cybersecurity))*

The research inclusion criteria were also incorporated: documents published in the last 5 years, restricted to articles and conference papers in the fields of Computer Science and Engineering, written in English or Portuguese. These criteria were applied after the exclusion criteria:

AND PUBYEAR > 2018 AND PUBYEAR < 2024 AND (LIMIT-TO (DOCTYPE, "ar") OR LIMIT-TO (DOCTYPE, "cp")) AND (LIMIT-TO (SUBJAREA, "COMP") OR LIMIT-TO (SUBJAREA, "ENGI")) AND (LIMIT-TO (LANGUAGE, "English") OR LIMIT-TO (LANGUAGE, "Portuguese"))

Results

The search string was used to query Scopus database in its "advanced search" mode, resulting in 390 documents. A new analysis of the word network visualized in VOSviewer confirmed the accuracy of our search targeting. Upon analyzing the documents retrieved from the Scopus search, it was observed that 2023 had the highest scientific production on this topic, with 88 papers published, compared to 64 publications in 2021, indicating a noticeable upward trend in the exploration of innovative teaching methodologies (Figure 1).



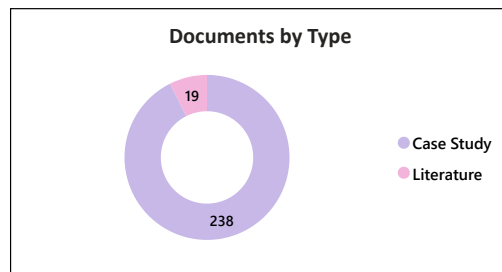
■ **Figure 1** Documents by year.

The three areas in which this research was most concentrated were Computer Science, Engineering and Social Sciences. Leading contributors to scientific production in this field included the United States, China, and Germany, with Portugal ranking eleventh based on 13 articles published in the last five years out of a total of 390 documents. Of these, 119 were scientific articles, and the remaining 271 were conference papers.

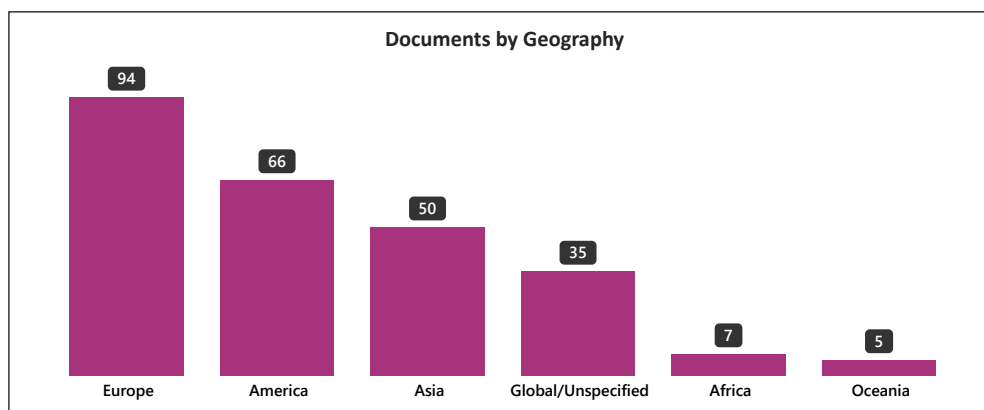
In the first analysis of the 390 documents, 9 duplicate articles were identified and excluded. The analysis of the remaining 381 documents was based on the title, author keywords and abstract fields, resulting in the exclusion of 124 documents deemed outside the scope of the research. These documents covered topics such as oil well exploration, medicine, sustainability, labor studies, and more specific topics like artificial intelligence and learning models, industrial

robotics, application programming interfaces (APIs), optimization and aerospace programs, which were not considered teaching methodologies. This selection process yielded a first final set of 257 papers.

The documents selected for a more in-depth analysis were categorized based on the following criteria: type of research (literature review or case study), geographical scope (country and continent; where not specified, author affiliation was used to determine geographical context), educational setting (educational level where the study was conducted), curricular area associated with each study, primary and additional subjects, and whether these subjects could be considered as approaches or methodologies for teaching programming. The results of this categorization are presented in Figures 2, 3, and 4.



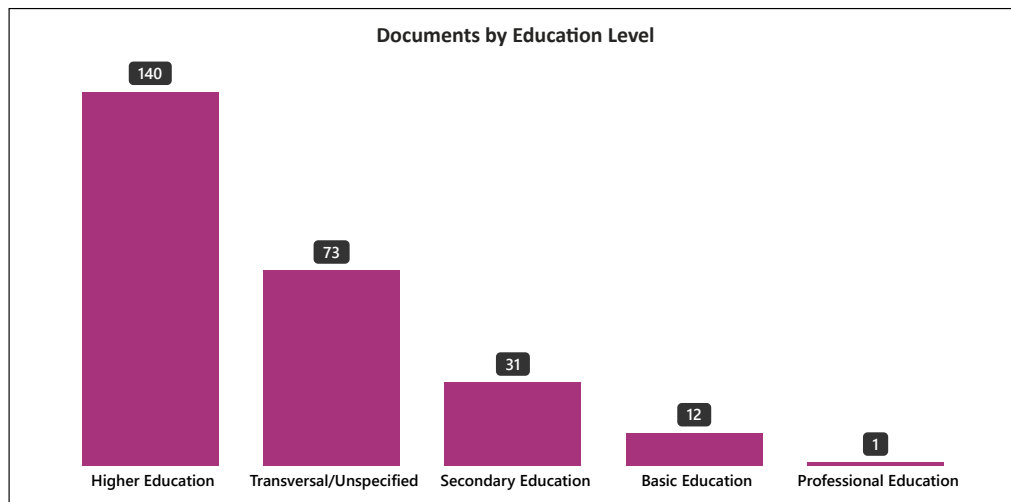
■ **Figure 2** Documents by type.



■ **Figure 3** Documents by geography.

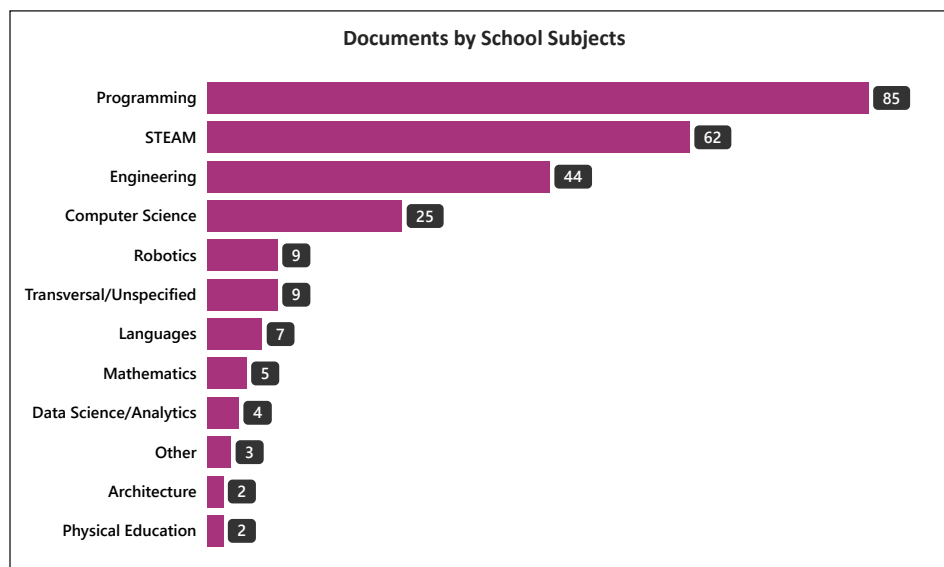
It should be noted that the three countries with the highest number of studies were the United States with 54, Spain with 14, and Germany with 10. Portugal conducted 5 studies in this research.

8:6 Mapping Innovative Pedagogies in STEAM and Programming Education



■ **Figure 4** Documents by educational level.

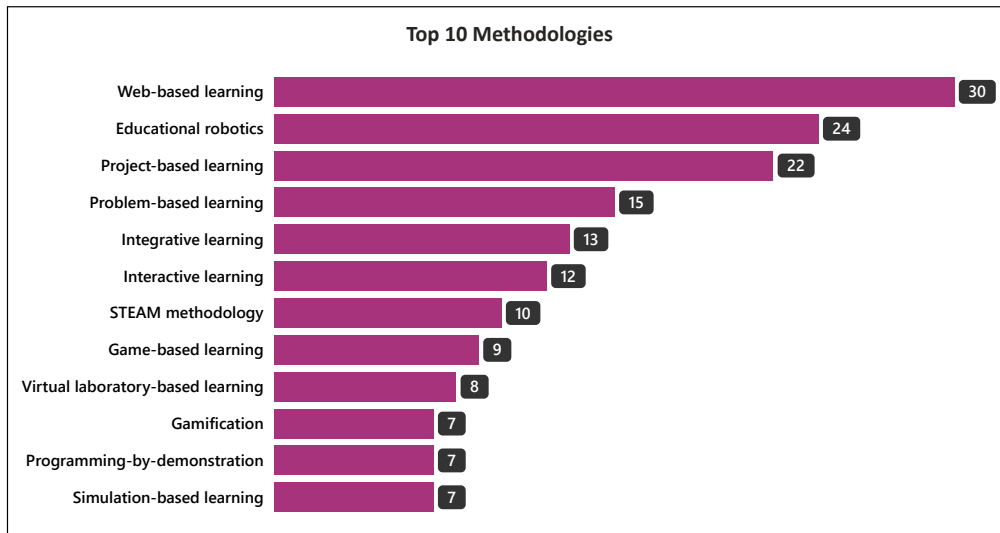
The information regarding the school subjects is presented in Figure 5.



■ **Figure 5** Documents by school subjects.

Considering the main themes of the analyzed studies, while all may be relevant for implementing significant improvements in the education system, it is significant to determine whether each identified theme can attend as an approach or methodology in programming teaching. Among the subjects identified in the 257 publications analyzed, 67 were deemed unsuitable for implementation as innovative approaches to programming education and were excluded. The five core subjects most frequently mentioned in these publications were accessible/inclusive education, learning analytics, teacher development, ethics, and human behavior.

The remaining 190 publications addressed core or additional subjects related to potential teaching methodologies, techniques, or approaches. Figure 6 presents the most frequently mentioned topics, based on the total number of occurrences across both core and additional subjects.



■ **Figure 6** Top 10 methodologies caption*.

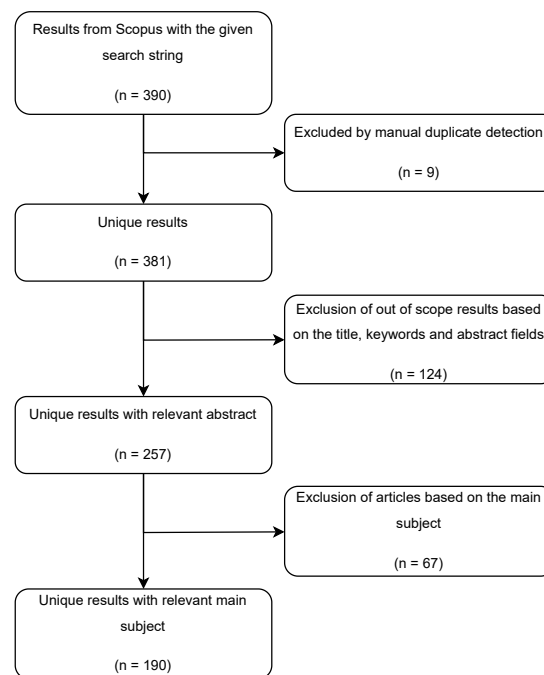
* The figure displays twelve results, as the last three entries record the same number of occurrences.

There is a clear trend towards studying “web-based learning” pedagogical approaches, which include the use of web-based platforms as educational resources. Some familiar terms, such as “project-based learning” (which involves the development of projects as the basis of learning), “problem-based learning” (where students learn by solving real-world problems), or even the “STEAM methodology” (an integrated learning approach that requires an intentional linking of rules, assessments, and the design and implementation of lessons between two or more STEAM subjects to be taught, assessed in, and through each other) continue to be investigated as pedagogical approaches that can create significant changes in teaching.

The synergy between curriculum areas is particularly noteworthy, especially with the integration of “educational robotics” to enhance programming education. Equally significant are methodologies related to the use of games in the educational context (“gamification” or “game-based learning”), which are increasingly studied as innovative teaching approaches. Key terms such as “outcome-based learning”, “augmented reality”, “block-based programming”, “educational Internet of Things” (IoT), “intelligent tutoring systems” (ITS), “mobile-based learning”, and “virtual reality”, along with other less frequently mentioned terms, were also identified.

A special note on the following concepts: “agile teaching”, an educational approach that applies agile principles widely used in industry, identified in papers [8, 18, 1]; “educational escape rooms” (EER or DEER) [12, 17], an emerging concept linked with gamified or game-based learning methodologies that introduces a potentially innovative teaching approach known as “out-of-the-box learning”; and “work-integrated learning”, which incorporates work-based training as an educational approach [15].

Figure 7 illustrates the flowchart used for selecting the relevant articles for the SLR.



■ **Figure 7** SLR workflow.

3 Discussion

The results obtained from literature mapping have enabled us to draw initial conclusions regarding recent efforts focused on analyzing specific methodologies. Figure 6 illustrates the methodologies or approaches identified most frequently in the studies analyzed.

Each methodology possesses unique characteristics, ranging from its objectives and required resources or knowledge, to the time-frame for effective implementation, the additional efforts educators need to invest for success, and its anticipated impact on students, among other factors. These characteristics yield both advantages and disadvantages that must be carefully assessed, as they significantly influence the choice of methodology to adopt.

A classification proposal of the main methodologies identified in Figure 6 is presented, considering the characteristics that emerged from reading and analyzing the articles referring to them. No additional studies have been conducted on specific approaches to classifying the use of teaching methodologies. For this classification proposal, three dimensions were considered: resources, implementation and receptiveness.

In the resources dimension, situations requiring additional infrastructures (technological or otherwise) not typically found in standard educational settings were considered. This dimension focuses on analyzing the potential impacts of costs (for developing or acquiring these infrastructures) and support (such as the need for a specialized infrastructure support team).

In the implementation dimension, which examines the actual application of the identified methodology in real educational contexts, factors of difficulty and time were considered. Difficulty refers to the level of challenge in implementing the methodology, such as interdis-

ciplinary requirements, collaborative efforts among teachers, or integration difficulties into traditional curricula. Time indicates the expected implementation duration, whether the methodology can be implemented immediately or requires a longer timeframe.

Lastly, the receptiveness dimension assesses the acceptance of the innovative methodology, considering both the educator pillar (which examines additional challenges or those that require greater efforts from teachers) and the student pillar (which reflects the motivation with which students might embrace this methodology, particularly if it is deemed innovative, disruptive, and aligns with their educational expectations).

For each of the six criteria mentioned, a scale of 1 to 5 was utilized, with 1 indicating “very poor” and 5 indicating “excellent”. Figure 8 illustrates the ratings assigned to each dimension and provides an overall score for each methodology.

Methodology	Resources		Implementation		Receptiveness		Overall Score
	Cost	Support	Difficulty	Time	Educator	Student	
Web-based learning	3,00	3,00	4,00	4,00	3,00	4,00	3,50
Educational robotics	2,00	2,00	3,00	3,00	3,00	5,00	3,00
Project-based learning	4,00	4,00	3,00	4,00	3,00	4,00	3,67
Problem-based learning	5,00	5,00	3,00	5,00	4,00	3,00	4,17
Integrative learning	3,00	3,00	2,00	3,00	3,00	5,00	3,17
Interactive learning	4,00	5,00	4,00	5,00	3,00	3,00	4,00
STEAM methodology	2,00	2,00	2,00	2,00	3,00	5,00	2,67
Game-based learning	3,00	3,00	3,00	4,00	4,00	5,00	3,67
Virtual-laboratory based learning	1,00	2,00	3,00	3,00	3,00	5,00	2,83
Gamification	3,00	3,00	3,00	4,00	4,00	5,00	3,67
Programming-by-demonstration	4,00	4,00	3,00	4,00	3,00	4,00	3,67
Simulation-based learning	2,00	2,00	3,00	3,00	3,00	5,00	3,00

■ **Figure 8** Top 10 methodologies classification.

Through the analysis of the overall classification, we can observe that “problem-based learning” and “interactive learning” are the two methodologies with the highest scores. It is noteworthy that these methodologies do not require additional resources in such an indispensable manner, and their ratings in this dimension are higher compared to other methodologies, which influenced the final score. They are also methodologies that can be implemented in less time and do not involve excessive implementation difficulties. However, they are the most penalized in terms of receptiveness, considering, notably, their potentially lower attractiveness to students.

On the other hand, upon analyzing from the student’s perspective, the methodologies identified with the highest potential receptiveness and overall score were “gamification” and “game-based learning”, followed by “integrative learning”. The methodologies “educational robotics”, “STEAM methodology”, “virtual-laboratory based learning”, and “simulation-based learning”, although they also score highly in student receptiveness, are the ones with more penalizing indicators of additional resources.

From the implementation perspective, considering difficulty and time factors, the methodologies that allow for more immediate implementation are “problem-based learning”, “web-based learning”, and “interactive learning”, contrasting with “integrative learning” and “STEAM methodology”, which pose greater implementation challenges.

8:10 Mapping Innovative Pedagogies in STEAM and Programming Education

The methodology identified with the lowest overall score was “STEAM methodology”, particularly impacted by its interdisciplinary nature, which involves the integrated teaching of two or more STEAM disciplines. It requires significant additional resources, both in terms of cost and support, and presents increased difficulties in implementing this interdisciplinary approach.

This concludes the main objective of this study with the answer to the RQ: “What innovative methodologies and pedagogical approaches have been identified, tested and developed to improve the teaching of programming?”.

While the primary objective of this research has been achieved, it is important to reflect on certain methodologies identified in the study. These methodologies, while not the most frequently mentioned, are considered to have significant potential for future use in programming education as emerging approaches. Figure 9 presents these methodologies, classified with the same parameters as the previous ones.

Methodology	Resources		Implementation		Receptiveness		Overall Score
	Cost	Support	Difficulty	Time	Educator	Student	
Agile teaching	4,00	3,00	3,00	4,00	3,00	4,00	3,50
Digital educational escape rooms	3,00	3,00	3,00	4,00	4,00	5,00	3,67
Work-integrated learning	5,00	4,00	3,00	4,00	4,00	5,00	4,17

■ **Figure 9** Emerging methodologies classification.

4 Conclusion

The problems affecting traditional teaching, especially in STEM (Science, Technology, Engineering and Mathematics), are a worldwide reality. This article conducts a systematic literature review and mapping study focused on innovative methodologies or approaches applied in the field of education, particularly in programming education. It is part of a larger, more comprehensive research project that is currently nearing completion. The primary objective was to identify methodologies that can support future research in this area.

The conclusions derived from this research provide significant contributions to the field of education. The mapping conducted enabled the aggregation of studies into geographic analyses, educational strengths, curricular areas, and themes addressed. In the quantitative analysis, the methodologies with the highest number of occurrences in the articles were mentioned, with particular emphasis on the top three: “web-based learning”, “educational robotics”, and “project-based learning”.

A classification proposal for these methodologies was also presented, considering their characteristics and weighting across the dimensions of resources, implementation, and receptiveness. From the analysis and classification of each methodology, it was concluded that “problem-based learning” and “interactive learning” are the two methodologies with the highest overall rating. They achieved consistent scores across all considered dimensions, highlighting the practicality of their implementation. The methodology with the lowest overall score was the “STEAM methodology”, primarily due to the challenges associated with implementing interdisciplinary learning across various STEAM disciplines.

This research also enabled a final reflection on emerging methodologies that, while not the most referenced, are believed to hold significant potential for future use in programming education. These include pedagogical approaches such as “agile teaching”, “educational

escape rooms” (EER) or “digital educational escape rooms” (DEER) and “work-integrated learning”. These methodologies should be considered for inclusion in new research efforts that incorporate practical applications.

Considering the more extensive research that is still ongoing, the aggregate analysis of this study has enabled the compilation of a comprehensive list of all teaching methodologies mentioned in the publications obtained through our search strategy. Each methodology will be accompanied by a brief description, chronological origin, and its main advantages and disadvantages. This list will be made available in due course and upon completion, both as a shared dataset and in a new publication.

The SLR presented in this article will also continue with a thorough reading of relevant articles to identify additional findings.

As a suggestion for future work, new dimensions could be investigated for the classification framework proposed, such as sustainability of the analyzed methodology or other relevant considerations. Further research could explore existing classification methodologies that might replace the proposed framework or potentially test the accuracy and effectiveness of the proposed classification in real-world environments.


References

- 1 Paulo André Pimenta Aragão and Rogéria Cristiane Gratão de Souza. Scrum xperience: A new approach for agile teaching. In Marcelo de Almeida Maia, Fabiano A. Dorça, Rafael Dias Araújo, Christina von Flach, Elisa Yumi Nakagawa, and Edna Dias Canedo, editors, *SBES 2022: XXXVI Brazilian Symposium on Software Engineering, Virtual Event Brazil, October 5 - 7, 2022*, pages 134–142, New York, NY, USA, October 2022. ACM. doi:10.1145/3555228.3555255.
- 2 Michael E. Caspersen and Jens Bennedsen. Instructional design of a programming course: a learning theoretic approach. In Richard J. Anderson, Sally Fincher, and Mark Guzdial, editors, *International Computing Education Research Workshop, ICER '07, Atlanta, GA, USA, September 15-16, 2007*, pages 111–122, New York, NY, USA, September 2007. ACM. doi:10.1145/1288580.1288595.
- 3 I Daukseviciute. Unlocking the full potential of digital native learners. *Henley Business School*, 2016.
- 4 Edward L. Deci, Robert J. Vallerand, Luc G. Pelletier, and Richard M. Ryan. Motivation and Education: The Self-Determination Perspective. *Educational Psychologist*, 26(3-4):325–346, June 1991. doi:10.1080/00461520.1991.9653137.
- 5 Darina Dicheva, Christo Dichev, Gennady Agre, and Galia 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.
- 6 D. Randy Garrison. *E-Learning in the 21st Century*. Routledge, March 2011. doi:10.4324/9780203838761.
- 7 Re’Shanda Grace-Bridges. Generation Z Goes to College. *Journal of College Orientation, Transition, and Retention*, 25(1), January 2019. doi:10.24926/jcotr.v25i1.2919.
- 8 Hélia Guerra, Luís Mendes Gomes, and Alberto Cardoso. Agile approach to a cs2-based course using the jupyter notebook in lab classes. In *2019 5th Experiment International Conference (exp.at'19), Funchal (Madeira Island), Portugal, June 12-14, 2019*, pages 177–182. IEEE, June 2019. doi:10.1109/EXPAT.2019.8876536.
- 9 Louise Hainline, Michael Gaines, Cheryl Long Feather, Elaine Padilla, and Esther Terry. Changing students, faculty, and institutions in the twenty-first century. *Peer Review*, 12(3):7–11, 2010.
- 10 Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

8:12 Mapping Innovative Pedagogies in STEAM and Programming Education

- 11 Barbara Kitchenham, Stuart Charters, et al. Guidelines for performing systematic literature reviews in software engineering, 2007.
- 12 Tilman Michaeli and Ralf Romeike. Investigating students' preexisting debugging traits: A real world escape room study. In Nick Falkner and Otto Seppälä, editors, *Koli Calling '20: 20th Koli Calling International Conference on Computing Education Research, Koli, Finland, November 19-22, 2020*, pages 15:1–15:10, New York, NY, USA, November 2020. ACM. doi:10.1145/3428029.3428044.
- 13 Diana G. Oblinger. The Next Generation of Educational Engagement. *Journal of Interactive Media in Education*, 2004(1):10, May 2004. doi:10.5334/2004-8-oblinger.
- 14 Marc Prensky. Digital Natives, Digital Immigrants Part 2: Do They Really Think Differently? *On the Horizon*, 9(6):1–6, January 2001. doi:10.1108/10748120110424843.
- 15 Lisa Romkey, Daniel Munro, Virginia Hall, and Tracy Ross. Evaluation of a Work-Integrated Learning Program for Undergraduate STEM Outreach Instructors. In *2023 ASEE Annual Conference & Exposition Proceedings*. ASEE Conferences, 2023. doi:10.18260/1-2--43464.
- 16 Darla Rothman. A tsunami of learners called generation z, 2016.
- 17 Tatjana Sidekerskienė and Robertas Damaševičius. Out-of-the-Box Learning: Digital Escape Rooms as a Metaphor for Breaking Down Barriers in STEM Education. *Sustainability*, 15(9):7393, April 2023. doi:10.3390/su15097393.
- 18 Rainer Telesko, Maja Spahic-Bogdanovic, Knut Hinkelmann, and Charuta Pande. A new approach for teaching programming: Model-based agile programming (MBAD). In *The 8th International Conference on Information and Education Innovations, ICIEI 2023, Manchester, United Kingdom, April 13-15, 2023*, pages 13–18, New York, NY, USA, April 2023. ACM. doi:10.1145/3594441.3594445.
- 19 Jean M Twenge. *iGen: Why today's super-connected kids are growing up less rebellious, more tolerant, less happy—and completely unprepared for adulthood—and what that means for the rest of us*. Simon and Schuster, 2017.
- 20 Elena Verdú, Luisa M. Regueras, María J. Verdú, José P. Leal, Juan P. de Castro, and Ricardo Queirós. A distributed system for learning programming on-line. *Computers & Education*, 58(1):1–10, January 2012. doi:10.1016/j.compedu.2011.08.015.
- 21 Jane Webster and Richard T Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS quarterly*, pages xiii–xxiii, 2002.

Teaching Programming Courses with Digital Educational Escape Rooms (DEER): A Conceptual Proposal Conducive to Learning by Trial and Error

Antonio Trigo¹ ✉ 

Polytechnic University of Coimbra, Portugal
CEOS.PP, ISCAP, Polytechnic of Porto, Portugal

Margarida Antunes ✉ 

Polytechnic University of Coimbra, Portugal

Abstract

In the field of programming education, the advent of new technologies such as ChatGPT has reshaped the landscape, challenging traditional methods and requiring new approaches to engage students effectively. Conventional teaching techniques find it hard to compete in a world where digital requests with instant feedback are plentiful. This shift emphasises the importance of innovative strategies, such as educational digital escape rooms, in programming education. By taking advantage of immersive storytelling and interactive challenges, these digital environments captivate students' interest and facilitate active learning. Instead of passively consuming information, students have the ability to apply programming concepts in a dynamic and gamified environment, promoting a deeper understanding and retention of the concepts. This paper presents a first effort in a work in progress to build an educational platform based on the digital escape room concept to be used in the classroom (or at home) throughout the school term.

2012 ACM Subject Classification Applied computing → Interactive learning environments; Software and its engineering → General programming languages

Keywords and phrases Education, University, Programming, Serious Games, Escape Rooms, Gamification

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.9

Funding This research was funded by Portuguese national funds through FCT – Fundação para a Ciência e Tecnologia, under the project UIDB/05422/2020.

1 Introduction

In recent years, the ability of students to concentrate in the classroom, either listening or doing practical problems, has diminished considerably, and it has become increasingly difficult to convey the concepts that are essential to their learning, especially in the first years of bachelor's programmes. From an early age, they get used to having the answers almost immediately, thanks to electronic devices, not giving them the opportunity for real learning (reflection and critical thinking), worrying only about the results. The emergence of artificial intelligence (AI) tools such as ChatGPT has made the process even more difficult, as they feel they can solve everything with such a tool and that the knowledge is all there. Although this is partly true, they have overlooked the necessity of having someone, specifically the teacher, to pose the questions that need answers and to develop a structured plan for their learning. Thus, the inability to keep students engaged and focused during lessons makes it urgent to look for innovative and motivating alternatives. In this context, gamified

¹ Corresponding author

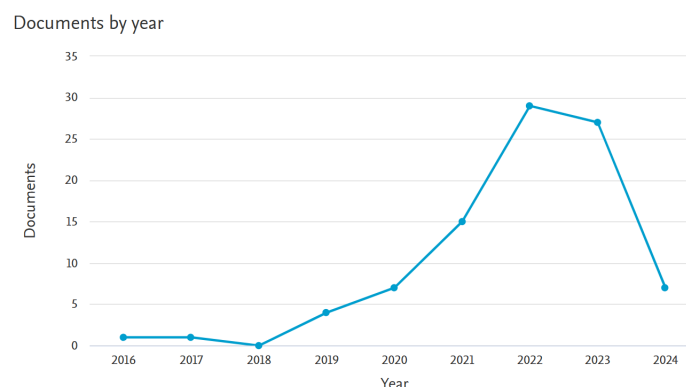


or game-based teaching, with an emphasis on “digital escape rooms”, has emerged as a promising tool for overcoming the challenges of contemporary education. This dynamic and immersive approach offers a stimulating and engaging learning environment, capable of capturing and maintaining the attention of students, even those with greater difficulty concentrating. Digital educational escape rooms (DEER) simulate challenging scenarios that require students to solve puzzles, use logical reasoning and collaborate as a team to achieve a common goal [6, 5, 7]. Through the inclusion of gaming elements, this methodology transforms learning into a fun and rewarding experience, awakening students’ interest in programming in a natural and organic way. Instead of feeling pressurised by deadlines and rigid objectives, students are encouraged to explore, experiment and make mistakes without fear, in a safe environment conducive to learning by trial and error. This autonomy and freedom of action contributes to the development of creativity, problem-solving and critical thinking, essential skills for success in the digital age [4, 8]. In addition, game-oriented methods using DEER promotes social interaction and teamwork, fundamental elements in the formation of complete citizens prepared for the challenges of the labour market. Students learn to collaborate with each other, communicate their ideas effectively and deal with different perspectives, essential skills for success in any professional field.

The challenges in creating DEER are, on the one hand, to ensure that they cover the learning and assessment of all the skills in a course unit, such as programming, and, on the other hand, the difficulty of constantly creating challenges/inventive games for students that don’t fall back on what they already know. In this paper we present a structuring proposal with examples for teaching programming through DEER.

2 Literature review

Using the search expression (*TITLE-ABS-KEY (escape PRE/0 room* AND (virtual OR digital) AND education*)) AND (LIMIT-TO (SUBJAREA , ‘COMP’) OR LIMIT-TO (SUBJAREA , ‘ENGI’))* in SCOPUS, limited to the areas of “Computer Science” and “Engineering”, 91 articles were obtained, as shown in Figure 1, where it is possible to verify the growing popularity of the topic in recent years.



■ **Figure 1** Search result in scopus database relative to DEER (april 2024).

From the list of articles retrieved from the search, we selected those accessible through our institution’s SCOPUS subscription that included concrete examples of DEER implementation, aiming to identify methodologies and technologies useful for the design and development of our proposal. The studies are presented in the order in which they appeared in SCOPUS, with a summary of the most relevant aspects for our proposal at the end.

The first work analysed presents the concepts involved in developing an escape room for teaching International Business [6]. The Escape Room creation involves defining learning objectives, developing a storyline, selecting game duration and language, creating challenges, designing clues, and testing the scenario. The authors based the development of their project on the AIDA model (Attention, Interest, Desire, Action). With regard to the technologies used to create the escape room, they used Genial.ly and Google Forms. According to the authors, the students were able to complete the game, with the students emphasising the need for teamwork in order to successfully complete the game.

As part of the German government's "Education in the Digital World" strategy, a DEER was created for the promotion and transmission of associated content in a playful way [4]. In this project, a digital serious game was designed and developed in a 2D environment, where various interactive elements, tasks and puzzles were incorporated into a digital educational Escape Room. The tests and evaluations carried out on the students afterwards showed a notable increase in their motivation and demonstrated their ability to solve tasks requiring the necessary skills.

In the work [8], the authors propose the use of DEER in STEAM (science, technology, engineering, arts and maths) education. They highlight the potential of DEER as a pedagogical tool to promote active learning and increase student engagement in STEAM education. To produce DEER, they propose a seven-step methodology: 1) Identify learning objectives: Understand what knowledge and skills students need to acquire; 2) Choose the platform: Select a digital escape room platform that suits your objectives; 3) Create a storyline and theme: Devise a narrative that engages the students and aligns with the objectives; 4) Design puzzles and challenges: Develop challenges that stimulate critical thinking without frustration; 5) Chart the learning path: Plan a sequence of puzzles that lead to achieving the learning objectives; 6) Incorporate feedback: Provide hints or clues to keep students engaged without giving away the solutions; 7) Test and Evaluate: Evaluate effectiveness with a test group to refine and improve the experience. In their work, they identified the following patterns for developing escape rooms: 1) Search and find: Players look for clues and objects within a room to solve puzzles and advance; 2) Lock and key: Players find keys or combinations to open doors or containers that hold essential items; 3) Observation and deduction: Players use visual and audio clues to solve puzzles through careful observation and deduction; 4) Sequence and order: Players must determine the correct sequence of actions or the order in which to solve the puzzle; 5) Communication and collaboration: Players work together, communicating effectively to solve the puzzles; and 6) Misleading clues: Misleading clues divert players' attention from the real solution, increasing complexity. With regard to technologies they identified the use of Breakout EDU, Escape Classroom, EdPuzzle, and Gdevelop. The authors found that 91.2% of students who responded to the questionnaire indicated that it enhanced their comprehension of the subject matter.

The work of [12] examines the use of DEER for teaching calculus as part of the engineering curriculum at a university in Spain, carried out entirely online. The Genial.ly platform was used to develop DEER. The authors conclude that incorporating DEER into calculus teaching in engineering proves to be an effective tool in engineering education. In particular, student satisfaction with the experience was remarkably high, with a significant demand for similar learning opportunities in the future. Another project that used the Genial.ly platform was [1], which designed DEER with the aim of preparing and revising the concepts and topics of the subject before the final exam. Not many details are given about the design of DEER, but the document does give examples of DEER. The students' opinions on the experience were mostly positive, so the activity can be used to motivate interaction in an online course.

9:4 Teaching Programming Courses with DEER: A Conceptual Proposal

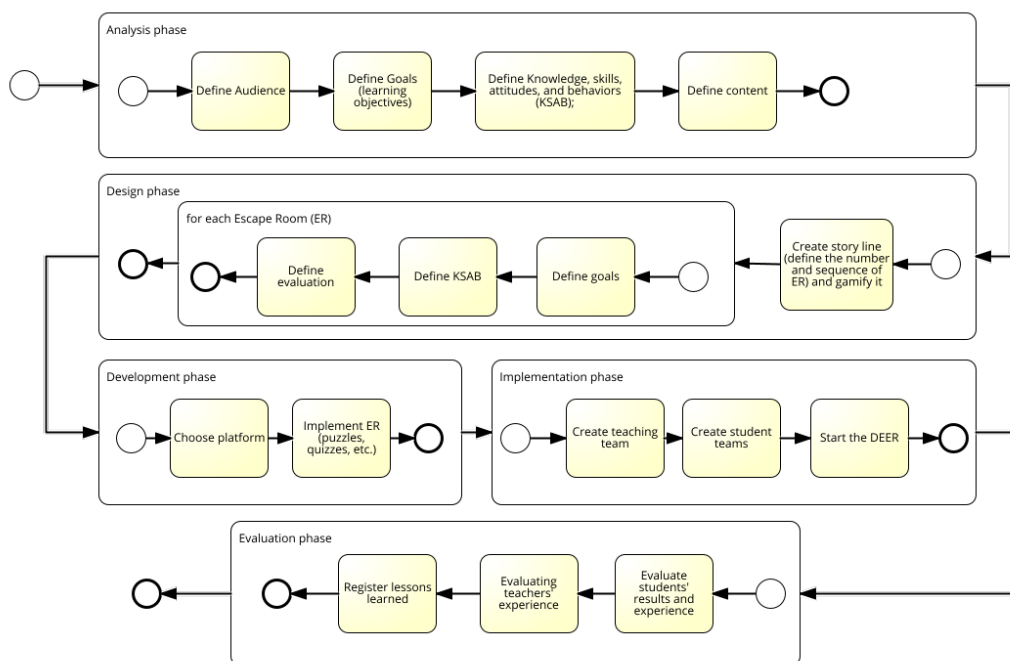
[7] work focuses on the development of DEER by 65 university students to be used by first graders. Forty-one DEER were created using the Design-Based Research methodology for different teaching areas: History, Grammar, English, Geography, Science and Maths. Although the authors have not provided a guide for designing DEER, they are very concerned with the learning objectives vs. the playfulness of the game (educational design vs. game design), in other words with designing effective DEER. The authors conclude that DEER provide innovative learning environments that cultivate thinking skills and social competences. Maintaining a balance between game design and learning design during development is crucial to maximising the relevance of DEER.

[3] present a platform developed with the Unity game engine for escape rooms based on virtual reality (VR) for teaching programming, which they consider to be a promising solution for improving students' results and motivation, allowing them to develop skills such as practical knowledge, creativity, and problem-solving skills. The fact that it's multiplayer encourages collaboration, enriching the learning experience. The platform allows the creation of escape rooms with different types of puzzle (hort answer, multiple choice, and jigsaw puzzles) for students to use, thus promoting problem-solving and critical thinking. The initial findings indicate that students exhibit a markedly positive attitude towards our VR-based DEER platform. Within the topic of VR-based DEER development, some works were found, such as [10, 9, 11], many of them more advanced than the work of [3], also including gamification concepts in the platforms [9, 11]. VR-based DEER are seen as the solution of the future for implementing DEER, since they allow interaction between the different players in the game itself and are therefore truly immersive, whereas many DEER proposals do not allow such interaction.

The work by [5] differs from the previous ones in that the authors focus mainly on the learning assessment process. As they point out in their own work and as we can also confirm, most of the studies we've come across focus a lot on the design and implementation of DEER but don't delve much into the assessment of learning, mostly presenting questionnaires answered by students about their level of satisfaction with DEER, lacking a more rigorous assessment of learning. Based on this assumption, the authors propose using learning analytics techniques to evaluate learning. In particular, they took advantage of new developments in sequencemining methods to analyse the temporal and sequential patterns of the actions performed by the students during the activity. In addition, they use clustering to identify different player profiles according to the sequential unfolding of the students' actions and analyse their acquisition of knowledge. The study was carried out as part of an undergraduate course on front-end programming at the Polytechnic University of Madrid. A DEER activity was implemented as an optional reinforcement exercise to solidify the fundamental concepts taught in a core segment of the course. These concepts encompassed the basics of HTML, CSS, and JavaScript, as well as more advanced technologies like React, Redux, and React Native. Students were paired up based on their own preferences, enabling them to benefit from collaborative learning and the advantages of pair programming. A total of 96 students participated in the study, organized into teams of two. When encountering difficulties with solving the various puzzles, students were permitted to request hints from a predefined set prepared by the instructors. However, to access these hints, students first had to earn them by successfully completing a brief online quiz covering the theoretical content of the course. This quiz served as a complement to the practical programming skills targeted by the escape room activity, enhancing the overall learning experience.

3 DEER Conceptual Proposal

The literature review showed that there are already many experiments underway in the use of DEER using different technologies. One of the most notable technologies is the trend toward using virtual reality to develop more advanced DEER, enabling players to interact with each other within the DEER environment. Many of these experiences are presented as small experiments to be used as a complement to teaching. We didn't find any experiments that lasted an entire school term, i.e. in which lessons were given with constant recourse to this type of platform, which is something we're trying to implement, given young people's receptiveness to this type of solution. In the literature review, it was possible to see the concern with defining the objectives/competences to be acquired by the students and, in the latest work presented [5], the concern with assessing learning. Figure 2 shows our proposed workflow for the implementation of DEER lasting one school term, which follows the traditional phases of the ADDIE model [2]: Analysis, Design, Development, Implementation and Evaluation.



■ **Figure 2** DEER conceptual proposal.

The model presented in Figure 2 outlines the stages for creating a DEER. The model does not include the technologies (virtual reality, generative AI, etc.), types of escape rooms, or how student learning control, including gamification should be implemented, as these depend on the choices of those implementing the DEER. Nonetheless, all the necessary stages for implementing a DEER are clearly outlined. Nonetheless, the model outlines all the stages and requisite elements, drawn from the literature review, for implementing effective and efficient DEER to bolster student learning. During the analysis phase, alongside delineating objectives, it is imperative to specify the Knowledge, Skills, Attitudes, and Behaviors (KSAB) students are expected to attain and how their acquisition will be assessed throughout the design and implementation phases. Furthermore, apart from acquiring the KSAB, it is crucial

to solicit feedback on students' experiences by conducting a satisfaction questionnaire or evaluation of the entire process upon its conclusion, a practice already commonplace in both physical and virtual curricular units.

4 Examples of escape rooms

In this section we present two possible challenges that could be included in the implementation of escape rooms, in this case for teaching Python programming.

Challenge 1: Title: "Numbers and codes"

- Puzzle Description:
 - Look at the Python script and execute the correct sequence of actions to unlock the door and escape the room.
 - To escape the room, you'll have to go through all the stages.
- Hints
 - Pay attention to the comments within the script. They provide clues on what to do at each stage.
 - Think logically about how Python functions and code blocks work.

■ Listing 1 Challenge 1 Code.

```
def stage_one():
    print("Stage One!")
    roman_num = "MMD"
    # Insert code
    print("The arabic number is: 2500")
    print("Congratulations! Proceed to stage two.")
def stage_two():
    print("Stage Two!")
    print("You're presented with a sequence of numbers.")
    sequence = [2, 1, 1, 2, 3, 5, 8, 13, 21, 34]
    # Insert code to perform operations on the sequence
    print("The transformed sequence: [4, 2, 2, 4, 6, 10, 16, 26, 42, 68]")
    print("Congratulations! Proceed to stage three.")
def final_three():
    print("Stage Three!")
    print("You face a formidable challenge: the Enigma code machine.")
    enigma_message = "JXHFUJ WTRR XZHHJXX! BJQQ ITSJ!"
    # Insert code to decrypt the Enigma message using Python
    print("The Enigma message is decrypted:" +
          " 'ESCAPE ROOM SUCCESS! WELL DONE!'")
    print("The door unlocks, and you've successfully escaped the room.")
def main():
    print("Python Codebreaker Challenge: Decrypt the Enigma")
    # Insert code to call up the different stages in order
```

Challenge 2: Title: "Pick a number?"

In this second example, only an image is presented because the idea is for the readers of this article, like the students, to try to find a solution to the challenge. If you need a hint, please contact the article's first author.



■ **Figure 3** Pick a number challenge.

5 Conclusion

The work presented in this paper endeavors to address a significant contemporary trend in education, namely the diminishing ability of students to concentrate in the classroom. This trend is attributed in part to the prevalent use of electronic devices and AI tools like ChatGPT, which provide immediate answers and discourage deeper reflection and critical thinking. Consequently, educators face increasing difficulty in effectively conveying essential concepts, particularly in the foundational years of bachelor's programs. In response to this challenge, this work explores the potential of game-based teaching, with a focus on DEER, as an innovative and engaging alternative, as can be seen from the literature carried out. DEER offers a dynamic and immersive learning environment that captures students' attention and fosters essential skills such as problem-solving and teamwork.

However, the implementation of DEER poses its own set of challenges, including ensuring comprehensive coverage of learning objectives and maintaining student engagement with varied and inventive challenges. Despite these obstacles, the paper presents a conceptual proposal, which needs to be tested and improved, for teaching programming through DEER drawing on existing literature and emphasizing the importance of clearly defined learning objectives, ongoing assessment, and feedback mechanisms.

Ultimately, the paper advocates for the integration of innovative teaching methodologies like DEER to meet the evolving demands of contemporary education.

References

- 1 Angeles Carolina Aguirre Acosta and Gabriela Espinola Carballo. The use of immersive tools in higher education: Escape rooms. In *Proceedings of the 2022 6th International Conference on Education and E-Learning, ICEEL 2022, Yamanashi, Japan, November 21-23, 2022*, pages 75–80. ACM, November 2022. doi:10.1145/3578837.3578848.
- 2 RK Branson, GT Rayner, JL Cox, JP Furman, FJ King, and WH Hannum. Interservice procedures for instructional systems development: Executive summary, phase i, phase ii, phase iii, phase iv, and phase v. *TRADOC pam 350-30, Ft. Monroe, VA: US Army Training and Doctrine Command*, 1975.
- 3 Ali Darejeh. Empowering education through eerp: A customizable educational vr escape room platform. In *IEEE International Symposium on Mixed and Augmented Reality Adjunct, ISMAR 2023, Sydney, Australia, October 16-20, 2023*, pages 764–766. IEEE, October 2023. doi:10.1109/ISMAR-Adjunct60411.2023.00166.

- 4 Sven Jacobs, Timo Hardebusch, Niklas Gerhard, Esther Franke, Henning Peters, and Steffen Jaschke. Promoting competences for the digital world by an educational escape room. In *2nd IEEE German Education Conference, GECon 2023, Berlin, Germany, August 2-4, 2023*, pages 1–4. IEEE, August 2023. doi:10.1109/GECon58119.2023.10295150.
- 5 Sonsoles López-Pernas, Mohammed Saqr, Aldo Gordillo, and Enrique Barra. A learning analytics perspective on educational escape rooms. *Interactive Learning Environments*, 31:6509–6525, December 2023. doi:10.1080/10494820.2022.2041045.
- 6 Juana María Padilla Piernas, María Concepción Parra Meroño, and María del Pilar Flores Asenjo. Escape rooms virtuales: una herramienta de gamificación para potenciar la motivación en la educación a distancia. *RIED-Revista Iberoamericana de Educación a Distancia*, 27:61–85, September 2023. doi:10.5944/ried.27.1.37685.
- 7 Manuela Repetto, Barbara Bruschi, and Melania Talarico. Key issues and pedagogical implications in the design of digital educational escape rooms. *Journal of e-Learning and Knowledge Society*, 19:67–74, 2023.
- 8 Tatjana Sidekerskienė and Robertas Damaševičius. Out-of-the-box learning: Digital escape rooms as a metaphor for breaking down barriers in stem education. *Sustainability*, 15:7393, April 2023. doi:10.3390/su15097393.
- 9 A. Staneva, T. Ivanova, K. Rasheva-Yordanova, and D. Borissova. Gamification in education: Building an escape room using vr technologies. In *46th MIPRO ICT and Electronics Convention, MIPRO 2023, Opatija, Croatia, May 22-26, 2023*, pages 678–683. IEEE, May 2023. doi:10.23919/MIPRO57284.2023.10159923.
- 10 Samira Yeasmin and Layla Abdulrahman Albabtain. Implementation of a virtual reality escape room game. In *2020 IEEE Graphics and Multimedia (GAME)*, pages 7–12. IEEE, November 2020. doi:10.1109/GAME50158.2020.9315039.
- 11 Piotr Zamojski, Norbert Barczyk, Marek Frankowski, Artur Cybulski, Konrad Nakonieczny, Marek Makowiec, and Magdalena Igras-Cybulska. Ohm vr: solving electronics escape room challenges on the roadmap towards gamified steam education. In *2023 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 532–535. IEEE, March 2023. doi:10.1109/VRW58643.2023.00117.
- 12 Ángel Alberto Magreñán, Cristina Jiménez, Lara Orcos, and Simón Roca. Teaching calculus in the first year of an engineering degree using a digital escape room in an online scenario. *Computer Applications in Engineering Education*, 31:676–695, May 2023. doi:10.1002/cae.22568.

Educational Program Visualizations Using Synthesized Execution Information

Rodrigo Mourato ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

André L. Santos ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL, Portugal

Abstract

Visualization is a powerful tool for explaining, understanding, and debugging computations. Over the years, several visualization tools have been developed for educational purposes. Most of these tools feed visualization engines using the raw program state data available provided by the debugger API. While this suffices in certain contexts, there are situations where additional relevant information could aid in building up more comprehensive visualizations. This paper presents two novel visualizations of Paddle, an educational programming environment based on synthesized program execution information. We generate execution traces and relevant program states through static and dynamic analysis of the execution data. The synthesized information captures program behaviors that facilitate the creation of comprehensive and rich visualizations involving arrays that depict position reads, writes, moves, and swaps.

2012 ACM Subject Classification Social and professional topics → Computer science education; Applied computing → Computer-assisted instruction

Keywords and phrases Introductory programming, visualization, comprehension

Digital Object Identifier 10.4230/OASISs.ICPEC.2024.10

1 Introduction

Programming educators commonly use illustrations to explain algorithms, in different forms, namely in their slides (possibly with animations), whiteboard explanations in the classroom, or on paper when addressing learners individually. Hence, program visualization tools appeal to many programming educators. However, a study [4] has shown that only about 20% of programming courses regularly use visualization tools and that almost half do not use them at all. The survey included responses from over 250 programming teachers and their students, who were asked about their use of visualization. Visualization tools are more often used by teachers working with younger students. The topics in which visualizations are most often used are introductory programming and data structures and algorithms.

Visualization tools are often integrated with debuggers or execution animators (e.g., [5, 1, 12, 2, 9]), where the tool renders the program state at each step. Except for PandionJ [9], these tools do not perform code analysis for capturing semantic aspects of the program (e.g., variable roles [8]) towards richer visualizations. The visualizations are often a mere alternative graphical representation of the information available in the call stack frames. Furthermore, debuggers do not provide the execution data regarding what happened before the program suspension at a breakpoint, making it difficult to illustrate the current program state in context. This leads to illustrations of program states that are less expressive than those hand-drawn by programming instructors [10], and the overall picture is lost through the debugging process.

In this paper, we describe automated program visualizations based on execution information synthesized from execution data, capturing traces and intents that are conventionally unavailable, such as expression-solving steps, array moves, and array swaps. Our main goal



© Rodrigo Mourato and André L. Santos;

licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 10; pp. 10:1–10:8

OpenAccess Series in Informatics



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

is to provide learners with a richer means to understand some programming basics and principles, such as recursion and expression resolution, and facilitate detailed observation of algorithmic behavior on arrays, including when errors occur. When using our tool, users execute programs normally, and only if needed, may switch views to gain more execution insights without requiring specialized tool knowledge.

We developed a web-based platform that supports a subset of Java, covering all the fundamental primitives for writing algorithms. We present two views with novel characteristics: (a) invocation tree with expression evaluation tracing; and (b) heap view with array history of reads and writes (capturing moves and swaps). These views aim to automate the hand-drawn illustrations of programming instructors using the results of a previous study [10]. In particular, the visualizations of array manipulations are novel concerning the state of the art, as we are unaware of any educational tool that illustrates moves and swaps explicitly (beyond depicting the raw program state step by step).

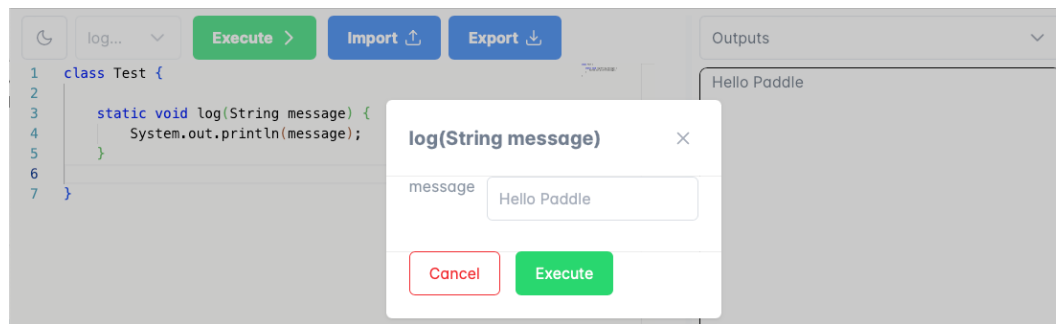
2 Related Work

Software visualization includes two broad areas, algorithm visualization, and program visualization, whereas the latter includes two further areas, visualization of static structures and visualization of runtime dynamics [11]. Algorithm visualization tools operate at a level of abstraction that is too high to be interesting for learning the basics of program execution. Our approach is focused on experimentation and debugging at an introductory level. Here we review tools that allow users to visualize the execution of their programs.

Jeliot [5] is a program animator supporting a subset of Java, where users play an animation of their programs. Visualizations are fine-grained, at the level of expression evaluation. Similarly, UUhistle [12] is a software tool to facilitate visual program simulation. It provides graphical elements that students can manipulate to indicate what happens during execution. The tool displays classes, functions, and operators that the program directly uses, enabling students to receive feedback on different types of errors, verify the accuracy of their answers, and obtain automated grading. These animation tools are useful to illustrate execution, but not practical when solving and debugging exercises because users have to go through the animation without traces of execution available. In our approach, we aim at an environment where programs are executed normally, i.e. in regular settings without any visualizations, and only if desired, behavior may be inspected in an aftermath manner through program traces that illustrate what happened.

Visualization tools are often integrated with debuggers within Integrated Development Environments (IDEs). JIVE is a declarative and visual debugging tool integrated with the Eclipse IDE [3]. jGRASP [2] is an IDE for visualizations to improve software comprehensibility through static and dynamic visualizations of programs. PandionJ [9] is an educational debugger for Java that combines static analysis and graphical visualization towards richer illustrations, such as depicting array iterator variables. While these tools are based on the standard Java debugger, in our present work we rely on a custom execution engine to collect more detailed information that is difficult to obtain otherwise (e.g., applying heavy program instrumentation).

Code Bubbles [7] is capable of displaying the debugging history as a UML sequence graph, the execution history of the current thread when it stops at a breakpoint, and information about a graphical user interface, including the widget hierarchy and the routines drawing at a selected pixel. Furthermore, it provides an interactive read-eval-print loop for the current context and a high-level view of the execution history in terms of threads, tasks, and



■ **Figure 1** Paddle environment: executing methods.

transactions. This view is generated automatically based on data collected during previous debugging runs. Code Bubbles targets a non-beginner audience, while our tool aims at the first stages of programming learning.

The SRec Visualization System [13] employs graphical representations to illustrate recursion trees. Each node corresponds to a recursive call composed of two halves: the upper half contains the parameter values of the call, while the lower half contains the invocation's result. WinHIPE [6] is an integrated development environment (IDE) for functional programming based on rewriting and visualization. It also includes a powerful visualization and animation system that automatically generates visualizations and animations as a side effect of program execution.

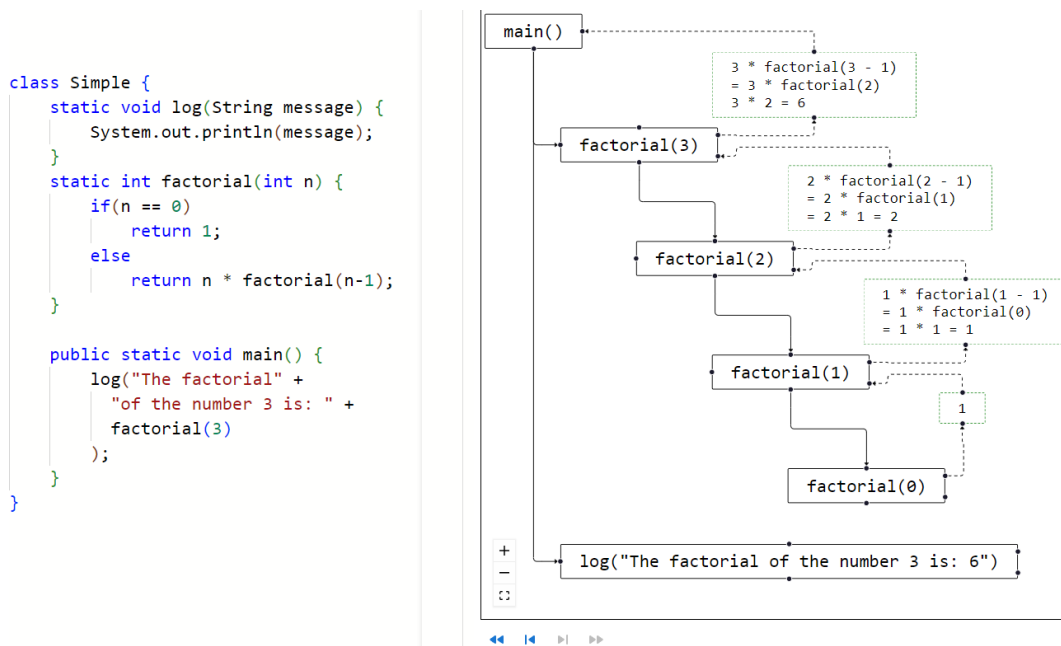
3 Paddle Environment

Paddle is an innovative educational programming environment providing visualizations that leverage synthesized program execution information. It generates representative execution traces and relevant program states through static and dynamic analysis of the execution data. The synthesized information captures diverse program behaviors to facilitate the creation of comprehensive and rich visualizations. The environment consists of a web application where the user can write code and obtain feedback about what happened during the execution as a trace illustration.

The user interface (UI) comprises two panels (see Figure 1): the left panel, where the user writes code and executes programs, and the right panel, where alternative visualization panels are presented. Figure 1 illustrates the elementary view for displaying console outputs. When clicking the “Execute” button, a dialog prompts the user to enter the values for each parameter, and the current code is sent to the server with the specified function and arguments. Afterward, the code result is returned to the web application, and the user may check the outputs and switch among the available visualization panels, which we detail next.

3.1 Invocation Tree View

Figure 2 presents a screenshot of the invocation tree view with the classic example of factorial calculation. Each node in the illustration represents one execution of a method, the solid edges represent invocations, and the dashed edges with the dashed nodes represent the return values of each invocation. If desired, the user may use the playback mode to go through each step, following the sequence of invocations. The related elements are selected in the code editor when clicking the view. When clicking an invocation node the function declaration is highlighted, whereas when clicking a value node the respective return expression is highlighted instead.



■ **Figure 2** Invocation tree view illustrating recursive calls (factorial calculation).

The main innovative feature of our view is the trace of expressions returned by the methods. In the example, the expression `3 * factorial(3 - 1)` is resolved to `3 * factorial(2)` and is finally resolved to `3 * 2`, which returns the final value of 6. This enables the user to understand the return value of each invocation and how it was calculated. This information is synthesized from execution data, and is not available when using debuggers (both educational and professional). For performance reasons, the total number of resolutions has a limit. Programming instructors often use similar illustrations to explain the execution of recursive calls [10].

3.2 Heap View

Figure 3 presents the heap view illustrating a function to check if an element is contained in an array. This view collects any array allocations performed in user code and renders its evolution through snapshots, from top to bottom. In this case, the array content remains the same because there are no side effects. The green background depicts that the highlighted position was read, whereas red denotes that a write was performed. In the illustration, we can observe that the last accessed position was the third one. The iterator variables for accessing array positions (*i* in the example) are depicted below the respective index (as in [9]). Programming instructors often use similar illustrations to explain computations that involve array iterations [10].

Figure 4 presents the heap view illustrating a procedure for left-shifting an array, exemplifying array writes. In the illustrations, a dashed arrow represents an array position move, that is, a value at one position is copied to another. This information is determined using a combination of static analysis and execution data.

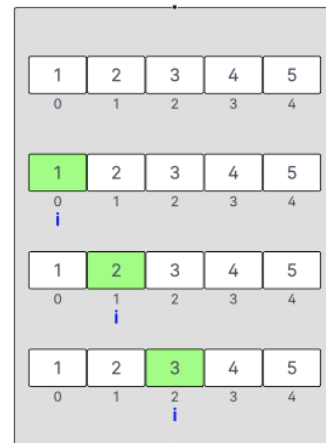
Figure 5 presents the heap view illustrating a procedure to reverse an array. The array was initialized with five elements and the reverse function was invoked, which internally invokes the function to swap two array elements given their indices. Special attention is paid


```

static boolean contains(int[] array, int n) {
    for(int i = 0; i < array.length; i++)
        if(array[i] == n)
            return true;
    return false;
}

static void main() {
    int[] a = {1, 2, 3, 4, 5};
    System.out.println(contains(a, 3));
}

```



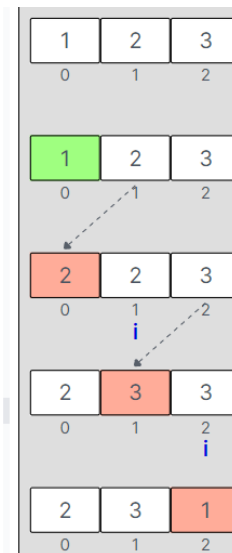
■ **Figure 3** Heap view illustrating array reads (check if element exists).

```

static void shift(int[] v) {
    int t = v[0];
    for(int i = 1; i < v.length; i++)
        v[i-1] = v[i];
    v[v.length-1] = t;
}

public static void main() {
    int[] array = {1, 2, 3};
    shift(array);
}

```



■ **Figure 4** Heap view illustrating array moves (left shift of array elements).

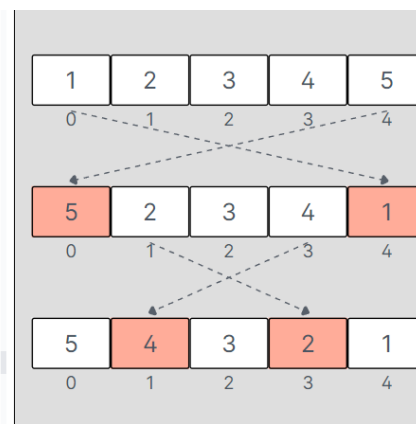
```

static void swap(int[] a, int i, int j) {
    int t = a[i];
    a[i] = a[j];
    a[j] = t;
}

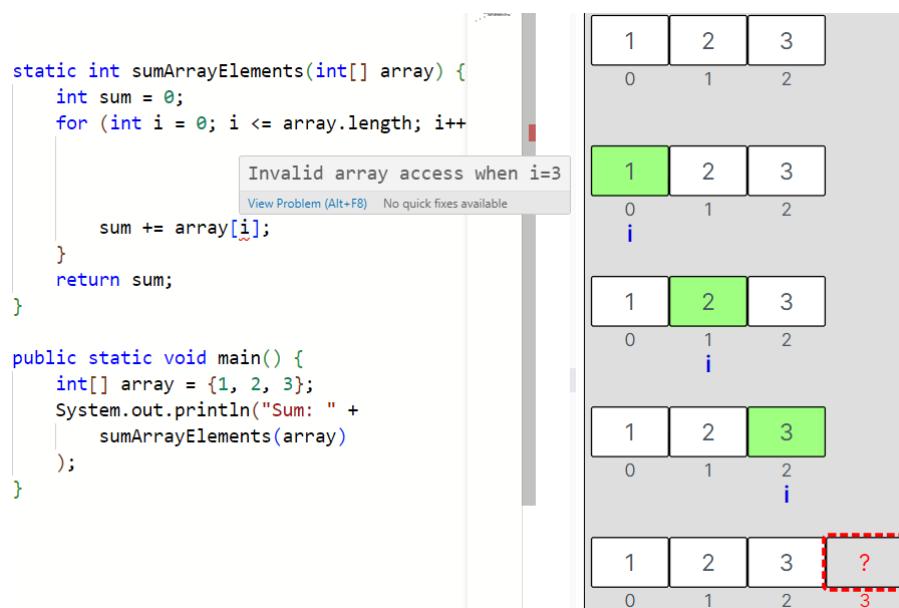
static void reverse(int[] a) {
    int n = a.length;
    for (int i = 0; i < n / 2; i++) {
        swap(a, i, n - i - 1);
    }
}

public static void main() {
    int[] array = {1, 2, 3, 4, 5};
    reverse(array);
}

```



■ **Figure 5** Heap view illustrating array swaps (reverse the array).



■ **Figure 6** Heap view illustrating an illegal access to an array position.

to array swaps – information synthesized from execution data. As in array moves, a dashed arrow represents a move. Since a swap consists of two moves that exchange the values of the positions, the corresponding arrows are depicted simultaneously.

If an array index out-of-bounds error occurs during execution, we illustrate the error in the view, as depicted in Figure 6. The expression that led to the invalid index is also marked with precision in the code. Recall that conventional support for this type of error typically consists of an error message that only includes the line number and invalid index (if multiple array accesses are in that line, the user must figure out which is causing the problem).

4 Implementation

The implementation of our prototype is based on a REST API, where program executions are performed, and a web-based frontend to display the results and visualizations. Ideally, the whole application could run on the browser, but we needed unavailable JavaScript libraries to execute the Java programs and synthesize the required information for the visualizations.

The backend was constructed using Spring Boot¹, a JVM-based framework that simplifies the development of standalone application servers. The API calls respond JSON messages holding the execution results, outputs, traces, etc, that are necessary for building the visualizations.

Program execution and analysis are performed using Strudel², a programming library comprising classes that model structured programming, providing a virtual machine capable of interpreting those models, simulating the call stack-based execution. This enables clients to observe every aspect of execution in detail, including errors, tracking variable values, loop iterations, call stack, and memory allocation. We developed execution listeners to gather the

¹ <https://spring.io/projects/spring-boot>

² <https://github.com/andre-santos-pt/strudel>

necessary information to render the views. Regarding the resolution of expressions, EvalEx³ was employed. EvalEx is a convenient expression evaluator for Java that enables the parsing and evaluation of expression strings.

The user interface was implemented using React⁴, a popular JavaScript library for user interface development. The Redux Toolkit⁵ was used for store management, providing utilities and abstractions to streamline common Redux tasks, such as creating actions, reducers, and store configuration. The code editor is provided by Microsoft Monaco⁶, a lightweight, browser-based, highly versatile code editor providing features such as syntax highlighting, code completion, and IntelliSense. Monaco is the engine behind the Visual Studio Code editing experience and can be embedded in Web applications to edit code directly in the browser. Finally, the visualizations were implemented using React Flow library⁷, a JavaScript library for developing interactive and visual flowcharts, diagrams, and graphs within React applications. It offers a flexible and customizable API to develop complex data visualization components, thereby enabling developers to incorporate drag-and-drop functionality, node-based layouts, and connection handling with relative ease. This library enabled the creation of custom nodes and edges, as illustrated in this paper's figures.

5 Conclusions and Future Work

Our prototype demonstrates that rich program visualizations can be obtained in a post-execution manner by making use of synthesized execution information. Our visualizations are inspired by illustrations often made by programming instructors (e.g., in slides, animations, or hand-drawn). In particular, the array manipulation illustrations are unavailable in other visualization tools supporting arbitrary user code, and without having to execute the program step-by-step (as when using a debugger). We argue that our views are a quick means to illustrate the execution of simple programs involving invocations and arrays, with minimal need to learn any particular tool features.

As future work, we plan to evaluate how programming instructors perceive the usefulness of our visualizations. Evaluating the tool from the perspective of programming beginners could also inform how easily and accurately they interpret the visualizations. Even if the visualizations have no expressive effect on novices working autonomously, they may serve as an aid to instructors when assisting learners in lab classes or remotely, sparing time that otherwise would be spent on figuring out what went wrong with the program execution and manually drawing illustrations for further explanations.

Regarding tool improvements, we plan to support objects in the heap view, which are important to illustrate elementary data structures such as linked lists and trees and to elaborate on the illustrations of errors (e.g., stack overflows). Furthermore, we believe that more interactivity between the views and the source code could improve the user experience, and we acknowledge that strategies to cope with large drawings are necessary for good usability.

³ <https://github.com/ezylang/EvalEx>

⁴ <https://react.dev>

⁵ <https://redux-toolkit.js.org>

⁶ <https://microsoft.github.io/monaco-editor>

⁷ <https://reactflow.dev>

References

- 1 G. Cattaneo, P. Faruolo, U.F. Petrillo, and G.F. Italiano. Jive: Java interactive software visualization environment. In *2004 IEEE Symposium on Visual Languages - Human Centric Computing*, pages 41–43, 2004. doi:10.1109/VLHCC.2004.34.
- 2 James Cross, Dean Hendrix, Larry Barowski, and David Umphress. Dynamic program visualizations: An experience report. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education, SIGCSE '14*, pages 609–614, New York, NY, USA, 2014. ACM. doi:10.1145/2538862.2538958.
- 3 Jeffrey K. Czyz and Bharat Jayaraman. Declarative and visual debugging in Eclipse. In *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange, eclipse '07*, pages 31–35, New York, NY, USA, 2007. ACM. doi:10.1145/1328279.1328286.
- 4 Essi Isohanni and Hannu-Matti Järvinen. Are visualization tools used in programming education?: By whom, how, why, and why not? In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research, Koli Calling '14*, pages 35–40, New York, NY, USA, 2014. ACM. doi:10.1145/2674683.2674688.
- 5 Ronit Ben-Bassat Levy, Mordechai Ben-Ari, and Pekka A. Uronen. The Jeliot 2000 program animation system. *Computers and Education*, 40(1):1–15, 2003.
- 6 Cristóbal Pareja-Flores, Jamie Urquiza-Fuentes, and J. Ángel Velázquez-Iturbide. WinHIPE: an ide for functional programming based on rewriting and visualization. *SIGPLAN Not.*, 42(3):14–23, March 2007. doi:10.1145/1273039.1273042.
- 7 Steven P. Reiss. The challenge of helping the programmer during debugging. In *2014 Second IEEE Working Conference on Software Visualization*, pages 112–116, 2014. doi:10.1109/VISSOFT.2014.27.
- 8 Jorma Sajaniemi. An empirical analysis of roles of variables in novice-level procedural programs. In *Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02), HCC '02*, pages 37–, Washington, DC, USA, 2002. IEEE Computer Society. URL: <http://dl.acm.org/citation.cfm?id=795687.797809>.
- 9 André L. Santos. Enhancing visualizations in pedagogical debuggers by leveraging on code analysis. In Mike Joy and Petri Ihantola, editors, *Proceedings of the 18th Koli Calling International Conference on Computing Education Research, Koli, Finland, November 22-25, 2018*, pages 11:1–11:9. ACM, 2018. doi:10.1145/3279720.3279732.
- 10 André L. Santos and Hugo Sousa. An exploratory study of how programming instructors illustrate variables and control flow. In *Proceedings of the 17th Koli Calling Conference on Computing Education Research, Koli, Finland, November 16-19, 2017*, pages 173–177, 2017. doi:10.1145/3141880.3141892.
- 11 Juha Sorva, Ville Karavirta, and Lauri Malmi. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education*, 13(4):15.1–15.64, 2013. doi:10.1145/2490822.
- 12 Juha Sorva and Teemu Sirkiä. Uuhistle: a software tool for visual program simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research, Koli Calling '10*, pages 49–54, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1930464.1930471.
- 13 J. Ángel Velázquez-Iturbide and Antonio Pérez-Carrasco. How to use the SRec visualization system in programming and algorithm courses. *ACM Inroads*, 7(3):42–49, August 2016. doi:10.1145/2948070.

Client-Side Gamification Engine for Enhanced Programming Learning

Ricardo Queirós ✉ 🏠 

School of Media Arts and Design & CRACS – INESC TEC, Polytechnic of Porto, Portugal

Robertas Damaševičius ✉ 

Department of Applied Informatics, Vytautas Magnus University, Vilnius, Lithuania

Rytis Maskeliūnas ✉ 

Centre of Real Time Computer Systems, Kaunas University of Technology, Lithuania

Jakub Swacha ✉ 🏠 

Department of IT in Management, University of Szczecin, Poland

Abstract

This study introduces the development of a client-based software layer within the FGPE project, aimed at enhancing the usability of the FGPE programming learning environment through client-side processing. The primary goal is to enable the evaluation of programming exercises and the application of gamification rules directly on the client-side, thereby facilitating offline functionality. This approach is particularly beneficial in regions with unreliable internet connectivity, as it allows continuous student interaction and feedback without the need for a constant server connection. The implementation promises to reduce server load significantly by shifting the evaluation workload to the client-side. This not only improves response times but also alleviates the burden on server resources, enhancing overall system efficiency. Two main strategies are explored: 1) caching the gamification service interface on the client-side, and 2) implementing a complete client-side gamification service that synchronizes with the server when online. Each approach is evaluated in terms of its impact on user experience, system performance, and potential security concerns. The findings suggest that while client-side processing offers considerable benefits in terms of scalability and user engagement, it also introduces challenges such as increased system complexity and potential data synchronization issues. The study concludes with recommendations for balancing these factors to optimize the design and implementation of client-based systems for educational environments.

2012 ACM Subject Classification Social and professional topics → Computer science education

Keywords and phrases Code generation, Computer Programming, Gamification

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.11

Funding This work is co-funded by the Erasmus+ Programme of the European Union within the project FGPEPlusPlus, with Agreement Number 2023-1-PL01-KA220-HED-000164696.

1 Introduction

Interactive and accessible learning platforms have become crucial in providing quality education to a geographically and economically diverse student population. Traditionally, such platforms rely heavily on constant server connectivity to function effectively, offering real-time feedback and interactive experiences that are central to modern pedagogical methodologies. However, this dependence on server-side processing presents significant challenges, particularly in areas with unstable internet access or limited server resources [32]. The evolution of client-side technologies, such as service workers and local storage, has introduced the potential for mitigating these issues by shifting some of the computational responsibilities from the server to the client. This shift not only promises to enhance the resilience of educational platforms against connectivity fluctuations but also aims to distribute the computational load more evenly, thus improving responsiveness and scalability [12]. The



© Ricardo Queirós, Robertas Damaševičius, Rytis Maskeliūnas, and Jakub Swacha; licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 11; pp. 11:1–11:12

OpenAccess Series in Informatics



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

advent of such technologies beckons a pivotal shift in the architecture of educational platforms, from a predominantly server-reliant model to a more decentralized approach [27, 7]. This transformation could revolutionize how educational services are delivered, ensuring more reliable and accessible learning experiences regardless of external network conditions.

The Framework for Gamified Programming Education (FGPE)¹ programming learning environment, thanks to the development of the FGPE Plus project² [21, 20], is currently provided as a Progressive Web Application, which means, once loaded, it could be used without sustained Internet connection. The problem is, for the students' solution of an exercise to be evaluated, and the relevant gamification rules triggered, there must be an active Internet connection at the time of submission. The goal of FGPE++³ is to remove this barrier by providing a way to assess the programming exercise solutions client-side. This will both greatly improve the FGPE platform usability in areas in which users experience Internet connection problems and will also reduce the load on university servers, allowing to limit their role to batch processing of the student progress data on synchronization, instead of running the full evaluation server-side, which with many concurrent students on university servers having limited resources caused noticeable delays and sometimes even stalls that ruined all user experience. Consequently, this paves the way for providing large-scale gamified programming courses with limited technical resources, and will support the sustained growth of the number of the FGPE ecosystem users. The envisaged client-side software layer for the assessment of gamified programming exercises and gamification rule processing will be capable of producing instant feedback to the users even without active connection with the server, caching data on user progress and achievements, and synchronizing students' progress cached on the client with the global state stored at the server.

This study seeks to explore this transition within the context of the FGPE platform, aiming to develop a client-based version that maintains high functionality offline while synchronizing with the server when connectivity permits. This approach is particularly pertinent for educational institutions with limited IT infrastructure, as it allows them to offer a high-quality, interactive programming education with reduced dependence on robust internet services.

The rest of the article is structured in four sections: the second section presents the base project of all this work. The next section, depicts the most used strategies for supporting offline features in client-server applications. The following two sections present the execution plan for the development of the client-side evaluation engine, identifying two approaches. Finally, the contributions of this article to the scientific community are presented as well as the future work.

2 The FGPE environment

Framework for Gamified Programming Education (FGPE) is an open, programming-language-agnostic set of exercise formats, exemplary exercises, and the supporting software [29]. The framework addresses the needs of both students (for a more engaging programming education) and teachers (for a customizable web platform for gamified teaching of programming). The framework has been developed by a consortium of five European higher-education institutions: University of Szczecin (Szczecin, Poland), INESC TEC (Porto, Portugal), Aalborg University

¹ <https://fgpe.usz.edu.pl/>

² <https://fgpeplus.usz.edu.pl/>

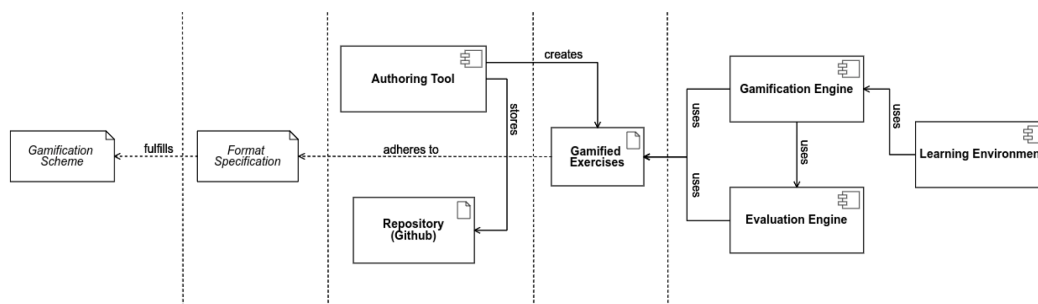
³ <https://fgpeplus2.usz.edu.pl/>

(Copenhagen, Denmark), University of Naples Parthenope (Naples, Italy), and Kaunas University of Technology (Kaunas, Lithuania), and financially supported from the Erasmus+ program⁴.

The key software components of the FGPE ecosystem include:

- the editor (FGPE AuthorKit) which gives the exercise creators the capability to design and implement both exercises and gamified scenarios with an intuitive wizard-like form-based user interface [25],
- gamification service (FGPE GS) that processes gamification rules and manages the overall game state [24],
- the interactive learning environment (FGPE PLE), which lets students access gamified exercises, solve them, and receive graded feedback, and lets teachers organize exercise sets, grant students access, and monitor their learning progress [21].

Figure 1 provides a comprehensive visualization of the FGPE architecture.



■ **Figure 1** FGPE Architecture.

Recently, FGPE platform has been enriched with an LTI-compliant interface allowing the FGPE-run exercises to be embedded in LMS-based courses, developed using any of the popular systems (e.g., Moodle or Open edX) [17].

To evaluate the solutions of the programming exercises submitted by students, the FGPE platform uses the reliable and secure Mooshak system [16]. As both the FGPE GS and Mooshak are run server-side, although the students can edit their code in FGPE PLE, which is currently provided as Progressive Web App, without the Internet access, they cannot submit it and get feedback until the connection is restored, which can be an issue for users traveling or staying in areas with poor Internet connectivity.

This defines the problem that the research presented in this paper aims to address.

3 Related work

This section reviews the existing literature and developments in the area of offline capabilities for educational platforms and the broader spectrum of tools that facilitate offline functionality in client-server applications. It sets the groundwork for understanding the technological and conceptual frameworks that our research builds upon.

⁴ <https://fgpeplus2.usz.edu.pl/>

3.1 Offline Strategies

The evolution from traditional client-server setups to architectures where significant functionalities are executed on the client-side has garnered considerable attention in research circles [15]. This paradigm shift holds particular significance in educational technology landscapes, where the necessity for continuous server interaction can pose accessibility and usability challenges, especially in areas with limited connectivity.

One pivotal approach within this realm revolves around leveraging service workers, which are scripts operating in the background on the client side, independent of the active web page [10]. These workers facilitate resource caching and streamline data synchronization processes, thus augmenting user experiences by enabling interactive engagements even in offline scenarios. Subsequently, they efficiently synchronize data with the server upon restoration of connectivity [23]. In addition to service workers, the adoption of local storage mechanisms and indexed databases such as IndexedDB stands out as another prominent strategy [2]. These technologies empower educational platforms to locally store substantial volumes of data, ranging from user progress metrics to interactive state information. By doing so, they alleviate the reliance on continuous data retrieval from the server, thereby enhancing system efficiency and responsiveness [13]. Finally, the integration of differential synchronization emerged as an interesting concept in this landscape [3]. This approach entails transmitting only the changes or differences between the client and server, rather than entire datasets. By minimizing data transfer requirements, it significantly contributes to optimizing system performance and bolstering application responsiveness [33].

3.2 Other Tools

In addition to strategies tailored explicitly for offline functionality, a plethora of tools and frameworks exist to support the development of client-heavy applications. Google's Workbox stands out among these, offering libraries and Node modules that streamline the management of service workers, making it more accessible and robust. Workbox boasts extensive features for precaching, runtime caching, and efficient data retrieval and storage strategies [1]. React, a JavaScript library renowned for building user interfaces, plays a crucial role in creating responsive and dynamic client-side applications. Its capability to manage state locally and render components based on user interactions makes it a preferred choice for developing educational platforms with minimal reliance on server-side rendering [31]. Frameworks such as Electron broaden the horizons by enabling the creation of native applications using web technologies. These applications can function as standalone desktop applications, eliminating the need for an active internet connection. By harnessing Electron, web-based educational platforms can seamlessly transition into fully functional desktop applications [28].

Collectively, these tools and strategies align with the overarching objective of bolstering the resilience and accessibility of educational platforms [11], especially in environments grappling with limited connectivity. Our study endeavors to leverage these advancements to propose a novel approach to managing offline functionalities within the FGPE platform, with the aim of delivering a seamless and uninterrupted learning experience.

4 The offline client-server optimization model

Bellow we explain optimization problem with five objective functions to maximize, representing usability, performance, server load, autonomy, and security. It defines decision variables and constraints related to two options, each with specific criteria and thresholds.

Through our **objective functions**, we aim to maximize the following criteria:

1. Usability (C_1)
2. Performance (C_2)
3. Server Load (C_3)
4. Autonomy (C_4)
5. Security (C_5)

Let us define **decision variables** used in our model:

- x_1 represent Option 1: Cached interface for Gamification Service
- x_2 represent Option 2: Client-side Gamification Service

Further we will define **constraints** for each option x_i , where $i = 1, 2$, related to:

- Client-side Complexity (D_1, D_6)
- Data Synchronization (D_2, D_7)
- Control (D_3, D_8)
- Duplication of Logic (D_4, D_9)
- Resource Requirements (D_5, D_{10})

We have chosen Non-dominated Sorting Genetic Algorithm II (NSGA-II [5]) to explore the solution space by maintaining a diverse population of non-dominated solutions and iteratively evolving it towards the Pareto optimal front. It is well-suited for multi-objective optimization problems like the FGPE approach, where multiple conflicting objectives need to be considered simultaneously.

The **objective function** for NSGA-II combines the weighted sum of criteria:

$$\begin{aligned} \text{Maximize } f_1(x) = & w_1 \cdot C_1(x) + w_2 \cdot C_2(x) + w_3 \cdot C_3(x) \\ & + w_4 \cdot C_4(x) + w_5 \cdot C_5(x) \end{aligned}$$

where $C_i(x)$ represents the value of criterion i for option x , and w_i are the weights assigned to each criterion.

The **constraints** ensure that each option satisfies the respective thresholds: D_1, D_2, D_3, D_4, D_5 for x_1 , and $D_6, D_7, D_8, D_9, D_{10}$ for x_2 .

Below is a pseudo-code representation of a multi-objective evolutionary algorithm inspired by NSGA-II for solving the optimization problem described:

■ **Algorithm 1** Multi-Objective Evolutionary Algorithm.

-
- 1: **Initialize** population P randomly or using a heuristic method
 - 2: **Evaluate**(P) // Evaluate the objective functions for each individual considering the constraints
 - 3: **repeat**
 - 4: Create an empty offspring population Q
 - 5: **while** $|Q| < |P|$ **do**
 - 6: Select parents from P based on non-dominated sorting and crowding distance, considering the constraints
 - 7: Perform crossover and mutation to create offspring, ensuring constraints are satisfied
 - 8: **Evaluate**(Q) // Evaluate the objective functions for each offspring considering the constraints
 - 9: Merge P and Q to form R (combined population)
 - 10: Perform non-dominated sorting on R considering the constraints
 - 11: Select top $|P|$ individuals from R based on non-domination and crowding distance
 - 12: **end while**
 - 13: Replace P with the selected individuals from R
 - 14: **until** termination criteria are met
-

11:6 Client-Side Gamification Engine for Enhanced Programming Learning

In the pseudo-code:

- *Initialize* function initializes the population of candidate solutions considering the decision variables.
- *Evaluate* function evaluates the objective function values for each individual in the population, taking into account the constraints.
- *Select* function selects parents for reproduction based on non-dominated sorting and crowding distance, ensuring feasibility with constraints.
- *Crossover* and *Mutation* functions create offspring from selected parents while satisfying the constraints.
- *Merge* function combines the parent population P and offspring population Q to form a combined population R .
- *Non-dominated sorting* sorts individuals in R based on non-domination, creating fronts of non-dominated solutions, considering the constraints.
- *Replace* function selects top individuals from R to form the next generation population P , based on non-domination and crowding distance, while ensuring feasibility with constraints.
- The algorithm continues iterating until a termination condition is met, such as reaching a maximum number of generations or evaluations.

5 The implementation of Client-side Evaluation Engine

In the scope of the FGPE project, two options were identified:

- Client-side execution environment + cached interface for Gamification Service
- Client-side execution environment + client-side Gamification Service (synchronized with server when available)

Both are detailed in the next subsections.

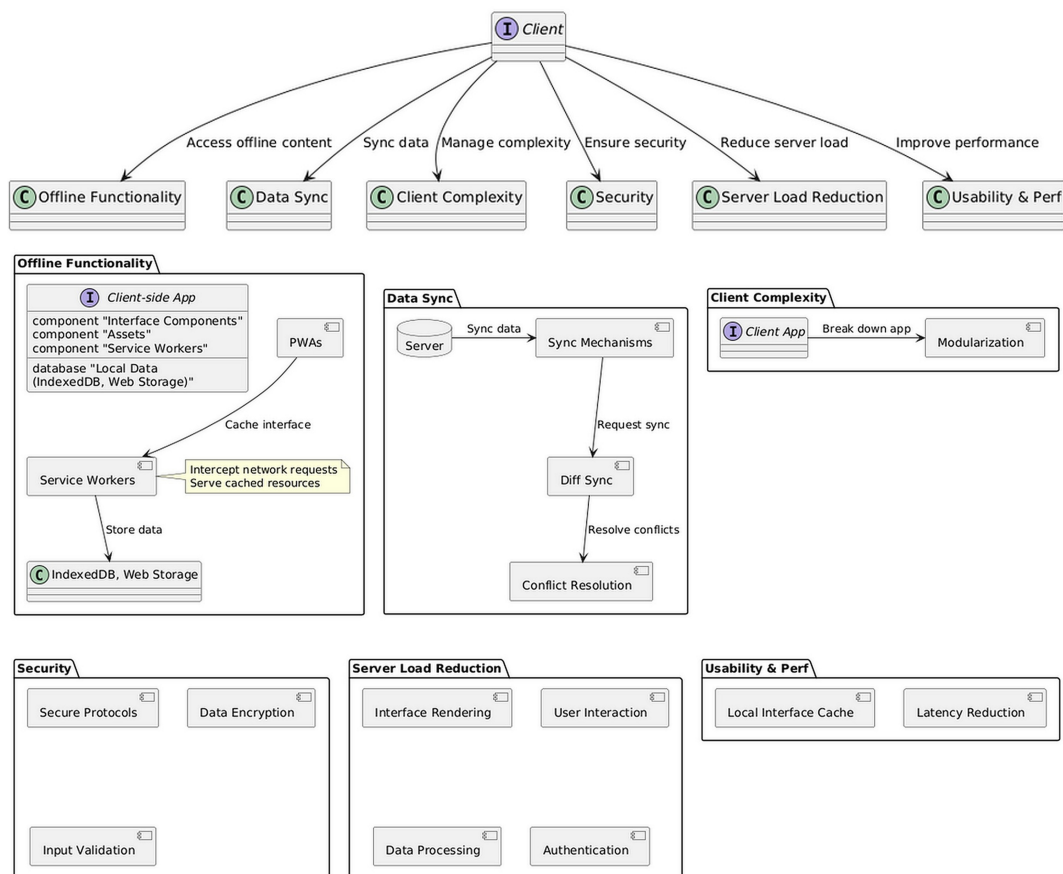
5.1 Cached interface for Gamification Service

This option focuses on caching the interface for the Gamification Service on the client-side. It means that even if there's no active internet connection, the user can still interact with the interface and work on their exercises. The client-side execution environment would allow the user to submit their solution regardless of the server's availability. The cached interface would ensure that the user can still receive feedback and engage with gamification elements locally, enhancing usability in areas with poor internet connectivity.

Bellow we explore the key elements behind our approach (see Figure 2 for reference).

1. First, the system must have offline functionality. The client-side application needs to store the interface components, assets, and relevant data locally using technologies like IndexedDB, Web Storage, or Service Workers. Progressive Web Apps (PWAs) [19] are employed to ensure key interface elements are cached for offline use. Service Workers are used rto intercept network requests, allowing the application to function even without an active internet connection by serving cached resources.
2. Next we must ensure data synchronization by implementing robust synchronization mechanisms [8] to ensure data consistency between the client and server once the internet connection is restored. We have used differential synchronization to sync data while minimizing bandwidth usage and conflicts. Conflict resolution strategies were employed to handle scenarios where changes are made both locally and on the server during offline usage.

3. Third element was solving client-side complexity as caching the interface on the client-side introduces additional complexity to the frontend codebase [18]. We have used modularization to help manage this complexity by breaking down the application into smaller, manageable parts.
4. Fourth element was security considerations, as with increased client-side execution, security risks such as data tampering, injection attacks, and unauthorized access become more pronounced [30]. We have implemented secure communication protocols, data encryption, and input validation on both client and server sides to mitigate these risks.
5. Fifth element was server load reduction, which we solved by offloading interface rendering and user interaction to the client-side [22], thus the server's role shifts to primarily handling data processing and authentication. This led to reduced server load and improved scalability, as the server no longer needed to handle as many concurrent interface requests, leaving our server to efficiently handle batch processing tasks and scale with increasing user demand.
6. Final element is usability and performance. Main improvement was done by caching the interface locally, which enhances usability in areas with poor internet connectivity, providing users with a seamless experience [26]. Performance gains were also achieved by reducing latency associated with fetching interface resources from the server, especially for frequently accessed components using cache policies.



■ **Figure 2** Cached interface for Gamification Service.

11:8 Client-Side Gamification Engine for Enhanced Programming Learning

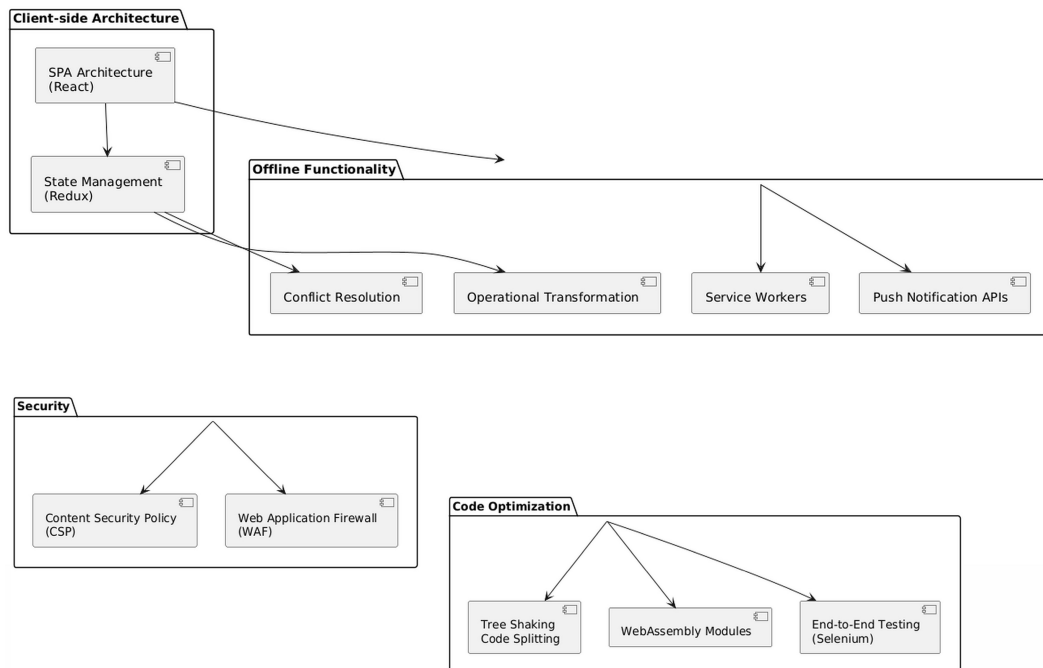
Following this approach can lead to two main advantages:

- Seamless user experience: users can continue working on exercises without interruptions due to server downtime or poor internet connection.
- Reduced server load: by caching the interface locally, the server's role is minimized to batch processing, reducing the strain on resources and potentially improving overall performance.

However, there are also disadvantages such as increased client-side complexity, data synchronization challenges and limited server-side control. Firstly, implementing a gamification service cached interface on the client-side adds complexity to the frontend codebase, potentially making maintenance and debugging more challenging. Secondly, caching the interface for the Gamification Service locally may introduce challenges in synchronizing data between the client and server when an internet connection is available. Finally, when relying on client-side execution, there may be limited control from the server-side, which could potentially lead to security vulnerabilities in data management.

5.2 Client-side Gamification Service

This option involves implementing the Gamification Service entirely on the client-side, synchronized with the server when an internet connection is available. It allows for more autonomy on the client-side, as the gamification elements are not dependent on server availability. However, synchronization would be necessary to update the global state and ensure consistency across users (see Figure 3 for reference).



■ **Figure 3** Client-side Gamification Service.

To implement efficient Gamification Service entirely on the client-side we needed an advanced Client-side Architecture. We have chosen single page application (SPA) architecture [14], based on React for efficient client-side rendering and seamless user experience. State management was implemented using Redux to manage complex gamification state across different components of the application.

Offline Functionality was first assured with conflict resolution mechanisms [4] to handle data conflicts that may arise during offline usage, ensuring data integrity when synchronizing with the server. We have also implemented operational transformation algorithms to reconcile concurrent edits made to the gamification data by multiple clients during offline usage, minimizing data inconsistencies. The progressive web application (PWA) had service workers implemented to enable offline caching and background synchronization of gamification data, transforming the application into a full-fledged PWA. We have also considered integrating push notification APIs to notify users of gamification updates and achievements, enhancing user engagement and retention.

Security was assured by applying content security policy (CSP) [6] to mitigate cross-site scripting (XSS) attacks and prevent execution of unauthorized scripts, enhancing the security of the client-side code. Web Application Firewall (WAF) was also used to monitor and filter HTTP traffic, detecting and blocking malicious requests targeting the client-side gamification service.

Code optimization was done through tree shaking and code splitting to eliminate dead code and reduce bundle size, optimizing performance and loading times of the client-side application. We have also integrated WebAssembly modules for computationally intensive gamification algorithms, leveraging the performance benefits of low-level bytecode execution in modern web browsers. End-to-end testing was done using Selenium to validate the functionality and performance of the client-side gamification service across different scenarios and environments.

Using this approach has the following advantages:

- Increased autonomy: Users can receive instant feedback and engage with gamification elements even without an internet connection, as the service is entirely client-side.
- Reduced dependency on server: By moving the gamification logic to the client-side, there's less reliance on server availability, potentially reducing downtime and improving user experience.

This approach lead also to several disadvantages such as security risks, duplication of logic and dependency on client resources. In the former, handling sensitive gamification logic and data on the client-side could pose security risks, as client-side code is inherently less secure and more susceptible to tampering or exploitation compared to server-side code. Other pitfall is the duplication of logic, since moving the gamification logic entirely to the client-side may result in duplication of code and logic, as similar functionalities may need to be implemented both on the client and server sides. This lead to code redundancy and maintenance overhead. Finally, relocating the gamification service to the client-side may increase the resource requirements on the client devices, especially for complex gamification algorithms or large datasets. This could impact performance on devices with limited processing power or memory.

6 Conclusions

The main result of this work is sketching the blueprint for the development of a client-side software layer for the assessment of gamified programming exercises within the FGPE project. This component will be responsible for producing instant feedback to the users even without active connection with the server, caching data on user progress and achievements, and synchronizing students' progress cached on the client with the global state stored at the server.

In order to tackle this challenge, two options were identified: 1) caching the interface for the Gamification Service on the client-side and 2) implementing the Gamification Service entirely on the client-side.

11:10 Client-Side Gamification Engine for Enhanced Programming Learning

While both options offer solutions to enable client-side execution and offline capabilities, they also come with their own set of challenges and drawbacks. The choice between the two options should consider factors such as the specific requirements of the application, the desired level of control and security, and the technical capabilities (and constraints) of the client devices and server infrastructure.

The software is planned was developed using two-way communication between web browser and web server [9]. The software will be freely distributed on the GitHub platform in the FGPE repository⁵ under the GNU General Public License v3 open-source license⁶, so that any educational institution can use it and, if necessary, adopt it for their specific purposes free of charge.

References

- 1 Workbox: Powerful tools for your service workers, 2021.
- 2 Thamer Al-Rousan. An investigation of user privacy and data protection on user-side storage, 2019.
- 3 Wahab Kh Arabo. The web and ai influences on distributed consensus protocols in cloud computing: A review of challenges and opportunities. *Journal of Information Technology and Informatics*, 3(1), 2024.
- 4 Rebecca Kai Cassar, Joseph Vella, and Joshua Ellul. A conflict resolution abstraction layer for eventually consistent databases. In *2016 International Conference on Engineering & MIS (ICEMIS)*, pages 1–5. IEEE, 2016.
- 5 Sankhadeep Chatterjee, Sarbartha Sarkar, Nilanjan Dey, Amira S Ashour, and Soumya Sen. Hybrid non-dominated sorting genetic algorithm: Ii-neural network approach. In *Advancements in Applied Metaheuristic Computing*, pages 264–286. IGI Global, 2018.
- 6 Hsing-Chung Chen, Aristophane Nshimiyimana, Cahya Damarjati, and Pi-Hsien Chang. Detection and prevention of cross-site scripting attack with combined approaches. In *2021 International Conference on Electronics, Information, and Communication (ICEIC)*, pages 1–4. IEEE, 2021.
- 7 Xiaodan Chen, Jun Lu, Mei Gong, Benjun Guo, and Yuanping Xu. Design and implementation of decentralized online education platform. In *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, 2020. doi:10.1109/ICMCCE51767.2020.00212.
- 8 Mohammad Faiz and Udai Shanker. Data synchronization in distributed client-server applications. In *2016 IEEE International Conference on Engineering and Technology (ICETECH)*, pages 611–616. IEEE, 2016.
- 9 Ian Fette and Alexey Melnikov. The websocket protocol. Technical Report RFC 6455, IETF, 2011.
- 10 Joy M Field, Liana Victorino, Ryan W Buell, Michael J Dixon, Susan Meyer Goldstein, Larry J Menor, Madeleine E Pullman, Aleda V Roth, Enrico Secchi, and Jie J Zhang. Service operations: what’s next? *Journal of Service Management*, 29(1):55–97, 2018.
- 11 Miftachul Huda. Between accessibility and adaptability of digital platform: investigating learners’ perspectives on digital learning infrastructure. *Higher Education, Skills and Work-Based Learning*, 14(1):1–21, 2024.
- 12 Mayssa Jemel and A. Serhrouchni. Toward user’s devices collaboration to distribute securely the client side storage. In *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, 2015. doi:10.1109/NOTERE.2015.7293479.

⁵ <https://github.com/FGPE-Erasmus>

⁶ <https://www.gnu.org/licenses/gpl-3.0.en.html>

- 13 Young joo Shin and Kwangjo Kim. Differentially private client-side data deduplication protocol for cloud storage services. *Secur. Commun. Networks*, 8:2114–2123, 2015. doi:10.1002/sec.1159.
- 14 DV Kornienko, SV Mishina, and MO Melnikov. The single page application architecture when developing secure web services. In *Journal of Physics: Conference Series*, volume 2091(1), page 012065. IOP Publishing, 2021.
- 15 Raoni Kulesza, Marcelo Fernandes de Sousa, Matheus Lima Moura de Araújo, Claudiomar Pereira de Araújo, and Aguinaldo Macedo Filho. Evolution of web systems architectures: a roadmap. *Special Topics in Multimedia, IoT and Web Technologies*, pages 3–21, 2020.
- 16 José Paulo Leal and Fernando Silva. Mooshak: A web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003.
- 17 José Paulo Leal, Ricardo Queirós, Pedro Ferreirinha, and Jakub Swacha. A Roadmap to Convert Educational Web Applications into LTI Tools. In *Third International Computer Programming Education Conference, ICPEC 2022, Dagstuhl, Germany, 2022*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASICS.ICPEC.2022.12.
- 18 Vittorio Maniezzo, Marco A Boschetti, Antonella Carbonaro, Moreno Marzolla, and Francesco Strappaveccia. Client-side computational optimization. *ACM Transactions on Mathematical Software (TOMS)*, 45(2):1–16, 2019.
- 19 Rytis Maskeliunas, Robertas Damasevicius, Tomas Blazauskas, and Jakub Swacha. Evaluation of a progressive web application for gamified programming learning. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*, pages 1334–1334, 2022.
- 20 Rytis Maskeliunas, Robertas Damasevicius, Tomas Blazauskas, and Jakub Swacha. Evaluation of a progressive web application for gamified programming learning. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education, Volume 2, SIGCSE 2023, Toronto, ON, Canada, March 15-18, 2023*, volume 2, page 1334, 2023. doi:10.1145/3545947.3576385.
- 21 Rytis Maskeliūnas, Robertas Damaševičius, Tomas Blažauskas, Jakub Swacha, Ricardo Queirós, and José Carlos Paiva. FGPE+: The mobile FGPE environment and the pareto-optimized gamified programming exercise selection model—an empirical evaluation. *Computers*, 12(7), 2023. doi:10.3390/computers12070144.
- 22 Farouk Messaoudi, Adlen Ksentini, and Philippe Bertin. Toward a mobile gaming based-computation offloading. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.
- 23 J. Moscicki and L. Mascetti. Cloud storage services for file synchronization and sharing in science, education and research. *Future Gener. Comput. Syst.*, 78:1052–1054, 2018. doi:10.1016/j.future.2017.09.019.
- 24 José Carlos Paiva, Alicja Haraszczuk, Ricardo Queirós, José Paulo Leal, Jakub Swacha, and Sokol Kosta. FGPE Gamification Service: A graphql service to gamify online education. In *Trends and Applications in Information Systems and Technologies: Volume 4*, pages 480–489, Cham, Switzerland, 2021. Springer.
- 25 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, Jakub Swacha, and Filip Miernik. Managing gamified programming courses with the FGPE platform. *Information*, 13(2):45, 2022.
- 26 Adrian Petcu, Madalin Frunzete, and Dan Alexandru Stoichescu. Evolution of applications: From natively installed to web and decentralized. In *International Conference on Computational Science and Its Applications*, pages 253–270. Springer, 2023.
- 27 U. Rahardja, M. A. Ngadi, R. Budiarto, Q. Aini, Marviola Hardini, and Fitra Putri Oganda. Education exchange storage protocol: Transformation into decentralized learning platform. *Frontiers in Education*, 6, 2021. doi:10.3389/feduc.2021.782969.
- 28 T. Sproull and Bill Siever. Going native with your web dev skills: An introduction to react native for mobile app development, 2020.

11:12 Client-Side Gamification Engine for Enhanced Programming Learning

- 29 Jakub Swacha, Thomas Naprawski, Ricardo Queirós, José Carlos Paiva, José Paulo Leal, Ciro Giuseppe De Vita, Gennaro Mellone, Raffaele Montella, Davor Ljubenkovic, and Sokol Kosta. Open Source Collection of Gamified Programming Exercises. In *Proceedings of the thirty-seventh Information Systems Education Conference, ISECON 2021*, pages 120–123, Chicago, IL, USA, 2021. Foundation for IT education.
- 30 Hamed Tabrizchi and Marjan Kuchaki Rafsanjani. A survey on security challenges in cloud computing: issues, threats, and solutions. *The journal of supercomputing*, 76(12):9493–9532, 2020.
- 31 Mohit Thakkar. Reactjs: A comprehensive analysis of its features, performance, and suitability for modern web development, 2020.
- 32 B. Wang and Yimin Zhou. Research and implementation of multiple virtual management platform in multimedia classroom based on cloud storage. In *2011 International Conference on Multimedia Technology*, 2011. doi:10.1109/ICMT.2011.6002332.
- 33 Tian Wang, Jiyuan Zhou, Anfeng Liu, Md Zakirul Alam Bhuiyan, Guojun Wang, and W. Jia. Fog-based computing and storage offloading for data synchronization in iot. *IEEE Internet of Things Journal*, 6:4272–4282, 2019. doi:10.1109/JIOT.2018.2875915.

Game Development: Enhancing Creativity and Independent Creation in University Course

Lenka Bubenkova ✉

Department of Computers and Informatics, FEI TU of Košice, Slovakia

Emilia Pietrikova ✉ 

Department of Computers and Informatics, FEI TU of Košice, Slovakia

Abstract

In this study, we tested a novel method of teaching the Unity engine to computer game design and development students. Our objective was to determine if a flexible assignment structure is the most effective for students with minimal engine experience. The study demonstrated that independent work significantly improves students' comprehension and problem-solving skills. Key findings include a 90% increase in students achieving more than the minimum required grade, a significant improvement in self-reported confidence with Unity (with 66.3% of students moving from no experience to higher skill levels), and diverse, innovative final projects that exceeded initial expectations. These results suggest that the flexible assignment approach enhances creativity and maintains high expectations for student work, ensuring their success in the game development industry. The combination of student project grades, innovative project elements, and positive feedback indicates that this method is highly beneficial and could be applied effectively in various educational settings.

2012 ACM Subject Classification Social and professional topics → Student assessment

Keywords and phrases novice programmers, assessment, learning analytics, motivation, unity engine, game development, problem-solving skills

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.12

Funding This work was supported by project Kega No. 015TUKE-4/2024 “Modern Methods and Education Forms in the Cybersecurity Education”.

1 Introduction

In computer science and software engineering, it is essential to recognize the significance of incorporating fun and creative elements alongside traditional programming and practical tools. As computer games and other forms of digital entertainment continue to grow in popularity, educators must integrate them into their teaching methods. To remain up-to-date with the latest trends, universities must equip their students with the skills required for game development. One of the aspects of supporting this tutorial is, for example, this case study[24] of the concept of gamification used on a games development course. One practical approach to combining education and creativity is teaching students how to craft their games in the most imaginative way possible while leveraging gamification to deliver lessons through gameplay. This can be used in various courses, as described in this publication [21], where authors implemented the learning of object-oriented programming by playing computer games. While teaching students the basics of game development and the usage of engines, there is also a need to consider a theory that will help them create visually attractive and exciting games. One of the ways to develop such a game is using patterns, as is written in this publication [32], to keep players entertained and simulate and enhance reality.

Currently, there are many game engines available for creating games. Unity is a popular choice because many job opportunities require knowledge of this engine, as written in this article [17], and it is easy to use, even for students new to game development. Besides,



© Lenka Bubenkova and Emilia Pietrikova;
licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 12; pp. 12:1–12:13

OpenAccess Series in Informatics



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

12:2 Enhancing Creativity in University Course

Unity's ability to create multiplatform [13] games and projects is important for today's game development world. In addition to these aspects, the use of Unity Engine for teaching is also suitable from the point of view of the accessibility of various tutorials. As mentioned in [31], from an educational perspective, it is appropriate for students to independently find additional information about creation from professionals, programmers, and developers dedicated to creating just such instructions. In this study [8], using Unity Engine and Design-Based Research approach resulted in the high engagement of students, making an introduction to this engine smoother and more attractive. To add, based on this study [9], learning how to code by creating games promises high motivation. The aim is to teach students how to develop Unity Engine and provide enough information to continue their work even after this course.

Besides, not only game development uses Unity Engine, as we can see [19] or [34], but Unity can be used for developing various types of projects. In this example, we can see the usage of the Unity engine in the creation of the short movie "The Heretic" [10]. This provides a strong justification for teaching Unity Engine.

However, the challenge lies in making the teaching of game development exciting and engaging while encouraging students to be creative and imaginative. The traditional approach to game development can be time-consuming and challenging because students need to develop unique ideas to incorporate into their games. If students are given strict rules and guidelines to follow, they might not develop a good understanding of the engine and how to create games. As mentioned in [30], in addition to basic programming, advanced knowledge in computer graphics, databases, artificial intelligence, or, for example, physics is also necessary. Basic knowledge enables students to understand contexts better and work more effectively on assignments within the subject. Today, it is also essential to know various techniques and developments. Unity, for example, provides an excellent environment for developing virtual reality and XR in general. According to [29], the focus on the interface and, thus, the use of these XRs is extremely important for game development and projects. Each of these approaches is usable and thus feasible in the Unity Engine. Using the Unity engine for enhancing problem-solving skills and creativity for this course is also supported by outcomes of this research [26]. This study provides insights that could be applied by educators across various disciplines who wish to incorporate similar strategies to enhance engagement and learning outcomes.

How can creative labs improve the GameDev course?

Up to this point, game development education in our courses has been implemented through basic versions of preexisting games, where students complete brief tasks and gain an understanding of the environment. Teaching methods are straightforward, and there is no space for considering one's elements and improvements. The entire procedure consists of downloading the game, launching it in the Unity editor, and playing it. After the game calculates their scores, students submit their scores and receive their marks. A similar approach can be seen in this [27] game made with the Unity engine.

The new method involves the entire project creation and setup process to develop a functional game similar to a flying simulator. This approach consists of three tutorials and a step-by-step guide, from creating a project to adding a user interface. The main difference between the old and new approaches is that the old approach did not allow students to create something independently. Instead, it strictly guided them through various tasks.

On the other hand, the new approach guides students through creating games, but it always leaves their final work independent of the assignment form. This means that while they must follow various steps and learn various parts, the final form is always in their

imagination. This ensures they create something unique, go through the tutorials, find more exciting and engaging parts, and work on them more. Based on the experience outlined in the article [5], the decision has been made to adopt GitLab as the submission platform for student projects.

The remainder of this paper is organized as follows: Section 2 provides background and related work. Section 3 describes our proposed approach and details the three iterations of the tutorial. Section 4 outlines the experiment setup and conduction, including our conjectures. Section 5 presents the results and discusses the findings. Finally, Section 6 concludes the paper with interpretations of the results, challenges, limitations, and suggestions for future work.

2 Background

With a similar course outline, this paper [14] describes the slow approach and exciting engagement for students in creating games through the Unity engine. It encourages its students to create 2D games, preparing them for future careers. We also considered the importance of active learning, as seen in this example [25], where authors used active learning based on scenarios using Unity 3D. Another engaging hands-on project experience, as described in this article [7], is using step-by-step tutorials. This kind of tutorial provides a compact way of learning and understanding problems to their core. Thinking about using Blender, we also searched for existing implementations of this tool in teaching. One of the most used approaches is the creation of educational games using Unity. This approach can introduce students to development with the Unity engine in the form of a game. With this form of teaching, as is written in this research [16] that discusses educational games in Unity, we can effectively avoid limitations to traditional teaching methods and support students' creativity.

Another approach for this tutorial is using a teacher-student model, which was optional in the previous teaching method. With the availability of a teacher during the whole process, we may support students' innovative thinking and reasoning abilities. This idea of composing into this course was supported with research [11], that analyses this Teacher-Student model.

While using the Unity engine, we were also aware of the risk of issues in students' projects, like bad smells. As was discussed in this article [6], bad smells are detectable and can be divided into categories. The main target of this course was to prevent students from developing this kind of issue. Similarly, this study [23] also discusses bad smells, and its aim was directly onto this issue in game development. This article also discusses the negative impact of bad smells and the risk that they are not always critical.

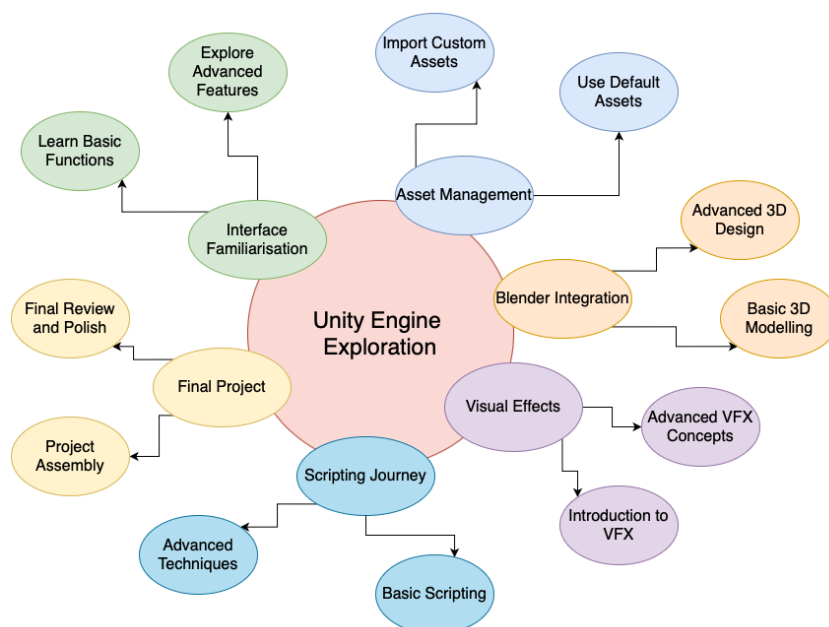
In conclusion, there is no need to push students back with their projects only because of the avoidance of bad smells. Instead, the fact that the Unity engine is fully programmable in C#, as is written in this article [33], there is a need to focus on correct and nonissue coding. However, at all times, we should be careful to avoid strictly controlling students with sets of rules to ensure creative and innovative thinking in their projects.

With the rise of immersive learning, we can also discuss the need for knowledge of the Unity engine. For example, this work [3] demonstrates the usage of Unity for the implementation of twin-screw, which is a process in the polymer. This work used Unity to implement the simulation in an interactive educational environment. A similar approach is used in the paper [35], where the problem of the informed purchase of toys is solved with the implementation of augmented reality technology using Unity 3D. Immersive interaction can also be done in full-body forms, as is described in this paper [15], where authors contributed to enabling this

kind of interaction in the metaverse. To achieve this, the authors worked with Unity 3D and other tools, successfully integrating the digital twins and immersive user experience. This work may show that students need more than the Unity engine to prepare for the industry's challenges. The usage of this information and its successful implementation in the education process is written in this article [20], where authors proved that this approach can be effective in the student's improvement in the learning process.

3 Proposed approach

Tutorials are created in the areas of computer game design and development. There are three tutorials. The goal of the first tutorial is to explain the engine and cover the fundamentals of project and game background creation. The second tutorial covers creating game effects using VFX Graph and explains the differences between different approaches in effect creations. The second tutorial also covers the fundamentals of the Blender[12] engine and how to create 3D models. The third tutorial served as an addition to help students add engaging components to their projects.



■ **Figure 1** The Unity Engine Learning Journey outlining the structured path from basic concepts to the final project.

Before discussion of the detailed structure of the tutorials, Figure 1 presents a comprehensive mind map of the Unity Engine Exploration course. This diagram encapsulates the learning journey from initial familiarization with the Unity interface to the final project assembly, containing advanced topics such as VFX and Blender integration.

3.1 First iteration

In the beginning, students are led through the installation process. Then, they are shown how to create and run the project. This part is crucial because creating and starting with work in Unity from the total basics is fundamental for students to develop a relationship with

the engine. It makes it easier for them to understand every part that will come next from the basics. After the introduction of the engine, the first tutorial covers the creation of the game world using terrain tools. Additionally, students are encouraged to use their assets, not just those shown in the tutorial. The essential part is introducing the Asset Store and working with assets in the editor. In the future, they must understand how to create and work with assets. The next important step is adding a player. They are encouraged to pick their avatar to play their game. The exciting part of this tutorial is adding a Timeline. A Timeline is generally used to animate movements and create exciting game moments. Students are shown the basics of work with a timeline and are not given the detailed needs of using a timeline for their game; instead, they use it in their way. Using a timeline is beneficial not only for game development but also for many other fields. Short movies and ads can be created using a timeline, so this incorporation into the curriculum may widen their abilities in the future in various fields. Students are also introduced to scripting in Unity, using C#. We are teaching 3rd graders, so they must have various programming skills. Because of that, this part is mainly informative and shows them good practices in scripting in Unity. At the end of this tutorial, students have developed their game world and the basic movement of players in this world. The crucial part is personification and uniqueness. This tutorial has a role in developing interest in creations and incorporating their own elements, which may benefit their careers and development.

3.1.1 First iteration outcome

In the first iteration, students learned the basics of Unity and created a simple game world. For instance, one student created a terrain with mountains and a river, utilizing assets from the Unity Asset Store. In contrast, another student designed a desert landscape with custom textures.

3.2 Second iteration

The second tutorial was expecting better skills with unity and overall orientation of students in the engine. Firstly, there is a comparison of the two most common tools for creating visual effects in unity: VFX Graph and Particle System. Next, students moved to work with VFXGraph, creating an elemental explosion made of more layers. In this article, we can see the point of teaching students the basics of VFX [18], also with a similar approach in this article [22], as this theme is not only valid with game development but also has broad usage in many other fields. Students are asked to develop their solution in the VFX Graph editor and, later in the tutorial, find their usage in their game. This tutorial also covers the basics of working with Blender. Blender is used to create a simple target for students' games. The main reason for incorporating Blender into this course is the need for 3D models in today's game. Teaching students the basics of working with Blender and coworking with Unity may benefit project uniqueness and originality. As written in the article that discusses modeling [4], Blender was the best choice. Later in the tutorial, students get a task to create something useful with Blender and use it in their games.

3.2.1 Second iteration outcome

In the second iteration, students learned the differences between the Particle system and VFX Graph in the Unity engine and worked on their unique solutions. They were led to create the effect of an explosion. Students created effects that contributed to their games.

12:6 Enhancing Creativity in University Course

Another aspect of this iteration was the usage of Blender. This engine helped students to be more creative with assets and learn the essentials of the work in Blender. Students created various 3D models, such as players, donuts, trees, and various equipment for their games. Students also worked with VFX Graph, creating explosions and other effects, like shooting and smoke.

3.3 Third iteration

The third tutorial is optional. Creating effects in Unity or modeling 3D objects in Blender is only interesting for some students. Primarily, these parts are required, but the extra work on their project can be done on other developing parts. The third tutorial covers the basics of the particle system, which can be used in various ways, not only in game development, for example, like [2] or [28], the creation of UI using Canvas, and its elements, like sliders or buttons. Also, students can see the creation of levels using Scene Manager. A similar approach can be seen in this article [1], where leveling is also based on scenes in a functional 3D Unity game. A combination of UI and scene manager can create an entry screen for players to their game, and in the final project, the game will look more professional. The last part of the third tutorial is bug fixing. Students are encouraged to fix issues in existing projects to enhance their problem-solving skills and general knowledge of game development, where fixing is common and vital.

3.3.1 Third iteration outcome

In the third iteration, students explored advanced features like the particle system and UI creation. For example, one student created a particle effect to simulate a magical spell, adding dynamic visual flair to their game. Another student designed a custom health bar and interactive menu, significantly improving the user experience. Additionally, one student developed a multi-level game with a main menu, level selection screen, and in-game HUD, demonstrating their ability to integrate UI elements with scene management.

3.4 Outcomes and Project Example

The existing project that students get to fix is available on the course page. In this project, all three tutorials are implemented, adding some unique elements and implementations to make students more curious and inspired. Students can create their own game in this project but are encouraged to create their own to solve issues that may be fixed in existing implementation and prepare them better for their future in this challenging field.

These tutorials should enhance students' independent work and support them in developing their games. Even if there are some mandatory steps for final grading, students still have the chance to add their elements and additions to their games and get the desired grade on the project. The reason is simple: game development is not a detailed, structured plan but more of an idea-adding process.

Following the detailed description of the tutorial structure, the following section clarifies the methodology utilized to evaluate its effectiveness. This analysis primarily focuses on assessing student feedback and project outcomes, which serve as critical metrics for measuring the educational impact of the implemented interventions.

4 Experiment setup and conduction

This study focuses on innovative and adaptable methods for teaching the fundamentals of using the Unity engine. Students' self-grading and individual development as project creators are crucial to a conclusion. This study employs a combination of self-reported data collection, enrolled students' grades, and final projects. Data from students' answers to questions were valid, and students were informed of the usage of their responses.

- Questionnaire – After grading, participants self-reported data to provide their perspectives and comments on the methodology.
- Evaluation – Using the results of final projects and their assignment grades; and also monitoring the quantity of work that exceeded the assignment's minimum.

4.1 Course Characteristics

This study was conducted during the 2023-2024 academic year. One hundred four third-year undergraduate students enrolled in a game development class during these academic year across both winter and summer semesters. The class included Slovak students as well as international students, and it was conducted in English and Slovak. Students had varying levels of prior experience with the Unity engine.

4.2 Conjectures

This paper posits several conjectures to evaluate the efficacy of a novel instructional approach in Unity engine development. These conjectures are implanted in the conviction that structured yet flexible learning environments can significantly enhance students' educational experience and outcomes, irrespective of their prior technical experience.

The research process involved three main iterations, each designed to build students' skills and creativity progressively. The first iteration focused on fundamental Unity skills, the second on advanced features like VFX and Blender, and the third on optional enhancements such as UI and bug fixing.

The following hypotheses are formulated to rigorously test the effectiveness of these educational methodologies through empirical evidence and data-driven analysis:

- ▶ **Conjecture 1.** *Students do not have to have prior skills and an introduction to Unity engine development to complete this course.*
- ▶ **Conjecture 2.** *If students are given more freedom in their assignments, they will be more creative and driven to work on their projects, which will ultimately result in better grades.*
- ▶ **Conjecture 3.** *After working more independently, students feel secure in their knowledge.*
- ▶ **Conjecture 4.** *Students will include unique elements and interesting strategies of their own.*

To verify these conjectures, employment of the various methods was used: Conjectures 1 and 2 were assessed using a structured questionnaire that studied student engagement and learning outcomes. For Conjecture 3, an analysis of existing student performance data was conducted to determine the impact of independence on learning security. Conjecture 4 was evaluated through project reviews that assessed the creativity and uniqueness of student submissions.

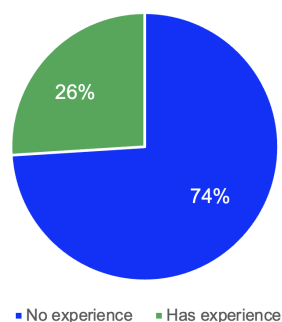
5 Results

The examination of students' results, final projects, and self-reported information from questionnaires was essential to analyze the effectiveness of the teaching method. This section presents the findings of these analyses, which help validate the conjectures proposed in this paper. An overview of the analyzed data and a summary of the findings related to the hypotheses are provided below.

5.1 Conjecture 1

In the following Figure Figure 5.1, it is visible that the number of students with some experience before attending this course was much smaller than those that had not previously worked with the Unity engine. Percentually, only 26% worked with the engine. This section aims to determine if these students will experience any resulting disadvantages.

Student's experience with Unity Engine



■ **Figure 2** Student's experience with Unity Engine before tutorials.

The Pearson correlation coefficient was used to examine whether prior experience with the Unity Engine statistically impacts the grading scores. The test yielded a coefficient of 0.12, considered a low degree of correlation. This result suggests no statistically significant score difference between individuals with and without previous experience using the Unity Engine. Consequently, previous knowledge of Unity only significantly influences the observed grading outcomes.

This proved Conjecture 1, where was supposed that students do not have to have prior knowledge to perform outstanding results in this course.

5.2 Conjecture 2

Immediately after the personal defense, students were awarded points for the developed projects based on a predetermined evaluation on the subject page at the end of each tutorial. The attached table Table 1 shows that the highest rating was 10 points, representing the maximum of the possible points obtained. Completing all the necessary steps in the tutorial could obtain the minimum number of points, which was 6. The result, which was 10 points, means that students did extra work on this project and added many particular tasks from all the tutorials. This point evaluation was obtained by more than half of all students. The second highest rating belongs to students with the same score, 14 for both 8 and 9 points. At the same time, a low assessment of the assignment occurred in only one case, in the form of one point. Therefore, completing the instructions by the students is exceptionally successful.

In the questionnaire, students had to answer the question, “Do you think that your point evaluation corresponds to the time you devoted to the assignment?” where the answer “Yes” reached the value of 94.2%, and the answer “No, I devoted a lot of time to this assignment” had a value of 3.5%, which can be considered as the result of a fair assessment and student satisfaction with this aspect of the university courses as well.

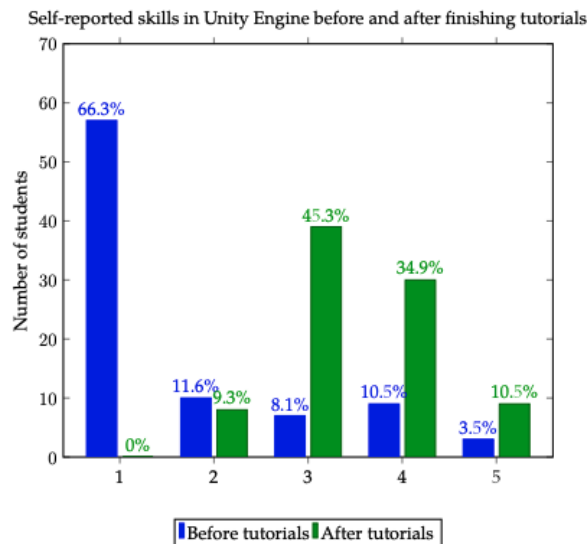
■ **Table 1** Table for students grades.

points	0p	1p	2p	3p	4p	5p	6p	7p	8p	9p	10p
students	0	1	0	0	3	3	3	7	14	14	55

This proved Conjecture 2, where was supposed that students would be motivated to work harder on their assignments if they were given some space for their creations. This also proved that 90% of students did more than the required work, graded with 6 points.

5.3 Conjecture 3

Students also had the opportunity to answer questions about their skills with Unity Engine before and after completing tutorials. As can be seen on the attached chart Figure 5.3, self-rating skills in Unity Engine, from rating to scale from 1 to 5, where 1 represents zero experience and five the highest level of ability properties, all students managed to move to a higher rating. In particular, the drop in the change from 66.3% to 0% in experience indicates excessive success. On the second side, self-assessment in the highest degree rose by 7%, assessment in the form of 4 out of 5 increased by 24.4%, and the rating on the 3 out of 5 scale increased by 37.2%.



■ **Figure 3** Self-reported skills in Unity Engine before and after finishing tutorials.

This proved Conjecture 3, where was supposed that students would feel more secure in their knowledge if they worked independently.

5.4 Conjecture 4

The students were instructed to create and add their elements independently. After collecting and evaluating the projects, it was evident that the students demonstrated fascinating diversity and creativity in their work. Each student's individuality was visible in the creation of their assignments. One exciting element was the use of Timeline in the form of game animation, which was used effectively in several projects. For example, a cinematic animation of a military plane flying over the sea and observing various elements such as ships, houses, and the sea. The manual mainly covers movement in the form of flying, but the resulting student projects brought many new solutions, such as walking. This means that encouraging the students to create and experiment on their own was enough, and when they were left to their own will, various game projects were created.

This proved Conjecture 4, where was expected that students would create unique and personal projects with various elements over the course outline.

6 Conclusion

The tutorial outcomes demonstrate considerable advancements in students' capabilities to produce creative outputs and solve complex problems. This section will analyze the implications of these results, contrasting them with traditional educational methodologies in game design and examining their potential transformative impacts on the curriculum.

6.1 Interpretation of Results

A notable aspect of the student projects was their diversity in game design approaches. While the core tutorials were designed around constructing simulations for flight simulators and first-person shooters, many students ventured beyond these confines. They crafted unique game environments, where characters navigated through complex worlds or diverged completely to create cinematic short animations. These projects not only stuck to the foundational elements of the tutorials but also incorporated innovative modifications and additions. For instance, Figure 6.1 illustrates a project featuring an interactive walking character, demonstrating the practical application of the skills acquired through the course. Positive feedback regarding the final projects was on the availability of project flying-hero, where were implemented all three tutorials with added addins and unique elements. As was said in the student feedback, the ability to see the existing games and their parts was beneficial, not only for students to see how the parts of the tutorial are implemented but also as an inspiration for their projects.



■ **Figure 4** Example of a student-developed game showcasing an interactive character.

6.2 Challenges and Limitations

One primary challenge was guiding students through managing extensive projects and utilizing comprehensive file systems within their version control repositories. Furthermore, the affiliate's familiarity with Unity Engine was initially low, posing a significant learning curve. However, most issues were swiftly addressed through online resources or direct interventions during lectures, enhancing the learning experience and problem-solving efficiency.

6.3 Future work

Future work on this course should contain more detailed explanations for problematic parts written in student responses. Also, students asked for additional content, such as AI in the form of enemies, more scripting in the course, or video tutorials, as they thought they would benefit more from them while working at home without the lecturer's presence. Detailed explanations of problematic topics may help them better adapt to future needs. Besides this, this course is up to date and should be sustainable for more years in the future, but after some time, some additions and work would be needed as the game development world moves further.

The study explores the integration of game design and development courses at the university, highlighting the inclusion of subjects like Unity Engine and Blender, which are popular among students and offer practical skills demanded in the industry. These courses introduce students to complex tools like effects, animations, interfaces, and advanced features like Timeline and VFX Graph.

The curriculum is designed not merely to engage students during lectures but to inspire independent work and creativity in their projects. This approach has proven effective, with students delivering highly successful projects that incorporate their unique solutions. The addition of Blender enriches the creative potential and deepens students' interest in game development and game design.

Student feedback regarding satisfaction with the course and the instructions' clarity has been overwhelmingly positive. The results met and exceeded initial expectations, with students developing innovative solutions promptly. Future enhancements include AI enemies, expanded scripting options, and multimedia instructional materials, reflecting the students' desire for more comprehensive and prolonged engagement with game development tools.

It can be considered, that using this approach in various courses can help to enhance students' learning and adaptability across different disciplines. This method may be integrated into other areas to foster technical skills, critical thinking, problem-solving, and creative design thinking. By implementing these techniques widely, educational institutions can better prepare students for the complexities of modern professional environments while equipping them with the necessary tools to excel in their chosen fields.

References

- 1 Shanmuk Srinivas Amiripalli, Mukkamala SNV Jitendra, Surendra Talari, Sannith Akkireddi, and D Sateesh Kumar. Design and implement an artificial intelligence based zombie's application using unity3d, 2020.
- 2 Natalia Ampilova, Igor Soloviev, and Michail Syasko. Computer modeling the effect of weak electromagnetic field on charged particles by unity engine. In *2020 International Conference and Exposition on Electrical And Power Engineering (EPE)*, pages 082–086, 2020. doi:10.1109/EPE50722.2020.9305579.

- 3 Pedro Santos Bartolomé, Daniel Just, Ariana Bampouli, Simon Kemmerling, Aleksandra Buczko, and Tom Van Gerven. Immersive learning through simulation: implementing twin screw extrusion in unity. In Antonios C. Kokossis, Michael C. Georgiadis, and Efstratios Pistikopoulos, editors, *33rd European Symposium on Computer Aided Process Engineering*, volume 52 of *Computer Aided Chemical Engineering*, pages 3489–3494. Elsevier, 2023.
- 4 Iliyyar Bikmullina and Enzhe Garaeva. The development of 3d object modeling techniques for use in the unity environmen. In *2020 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)*, pages 1–6, 2020. doi:10.1109/FarEastCon50210.2020.9271568.
- 5 Miroslav Binas. Version control system in cs1 course: Practical experience. In *2013 IEEE 11th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, October 2013. doi:10.1109/ICETA.2013.6674398.
- 6 Antonio Borrelli, Vittoria Nardone, Giuseppe A. Di Lucca, Gerardo Canfora, and Massimiliano Di Penta. Detecting video game-specific bad smells in unity projects. In *Proceedings of the 17th International Conference on Mining Software Repositories, MSR '20*, pages 198–208, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3379597.3387454.
- 7 Simon Bouvier-Zappa, Olivier Dionne, and David Hunt. Advanced use cases for animation rigging in unity. In *ACM SIGGRAPH 2019 Studio*, SIGGRAPH '19, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3306306.3328748.
- 8 Oswald Comber, Renate Motschnig, Hubert Mayer, and David Haselberger. Engaging students in computer science education through game development with unity. In *2019 IEEE Global Engineering Education Conference (EDUCON)*, pages 199–205, 2019. doi:10.1109/EDUCON.2019.8725135.
- 9 Oswald Comber, Renate Motschnig, Hubert Mayer, and David Haselberger. Engaging students in computer science education through game development with unity. In *2019 IEEE Global Engineering Education Conference (educon)*, pages 199–205. IEEE, 2019.
- 10 Veselin Efremov and Adrian Lazar. Real-time procedural vfx characters in unity's real-time short film "the heretic". In *ACM SIGGRAPH 2019 Real-Time Live!*, SIGGRAPH '19, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3306305.3332363.
- 11 Hui Fang, Hongmei Shi, Jiuzhou Zhang, and Marimuthu Karuppiah. Effective college english teaching based on teacher-student interactive model. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 22(3), March 2023. doi:10.1145/3486676.
- 12 Lance Flavell. *Beginning Blender: Open Source 3D Modeling, Animation, and Game Design*. Apress, USA, 1st edition, 2010.
- 13 Maxwell Foxman. United we stand: Platforms, tools and innovation with the unity game engine. *Social Media+ Society*, 5(4):2056305119880177, 2019.
- 14 Christopher L. Hideg and Debatosh Debnath. A programming course using video game design with platform projects. In *2018 IEEE International Conference on Electro/Information Technology (EIT)*, pages 0030–0034, 2018. doi:10.1109/EIT.2018.8500103.
- 15 Shimasadat Hosseini, Ali Abbasi, Luis G. Magalhaes, Jaime C. Fonseca, Nuno M.C. da Costa, António H.J. Moreira, and João Borges. Immersive interaction in digital factory: Metaverse in manufacturing. *Procedia Computer Science*, 232:2310–2320, 2024. 5th International Conference on Industry 4.0 and Smart Manufacturing (ISM 2023). doi:10.1016/j.procs.2024.02.050.
- 16 Zhiyong Hu, Qing Xu, and Guang Huang. Discussion on educational games based on unity. In *Proceedings of the 2022 6th International Conference on Education and E-Learning, ICEEL '22*, pages 67–74, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3578837.3578847.
- 17 Afzal Hussain, Haad Shakeel, Faizan Hussain, Nasir Uddin, and Turab Latif Ghouri. Unity game development engine: A technical survey. *Univ. Sindh J. Inf. Commun. Technol*, 4(2):73–81, 2020.

- 18 Manolya Kavakli and Cinzia Cremona. The virtual production studio concept – an emerging game changer in filmmaking. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 29–37, 2022. doi:10.1109/VR51125.2022.00020.
- 19 Liuxian Li and Zhiyang Fang. Earthquake escape simulator: A system for disaster knowledge popularization. In *Journal of Physics: Conference Series*, volume 2333, page 012002. IOP Publishing, 2022.
- 20 Xiaoxiao Liu, Yiming Shen, Yukari Nagai, and Hirokazu Kato. Use of a mixed-reality creative environment in design education. *Computers & Education: X Reality*, 4:100055, 2024.
- 21 Jakub Livovský and Jaroslav Porubán. Learning object-oriented paradigm by playing computer games: Concepts first approach. *Open Computer Science*, 2014. doi:10.2478/s13537-014-0209-2.
- 22 Jonathan Mortimer. How universities can better engage with the animation/vfx sector in scotland. *Animation Practice, Process & Production*, 7(1):157–173, 2018.
- 23 Vittoria Nardone, Biruk Muse, Mouna Abidi, Foutse Khomh, and Massimiliano Di Penta. Video game bad smells: What they are and how developers perceive them. *ACM Trans. Softw. Eng. Methodol.*, 32(4), May 2023. doi:10.1145/3563214.
- 24 Siobhan O’Donovan, James Gain, and Patrick Marais. A case study in the gamification of a university-level games development course. In *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference, SAICSIT ’13*, pages 242–251, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2513456.2513469.
- 25 Hyesung Park, Sean Yang, and Hongsik Choi. Scenario based active learning programming with unity 3d. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education, SIGCSE ’20*, page 1283, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3328778.3372582.
- 26 Banyapon Poolsawas and Winyu Niranatlamphong. Using a game development platform to improve advanced programming skills. *Journal of Reviews on Global Economics*, 6:328–334, 2017.
- 27 Vincent Schiller. Enc#ypted: An educational game for programming in the unity engine. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems, CHI EA ’21*, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3411763.3451852.
- 28 Jasmine Y. Shih, Kalina Borkiewicz, AJ Christensen, and Donna Cox. Interactive cinematic scientific visualization in unity. In *ACM SIGGRAPH 2019 Posters, SIGGRAPH ’19*, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3306214.3338588.
- 29 Branislav Sobota. *Computer Game Development*. IntechOpen, Rijeka, August 2022. doi:10.5772/intechopen.97983.
- 30 Branislav Sobota and Emília Pietriková. *Computer Science for Game Development and Game Development for Computer Science*. IntechOpen, Rijeka, November 2023. doi:10.5772/intechopen.1000364.
- 31 Branislav Sobota and Emília Pietriková. The role of game engines in game development and teaching. In Branislav Sobota and Emília Pietriková, editors, *Computer Science for Game Development and Game Development for Computer Science*, chapter 5. IntechOpen, Rijeka, 2023. doi:10.5772/intechopen.1002257.
- 32 Branislava Vranić and Valentino Vranić. Patterns of recreating reality in games. In *Proceedings of 29th Conference on Pattern Languages of Programs, PLoP*, 2022.
- 33 Ursula Wolz, Gail Carmichael, and Chris Dunne. Learning to code in the unity 3d development platform. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education, SIGCSE ’20*, page 1387, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3328778.3367010.
- 34 Haolong Yang, Chunqiang Hu, Guwei Li, and Jingchun Fan. A fire escape simulation system based on the dijkstra algorithm. *Comput. Syst. Sci. Eng.*, 39(3):365–372, 2021.
- 35 Lingxin Yu, Jiacheng Zhang, Xinyue Wang, Siru Chen, Xuehao Qin, Zhifei Ding, and Jiahao Han. Constructing immersive toy trial experience in mobile augmented reality. *Internet of Things and Cyber-Physical Systems*, 4:250–257, 2024. doi:10.1016/j.iotcps.2024.02.001.

Learning Paths: A New Teaching Strategy with Gamification

Filipe Portela  

Algoritmi Centre, University of Minho, Guimarães, Portugal

Abstract

Education's primary objective should be to equip students with the skills and knowledge necessary for professional success. However, the current education system often employs a one-size-fits-all approach, treating all students alike and expecting uniform performance. As students progress, their strengths and weaknesses become more apparent, yet they face the same challenges.

This article addresses this issue by challenging the conventional approach in higher education, demonstrating that it can accommodate diverse student needs and aspirations while maintaining academic rigour and incorporating gamification strategies. Within the TechTeach paradigm, a two-path strategy was developed and implemented with 114 students at the University of Minho.

The first path caters to students aiming for satisfactory grades, considering the subject's minimal relevance to their future careers. The second path is designed for aspiring experts in the subject.

The results indicate a high level of student approval, with 90% expressing satisfaction. Additionally, 49.43% of students achieved final grades between 14 and 18, highlighting the effectiveness of tailored learning pathways in meeting diverse student needs and goals.

2012 ACM Subject Classification Information systems → Information retrieval

Keywords and phrases TechTeach, Information Systems, Gamification, Higher Education, Learning Paths, Personalized Learning

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.13

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

Higher education students in Europe have significantly increased in recent years [13] - Portugal is not an exception [7] – and professors feel powerless to keep them motivated and focused.

As a professor, I have observed that many students are in classes to do the minimum, and typically, they are conditioning their colleagues who want to learn more about the subject and become experts. This suspicion was easily attested by the surveys answered by the students at the beginning of each class since 2016/17, where only 50% want to learn. The first strategy was to add gamification to the classes, and a new paradigm was created: TechTeach [9]. The results were promising, but professors still found a limitation. They were “forcing” students to learn and do things they were not motivated or ready for because their skills were more suitable to other subjects. Gamification is a strategy, but it is not enough. It is essential to innovate the curricula and make the students learn what they like based on the thresholds defined by the professor. Students also have many alternative knowledge sources, usually not scientifically validated, but they consider enough to learn, moving them away from the classes. The increase in students, verified in parallel with the decrease in professors and the respective diminution of accompanying tasks, puts the teaching systems and future professionals at risk. It is a real problem, and universities must find a new way to address it.

So, a new question was raised: “Why are we requesting the same for all the students if we know that is not positive for anyone?”. Instead of creating demotivated professionals, we must demand more from those who want to learn and ensure the minimum knowledge for those with other skills. In the end, all (the students and the professor) win. Analysing the



© Filipe Portela;

licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 13; pp. 13:1–13:12

OpenAccess Series in Informatics



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

13:2 Learning Paths: A New Teaching Strategy

issue stated, a new teaching strategy based on learning paths was proposed and tested for this proposal. This strategy is part of TeachTeach and splits the students into two groups: Traditional and Advanced. Each student must select a group, and the learning content, projects and grades vary according to the path chosen.

After designing the strategy, a case study was conducted during the 2022/2023 academic year at the University of Minho with Web Programming students in the Information Systems Engineering Course. For the first time, the students could select what they desired to be/have after the subject ended. The narrative was earlier defined and they knew from the beginning what they needed to do to get the expected grade. The narrative was earlier defined, and students knew from the beginning what they needed to do to get the expected grade. Two gamification mechanisms, Cards and Pairs Evaluation, were also used during the process in order to keep the students informed about whether their performance was suitable for the chosen path. The experience, conducted with 140 students, represents an innovative practice in engineering education and was revealed to be a success.

This paper describes the strategy and presents the case study and the first results. It is split into six sections. After briefly introducing the paper, the topics and related work are addressed in section two: background. Then, the material and methods section explains the research process. Section four describes the strategy created, while section five presents the case study deployed. Finally, section six remarks on the research and presents future work.

2 Background

This section presents the main topics of the work and some similar works.

2.1 Learning Paths and Outcomes

Pedagogical innovation is essential to teaching success, and learning paths are a simple way of personalizing learning. Personalizing learning is an alternative to the “one size fits all” learning [2] and refers to approaches that generate multiple learning paths considering the individual differences in learning preferences, goals, abilities, knowledge background, and others [5]. A learning path, embodying curriculum design, comprises a series of structured learning activities aimed at helping users attain predefined learning objectives. Such paths hold significant promise in reshaping professors’ approaches to learner support [4, 8]. The Learning paths need to be aligned with Learning outcomes. Learning outcomes are student-centred and describe the measurable skills, knowledge, or values that students should be able to demonstrate after completing a course [16].

2.2 TechTeach and Similar Works

In 2020, Filipe Portela created TechTeach, a new and gamified paradigm to engage students in the classroom [9]. This paradigm combines various digital tools and techniques (e.g. Gamification and project-based learning) to enhance learning results. Since the start, it has been optimized, and new approaches are being created every year, like the creation of assessment exercises using Kahoot! [11]. Motivated by the results, the new strategy presented in this article was improved by the gamification mechanism created at TeachTeach.

Several studies have explored the concept of learning paths. Charzyński discussed educational paths as a form of fieldwork, stressing the necessity of meticulous design and teacher support [3]. In 2021, Ramos [12] introduced a novel learning path model for e-learning systems, leveraging system data to visualize paths and analyze student behaviour. Focused on automatically creating learning paths using open educational resources, a study [15]

showed the techniques and algorithms essential for this process. These studies collectively underscore the significance of meticulous design, data analysis, and practical application in developing and utilising learning paths.

3 Material and Methods

This work is based on the case study method, which allows for exploring a specific occurrence or phenomenon [17]. The process of conducting a case study typically unfolds through various stages, including the development of the design, the collection of data, the analysis of findings, and the interpretation of the results [1]. The specific phases and activities involved in this case study are as follows:

- Design:
 - Understand student's motivation and goals about the subjects
 - Study a new way to increase students' motivation and results
 - Design a new learning strategy that suits students needs
 - Create students' opinion questions to evaluate opinions about the strategy designed
- Implementation:
 - Define the rules and strategy plan
 - Change subject evaluation plan to include strategy designed
 - Choose and implement the gamification mechanism from TechTeach
 - Create a project and put the strategy into practice
- Analysis:
 - Compare students' results with their expectations
 - Evaluate the project results and see the strategy impact
 - Verify students' opinion responses
- Interpretation:
 - Analyse the outcomes to conclude the strategies applied.
 - Evaluating how these results align with the existing learning results of the subject.
 - Exploring the wider implications of this strategy within the context of higher education learning.

The application of case study methodology is justified because it is necessary to study the suitability of creating distinct learning paths while keeping a subject's learning goals intact. This methodology confirmed their applicability to research in real-world settings [6] because it allowed professors to understand students' behaviour and compare it with the achieved results. Regarding the tools, Kahoot was used to receive students' feedback on the subject and their expectations at the beginning, middle, and end. Kahoot [10] was also used for mini-tests. ioEduc [14] was used for gamification mechanisms, such as a card system, quizzes, and peer evaluation.

4 Learning Path Model

This section presents the strategy developed to split teaching into two distinct paths.

Path 1: Basic Knowledge

1. **Focus:** Essential subject knowledge without diving into advanced or recent technologies.
2. **Goal:** Ensure the minimum subject knowledge.

13:4 Learning Paths: A New Teaching Strategy

3. **Ideal for:** Students who require an understanding of the subject for general knowledge, not for professional specialization.
4. **Outcome:** Students gain necessary foundational knowledge sufficient for non-specialists.

Path 2: Advanced Exploration

1. **Focus:** In-depth study that includes the latest technologies and trends in the field.
2. **Goal:** Explore the most recent technologies and do work near reality.
3. **Ideal for:** Students aiming to specialize in the subject and pursue it as a career.
4. **Outcome:** Students develop a comprehensive understanding and are well-prepared for professional roles in the field.

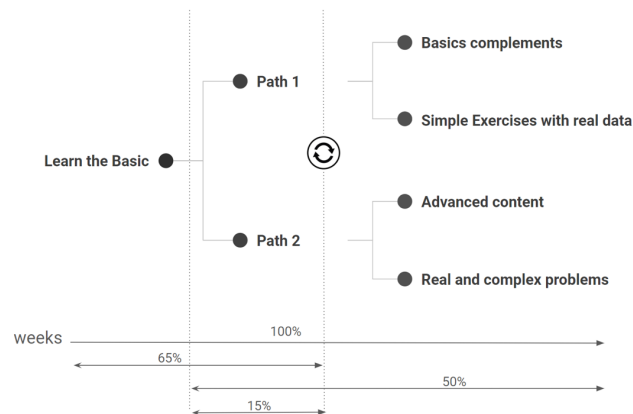
In detail,

- On path 1, students are faced with basic knowledge about the subject. They only need to know the essentials and do not need to experiment with the most recent technologies suitable to the study area. This path is usually ideal for students whose subject area is not their professional bet but who need to know the topics.
- Path 2 is advised to students who want to learn more about the subject and prepare to be professionals in that area. These students are typically encouraged to use the most prominent technologies and explore new trends. Ultimately, they will know more about the subject and learn advanced topics essential to becoming a future expert in the area.

Figure 1 represents an example of the learning path, where Classes have a limited time in weeks (100%). It represents, on average:

- Six weeks – Essential Content;
- Two weeks – Multi-content and path chosen;
- Four weeks – Specific Content of the path chosen.

With this strategy, students can balance their efforts and excel in the areas/subjects more suitable to their profile. It will ensure minimum experience in complementary areas and give them time to improve their knowledge in the main area.



■ **Figure 1** Learning Path Example.

Gamification can be used to motivate students. For example, professors can instigate good performance through badges or points. To best understand what can be used, readers can explore TechTeach.

4.1 Strategy Rules

The strategy allows students to choose one of two learning paths. In practice, it splits the students into two groups. The process is the following:

1. Characterize the learning environment and ask students about their desires, expectations and why they are in the classes;
2. Explain the learning paths narrative to students, their impacts and rules;
3. Show how it works and where it is applied;
4. Create a learning path and agenda for each path;
5. Prepare subject content (matter) according to the paths and ensure that the European Credit Transfer and Accumulation System (ECTS) is kept;
6. Allows students to change the path one time;
7. create a project and add challenges for each path
8. create Gamification methods to valorize those who do more than the expected
9. Grade students according to the path chosen.
10. Ask students for feedback

Students are the only ones responsible for defining their path; professors cannot obligate them to opt for one path or limit their knowledge. For example, if students choose Path 1, they can keep participating in classes directed to Path 2 students. This approach ensures equal opportunity equality following ECT guidelines (time efforts by credits).

Regarding the learning path, some rules must be taken into attention:

- (a) Each learning path should be designed according to the learning outcomes.
- (b) Validate the effort according to the European Credit Transfer and Accumulation System (ECTS).
- (c) Provide clear guidance and resources to support students' progression along the learning path.
- (d) Regularly assess and evaluate student progress to ensure alignment with learning objectives.
- (e) Offer flexibility to accommodate diverse learning styles and preferences.
- (f) Encourage active engagement and participation through interactive learning activities.
- (g) Foster collaboration and peer interaction to enhance learning outcomes.

Gamification can be used to achieve some of the goals above.

4.2 Gamification Mechanisms

The gamification model enhances learning, but the narrative must be highlighted initially. Professors can use gamification to

- (a) Evaluate student progress;
- (b) Promote pairs assessment;
- (c) Help students achieve individual goals.

To a better comprehension of how it can be done, section 5 presents the case study.

5 Case Study

This case study regards web programming subject at the University of Minho with 114 active students:

- **Subject:** Web Programming | **Degree:** Engineering and Management of Information System.

13:6 Learning Paths: A New Teaching Strategy

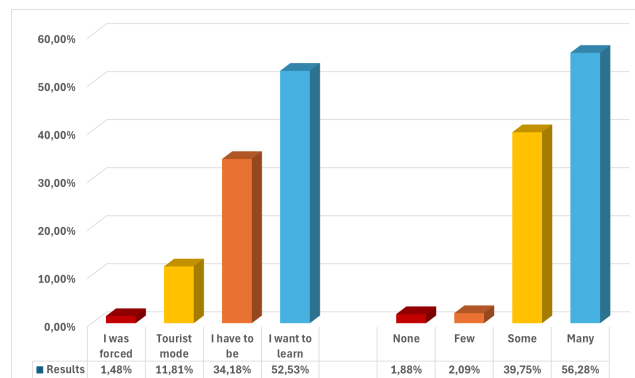
- **Academic degree:** Bachelor | **ECTS:** Five.
- **Academic Year:** Third | **Semester:** Second.
- **Weeks:** Fifteen (twelve of contact).
- **Weekly Classes:** One Theoretic (2hours) and One practice (2hours).
- **Main Scientific Area:** Information Systems and Technologies.

5.1 Design

From the beginning (2018) and in the first class, students were asked to explain why they were in classes and the subject relevant to their future. The questions are

1. Why are you here?
 - a. I was forced
 - b. I want to learn web programming
 - c. I have to be otherwise I won't finish the course
 - d. I'm in tourist mode
2. Importance of PW for your future
 - a. None
 - b. Few
 - c. Little
 - d. Much

Figure 2 presents the results of the questionnaires that were answered from 2018 to 2023 by 478 students.



■ **Figure 2** Students opinion at the subject begin.

Analysing figure 2), it is possible to observe that only one-half of students on average want to learn (52,53%) and consider that Web Programming has much relevance to the future (56,28%). This means that around 250 of the 487 students asked were really interested in the classes.

So, *what should we demand the same from all?* The answer is *“we should not”*. We must ensure the same opportunities for all students. and bring responsibility to them. Based on these answers, the learning path was created and implemented for the first time in the curricular year of 2022/2023. At the beginning of the second semester, professors followed the strategy explained in section 4 and defined the following paths: 1. Traditional and 2. expert.

5.2 Implementation

In terms of implementation, the strategy design brings changes to classes and projects. The class content was split according to the path, and a project was created to consider both developments.

■ **Table 1** Units content overview.

ID	Week	Description	Traditional	Expert
U1	1	Web Programming Introduction	✓	✓
	1	– Web 1.0 to Web 4.0	✓	✓
	1	– Cloud Computing	✓	✓
	2	– Client and Server	✓	✓
	2	– Services on the web (IaaS, PaaS, SaaS, FaaS)	✓	✓
U2	2	Client and Server	✓	✓
U3	3	Static Web Pages (HTML)	✓	✓
U4	4	Page Layout	✓	✓
	4	– Web Templating and Design	✓	✓
	4	– Cascade Style Sheets (CSS)	✓	✓
	4	– Syntactically Awesome Style Sheets (SASS)		✓
	4	– Media Queries		✓
U5	5	Frameworks and CMS (client)	✓	✓
	5	– Bootstrap	✓	✓
	11	– CMS and WordPress	✓	✓
	9,10	– Vue.js		✓
U6	6	Dynamic web pages (client)	✓	✓
	6	– JavaScript	✓	✓
	7	– DOM	✓	✓
	8	– Storage, Cookies and Sessions		✓
U7	8	Client-Server Connection		✓
	8	– API and Web Services		✓
	8	– Fetch		✓
U8	11	New Trends and Going Live	✓	✓
U9	2	Web Tools	✓	✓
	2	– AI to coding	✓	✓
	2	– Project tools	✓	✓
U10	5-15	Project	✓	✓

In the beginning, 83 students (72.81%) chose to be specialists, and 31 selected the traditional path. During the weeks, they could change the learning path. The number of specialists at the end was 72 (63.16%).

5.2.1 Subject Plan

The content of the subject and theoretical classes were split into ten units. Table 1 presents the subject organization according to each learning path.

For example, Unit 1 is directed at all students, but Unit 4 has mixed content, where the SaaS and Media Queries content is more suitable for the expert path. Although this organization, classes and content are available to all, independent of their choice. This option does not block the content but clarifies what is or is not essential.

13:8 Learning Paths: A New Teaching Strategy

In a weekly plan, U1, U2, U3, U4, U9, and partially (U5 and U6) were taught during the first six weeks. Then, U6 (remaining) and U7 were taught parallelly to both paths. In this period, some content was more suitable for path 2 (expert) than one, but all must participate in the classes. Finally, the remaining U5 was available to all, as Vue.js was “obligated” to path 2, and CMS and WordPress were advisable for path 1. U10 represented the project that started in week five and lasted until the end of classes (week 15).

5.2.2 Assessment Methods

This subject comprises three assessment methods: participation, mini-tests and project. In this first experiment, only participation (quiz bonus for each unit) and project differed in each path. Mini-tests will be experimented with in the future. The final grade was calculated using the formula: $15\% * participation(quiz) + 25\% * mini-tests + 60\% * project$

In terms of participation, classes were not obligatory, but students’ attendance was recorded. By attending classes, students can get bonuses and unlock weekly quizzes. Quizzes with bonuses [9] are a gamification strategy that tests the main contents of each unit, prepares students for the mini-tests, and allows students to earn participation points.

■ **Table 2** Project Learning Paths.

Category		Traditional	Expert
Technologies	HTML	✓	✓
	CSS	✓	✓
	JS	✓	✓
	Storage	–	✓
	APIs	–	✓
	Vue.js	–	✓
Front-office		Complete	Complete
Back-office		50%	Complete
Maximum Grade		15	20

The project represented a real problem. The statement was: “*Activities4All is an external entity with a group of people who organise events in different areas. This company coordinates group activities and needs a web interface to help manage events. The project includes two aspects of development: traditional and expert. Each group must choose which path to follow. It is important to note that the choice must be made when the group is created and may be changed until the specifications are delivered (Week 9). Once the choice has been formalized, it is impossible to change it, and each group will be evaluated according to their chosen path. The groups comprised 5 (five) elements (+/-1). The functionalities of each element must be divided between the front office (without login) and the back office (with login). Each group must define the tasks per element and include this information in the team’s specifications.*

Table 2 presents the project rules by learning path being the maximum grade truncated on the defined rules. After selecting the path, groups/students can change it once. They must choose it according to the expected results, as their assessment is limited by the path chosen. A group from the traditional path cannot have more than fifteen, but an expert group can have less than 15 if they don’t meet the guidelines.

Regarding project evaluation, there was one control point (week 9) and a final presentation. At the control point, the professor assessed the quality of project development according to the chosen path and proposal. He advised students about development and informed them whether they were on the right path to the defined target. The professor also gave cards to the highlighted students (good and bad).

This learning path strategy also helps students train soft skills and create personalized learning plans. The decision on the path is made in groups; i.e., if some group students want to get a good grade (expert path), they must convince their colleagues, or they will need to choose another group. After creating the group and choosing the path, students write a working contract explaining their project proposal to the professors and firm their commitment. They explain what they will use (technologies), do (functional and non-functional features), and the desired grade at the end. After finishing this starting process, the practical class professor validated the group contract and started a weekly accompanying and validation of working according to the rules (contract) proposed by the students.

5.2.3 Gamification

Gamification mechanisms were used during the classes to help students achieve their goals. Quiz with bonus allowed students to keep the focus and train weekly content. Attendants were randomly selected to have a weekly bonus (double points in the quiz). After the class, they answered multiple questions about the matter taught on ioEduc. In the end, the best performance (without bonus) had a grade of 20, and the other students with a bonus with equal or higher results also had 20, and the remaining had a relative grade.

Professors used card systems to alert/award students to their performance and show whether groups were doing the work expected in their path. The card system is presented in the table 3.

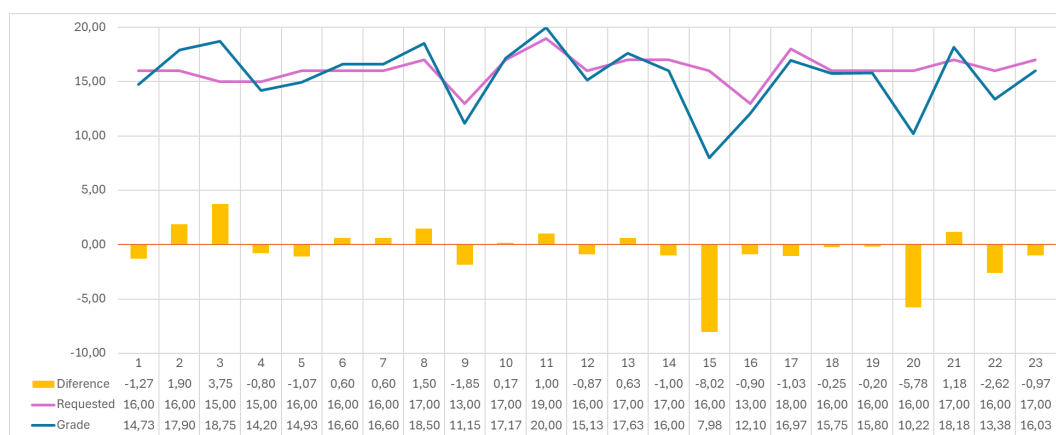
■ **Table 3** Student Evaluation Criteria.

Color	Name	Description
Yellow	First warning	The work performed is below the expected;
Orange	Second warning	The performance is negative, and the student's future at the CUnit is critical;
Red	Student failed	The student did not try to improve their participation. He did not do the minimum acceptable amount, and the level of knowledge is too low, so we cannot do the work.
White	Good Work	The student is working very well, and the professor recognises some extra effort when he is compared to the class.
Blue	Superb Work	The Student is a good example. The commitment level with the CUnit is high, and he deserves to be rewarded.

Each group member also evaluated the project (0-20) and carried out a self-assessment and a hetero-assessment of the group's members. The straight self-assessment was based on N (overall grade) and demonstrated the student's vision regarding their work and the work carried out by the other members of the Group (N, N+1, N+2, N-1, N -2, ...), ensuring that the sum of grades is N. Regarding subject grades, Mini-tests are the only ones not affected by the path until now because they only aim to guarantee that all students have the minimum knowledge required. Class participation gives access to quizzes according to the subject taught, and projects have different goals for each learning path. It also shows that the path chosen does not ensure a grade.

Figure 3 presents an overview of the final grades. In this chart, it is possible to see the grades requested by each group (at the project start and according to the chosen path), the final classification and the difference between the requested and achieved grades. For example, group 11 pointed to 19 but achieved 20 with a difference of one point. Conversely, group 15 did not measure their knowledge/expectations very well and failed (grade <8).

13:10 Learning Paths: A New Teaching Strategy



■ **Figure 3** Grades by Project.

5.3 Analysis of the results

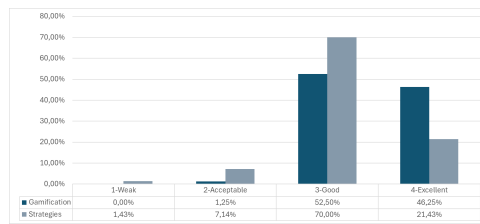
The strategy can be measured in two ways: students' opinions about it and the groups' final grades. Top projects showed significant improvement compared to previous ones, while others performed similarly but with less effort (more suitable to ECTS guidelines). Thus, the strategy effectively helped develop more skilled specialists in Web Programming. However, accurately quantifying this difference is challenging due to changes in the credit value of the Web Programming subject, reduced from 10 ECTS to 5 ECTS, with the introduction of a new curriculum plan in the academic year 2022-23. The results show the relevance of choosing the right path and being honest with the team and professors. The complexity of a subject like web programming requires extra effort to get higher grades. After observing the group's grades, it was easy to see that most groups should select the traditional path.

These observations allowed professors to understand the reality and alert the students in the new academic year(2023/24) to the strategy's effectiveness. Based on this, students were more realistic in the new academic year (2023/24), and the number of experts was reduced to (10/18). However, this subject is running, and the final impact of it cannot be measured (an extended version of the paper will be prepared showing the results).

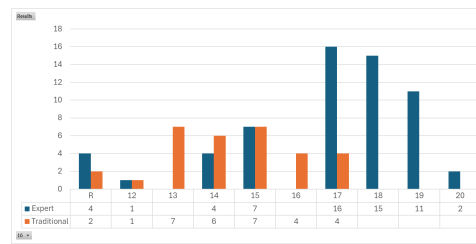
Regarding students' feedback, they were asked several questions, two of which related to the strategy presented in this paper:

1. How do you evaluate the gamification mechanisms?
2. How do you assess the adequacy of strategies and methodologies adopted by the teaching team?

Figure 4a shows the results, where it is possible to observe that the percentage of students that do not consider the gamification mechanism or teaching strategies well-suitable (weak or acceptable) is less than 10%. As shown in figure 4b good students on the traditional path can achieve a final value higher than fifteen, and "expert students" can also have low grades. Only students on the expert path had grades of excellence(higher and equal to heightening). In conclusion, there is an evident separation between the students being 23.50% with grades less than 14 and 25,93% with grades higher or equal to 18.



(a) Subject evaluation.



(b) Students final grades.

■ **Figure 4** Evaluation and Final Grades.

6 Conclusion

This article introduces a novel teaching strategy for Higher Education framed in the TechTeach paradigm. In response to the need for enhanced teaching quality and improved student outcomes, professors developed two distinct learning paths to provide tailored learning experiences and avoid the one-size-fits-all approach. These paths were carefully designed to impact various aspects of the subject program, including exercises and project development.

Each path serves specific objectives: the first path emphasizes foundational understanding, ensuring that students know the subject's fundamental concepts, while the second path is tailored for students aspiring to specialize in the subject and pursue a career in the field. This strategy suggests that 50% of the subject time has the same content for both paths, and 15% has multiple contents (during this time, students can select/change their path). Finally, the last 35% must have specific content according to the designed path.

To validate the effectiveness of the proposed strategy, a case study was conducted during the academic years 2022/2023 in an Engineering course at the University of Minho. The subject chosen for the study was Web Programming, and 114 students participated. Over 15 weeks, students learned the basics and followed their chosen path, doing exercises and completing a project aligned with their learning objectives. Gamification played a relevant role through the use of quiz with bonus and card systems.

Upon evaluation, 90% of the students expressed satisfaction with the strategy, highlighting its positive impact on their learning experience. Notably, the final grades obtained by students were aligned with their chosen paths and corresponding efforts. Students following the second path achieved higher grades on average, reflecting their deeper engagement and commitment to the subject. However, it was observed that students on the second path (exported) who exerted less effort received lower grades than those on the traditional path.

Moving forward, the strategy continues to undergo refinement based on feedback from both professors and students. The 2023/2024 academic year results will be carefully analyzed to optimize the plan further. Additionally, future endeavours will explore implementing this strategy across a broader range of subjects to provide tailored and compelling learning experiences to students across various disciplines.


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 Barbara Bray and Kathleen McClaskey. A step-by-step guide to personalize learning. *Learning & Leading with Technology*, 40(7):12–19, 2013.

13:12 Learning Paths: A New Teaching Strategy

- 3 Przemysław Charzyński, Zbigniew Podgórski, Dariusz Brykała, Aleksandra Zaparucha, and Sylwia Barwińska. Fieldwork on selected educational paths. In *the Vistula banks – fieldwork in bilingual education*, 2015.
- 4 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.
- 5 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.
- 6 Kathleen M Eisenhardt. Building theories from case study research. *The Academy of Management Review*, 14(4):532–550, 1989.
- 7 Portugal Gov. Portugal beats records of students in higher education – portugalgov.pt. <https://www.portugal.gov.pt/en/gc23/communication/news-item?i=portugal-beats-records-of-students-in-higher-education>. [Accessed 15-04-2024].
- 8 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.
- 9 Filipe Portela. Techteach – An innovative method to increase the students’ engagement at classrooms. *Information*, 11(10), 2020. doi:10.3390/info11100483.
- 10 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 (OASISs)*, pages 8:1–8:12, 2023. doi:10.4230/OASISs.ICPEC.2023.8.
- 11 Filipe Portela. A new approach to perform individual assessments at higher education using gamification systems. In *4th International Computer Programming Education Conference (ICPEC 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.
- 12 David Brito Ramos, Ilmara Monteverde Martins Ramos, Isabela Gasparini, and Elaine Harada Teixeira de Oliveira. A new learning path model for e-learning systems. *Int. J. Distance Educ. Technol.*, 19:34–54, 2021.
- 13 Evan Schofer, Francisco O Ramirez, and John W Meyer. The societal consequences of higher education. *Sociology of Education*, 94(1):1–19, 2021.
- 14 Miguel Silva, Diogo Ferreira, and Filipe Portela. IoEduc – Bring Your Own Device to the Classroom. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *Open Access Series in Informatics (OASISs)*, pages 23:1–23:9, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASISs.ICPEC.2020.23.
- 15 Anna Sirén and Vassilios Tzerpos. Automatic learning path creation using oer: A systematic literature mapping. *IEEE Transactions on Learning Technologies*, 15:493–507, 2022. URL: <https://api.semanticscholar.org/CorpusID:251077865>.
- 16 USC. Learning Outcomes – Center for Teaching Excellence | University of South Carolina – sc.edu. https://sc.edu/about/offices_and_divisions/cte/teaching_resources/course_design_development_delivery/learning_outcomes/index.php. [Accessed 15-04-2024].
- 17 Robert K Yin. *Case study research and applications: Design and methods*. SAGE Publications, 2017.

Code Review for CyberSecurity in the Industry: Insights from Gameplay Analytics

Andrei-Cristian Iosif ✉ 


Universität der Bundeswehr München, Germany
Siemens AG, München, Germany

Ulrike Lechner ✉ 

Universität der Bundeswehr München, Germany

Maria Pinto-Albuquerque ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Tiago Espinha Gasiba ✉ 

Siemens AG, München, Germany

Abstract

In pursuing a secure software development lifecycle, industrial developers employ a combination of automated and manual techniques to mitigate vulnerabilities in source code. Among manual techniques, code review is a promising approach, with growing interest within the industry around it. However, the effectiveness of code reviews for security purposes relies on developers' empowerment and awareness, particularly in the domain-specific knowledge required for identifying security issues. Our study explores the use of *DuckDebugger*, a serious game designed specifically to enhance industrial practitioners' security knowledge for code reviews. By exploring analytics data collected from game interactions, we provide insights into player behavior and explore how the game influences their approach to security-focused code reviews. Altogether, we explore data from 13 events conducted in the industry together with 224 practitioners, and derive metrics such as the time it takes participants spend to reviewing a line of code and the time required to compose a comment. We offer empirical indicators on how serious games may effectively be utilized to empower developers, propose potential design improvements for educational tools, and discuss broader implications for the use of Serious Games in industrial settings. Furthermore, our discussion extends to include a discussion outlining the next steps for our work, together with possible limitations.

2012 ACM Subject Classification Security and privacy → Software and application security; Applied computing → Collaborative learning; Applied computing → E-learning; Security and privacy → Software security engineering

Keywords and phrases Cybersecurity, Code Review, Developer Empowerment

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.14

Funding This work is partially financed by Portuguese national funds through FCT – Fundação para a Ciência e Tecnologia, I.P., under the projects FCT UIDB/04466/2020 and FCT UIDP/04466/2020. Furthermore, the third author thanks the Instituto Universitário de Lisboa and ISTAR, for their support. We acknowledge funding for project LIONS by dtec.bw. Andrei-Cristian Iosif and Tiago Gasiba acknowledge the funding provided by the Bundesministerium für Bildung und Forschung (BMBF) for the project CONTAIN (FKZ 13N16585).

1 Introduction

In recent years, the cybersecurity community has witnessed how purposefully introduced software vulnerabilities can be weaponized into sophisticated supply chain attacks. One such example of this is the Lazarus group [10] Advanced Persistent Threats (APT). Such attacks abuse trust relationships and result in malicious code being injected into legitimate software.



© Andrei-Cristian Iosif, Ulrike Lechner, Maria Pinto-Albuquerque, and Tiago Espinha Gasiba; licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 14; pp. 14:1–14:11

OpenAccess Series in Informatics



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

One way to combat this is through rigorous code review processes. Reviews can serve as an additional security measure against targeted breaches, and also catch less malignant security issues from legitimate developers, that might otherwise be accidentally overlooked.

Industrial software systems, particularly those that manage critical infrastructure, are an area of special concern when accounting for cybersecurity - with stakes including economic concerns and public safety. Consequently, such systems must adhere to stringent standards and compliance regulations, such as the IEC 62433 [7].

One way to achieve security and compliance in an industrial environment is a bottom-up approach that begins with empowering developers. Training programs focused on security can significantly enhance developers' ability to detect and mitigate vulnerabilities earlier in the development process.

One innovative tool in this effort is DuckDebugger, a serious game (SG) designed specifically to teach software developers to perform code reviews with a focus on cybersecurity. We use SGs as a medium for information delivery, as the authors possess extensive knowledge in this area. The proposed game offers an environment where developers are asked to review snippets of vulnerable code, and are able to consult the output from security tools. Through engaging with our game, developers can hone their skills in a practical and realistic environment.

This paper presents the results from analyzing gameplay data from events conducted in industrial contexts. Our objective is to understand the interactions taking place in the DuckDebugger. By exploring player behavior, our analysis takes a first step towards measuring how players engage in code review, in order to gauge how the game sizes towards empowering practitioners. The insights from our study show the potential for serious games to improve security training in software development. Furthermore, we identify design implications for our game and optimization of the game experience for the participants.

The paper is organized as follows: Section 2 will introduce related work relevant to our study. In Section 3, we present our work's context and showcase the game artifact's relevant aspects. Section 4 presents our findings, exploring their implications and discussing potential limitations. Lastly, Section 5 lays out the conclusions of our work, together with discussing further steps.

2 Related Work

This section introduces related work in the two main areas of interest under which our game falls, namely cybersecurity training through serious games, and code review.

2.1 Serious Games for Cybersecurity Education

A serious game is defined by Susi et al. as “a game designed with a primary objective other than pure entertainment” [18]. Furthermore, indicated they are effective in disseminating takeaways to their users, as shown in an experience report of Namin et al [17] and through a literature review conducted by Hendrix et al [4].

Roepke et al. provide a comprehensive overview of SGs focused on cybersecurity, and explore how many games available for end-users without prior knowledge exist, as well as whether they teach sustainable knowledge and skills [15]. Their findings show that although there is a growing number of SGs for cybersecurity, the content they target is often lacking relevance, focusing rather on factual knowledge without context.

Capture The Flag (CTF) competitions, a popular form of competitive serious games in cybersecurity, have been extensively studied for their educational value. Research by Culliane et al. [2] and Švábenský et al. [20] provide an overview of CTFs seen through the educational angle.

Unlike traditional CTFs and challenges thereof, which often emphasize offensive practices, the game discussed in this work focuses on defense and patching.

Previous work done by the authors on this topic explores the requirements of defensive cybersecurity SGs [3], defining 15 challenge requirements suitable for industrial requirements. The game is developed using an iterative design approach based on Design Science Research (DSR) principles, as outlined by Sein [16]. In this second iteration, we have incorporated feedback from participants, specific to DSR guidelines, and have successfully met 11 out of the 15 requirements. We plan to address the remaining requirements, should the participants deem it necessary.

Švábenský et al. [19] review how cybersecurity training data can improve educational research. Their work offers an overview on how training data can to understand and support cybersecurity learning. However, their approach notes that it only targets academic settings and exercises with an offensive focus.

2.2 Industrial Code Review

Regarding standardization within the industry, Moyon et al. [13] delve into the practical challenges related to integrating security into an agile software development and explore how the requirement for industrial compliance shapes this process.

The IEC 62443 series of standards was developed to cover the security of industrial automation and control systems throughout their lifecycle. Relevant to our work, the IEC 62443-4-1 [7] and IEC 62443-4-2 [8] standards underscore how code review in the development lifecycle is important for cybersecurity, and the emphasize the need for reviewers to possess specialized knowledge on performing secure code review.

Other standards of interest to our work are: ISO/IEC 20246 [6], which provides a generic framework for work product reviews applicable across various organizational roles, and the ISO/IEC TR 24772-1 standard [9], which offers programming-language agnostic guidance on avoiding coding vulnerabilities.

We seek to address the most prevalent vulnerabilities encountered in software by introducing vulnerabilities. To this end, we draw from the the CWE Top25 [12] and OWASP Top10 [14], as these resources are well-known standards, with industry-wide acceptance.

MacLeod et al. show that code reviews often do not find critical bugs that would block a code submission, instead highlighting issues related to long-term code maintainability [11]. They argue that effective code reviews require specific skills and that the social dynamics within teams significantly influence the reviewing process. They suggest that the current practices in code reviews are often inefficient and call for a more sophisticated approach to integrating code reviews into software engineering workflows. Building on these findings, our work seeks to address these deficiencies by employing the DuckDebugger as a means to enhance practitioners' skills necessary for effective code review.

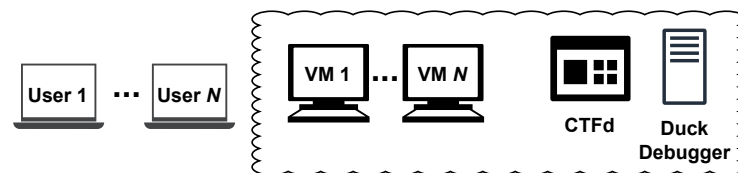
Bosu et al. analyze the effectiveness of code review comments in Microsoft projects and find that useful feedback improves both code quality and developers' skills [1]. They show that the usefulness of comments tends to increase with the reviewer's experience but is negatively correlated with the size of the review. The study provides recommendations for increasing the effectiveness of code reviews, such as optimizing the number of files in a review and enhancing reviewer experience through targeted training. We integrate their findings in the design of the game, which we use for training industrial developers.

3 Methodology

In this section, we describe the methodology behind the collected data. First, we present the industrial context in which the DuckDebugger is deployed. We proceed by introducing the game artifact itself, highlighting the important elements of our data.

3.1 Context

Practitioners are invited to try out our serious game as part of a workshop on developer empowerment on the topic of cybersecurity. Such workshops consist of two parts: first, a classroom-style presentation of common attacks and mitigations (1-2 days, depending on the event), followed by a full-day CTF-style event. On the last day, participants organize themselves in teams, where they compete against each-other by solving various types of security-related challenges. One of the types are the code review challenges discussed in this work. Although we integrate the delivery of our game in a CTF setting for our events, this is not strictly necessary, as the game is designed to be a self-standing application.



■ **Figure 1** Architecture.

Figure 1 presents an overview of the infrastructure behind the hands-on part of the event. Participants only require a laptop, with which they can connect to a fully provisioned virtual machine for all the available challenges. Upon solving a challenge, players redeem points for their team in a dashboard (CTFd). We utilize the dashboard mechanic of the CTF genre to foster competitiveness among participants.

When the event concludes, the winning team is announced and congratulated. This is followed by a gathering feedback from the participants (through surveys and semi-structured interviews), and a wrap-up session which discusses challenges which the players found difficult.

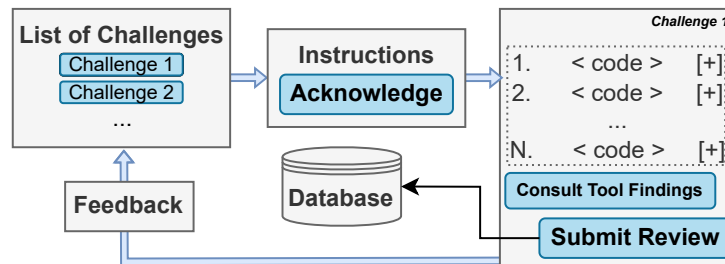
At the time of writing, the DuckDebugger has been trialed across 13 industrial events, with a total of 224 participants. An overview of these events is presented in Table 1.

■ **Table 1** Events Overview.

Event Number	1	2	3	4	5	6	7	8	9	10	11	12	13
Date	05.2023	05.2023	06.2023	11.2023	11.2023	11.2023	12.2023	12.2023	01.2024	01.2024	02.2024	02.2024	02.2024
Place	Online	Online	United Kingdom	Germany (Hybrid)	Germany	China	Online	China	Online	Germany	Germany	Online	Germany (Hybrid)
Number of Participants (Total: 224)	14	7	16	9	20	24	30	20	16	15	24	17	12

3.2 Game Artifact

The focus of this work, the DuckDebugger platform, is a self-standing web application and is hosted as a separate server in the same cloud environment as the participants' virtual machines, in an AWS Virtual Private Cloud. This design choice was taken to ensure that the game can offer flexibility in how it can be delivered.



■ **Figure 2** Artifact Overview.

Figure 2 introduces an interaction diagram for the game: The user starts by viewing a welcome screen with platform instructions. After this, they choose from various code review challenges categorized by programming language and application type, such as web or embedded programming. They then receive a source code snippet with vulnerabilities and poor practices, which they are asked to annotate with comments about security findings.

At the time of writing, participants can choose from 28 exercises that cover 4 programming languages: Java, C#, Python, and JavaScript. The vulnerabilities in our game’s snippets include common security malpractices, from the CWE Top25 [12] and OWASP Top10 [14].

Figure 3 shows the main game interface, organized in tabular form, with columns for user comments, the code under review, and the intended solution (to be displayed after a user successfully solves a challenge). Users input their comments and can reference SAST tool findings alongside the code. The integration of SAST aims to enhance users’ skills with real-world tools and educate them on recognizing false positives/negatives.

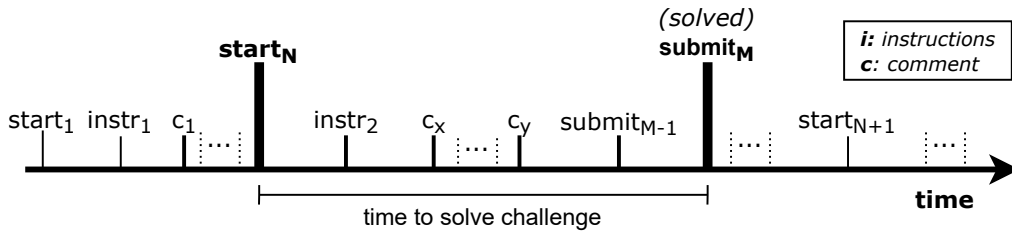


■ **Figure 3** Interface.

Upon submitting their review comments, the DuckDebugger evaluates how many vulnerabilities a player has found, and records the interactions with the platform into a database for later analysis. If a player identifies at least 50% of the vulnerabilities, they are presented the full solution as feedback. This benchmark is based on prior research from the authors [5], where it was found that trainees identify, on average, about half of the vulnerabilities detected by experts.

4 Results and Discussion

To help understand the player behavior analysis, we consider the following: Players have the autonomy to engage with, resolve, or abandon any challenge at will, in any preferred order.



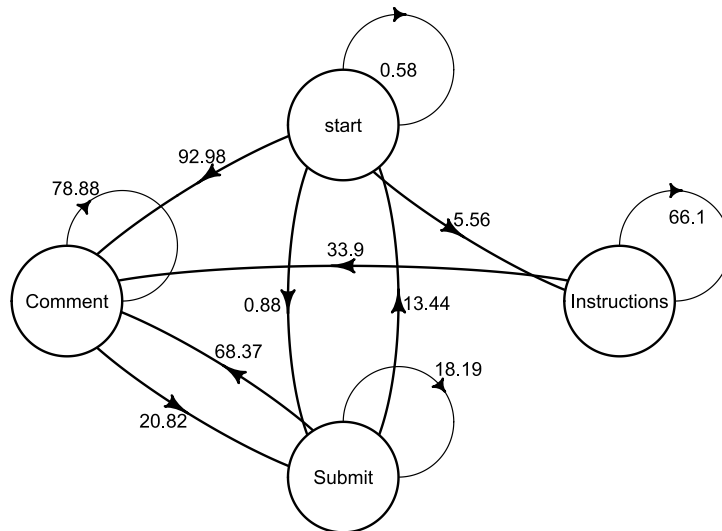
■ **Figure 4** Timeline of interactions for a (user,challenge) pair.

An example timeline of interactions is presented in Figure 4, illustrating a *single user's* interaction sequence for a *single challenge*. Possible interactions for a user are: starting the challenge (**start**), consulting the instructions (**i**), writing a comment (**c**), and submitting their solution attempt (**submit**).

The figure has been simplified for clarity – the collected data covers the multiple users' interactions, for multiple challenges, where all entries of users and challenges are chronologically interlaced.

The time it took a given user to solve a specific challenge is computed as the timestamp difference between the first occurrence of a **submit** interaction marked as *solved* by the platform, and the first **start** interaction which precedes it.

4.1 Player Behavior Model



■ **Figure 5** Platform Interactions: Player behavior.

Based on the collected players' interactions, we can construct a transition matrix between possible actions in the game. Figure 5 models the player behavior and highlights the transition probabilities between actions. Most notably in this figure, we can observe that:

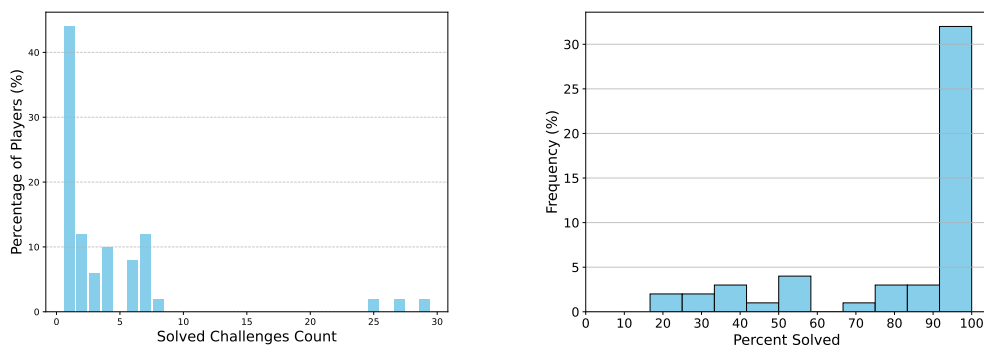
- 66,1% of players re-check the instructions. This could likely be due to habitual interactions with similar interface elements, suggesting a conditioned response to dismiss such overlays without thorough engagement. Empirical surveys, however, indicate that the majority find the instructions clear. Additionally, the absence of a **submit** → **instructions** transition indicates that the task is well understood.

Similarly, the **start** → **instructions** transition occurs because users are accustomed to dismissing pop-ups quickly. This would suggest a need for different UI choices, such as waiting for a timer to expire before being able to click away the instructions pop-up.

- 18.19% of users press the submit button repeatedly (**submit** → **submit**). This would indicate a need for a visual cue to confirm that the button has been activated successfully.
- **start** → **start** interactions account for 0.58% of interactions, possibly stemming from accidental page refreshes.
- One in five comments is immediately followed by a submission attempt – 20.82% on the **comment** → **submit** transition. If this percentage were lower, it would suggest that users are overthinking their submissions. Conversely, a higher percentage might indicate players trying to exploit the system for a competitive edge by rapidly resubmitting their comments.

4.2 Solved Challenges Counts and Percentages

Another perspective on player behavior involves examining the number of challenges each player successfully completes. It is useful to observe whether players abandon challenges before finishing them.



(a) Count Solved.

(b) Percent Solved.

■ **Figure 6** Metrics: Solving Challenges.

Figure 6a illustrates that the majority of participants solve fewer than 10 challenges, with a noticeable peak at just one challenge. This trend may be attributed to the specific conditions of the event, such as time constraints and the diversity of available challenges.

Figure 6b shows that more than 30% of players solve all challenges they start, and more than 50% solve half or more. This demonstrates a commendable level of persistence among the players, indicating that they generally complete the challenges they begin.

4.3 Time to solve a challenge

Based on the time it takes players to solve a challenge, we derive and examine the duration participants spend reviewing a line of code (LoC) and the time required to compose a comment. As challenges have code snippets of varying length, the times are normalized to each challenge's LoC. We group our data by programming language, average it, and present our findings in Table 2.

■ **Table 2** Times.

Programming Language	Avg. time to review one Line of Code (s.)	Avg. time to write one comment (s.)
csharp	12.56	4.87
go	3.44	1.94
java	1.57	1.67
javascript	3.47	1.95
python	12.57	9.32

In the case of Java, the lower numbers can be attributed to the language’s syntactic noise (*boilerplate code*), which may result in a lower time per LoC. Additional influencing factors include the diverse backgrounds of the participants (i.e. few proficient Python programmers across all events), which could also play a significant role in the observed results.

The correlation coefficient between the two metrics presented in Table 2 is 0,873. This indicates that the time participants take to comment is relatively consistent across different programming languages. Such strong correlation could indicate that the measured times are an indicator of the players’ proficiency levels with the programming languages, rather than an indicator of review times being dependent on the programming language. Nonetheless, further studies would be required to establish a definitive conclusion in this direction.

Furthermore, we can explore how the participants’ solving times evolve between consecutive challenges. Table 3 presents the median time it takes a player to solve a challenge. We truncate our findings at 7 challenges, as Figure 6a shows that few players only solve more than 7 challenges, which would challenge the statistical relevance of these findings. Although our data is truncated, preliminary findings indicate a non-linear improvement pattern.

■ **Table 3** Time to solve a challenge.

Number of Solved Challenges	1	2	3	4	6	7
Median Time To Solve (s.) (normalized to challenge LoC)	7.72	18.87	3.29	7.55	6.48	3.14

We can observe that there is no steady increase or decrease in solving time, between the number of challenges a players solves. Based on our experience in designing serious games, this non-linear time pattern could point to multiple insights and/or reflect our design choices: the challenges have variable difficulty, and participants have diverse problem solving approaches as they learn and adapt to the platform. Had there been a steady increase, this would indicate that the challenges result in tiredness or a plateau in learning efficiency. Conversely, the lack of a steady decrease in the solve time indicates that the DuckDebugger has a varied repertoire of code review challenges, where players cannot *game the system* through recalling the solutions to the challenges they previously encountered and solved.

4.4 Knowledge Exchange

The nature of the event has players organized in teams which compete for points across multiple challenge categories, some of which are code review challenges delivered through the DuckDebugger. We observed during the events that most, if not all teams competitively optimize their strategy by splitting up across challenge types. With proper intra-team coordination, this would translate into one player per team solving the DuckDebugger challenges at any given time.

Nonetheless, while conducting the events, we observed players of the same teams often pause to share newfound knowledge from the challenges. We back this statement through an observation in our aggregated data: 28.2% of players revisit a challenge after having solved it. This indicates to us that more than a quarter of the participants likely exchange the gained knowledge with their team-mates, to share newly gained information.

Since the dashboard interface indicates to the other players of a team that a challenge is already solved, this would rule out most accidental revisits of a challenge. Furthermore, there is no competitive advantage in solving a challenge again.

This finding about players' knowledge exchange underlines the relevance of code review as an industrial practice fostering developer empowerment, as it shows an organic tendency towards knowledge sharing even under a competitive setting. We thus reinforce the importance of collaborative learning and the dissemination of best practices across the workforce, and show that code review is a good vector towards achieving this. This observation reinforces the value of code review not only as a skill but also as a catalyst for fostering developer empowerment and the spread of best practices across the industry.

The competitive yet collaborative environment created by the DuckDebugger game illustrates how serious games can bridge the gap between individual learning and team-based knowledge dissemination. This aspect is crucial in real-world applications where cybersecurity is a collective responsibility. By integrating these insights into future game iterations, DuckDebugger can further enhance its impact on cybersecurity training.

4.5 Discussion on Validity and Limitations

The study on the use of DuckDebugger for code review training in cybersecurity presents potential threats to validity, particularly in the context of generalization and external validity.

First, the events are conducted in a controlled setting, which may not accurately reflect developers' typical working environment. This difference might influence the behavior and performance of participants, as developers in a competitive, time-constrained event might interact with the game differently compared to a regular work setting.

The analysis of gameplay data might overlook deeper cognitive and learning processes involved in vulnerability identification and mitigation, focusing primarily on observable metrics such as time spent and challenge completion rates. Furthermore, addressing the long-term impact of the game is unfeasible in an industrial setting, as developer teams and individual responsibilities shift with time. Learning trends, solving efficiency and proficiency can thus only be observed within the scope of individual events. Nevertheless, preliminary findings show neither a learning plateau nor a saturation of gained knowledge.

Furthermore, the event's structure, which organizes individual players into teams, likely impacted data collection in terms of sample size. We observed during our the moderation of our event that users within the same team tend to split between challenge types, to gain more points in total. Due to how data is anonymized, we cannot present a percentage of players and teams that choose to follow this strategy. As we observed players dividing their efforts across different challenge types to optimize team points, this typically resulted in not all people from a team addressing the review challenges. Nonetheless, as the data is collected at an individual level, and not aggregated by teams, individual performance metrics are accurately represented, independent of team dynamics.

As a general note, although our game has been embedded in a CTF setting, though this is not strictly necessary. The implementation of the game makes it independent of a CTF setting. We believe our findings to be similar, had the game been introduced under a different structure.

Nonetheless, our conclusions align with previous industrial research around serious games. This, coupled with the typical limitations inherent to design science studies carried out in the industry, leads us to consider that our findings should not significantly diverge, had the event been restructured to optimize for data collection instead of learning outcome.

5 Conclusions

This study explores the DuckDebugger game as a tool for training developers in cybersecurity-focused code review through gameplay analytics. Our results extrapolate a player behavior model which can be used by practitioners to design similar games, and explore metrics related to players solving challenges. Notably, the game design, which incorporates elements of competition and immediate feedback, fosters engagement and learning among participants.

Notably, the finding that over a quarter of participants revisit challenges even after solving them highlights an organic tendency towards knowledge sharing within teams, emphasizing the collaborative nature of learning, even in competitive settings.

Our findings suggest that serious games like DuckDebugger can enhance cybersecurity education among industrial developers. This aligns with the results of previous related work, reinforcing the importance of tailored training tools in improving cybersecurity skills (i.e. code review) and fostering collaborative learning environments. The interactive and practical nature of our game provides a hands-on learning environment.

Our game contributes to academic research by exploring the use of use of SGs in an industrial setting, specifically centered around empowering developers through code review for cybersecurity. We follow DSR principles and focus on a *defensive* approach to disseminating cybersecurity knowledge, by targeting mitigation techniques in our game, instead of exploitation of vulnerabilities.

Future work aims to address the identified refinement needs, including game's design based on the collected feedback and interaction analytics. Additionally, we plan to test additional scenarios and further evaluate the utility of the game as perceived by the participants.

References

- 1 Amiangshu Bosu, Michaela Greiler, and Christian Bird. Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 146–156, Florence, Italy, 2015. IEEE. doi:10.1109/MSR.2015.21.
- 2 Ian Cullinane, Catherine Huang, Thomas Sharkey, and Shamsi Moussavi. Cyber security education through gaming cybersecurity games can be interactive, fun, educational and engaging. *J. Comput. Sci. Coll.*, 30(6):75–81, June 2015.
- 3 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.
- 4 Maurice Hendrix, Ali Al-Sherbaz, and Victoria Bloom. Game based cyber security training: are serious games suitable for cyber security training? *International Journal of Serious Games*, 3(1), March 2016. doi:10.17083/ijsg.v3i1.107.
- 5 Andrei-Cristian Iosif, Tiago Espinha Gasiba, Ulrike Lechner, and Maria-Pinto Albuquerque. Raising awareness in the industry on secure code review practices. In *CYBER 2023: The Eighth International Conference on Cyber-Technologies and Cyber-Systems*, pages 62–68. IARIA, September 2023.
- 6 ISO/IEC 20246:2017. Software and systems engineering – Work product reviews. Standard, International Organization for Standardization, Geneva, CH, 2017.

- 7 ISO/IEC 64223-4-1:2018-1. ISO/IEC 62443-4-1:2018 Security for industrial automation and control systems – Part 4-1: Secure product development lifecycle requirements. Standard, International Organization for Standardization, Geneva, CH, January 2018.
- 8 ISO/IEC 64223-4-2:2019-12. Security for Industrial Automation and Control Systems – Part 4-2: Technical Security Requirements for IACS Components. Standard, International Electrical Commission, Geneva, CH, January 2019. ISBN 978-2-8322-6597-0.
- 9 ISO/IEC TR 24772-1:2019. Programming languages – Guidance to avoiding vulnerabilities in programming languages – Part 1: Language-independent guidance. Standard, International Organization for Standardization, Geneva, CH, 2019.
- 10 Peter Kálnai. Lazarus campaigns and backdoors in 2022-2023. In *Proceedings of the Virus Bulletin International Conference*, London, United Kingdom, October 2023.
- 11 Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. Code reviewing in the trenches: Challenges & best practices. *IEEE Software*, 35(4):34–42, 2017.
- 12 MITRE Corporation. CWE Top 25 Most Dangerous Software Weaknesses. <http://bit.ly/mitre25>, 2023. Online, accessed 2023.07.24.
- 13 Fabiola Moyon, Daniel Mendez, Kristian Beckers, and Sebastian Klepper. How to integrate security compliance requirements with agile software engineering at scale? In Maurizio Morisio, Marco Torchiano, and Andreas Jedlitschka, editors, *Product-Focused Software Process Improvement*, pages 69–87, Cham, 2020. Springer International Publishing.
- 14 OWASP Foundation. OWASP Top10:2021. <https://owasp.org/Top10>, 2021. Online, accessed 2023.07.24.
- 15 Rene Roepke and Ulrik Schroeder. The problem with teaching defence against the dark arts: A review of game-based learning applications and serious games for cyber security education. In *Proceedings of the 11th International Conference on Computer Supported Education*. SCITEPRESS – Science and Technology Publications, 2019. doi:10.5220/0007706100580066.
- 16 Maung K. Sein, Ola Henfridsson, Sandeep Puro, Matti Rossi, and Rikard Lindgren. Action Design Research. *MIS Quarterly*, 35:37–56, 2011.
- 17 Akbar Siami Namin, Zenaida Aguirre-Muñoz, and Keith Jones. Teaching cyber security through competition an experience report about a participatory training workshop. In *7th Annual International Conference on Computer Science Education: Innovation & Technology (CSEIT 2016)*, CSEIT. Global Science & Technology Forum (GSTF), October 2016. doi:10.5176/2251-2195_cseit16.39.
- 18 Tarja Susi, Mikael Johannesson, and Per Backlund. Serious games: An overview. Technical report, IKI Technical Reports, 2007.
- 19 Valdemar Švábenský, Jan Vykopal, Pavel Čeleda, and Lydia Kraus. Applications of educational data mining and learning analytics on data from cybersecurity training. *Education and Information Technologies*, 27(9):12179–12212, May 2022. doi:10.1007/s10639-022-11093-6.
- 20 Valdemar Švábenský, Pavel Čeleda, Jan Vykopal, and Silvia Brišáková. Cybersecurity knowledge and skills taught in capture the flag challenges. *Computers and Security*, 102:102154, 2021. doi:10.1016/j.cose.2020.102154.

Implementing a Digital Twin for a Robotic Platform to Support Large-Scale Coding Classes

Michael Heeney ✉ 

Department of Computer Science, Middlesex University, London, UK

Kelly Androutsopoulos ✉ 

Department of Computer Science, Middlesex University, London, UK

Franco Raimondi ✉ 

Gran Sasso Science Institute, L'Aquila, Italy

Abstract

Constructionist learning involves learners that are actively engaged in the construction of an entity that reflects the learning achievements. When learning to code, such a physical entity can take the shape of a robot, or of a robotic arm, or any other hardware device that is used to manifest the effect of the code that students are writing. Hardware devices have been used in primary and secondary schools, and also in Higher Education. Unfortunately, the use of hardware devices is limited as it does not scale to large cohorts and requires a physical space for face-to-face teaching.

In this paper we introduce a digital twin for a robotic platform to replicate a classroom setting used for teaching first year undergraduate Computer Science students. We describe the architecture of the system and its implementation.

2012 ACM Subject Classification Applied computing → Interactive learning environments

Keywords and phrases digital twin, introductory programming, constructionism, robotics, computer science education

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.15

1 Introduction

According to constructionism, learning “*happens especially felicitously in a context where the learner is consciously engaged in constructing a public entity, whether it’s a sandcastle or a theory of the universe*” [24]. The case of *learning to code* is a special situation in which:

- 1) students need to learn a *static* syntax to encode an algorithm to achieve a desired goal, but also
- 2) students need to learn how to associate a dynamical behaviour to the code they have written, to make sure not only that the code is syntactically correct, but also that it achieves its desired goals. Typically, the dynamical behaviour of code is often hidden and students can only observe its final output, unless they use a debugger.

An approach based on constructionism can provide *physical manifestations* for code that is easier to understand than traces from a debugger, and thus help in providing a better strategy for students to understand the dynamical behaviour (and the consequences) of the code they write, in that students can observe not only the final state of a program, but also all the intermediate steps taken, if these correspond to states of a robot or any other hardware platform. Several approaches have investigated strategies to support constructionism in the setting of learning to code, see for instance [19] and references therein. Approaches based on actual robots have been used in primary schools to support the development of computational thinking and to also to support teachers’ confidence [9]. In Higher Education, robots have been used on a range of topics, including to support teaching of functional patterns [6].



© Michael Heeney, Kelly Androutsopoulos, and Franco Raimondi;
licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 15; pp. 15:1–15:12

OpenAccess Series in Informatics



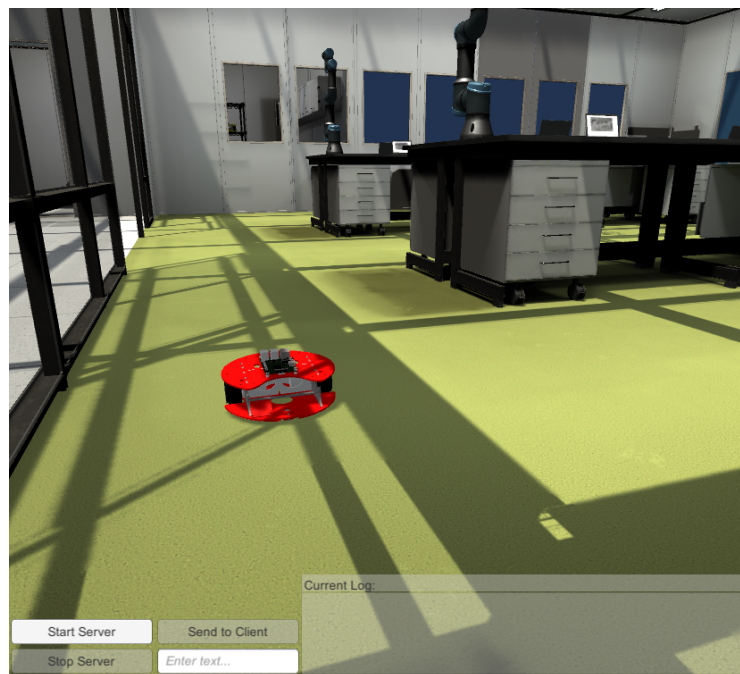
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

15:2 A Digital Twin to Support Large-Scale Coding Classes

Virtual platforms such as Scratch¹ can provide a replacement for physical robots. Unfortunately, these platforms have typically targeted younger users, with very few options (if any) available for the Higher Education sector.

We have developed a *digital twin* for a robotic platform with the explicit goal of providing physical manifestations of code for students in the Higher Education sector. At a high level, a digital twin is the virtual counterpart of a physical system, implementing all its functional properties and reproducing its physical characteristics with high accuracy. Digital twins have been used in a range of domains, including manufacturing, avionics, architecture, etc. One of the key differences between a digital twin and a simulation is the fact that a simulation may remove non-relevant features, for instance a car may be depicted as a rectangle in a 2-dimensional space, while a digital twin would reproduce not only the dynamical behaviour of a car, but also its other aspects, such as the ability to open doors etc.

In this paper we describe the details of a Digital Twin platform that we have developed in Unity². We present the architecture of the system, centred around the extension of an existing approach for service-based development of microcontrollers [4].



■ **Figure 1** The current prototype with text field for logs, information and sent/received data on the TCP/IP network.

2 Related Work

Constructionism is rooted in Piaget's constructivist approaches [1]. Both constructivism and constructionism have played a key role in the past five decades in STEM education, and advent of increasingly better connectivity has then resulted in the development e-Learning platforms for a range of subjects [27] and across year groups, from primary school

¹ <https://scratch.mit.edu/>

² <https://unity.com>

to higher education. In parallel, robots have been used to support the teaching of robotics, electronics, manufacturing and STEM subjects in general, building on the success of the maker movement [23].

Robots are used as a platform to teach many concepts in Computer Science. They provide an educational tool for introducing students to embedded systems and computational thinking for interacting with the environment around them. When hardware is introduced, students can conceptualise a topology of the system; from the microcontroller and General Purpose Input/Output (GPIO) functionality, to the software, libraries and interfacing/networking involved to carry out a simple action such as driving forward. Robotics in education naturally puts together continuous and discrete computation and provides an opportunity to reason about error and uncertainty, an important topic of study at an undergraduate level [29].

In regards to computational thinking, other forms of technologies have been developed to help students start to grasp the concepts of programming including augmented reality [21].

Within the task of building a robot, the layers of the system allow researchers and educators to deploy teaching strategies and subject specificity for different target audiences. For example, within the makers movement [23], there have been studies to understand the benefits of robotics to teach programming, electronics, fabrication and general STEM as an overall subject.

When creating a robot for teaching, there is an emphasis whether to buy an existing product, or to create a bespoke product which the educators can iterate over time. Developing a prototype in a maker movement [13] style allows open sourcing of hardware and software, collaboration between staff and students and integration with higher education facilities. One of the main benefits robot platforms following this tradition is the cost. Working robots can be physically built for under £50 with a Raspberry Pi Zero board [34]. However, as described by Correll et al [11], scalability is a factor. For example, designing complex systems becomes a labour of maintenance, explanation of hardware, etc., which can create bottlenecks for delivery.

When developing a simulation part of a platform, linking the virtual environment to this platform is similar to gaming platforms, also used in educational teaching strategies. Games are widely accepted as an engaging and motivating tool in the CS curriculum [18]. There are many advantages including the ability to increase learning interest, enhancing confidence in learning and also lead to long term knowledge retention. It is also noted that gaming environments can produce effectiveness in learners due to their response time. They can generate options to users which may not be available for the learner at the particular time. They are flexible and allow choices without risk, exploration for the user, which stimulates curiosity, discovering learning and perseverance [20].

With the emergence of digital twins, users are able to run counterparts to a physical system. This could be a digital copy of a robotic system, which runs in parallel; this is a common technology in the automotive industry [5]. These models have a plethora of uses in a design life cycle and have many benefits including time saving of ideas, increased quality of work, reduced risks in design and increased efficiency. Links have also been made to how digital twins serve well as an educational tool, bringing together many aspects of different learning theories [12]. Additionally, the manifestation of the understanding of how individual parts of a system, including sensors and actuators work, is beneficial to students [8].

A tangible artefact, such as a robot gives the user embodiment, allowing them to express their understanding of code through the robot e.g. movement, sensing the environment etc. Embodiment in robotic hardware has been seen to have positive benefits over an array of year groups [17, 22, 33]. Popular beneficial themes include engagement of material, self-efficacy,

15:4 A Digital Twin to Support Large-Scale Coding Classes

attitudes towards the subject and improvement of grades. These benefits are continued between different forms of platform, for example, from bespoke and custom hardware [10], to commercial robots or for virtual environments and simulations [3, 30].

Physical feedback obtained by using mobile robots has the ability to strengthen the concepts of programming and motivate the students. On one hand, students are able to become motivated by the technology and at the same time, have a better relationship between the notion of theory and how that relates to practical results [31]. By creating a realistic counterpart, the idea is to keep the irreplaceable position in the educational process [28], but also keep the realism of the physical feedback obtained with real hardware and show the similarities in class.

It has been reported [14] that experiences with physical manifestations and existing projects can bring students to be motivated into learning and be interested in the subject. Furthermore, students developed ideas further than originally set, setting additional goals once understanding the system and technology further.

When developing a robotic platform, or its digital counterpart, the engagement of students will allow them to spend more time developing programs and working on solutions. With repetition of exercises, seeing changes and working on a solution in iterations, it has been suggested [15] that students build better solutions overall.

MIRTO [2] is an example of a cheap, open-source, robotic platform used in face-to-face teaching. In this paper, the learning objectives were achieved much earlier than in traditional and theoretical models of teaching. This tool is influenced by the constructionism approach, and it allows the students to explore the capabilities of the robot independently. For example, before working towards the marking objectives of the module, introductory activities would include to understand wheel rotation and navigation. This includes defining the speeds of the motors, alignment and bump sensors. These tasks allow students to understand by trial and error the notion of real-time system and control, reinforcing knowledge and skills from previous work. Gamification in the classroom by means of line-following races led to further engagement [6].

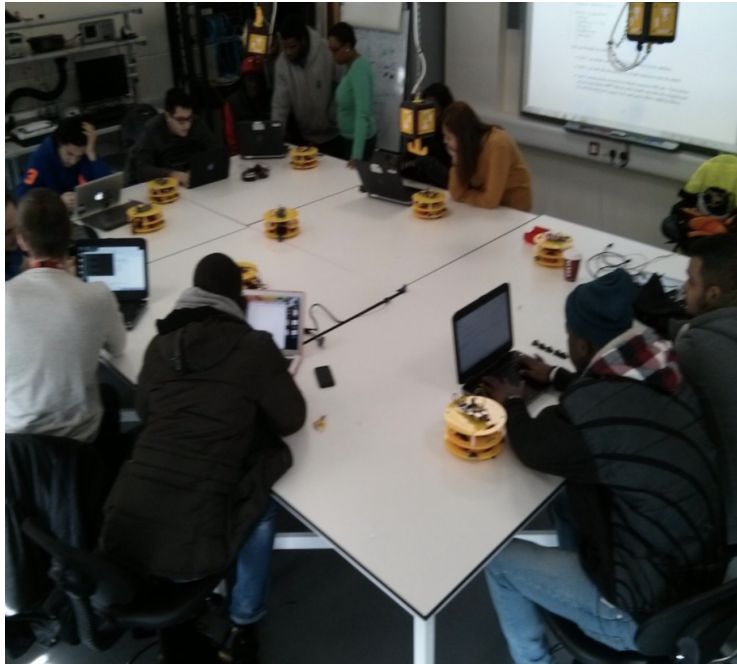
In 2020, with academic institutions going into lockdown due to the COVID-19 virus, the ability to create virtual labs for simulating practical skills [25] was essential to continue teaching practical skills in classes. Digital twins allow this to happen by creating a reliable and high fidelity prototype. However, with the development of this kind of systems, the scalability for experimentation should increase and provide feedback as close as possible to the original physical systems [32].

Existing literature shows the benefits of mobile robotics in Computer Science education. The robots can be virtual or tangible. However, most of the case studies are up to keystage 5 in the United Kingdom (16-18 year olds). To the best of our knowledge, there does not seem to be an undergraduate Computer Science course using a digital twin to support teaching programming to these students.

Testing the principles of constructionist approaches and how to measure the effect of the learning methodology has been subject to various studies. Kafai [16] created a comparative study, in which the classroom was split between a constructionist approach using a creation of video games which incorporated mathematical problems requiring an understanding of fractions to solve; with another group received traditional teaching for fraction instruction. This study concluded a better understanding of fractions, increased engagement/motivation of the students, better critical thinking and a development of soft skills through collaborative learning. Benefits of Scratch as a constructionist tool have been widely shown, particularly by Resnick and Brennan [7] in which results concluded positive improvements in the ability to develop an iterative design cycle with experimenting and iteration of code and objects, logical reasoning through testing and debugging, creativity and problem solving.

3 The MIRTO Robotic Platform

The MIRTO robotic platform was developed to support a specific pedagogy. We take a constructionist approach to teaching the fundamentals to first year undergraduate Computer Science students with MIRTO. Hardware is demonstrated using digital electronic circuits to discuss system architecture, microcontrollers and microprocessors such as Raspberry Pi. Programming concepts are taught with Racket (a dialect of Lisp). In the last third of the academic year, the fundamentals of these topics are brought together using a robot with the acronym MIRTO.



■ **Figure 2** Students programming robots in a classroom setting.

MIRTO is an open source robotic platform that contains the following components:

1. Raspberry Pi: A custom image of Raspbian extended with Racket to compile code on the Pi. This also enables networking opportunities such as custom wifi networks, editing of a Linux image, ssh, ethernet etc. This can be networked with an Arduino microcontroller to control GPIO or these two boards can be swapped with a Raspberry Pi Pico.
2. Custom PCB: The top-layer of the robot has all of the GPIO options of the robot available in one place including motorshield for controlling 2 x DC motors, an RGB LED, an infrared array for line following, 2 tilt switches for hit detection, a speaker and a LCD for displaying messages and connection information.
3. ASIP: Arduino Service Interface Protocol³ enables a computer to discover, configure, read, write a microcontrollers general purpose IO pins. As standard, ASIP uses a serial connection to the Raspberry Pi.
4. Along with ASIP, there are various libraries to work with programming languages including Java and Python. For first year students the focus is on the Racket⁴ library.

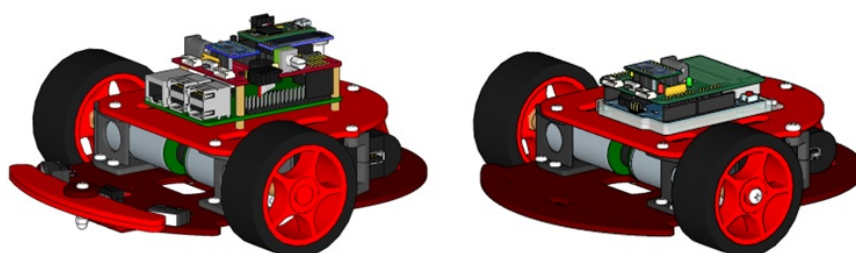
³ <https://github.com/michaelmargolis/asip/>

⁴ <https://github.com/fraimondi/racket-asip>

15:6 A Digital Twin to Support Large-Scale Coding Classes

To support iteration of teaching material, anonymous questionnaires are delivered to students, in which many reported enjoyment using MIRTO and use of hardware across the year. However, in order to scale the use of robots for large classes and for online teaching, we developed a digital twin to serve as a real-time counterpart of the physical robots in class. Other reasons for designing a digital twin include:

1. Sustainability of manufacturing: The robots take time to design, build and repair, which could be labour-intensive for large cohorts.
2. Time management: Students only have a 2-hour class per week to work on the robots. The demand was very high outside the classroom for students to test, but due to the last point, we were unable to loan these with fairness.
3. Training: The robots need a range of expertise across various disciplines to operate and maintain and in scale, this was proved difficult for staff availability.



■ **Figure 3** The latest iteration of MIRTO 2024.

3.1 Software for MIRTO

Students can complete exercises independently or in groups on software problems that involve MIRTO. They are given a selection of exercises to try various GPIO of the robot and then given examples of code to run and understand what it is doing. The exercises consist of moving the robot in a room at different speeds and for a certain time (understanding the timing issues) avoiding obstacles and bumping into walls. More advanced exercises involve following a line and improving on precision. Examples of instructions are given in Table 1.

■ **Table 1** Instructions for the MIRTO using DrRacket.

Instruction	Description
(setMotor 0 0)	Set the speed for the motors. The first integer assigns the left (0) or right motor (1). The second integer assigns the speed of the motor. Value range is from -255 to 255.
(stopMotor 0)	This will stop a single motor. The integer assigns the motor as above.
(stopMotors)	Stops both the motors.
(getIR 0)	Prints the reflected value from a infrared sensor array. The integer is the sensor number in the array. Currently there are 3 sensors. Value range is from 0 - 255.
(setLCDMessages "message" 0)	Write a message to a five line liquid crystal display. The text in " " is what will be printed. The integer is the line on the display.
(leftBump?)	This will check the state of a bump sensor, printed as a 0 or 1 for true or false. There are two sensors on the robot, leftBump and rightBump.
(analog-read 0)	Reads the on board potentiometer on the robots PCB. This could be any analog sensor which is assigned by the integer.
(playTone 0 0)	Plays a tone on a piezo buzzer based on a particular frequency. The first integer assigns the frequency. The second assigns the duration in seconds.

With the majority of the exercises, students are given the freedom to explore a snippet of code and guess what it will do. Then when they compile the program, they can see how the robot will react. With the exploration of the wheels, sensors and other input/output of the robot, students embed themselves in a creative learning process. This supports reflection in class and discussion between peers. During observations, an array of interpretations of solutions for the same task are created using sensors to determine movement, outputs of LCD messages or colour of LEDs to show a state or explanation of location etc. With this notion of creative learning intertwined with the technology, students are able to follow a creative learning spiral [26]. Examples of the concepts taught using MIRTO include higher-order functions, string processing and open vs closed loop systems [6].

4 Implementation of Digital Twin

We have implemented a digital twin in Unity, to replicate a classroom setting for students. Bespoke classroom objects such as the mobile robot, robotic arms and some furniture were all designed in Blender.

4.1 High-Level System Architecture

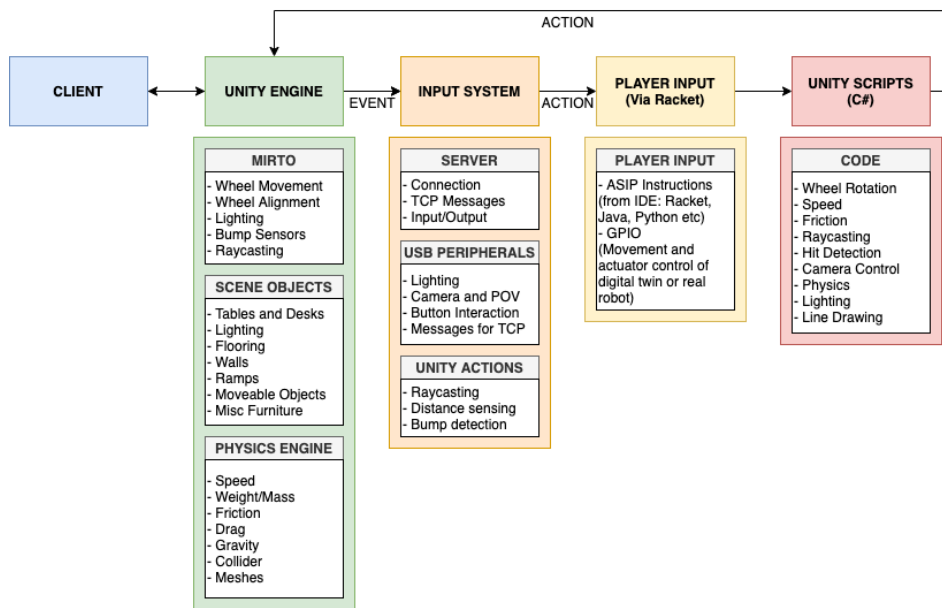


Figure 4 Digital Twin Architecture for the Online Provision.

The digital twin created in Unity replaces the robot described above and the overall architecture of the system is illustrated in Figure 4. At the high level, a student will control the client of the system. This is typically a program written in Racket. When run, the Racket program will establish a TCP connection with Unity and begin to translate Racket messages into instructions. These instructions are then converted to C# scripts which will control many aspects of the scene of the digital twin including the orientation and direction of the robot, the speed and movement of the robot, control and reading of the sensors of the robot (bump sensors, infrared sensors, LEDs etc). In turn, in some of these instances, Unity will send back string messages to Racket, to let the program know which sensors have

15:8 A Digital Twin to Support Large-Scale Coding Classes

been interacted with, looking for a follow up instruction if needed. For example, students are taught different ways to navigate with the robot. This could include driving, using bump sensors to turn if there are objects in the way, line following etc.

In Unity, the digital twin is created on the same part list used for the physical manufacturing of the robot. Therefore, the parts are the same scale and the scene replicates the classroom setting. The scene is created using a range of furniture, walls and lighting. All these components contain physics properties and meshes so the robot (and user) is able to have a realistic interaction in the environment.

The digital twin incorporates wheel movement by operating scripts to control wheel colliders, which are a slip-based tyre friction model generated for wheels. Along with the collider, there is a rotation animation, to give realism and understanding of speed for the user.



■ **Figure 5** The digital twin has 3 rooms for students to explore and run their programs in.

For the sensors of the robot, Unity uses raycasting for the bump sensors. With raycasting, the bump sensors on the 3D model send out a “ray” from a camera point until it finds a surface it collides with. The ray sent is very short, to mimic being close enough to the object to “hitting” it. Once hit, a boolean value is triggered, sending a message back to the Racket client. Similarly, with the infrared sensor, 3 raycasts are sent (as separate threads) to the floor to detect colour in the flooring, this is to mimic lines on the surface for students to complete line following exercises.

The environment uses an input system with various methods. For example, as described above, Racket will send messages via a TCP server to control the robot; Unity can also output messages back to Racket. USB peripherals can also control the movement, alignment and camera position of the user in the scene. Additionally, Unity actions such as the raycasting can provide input to control the environment. Racket is also the main input for the environment, controlling the digital twin and its movement. All of these methods of inputs will generate the scripts written in C to control the speed, direction and friction of the wheels, line drawing (to help with navigation) and sensor and input/output control.

4.2 Implementation

Let's consider an example to see how the digital twin works in practice. Assume that a student needs to write code for the following task: "drive the robot forward for two seconds then stop"; the Racket code would be the following:

```

1 #lang racket
2 (require "AsipMain.rkt")
3   (open-asip)
4   (setMotors 150 150)
5   (sleep 2)
6   (stopMotors)
7   (close-asip)

```

This is identical for a physical robot or for the digital twin. In the case of the digital twin, instead of these commands being sent to the serial port of the robot, ASIP⁵ will send a string to the TCP server running in Unity. The above code does the following:

- 1: Include the AsipMain Racket library
- 3: We setup the connection to the robot. This will connect to the localhost at port 54010 for a digital twin.
- 5: The robot is asked to move both wheels forward. The power selected is 150 (the range is between -255 to 255).
- 6: There is a sleep for 2 seconds, which allows the wheels to run during this time.
- 7: The wheels are turned off, the speed is set to 0.
- 8: Close the TCP connection

For the example above, the actual strings sent over TCP would be in two parts, to turn the motors on and then off again. Concretely, the following messages will be sent:

```
"M,m,0,150"; "M,m,1,150"; "M,m,0,0"; "M,m,1,0"
```

When a motor command is sent, it is translated to two commands back to control each motor separately. This allows us to also set individual motor movement for turning and concise movements.

In Unity, once the string is received, it is split into four different variables to understand its role in ASIP and what it should be controlling. Each string starts with a letter, which is the header of the request; this could include M (motors), E (encoders), T (tone – for a piezo), P (RGB LED control), L (LCD on the robot), I (port to pin mapping) etc. Each of these strings has separators (,) and other tags and numeric values that go with it. Once the command above is identified as a motor command, wheel colliders in Unity that control the wheels will move using the following command:

⁵ <https://github.com/michaelmargolis/asip/tree/master/documents>

15:10 A Digital Twin to Support Large-Scale Coding Classes

```
leftWheelInput.motorTorque = asipLeftWheelInput * motorForce;  
rightWheelInput.motorTorque = asipRightWheelInput * motorForce;
```

Within the movement calculations, there are parameters set to increase the wheel speed in the virtual space, similar to the classroom setting. For example, 0 being the no movement, 255 being the fastest speed and -255 reversing at the maximum speed. At a later date, there will be planned tests to ensure accurate comparisons between the virtual and physical spaces.

For reference of the user, in the Unity GUI, there is a display box to show all of the messages received from ASIP, as well as messages sent and error messages etc. This feature can be shown/hidden by the user as well as the ability to draw a line showing the movement path of the digital twin to help students understand path planning and movement instructions further, similar to designs of LOGO and turtle robots in the 1980s [24].

5 Conclusion and future work

We have presented the development of a multi-programming language digital twin platform to replicate a classroom setting used for teaching Computer Science to undergraduate students. The digital twin is planned to be released as an open-source platform. The digital twin can run side-by-side with its physical counterpart. Students are able to use the physical model in classes and once resources are not available, for example out of class, students can login to a virtual world to continue their programming assignments and continue developing their computational thinking. The online platform and physical platform both use Racket (a dialect of Lisp) as its main programming language. However, our virtual environment can be used by students across year groups and can be programmed in multiple languages including Java, Python, as well as Racket.

While the use of the digital twin in classrooms has shown to be promising, we plan to conduct further experiments with students to empirically evaluate its impact. Current surveys suggest there is a strong link to engagement and computational thinking when using the digital twin. However, more work is to be carried out to fortify these claims.

For future work, more work is needed for creating an open source and robust digital twin for all to host, deploy and use for their teaching and training. Emphasis will be made on the GUI, to deploy visual indicators on all sensors and actuators, so students and staff can get visual feedback of the tools they are using, rather than printed syntax and text. Within the GUI, students will also be able to select a lab space, to best replicate the room they are in for size. Additional to this, menus will be designed to enable students to select the connection type; this could include web socket connections, TCP/IP as stated previously or how they want to connect, via simulation only or as a digital twin.

References

- 1 Edith Ackermann. Piaget's constructivism, papert's constructionism: What's the difference? In *Constructivism: Uses and perspectives in education.*, pages 85–94, 2001.
- 2 Kelly Androutopoulos, Leonidas Aristodemou, Jaap Boender, Michele Bottone, Edward Currie, Inas El-Aroussi, Bob Fields, Lorenzo Gheri, Nikos Gorogiannis, Michael Heeney, et al. Mirto: an open-source robotic platform for education. In *Proceedings of the 3rd European Conference of Software Engineering Education*, pages 55–62, 2018.
- 3 Mikko Apiola, Matti Lattu, and Tomi A Pasanen. Creativity and intrinsic motivation in computer science education: experimenting with robots. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, pages 199–203, 2010. doi:10.1145/1822090.1822147.

- 4 Gianluca Barbon, Michael Margolis, Filippo Palumbo, Franco Raimondi, and Nick Weldin. Taking arduino to the internet of things: The asip programming model. *Computer Communications*, 89:128–140, 2016. doi:10.1016/j.comcom.2016.03.016.
- 5 Florian Biesinger and Michael Weyrich. The facets of digital twins in production and the automotive industry. In *2019 23rd international conference on mechatronics technology (ICMT)*, pages 1–6. IEEE, 2019.
- 6 Jaap Boender, Ed Currie, Martin Loomes, Franco Raimondi, and Giuseppe Primiero. Teaching functional patterns through robotic applications. In *Trends in Functional Programming in Education 2015*, June 2015.
- 7 Karen Brennan and Mitchel Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*, volume 1, page 25, 2012.
- 8 Rolando Antonio Chacón Flores, Martí Sánchez Juny, Esther Real Saladrigas, Xavier Gironella Cobos, Jaume Puigagut Juárez, and Alberto Ledesma Villalba. Digital twins in civil and environmental engineering classrooms. In *EUCEET 2018: 4th International Conference on Civil Engineering Education: Challenges for the Third Millennium*, pages 1–10. International Centre for Numerical Methods in Engineering (CIMNE), 2018.
- 9 Christina Chalmers. Robotics and computational thinking in primary school. *International Journal of Child-Computer Interaction*, 17, July 2018. doi:10.1016/j.ijcci.2018.06.005.
- 10 CTE STEM Coach, David Archer, Lois Delcambre, Scott Britell, and Ananth Mohan-Contributors. K-12 stem robotics: Stealth computer science for the masses. *The Journal of Computing Sciences in Colleges*, page 152, 2011.
- 11 Nikolaus Correll, Rowan Wing, and David Coleman. A one-year introductory robotics curriculum for computer science upperclassmen. *IEEE Transactions on Education*, 56(1):54–60, 2012. doi:10.1109/TE.2012.2220774.
- 12 Joe David, Andrei Lobov, and Minna Lanz. Learning experiences involving digital twins. In *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, pages 3681–3686. IEEE, 2018. doi:10.1109/IECON.2018.8591460.
- 13 Erica Rosenfeld Halverson and Kimberly Sheridan. The maker movement in education. *Harvard educational review*, 84(4):495–504, 2014.
- 14 Cindy K Harnett, Thomas R Tretter, and Stephanie B Philipp. Hackerspaces and engineering education. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–8. IEEE, 2014.
- 15 Michael Jonas. Do it again: Learning complex coding through repetition. In *Proceedings of the 18th Annual Conference on Information Technology Education*, pages 121–125, 2017. doi:10.1145/3125659.3125690.
- 16 Yasmin B Kafai. *Minds in play: Computer game design as a context for children’s learning*. Routledge, 2012.
- 17 Christopher Kitts and Neil Quinn. An interdisciplinary field robotics program for undergraduate computer science and engineering education. *Journal on Educational Resources in Computing (JERIC)*, 4(2):3–es, 2004.
- 18 Stan Kurkovsky. Making computing attractive for non-majors: a course design. *Journal of Computing Sciences in Colleges*, 22(3):90–97, 2007.
- 19 Michael Lodi, Dario Malchiodi, Mattia Monga, Anna Morpurgo, and Bernadette Spieler. Constructionist Attempts at Supporting the Learning of Computer Programming: A Survey. *Olympiads in Informatics: An International Journal*, 13:99–121, July 2019. doi:10.15388/loi.2019.07.
- 20 Ju Long. Just for fun: using programming games in software programming training and education. *Journal of Information Technology Education: Research*, 6(1):279–290, 2007.
- 21 Luis Carlos Martins, Lázaro Vinicius Lima, and Pedro Rangel Henriques. Lcsmar, an ar based tool to inspect imperative programs. In *4th International Computer Programming Education Conference (ICPEC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

- 22 Monica M McGill. Learning to program with personal robots: Influences on student motivation. *ACM Transactions on Computing Education (TOCE)*, 12(1):1–32, 2012. doi:10.1145/2133797.2133801.
- 23 Sofia Papavlasopoulou, Michail N. Giannakos, and Letizia Jaccheri. Empirical studies on the maker movement, a promising approach to learning: A literature review. *Entertainment Computing*, 18(C):57–78, 2017. doi:10.1016/j.entcom.2016.09.002.
- 24 Seymour Papert and Idit Harel. Situating constructionism. In Seymour Papert and Idit Harel, editors, *Constructionism*, chapter 1. Ablex Publishing Corporation, Norwood, NJ, 1991. URL: <http://www.papert.org/articles/SituatingConstructionism.html>.
- 25 Sandipan Ray and Sanjeeva Srivastava. Virtualization of science education: a lesson from the covid-19 pandemic. *Journal of proteins and proteomics*, 11:77–80, 2020.
- 26 Mitchel Resnick. Lifelong kindergarten: Cultivating creativity through projects, passion, peers and play. In “*Lifelong Kindergarten: Cultivating Creativity through Projects, Passion, Peers and Play*”, chapter 1. The MIT Press, Cambridge, Massachusetts, 2017.
- 27 T. Richter, S. Rudlof, B. Adjibadji, H. Bernlöhner, C. Grüninger, C.-D. Munz, A. Stock, C. Rohde, and R. Helmig. Viplab: a virtual programming laboratory for mathematics and engineering. *Interactive Technology and Smart Education*, 9:246–262, 2012. doi:10.1109/ISM.2011.95.
- 28 Carlos Rodriguez, Jose L Guzman, Manuel Berenguel, and Sebastian Dormido. Teaching real-time programming using mobile robots. *IFAC-PapersOnLine*, 49(6):10–15, 2016.
- 29 Daniela Rus. Teaching robotics everywhere. *IEEE Robotics & Automation Magazine*, 13(1):15–94, 2006. doi:10.1109/MRA.2006.1598048.
- 30 Ashraf Saad, Travis Shuff, Gabriel Loewen, and Kyle Burton. Supporting undergraduate computer science education using educational robots. In *Proceedings of the 50th Annual Southeast Regional Conference*, pages 343–344, 2012. doi:10.1145/2184512.2184596.
- 31 Payman Shakouri, Olga Duran, Andrzej Ordys, and Gordana Collier. Teaching fuzzy logic control based on a robotic implementation. *IFAC Proceedings Volumes*, 46(17):192–197, 2013. doi:10.3182/20130828-3-UK-2039.00047.
- 32 Jaroslav Sobota, Roman Pišl, Pavel Balda, and Miloš Schlegel. Raspberry pi and arduino boards in control education. *IFAC Proceedings Volumes*, 46(17):7–12, 2013.
- 33 Eben B Witherspoon, Ross M Higashi, Christian D Schunn, Emily C Baehr, and Robin Shoop. Developing computational thinking through a virtual robotics programming curriculum. *ACM Transactions on Computing Education (TOCE)*, 18(1):1–20, 2017. doi:10.1145/3104982.
- 34 Narasimha Saii Yamanoor and Srihari Yamanoor. High quality, low cost education with the raspberry pi. In *2017 IEEE Global Humanitarian Technology Conference (GHTC)*, pages 1–5. IEEE, 2017. doi:10.1109/GHTC.2017.8239274.

To Kill a Mocking Bug: Open Source Repo Mining of Security Patches for Programming Education

Andrei-Cristian Iosif ✉ 

Universität der Bundeswehr München, Germany
Siemens AG, München, Germany

Tiago Espinha Gasiba ✉ 

Siemens AG, München, Germany

Ulrike Lechner ✉ 

Universität der Bundeswehr München, Germany

Maria Pinto-Albuquerque ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Abstract

The use of third-party components (TPCs) and open-source software (OSS) has become increasingly popular in software development, and this trend has also increased the chance of detecting security vulnerabilities. Understanding practical recurring vulnerabilities that occur in real-world applications (TPCs and OSS) is a very important step to educate not only aspiring software developers, but also seasoned ones. To achieve this goal, we analyze publicly available OSS software on GitHub to identify the most common security vulnerabilities and their frequency of occurrence between 2009 and 2022. Our work looks at programming language and type of vulnerability and also analyses the number of code lines needed to be changed to fix different vulnerabilities. Furthermore, our work contributes to the understanding of real-world and human-made data quality required for training machine learning algorithms by highlighting the importance of homogeneous and complete data. We provide insights for both developers and researchers seeking to improve cybersecurity in software education and mitigate risks associated with OSS and TPCs. Finally, our analysis contributes to software education by shedding light on common sources of poor code quality and the effort required to fix different vulnerabilities.

2012 ACM Subject Classification Security and privacy → Software and application security; Software and its engineering → Collaboration in software development; Information systems → Open source software; Security and privacy → Vulnerability management

Keywords and phrases Open-source software, Software quality, Cybersecurity, Repository Mining

Digital Object Identifier 10.4230/OASICS.ICPEEC.2024.16

Funding This work is partially financed by Portuguese national funds through FCT – Fundação para a Ciência e Tecnologia, I.P., under the projects FCT UIDB/04466/2020 and FCT UIDP/04466/2020. Furthermore, the third author thanks the Instituto Universitário de Lisboa and ISTAR, for their support. We acknowledge funding for project LIONS by dtec.bw. Andrei-Cristian Iosif and Tiago Gasiba acknowledge the funding provided by the Bundesministerium für Bildung und Forschung (BMBF) for the project CONTAIN (FKZ 13N16585).

Acknowledgements The authors would like to thank Kaan Oguzhan for aiding in data collection, and also for the helpful, insightful, and constructive comments and discussions about the present work.

1 Introduction

Having secure software has always been essential for developing any product or service. Additionally, cybersecurity has gained more attention in recent years due to the ever-increasing reliance on the internet. Providing secure products or services is essential to



© Andrei-Cristian Iosif, Tiago Espinha Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque; licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 16; pp. 16:1–16:12

OpenAccess Series in Informatics



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

maintain customer and user trust. Moreover, it prevents potential damages resulting from security breaches. As such, it is important that the developers of the software be aware of security standards and best practices, e.g. through awareness campaigns or, in general, cybersecurity education.

Several industrial cybersecurity standards provide guidelines for the industry. One example is the IEC 62443 standard [5], which requires a secure software development life cycle (sSDLC) framework that covers all SDLC phases and provides detailed guidance to ensure software security. Another widely-used cybersecurity standard in the industry is the Common Weakness Enumeration (CWE) [12] developed by the MITRE Corporation, which is a community-developed list of both software and hardware vulnerabilities categorized by the type of vulnerability they introduce. The CWE also provides descriptions of the vulnerabilities and usually includes examples of both vulnerable and non-vulnerable code associated with the vulnerability. Apart from cybersecurity standards, there are also standards for software quality, such as ISO 25000 [6], which provides a set of metrics for evaluating the quality of software products and guidelines for producing high-quality software.

According to the U.S. Department of Homeland Security (DHS) [2], humans are responsible for more than 90% of software quality issues.

As software complexity increases, the use of third-party components, including commercial off-the-shelf (COTS) and open-source software (OSS), has become more widespread. According to a survey conducted by Black Duck Software, 78 percent of respondents reported that their companies use OSS for some or all of their operations [1]. Although OSS can benefit developers, detecting the most critical quality violations can pose a challenge. Attackers view the widespread use of TPCs as an opportunity for exploitation. Notably, TPCs have been the source of significant security vulnerabilities, such as the FREAK OpenSSL vulnerability (CVE-2015-0204) [10], Shellshock Vulnerability (CVE-2014-6271) [9], and Log4Shell (CVE-2021-44228) [11]. As the trend of using third-party components continues to rise, it is important to consider security when selecting and utilizing TPCs. This paper aims to analyze existing OSS software on GitHub and shed light on the trends of security vulnerabilities.

Security-oriented companies may prioritize their own interests when discussing software vulnerabilities, which in turn results in fewer studies with unbiased real-world data. We contribute to improving the understanding of cybersecurity issues in OSS software and providing insights for developers and researchers. In this work, we highlight the difficulty of identifying OSS software's most frequent security issues.

In this work, by exploring GitHub repositories and commits between 2009 and 2022, we aim to gather insight into the following Research Questions:

RQ1: What are the most common security vulnerabilities in OSS software?

RQ2: How do these vulnerabilities vary by programming language and over time?

RQ3: What is the effort required to fix these vulnerabilities in terms of code changes?

To conduct our analysis, we scraped publicly available GitHub repositories, identified commits aimed at fixing known software vulnerabilities by looking at their commit messages, and analyzed trends and distributions of CWE categories. Our findings can benefit development teams and researchers seeking to improve software quality by exploring the most common sources of poor code quality. Therefore, in this paper, we present a summary of the most common security vulnerabilities and their frequency of occurrence in OSS software.

This paper is structured as follows: Section 2 will introduce relevant related research. Next, we present our approach in Section 3, followed by results in Section 4. We discuss our results in Section 5. Next, Section 6 explores the threats to validity to our study. Finally, Section 7 reiterates through our work and explores possible next steps.

2 Related Work

In this section, we present similar large-scale studies which explore security through repository metadata.

Iannone et al [4] explored the effects of refactoring on security by measuring security-related technical debt. Their large-scale investigation involved running git-blame commands to analyze commits, providing insights into the correlation between refactoring practices and security implications in software development.

Li and Paxson [7] performed a comprehensive empirical study of security patches, analyzing a diverse set of repositories to generate generalizable insights about security practices. They utilized Git commit links to identify and review security-related changes, contributing significantly to understanding security patch dynamics.

Wang et al. [18] developed PatchDB, a large-scale security patch dataset to facilitate the manual checking of security-related commits. Their work highlights the consistency and challenges in identifying security-focused changes across large datasets.

Wang and Nagappan [17] characterized software developer networks by conducting a large-scale empirical study to distinguish between security and non-security-related commits. Their findings provide insights into the network dynamics of developers engaged in security-related software development.

While all these approaches explore repository metadata, our work tackles a purely quantitative analysis of security-related git commit messages across OSS repositories, where we explore: language-specific vulnerability analysis, SAST Tool integrations, and commit message quality.

3 Approach

In this section, we provide details about the steps taken to collect the data driving our study. Our search space includes all public GitHub repositories created between January 1, 2009, and December 31, 2022. This arbitrary cut-off date is motivated by recent developments in AI-assisted programming – according to a Microsoft executive’s statement from March 2023, 40% of GitHub Copilot users check in “AI-generated and unmodified” code [8]. Hence, we purposefully restrict our search space to include *mostly human* code changes.

3.1 Querying and Scraping GitHub for Repositories

To gather the necessary data for our study, we developed a scraper to query GitHub for repositories. The scraper searched for mentions of “CWE ID” in commit messages to identify repositories containing fixes for vulnerabilities listed in the CWE catalog.

Due to limitations on the branch type for queries, we could only search for commits on the master/main branch. We also employed a filter to exclude commits that contained more than a single file change, as this helped to ensure that the commit was not a general commit that happened to include a fix for a vulnerability. Changes on non-code files such as README.md or LICENSE were not counted towards the one file change limit.

To parallelize the scraping process, we use Terraform to deploy multiple instances of our scraper on the cloud. We selected instances optimized for high network bandwidth, with a reported capacity of *up to 10 Gigabit*¹, as well as 8GB of RAM and four vCPUs. These specifications helped ensure the instances could efficiently clone and analyze repositories with high commit volumes.

¹ Available instance bandwidth reported by Amazon AWS

16:4 Open Source Repo Mining of Security Patches for Programming Education

We conducted our analysis by running 13 instances over approximately seven days, each assigned to scrape a different subset of repositories. We faced challenges with GitHub API's query limit, which restricts results to the first 1000 entries per query. We implemented 6-hour query intervals to mitigate this, allowing us to cover most of the relevant repositories. Despite the time windowing approach, some commits were still missed, but we deemed this an acceptable trade-off between the number of repositories we could scrape and the time it took to scrape them.

3.2 Data Extraction

Once our scraper finished detecting all candidate repositories within the given year range, it began cloning the repositories and analyzing their commits. We applied strict rules to classify a commit as a “vulnerability fixing commit.” These rules required a commit message containing the strings “fix” and “CWE ID,” where ID is any possible number in the CWE catalog. We also imposed a limit of 1 file change per commit, excluding changes on non-programming-language files such as `Readme.md` or `License`. These restrictions were implemented to keep the results as relevant as possible. For each commit detected as fixing a CWE vulnerability, we kept small metadata about the commit, date, repository, and the before and after versions of the file(s) that were changed.

3.3 Analysis Method

In this section, we will describe our data analysis approach. We processed the scraped data to extract insights such as the popularity of CWE vulnerabilities across different programming languages and the number of commits per year. Our analysis involved plotting the results for visualization and applying data manipulation techniques such as grouping and sorting to determine the frequency of CWE vulnerabilities by programming language and year.

We analyzed whether some vulnerabilities were more common than others, whether the frequency of specific vulnerabilities varied over time or varied by programming language, and whether some programming languages were more prone to certain types of vulnerabilities. Additionally, we statistically analyzed the number of lines changed in the commits to fix specific vulnerabilities and their variation between different CWEs.

Regarding limitations, it is worth noting that our results are based solely on the commit messages of the commits. This approach may result in some limitations in the accuracy of our data, as we would miss all commits that don't mention the fix in the commit message. However, we believe this limitation is acceptable, given the size of our dataset and the constraints of our approach. The results of our analysis are presented in the next section.

4 Results

In this section, we present the results of our analysis of the commits aimed at fixing CWE vulnerabilities in open-source repositories found on GitHub. By focusing on repositories created from 2009 onwards, we were able to acquire 1934 repositories that contained at least one commit aimed at fixing a CWE vulnerability. Overall, we observed 7093 such commits, as shown in Table 1.

Upon analyzing public GitHub repositories, we found that some programming languages have received more Common Weakness Enumeration (CWE) vulnerability fixes compared to others. Notably, the languages *C*, *Java*, and *C++* received the highest number of CWE vulnerability fixes, followed by *JavaScript* and *Python*. The number of CWE vulnerability fixes for each language is shown in Table 2.

■ **Table 1** Number of GitHub repositories containing at least one commit aimed at addressing CWE vulnerabilities, and the overall count of commits dedicated to fixing CWE vulnerabilities on the platform.

Type	Count
Number of Repositories	1934
Number of Commits	7093

■ **Table 2** A breakdown of the top programming languages that were detected in the commits, along with their corresponding overall percentages. Only the top 5 languages are shown.

Programming Language	Count
C	1784
Java	772
C++	547
JavaScript	301
Python	265

■ **Table 3** Top-5 popular programming languages according to multiple sources for the year 2022.

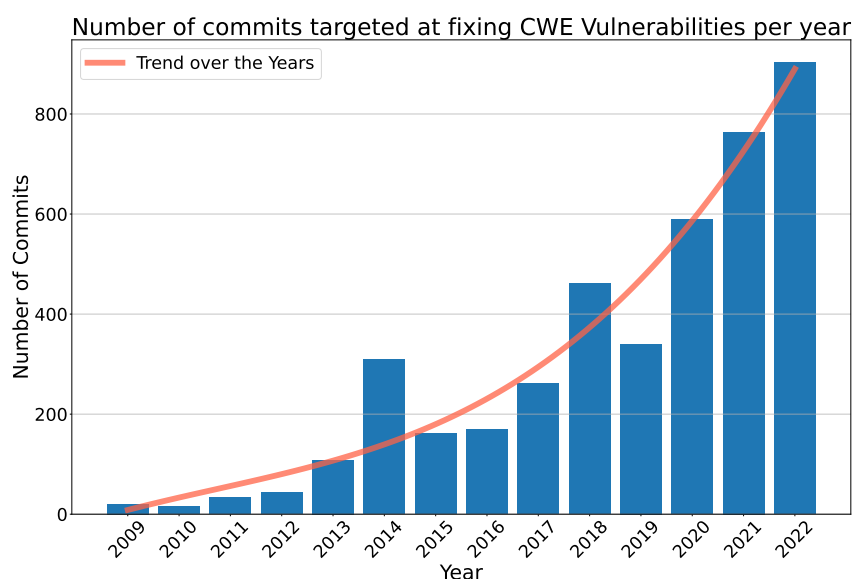
Rank	Octoverse [3]	TIOBE [16]	RedMonk [14]
1	JavaScript	Python	JavaScript
2	Python	C	Python
3	Java	Java	Java
4	TypeScript	C++	PHP
5	C#	C#	C#

In 2022, *JavaScript* and *Python* were the most popular programming languages, according to Octoverse, TIOBE, and RedMonk. Surprisingly, *C*, which is the top language for receiving CWE vulnerability fixes, is not even in the Top-5 of Octoverse and RedMonk. This indicates a significant disparity between the popularity of programming languages on GitHub and the number of security vulnerability-fixing commits. It is possible that developers using popular programming languages lack cybersecurity awareness or are fixing vulnerabilities without explicitly mentioning them in their commit messages. Notably, this analysis is limited to publicly available repositories on GitHub, and further investigation is necessary to comprehend the correlation between the popularity of programming languages and the number of CWE vulnerability fixes, as well as the security implications of language choice.

Additional analysis was conducted on the commits within GitHub repositories to determine basic statistics on the number of lines changed in each commit. The results indicated that the average number of lines changed in a commit was 112.89, while the median was 7, demonstrating that the distribution of the number of files changed is skewed to the right. Investigation revealed that the skew was due to several factors, including the use of XML files with named versioning in some repositories and Notebook-style coding. To minimize the impact of outliers on the results commits with more than 100 lines changed were excluded, shifting the mean and median to 13.86 and 6.00, respectively. Although this exclusion may have affected the accuracy of the analysis, it was necessary to reduce the impact of outliers.

4.1 Number of Commits with mentions of CWE per Year

Figure 1 shows the number of publicly available GitHub commits that aimed to fix software vulnerabilities and were tagged with the associated CWE ID. From 2009 to 2022, the number of commits steadily increased from about 20 to around 1000. This trend indicates that the awareness of cybersecurity is on the rise, and people are taking steps to address vulnerabilities in their code.



■ **Figure 1** Trend in the number of commits made on publicly available repositories on GitHub, aimed at fixing software vulnerabilities and tagged with the associated CWE ID. The figure displays a clear upward trend in the number of commits over the years. Over the years, the number of commits has steadily increased from around 20 in 2009 to approximately 1000 in 2022.

However, our analysis suggests that the number of commits should be higher, especially in recent years. This may be due to two reasons: either people are fixing vulnerabilities without mentioning them or not enough people are addressing them. Additionally, our data collection method may have limited the number of commits we collected, which highlights the need for further research on this critical issue.

According to GitHub, there are approximately 28 million repositories, and assuming that all 800 commits collected are from different repositories, the percentage of repositories that had at least one software vulnerability fixing commit is only 0.003%. This suggests that our data collection method may not be exhaustive, and more commits may have been missed. Nevertheless, our study emphasizes the importance of addressing software vulnerabilities and encourages further research on this critical issue.

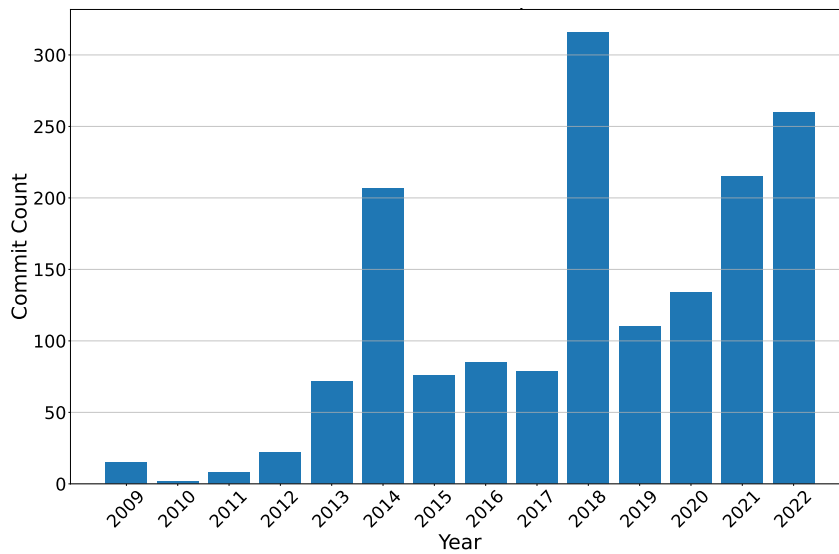
Alternatively, it is possible that people are indeed correcting vulnerabilities but not mentioning it specifically in their commits. This would explain the low number of commits, despite the increasing trend. It could be due to code analysis tools reporting vulnerabilities and people fixing them but do not mention the related CWE ID in their commits. Overall, the number of commits is increasing, which indicates that people are taking steps to address software vulnerabilities.

4.2 CWE per year

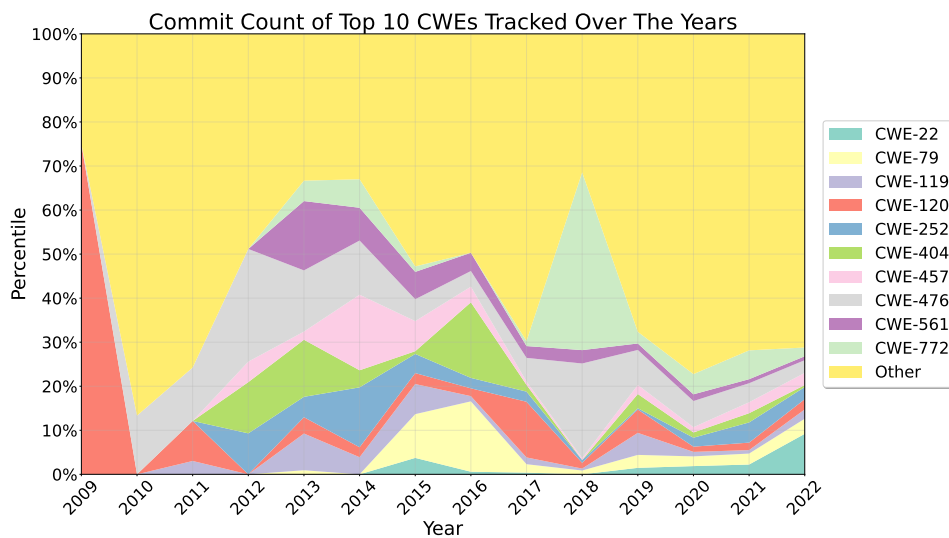
We repeat the trend analysis, this time only considering the top-10 CWEs.

Figure 2 shows a clear upward trend in the number of commits, indicating increasing cybersecurity awareness and action taken to detect, report, and fix vulnerabilities.

In 2018, a significant upward spike was observed, mainly due to a spike in CWE-772. The exact reasons for this spike are unclear, and further research is necessary to establish causality and determine the underlying reasons behind this observation.



■ **Figure 2** Number of commits between 2009 and 2022 targeted at fixing software vulnerabilities and tagged with the associated CWE ID, filtered for the top-10 CWEs.



■ **Figure 3** Percentile distribution of the Top-10 CWE categories from 2009 to 2022, along with the 'Others' category representing all remaining CWEs. The graph illustrates how the distribution of the Top-10 CWEs changes over time as a percentile. The graph provides valuable insights into whether the top-10 CWEs are decreasing or increasing over the years.

To the best of our knowledge, the spike could be due to high-profile cybersecurity incidents like Meltdown and Spectre or data scandals like Scandal, all of which may have led developers to review their code and fix vulnerabilities. Additionally, the General Data Protection Regulation (GDPR), which came into effect on May 25, 2018, may have contributed to an increase in vulnerability fixes, particularly those related to CWE-772.

We investigated whether the software development community is learning from the vulnerabilities present in the top-10 CWEs by grouping the top-10 CWEs and the remaining CWEs into eleven categories. Figure 3 suggests that the percentage of top-10 CWEs has been decreasing over time, while the “Others” category has been increasing. The significant spike observed in CWE-772 in 2018 is also visible in Figure 3.

The trends observed in Figure 3 indicate that lessons are being learned from the top-10 CWEs over time, and the software development community is focusing on addressing vulnerabilities other than the most common ones, which is positive. This suggests that the community is becoming more proactive in identifying and fixing vulnerabilities beyond the top-10 CWEs, which is crucial for improving software security.

Both Figure 2 and Figure 3 suggest that the software development community is becoming more effective in addressing vulnerabilities beyond the top-10 CWEs. Although the spike observed in CWE-772 in 2018 requires further investigation, the overall trends are positive and indicate proactive measures being taken to improve software security, which is also supported by Figure 1.

4.3 Programming Language vs CWE

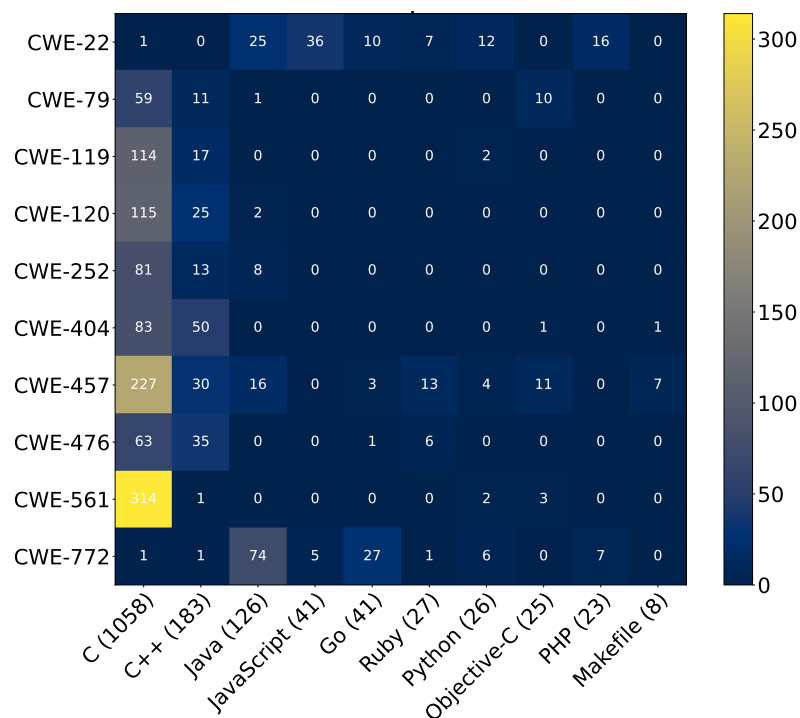


Figure 4 Commit count for the Top-10 CWEs across the Top-10 programming languages. The x-axis represents the programming language, the y-axis represents the CWE, and the color of the cells represents the number of commits.

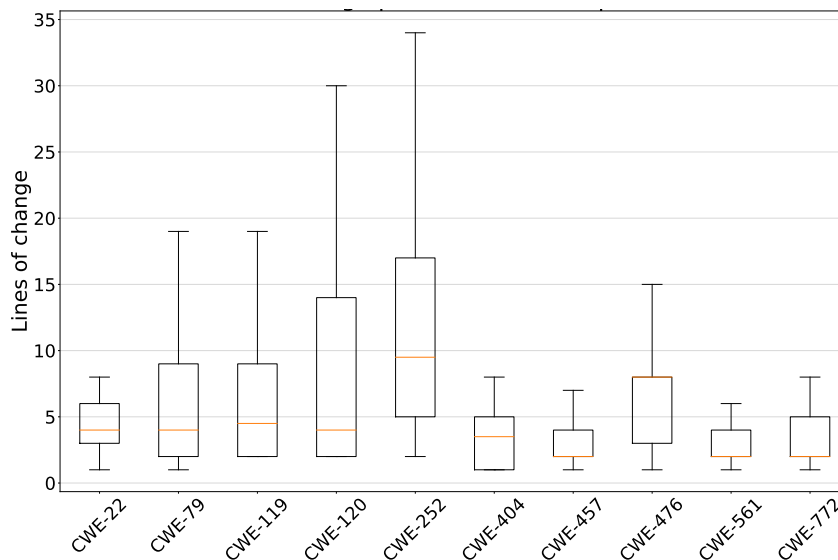
The heatmap in Figure 4 shows commit counts for the Top-10 CWEs across the Top-10 programming languages. The x-axis represents the programming language, the y-axis represents the CWE, and the color of the cells indicates the number of commits. Results show that C has significantly more vulnerability fixes compared to other programming languages.

This could be due to C's popularity in low-level programming, which is more prone to vulnerabilities. C is also widely used in operating systems, embedded systems, and other low-level programming applications, contributing to its high number of vulnerability fixes. Additionally, counting all .h files as C files has resulted in fewer C++ files, making C++ the second-highest in number of commits for the Top-10 CWEs. This could be due to C++'s popularity in low-level programming combined with C, resulting in a majority of the Top-10 CWEs being attributed to C and C++.

The heatmap also highlights a noticeable trend of high counts for C and C++ for many CWEs, but a count of zero for other programming languages. This further supports the hypothesis that results are skewed towards C and C++ for the Top-10 CWEs. The presence of many zeros also suggests that some programming languages may be inherently immune to certain CWEs. For example, the common “Buffer Overflow” vulnerability (CWE-120, CWE-121, CWE-122) is common in C/C++, but is 0 for Python, Go, Ruby, and other languages.

4.4 Lines changed vs CWE

We analyze the number of lines changed for each of the Top-10 CWEs, to provide insight into the effort required to fix these vulnerabilities.



■ **Figure 5** Number of code line that were changed to address a certain vulnerability (CWE).

Results from Figure 5 show that for 8 out of 10 CWEs, the mean lines of change is less than 5, with the exceptions being CWE-252 and CWE-476. The interquartile range (IQR) usually does not exceed 10. These findings indicate that addressing these CWEs can be done efficiently and effectively with relatively small changes to the codebase. The small IQR values suggest that the fixes are consistent across different instances of the same CWE. This indicates that there is a clear and standardized approach to addressing these vulnerabilities.

Overall, these findings provide valuable insights into the nature of common software vulnerabilities and how to address them effectively. However, it is important to note that this data alone is not sufficient to draw conclusions about the effort required for vulnerability detection and fixing. Additional research is necessary to investigate the required effort.

5 Discussion

In addressing our research questions, we explore our results to provide insights into the common security vulnerabilities, their variations, and the effort required to fix them. Next, we will discuss the results to provide an understanding of security vulnerabilities in OSS, their distribution, variation by programming language, evolution across time, and the effort required to mitigate them.

5.1 Implications for AI-Assisted Patching

Training machine learning (ML) algorithms to detect cybersecurity vulnerabilities is challenging due to the lack of a large and diverse dataset classified by experts [15]. ML applications require vast amounts of data for optimal performance, which is not always readily available.

To overcome the lack of diverse datasets, synthetically generated datasets such as Juliet [13] can be used. However, the use of synthetic datasets poses a potential risk where ML algorithms may learn characteristics of the dataset rather than vulnerability features, leading to data leakage towards the test set. To address this issue, it is important to evaluate the effectiveness of ML models on real-world examples rather than relying solely on synthetic datasets. Investigating the results of ML training on synthetic datasets and testing on real-world data to detect cybersecurity vulnerabilities would be interesting in the future.

5.2 Commits with CWE

The analysis shows an increase in commits with CWE over time, indicating a rise in awareness of software vulnerabilities and more software fixes. Future research could investigate factors contributing to this trend, such as the adoption of secure coding practices, improved vulnerability scanning tools, and cybersecurity training for developers. Examining the correlation between commits aimed at fixing vulnerabilities and repository numbers would yield valuable insights into the effectiveness of current practices and policies in mitigating software vulnerabilities.

5.3 Programming Language vs CWE

The results suggest that certain programming languages are more prone to specific CWEs than others due to their design and nature. C and C++ are susceptible to buffer overflow vulnerabilities because of their low-level nature and use in operating and embedded systems. In contrast, Python is immune to buffer overflow CWE-120 vulnerability, provided third-party libraries are not used. Developers should consider the programming language and its more prevalent vulnerabilities when designing and developing software. By understanding the likelihood of different vulnerabilities in different programming languages, developers can mitigate risks and ensure the security and integrity of their software. Staying up-to-date with the latest vulnerabilities and security trends is also important, as new vulnerabilities can emerge and existing ones can evolve over time.

5.4 Lines changed vs CWE

Overall, the average number of changed lines is usually less than 5, implying that the fixes are simple and can be done without introducing big changes to the codebase. Even for complex CWEs, such as CWE-120 and CWE-252, average number of changed lines is still relatively low. It is worth noting that some vulnerabilities may require more time to address their root cause, but overall, when the vulnerability is detected, the fix can be done efficiently.

6 Threats to Validity

We base our results solely on commit messages, which may limit data accuracy. It is possible that changes were collected, which should not have been accounted for due to bad comments, or that changes in our scope were missed, due to missing commit message information.

Throughout the data collection, we used a strict filter that specifically checks for the mention of CWE in the commit messages to eliminate false positives. However, this approach also has limitations. Specifically, we may have discarded changes relevant to vulnerability fixes that did not mention the related CWE-ID in the commit message. As a result, the data on GitHub may contain more vulnerability-fixing commits than what we captured, but we were not able to detect them due to the filtering process. Although we aim to increase the quality of our data by discarding changes that do not specifically mention a CWE-ID, this may result in a smaller sample size. Therefore, future studies could explore alternative methods for collecting data on vulnerability fixes in software development, such as scraping and labeling the code directly with static analysis tools.

7 Conclusions and Future Work

Software quality is crucial and must be addressed throughout the software lifecycle. The increase in commits targeting cybersecurity flaws in OSS repositories indicates developers are more aware of the need to address security concerns in software development. Evolving security standards mandate the implementation of secure software development processes, such as the secure Software Development Life Cycle (sSDLC) outlined in the IEC 62443 standard, to improve software quality. However, the increasing usage of OSS brings unknown vulnerabilities into one's own software. Our analysis shows that the awareness of software vulnerabilities in OSS is increasing over the years but still low, suggesting that there is room for improvement.

We found a correlation between the programming language and the type of vulnerabilities, indicating that developers should consider the strengths and weaknesses of the programming language in terms of security. By understanding the prevalent vulnerabilities in different programming languages, developers can mitigate risks and ensure the security and integrity of their software. Staying up-to-date with the latest vulnerabilities and security trends is also crucial as new vulnerabilities can emerge and existing ones can phase out over time.

Once vulnerabilities are detected, our research shows that fixes can be done efficiently without introducing significant changes or rebasing to the codebase. Our database of fixes can be used to cross-check the results of Static Application Security Testing (SAST) tools, to check whether software quality problems are detected by the tool and if vulnerabilities found match the mentioned CWE-ID in the commit message.

By outlining the findings and challenges associated with our large-scale data acquisition approach, our research also contributes to understanding the quality of data required for training machine learning algorithms, as data quality can significantly impact algorithm performance. Homogeneous and complete datasets are essential for training models that can accurately detect and classify software security vulnerabilities.

In a future work, the authors would like to extend their analysis by exploring a comparative analysis with post-2022 commits. Additionally, we would like to explore how the acquired data can be leveraged for AI-assisted vulnerability detection and classification.

References

- 1 North Bridge / Blackduck. <https://tinyurl.com/blackduck2k15>. [Accessed: 24 Apr. 2024].
- 2 Department of Homeland Security, US-CERT. Software Assurance. Online, Accessed 27 September 2020. URL: <https://tinyurl.com/y6pr9v42>.
- 3 GitHub Octoverse. Top programming languages in 2022. <https://tinyurl.com/octoverse2k22>, 2022. [Accessed: 4 Apr. 2024].

- 4 Emanuele Iannone, Zadia Codabux, Valentina Lenarduzzi, Andrea De Lucia, and Fabio Palomba. Rubbing salt in the wound? a large-scale investigation into the effects of refactoring on security. *Empirical Software Engineering*, 28(4), May 2023. doi:10.1007/s10664-023-10287-x.
- 5 International Electrotechnical Commission. IEC 62443-4-1 – Security for industrial automation and control systems – Part 4-1: Secure product development lifecycle requirements. Technical report, International Electrotechnical Commission, Geneva Switzerland, January 2018.
- 6 International Organization for Standardization. ISO/IEC 25000:2014 – Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. Technical report, International Organization for Standardization, Geneva, CH, March 2014. URL: <http://iso25000.com/index.php/en/iso-25000-standards>.
- 7 Frank Li and Vern Paxson. A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2201–2215, New York, NY, USA, 2017. ACM. doi:10.1145/3133956.3134072.
- 8 Microsoft. Morgan Stanley Technology, Media & Telecom Conference - FY2023. <https://tinyurl.com/ynepy7jw>, 2023. Accessed: 15. Apr. 2024.
- 9 MITRE. CVE-2014-6271. <https://tinyurl.com/4dk6yfpz>. [Accessed: 15 April 2024].
- 10 MITRE. CVE-2015-0204. <https://tinyurl.com/3prfckfj>. [Accessed: 15 Apr. 2024].
- 11 MITRE. CVE-2021-44228. <https://tinyurl.com/2dejmr3e>. [Accessed: 15 Apr. 2024].
- 12 MITRE. Common Weakness Enumeration. cwe.mitre.org, 2023. [Accessed: 22 Apr. 2024].
- 13 National Security Agency Center for Assured Software. Juliet Test Suite C/C++ 1.3. <https://tinyurl.com/bdd9csvz>, 2023. [Accessed: 20 Apr. 2023].
- 14 Stephen O’Grady. The redmonk programming language rankings: June 2022. <https://tinyurl.com/4xpdr83z>, 2022. [Accessed: 20 Apr. 2024].
- 15 Kaan Oguzhan, Tiago Espinha Gasiba, and Akram Louati. How good is openly available code snippets containing software vulnerabilities to train machine learning algorithms? In *CYBER 2022, The Seventh International Conference on Cyber-Technologies and Cyber-Systems*, volume ISBN: 978-1-61208-996-6, pages 25–33. ThinkMind, 2022. [ISSN: 2519-8599].
- 16 TIOBE. Tiobe index. <https://tiobe.com/tiobe-index/>, 2023. [Accessed: 25 Apr. 2024].
- 17 Song Wang and Nachiappan Nagappan. Characterizing and understanding software developer networks in security development. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 534–545, 2021. doi:10.1109/ISSRE52982.2021.00061.
- 18 Xinda Wang, Shu Wang, Pengbin Feng, Kun Sun, and Sushil Jajodia. Patchdb: A large-scale security patch dataset. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 149–160, 2021. doi:10.1109/DSN48987.2021.00030.

Improving Industrial Cybersecurity Training: Insights into Code Reviews Using Eye-Tracking

Samuel Riegel Correia ✉ 


Instituto Universitário de Lisboa (ISCTE-IUL), ISTA, Portugal

Maria Pinto-Albuquerque ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Tiago Espinha Gasiba ✉ 

Siemens AG, München, Germany

Andrei-Cristian Iosif ✉ 

Universität der Bundeswehr München, Germany

Siemens AG, München, Germany

Abstract

In industrial cybersecurity, effective mitigation of vulnerabilities is crucial. This study investigates the importance of code reviews among cybersecurity professionals and analyses their performance in identifying vulnerabilities using eye-tracking technology. With the insights gained from this study, we aim to inform future tools and training in cybersecurity, particularly in the context of code reviews. Through a survey of industry experts, we reveal what tasks industry professionals consider the most important in mitigating cybersecurity vulnerabilities. A study was conducted to analyse how industrial cybersecurity professionals look at code during code reviews. We determined the types of issues our participants most easily discovered and linked our results with patterns and data obtained from an eye-tracking device used during the study. Our findings underscore the pivotal role of code reviews in cybersecurity and provide valuable insights for industrial professionals and researchers alike.

2012 ACM Subject Classification Security and privacy → Software and application security; Software and its engineering → Collaboration in software development; Information systems → Open source software; Security and privacy → Vulnerability management

Keywords and phrases code review, cybersecurity, development lifecycle, eye-tracking

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.17

Funding This work is partially financed by Portuguese national funds through FCT – Fundação para a Ciência e Tecnologia, I.P., under the projects FCT UIDB/04466/2020 and FCT UIDP/04466/2020. Furthermore, the first and second author thank the Instituto Universitário de Lisboa and ISTAR, for their support. Tiago Gasiba and Andrei-Christian Iosif acknowledge the funding provided by the Bundesministerium für Bildung und Forschung (BMBF) for the project CONTAIN with the number 13N16585.

1 Introduction

The security of software applications is crucial in today's digital landscape, in which cybersecurity threats continue to evolve in their sophistication and frequency [1]. As developers strive to create secure programs, analysing and understanding developers' cognitive processes when creating secure programs becomes crucial.

Code reviewing, a key task in the software development lifecycle, plays a pivotal role in creating secure code. It serves as a crucial measure in detecting vulnerabilities and other issues in code, making it an important task in the development of secure programs. The task of conducting a code review primarily involves reading code, this makes the use of eye-tracking technologies a fitting approach when studying developers' cognitive processes.



© Samuel Riegel Correia, Maria Pinto-Albuquerque, Tiago Espinha Gasiba, and Andrei-Cristian Iosif; licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 17; pp. 17:1–17:9

OpenAccess Series in Informatics



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

17:2 Improving Industrial Cybersecurity Training

Although eye-tracking has been used in several coding and code interpretation studies, very few studies have focused on cybersecurity. This study aims to explore this underrepresented yet crucial subcategory of eye-tracking studies.

We have two main research questions:

- RQ1** What tasks in the software development life cycle do industrial cybersecurity professionals consider to be the most crucial in mitigating cybersecurity vulnerabilities?
- RQ2** How do industrial cybersecurity professionals analyse code while attempting to find cybersecurity vulnerabilities?
- a) How successful are industrial cybersecurity professionals at finding cybersecurity vulnerabilities while performing code reviews?
 - b) Is there a relation between the patterns revealed using eye-tracking technology and the code reviewers' success in spotting the vulnerabilities?

By gaining insight into the thought processes of industrial cybersecurity professionals during code reviews, we seek to determine which practices are best suited to enhancing software security. With this information, we can suggest how to improve training, resources, and processes to enhance development practices and make them more secure.

2 Related Work

Related work in this field includes literature reviews of objectives and techniques important to analysing software developers' coding behaviour (e.g., [5]). Considering the current state of the art, we determined the following tasks to be crucial in cybersecurity and, additionally, be suitable candidates for analysis through an eye-tracking study:

- a) Code reviewing
- b) Analysis of code review tool outputs
- c) Reading documentation
- d) Researching online resources (e.g. Stack Overflow or other community-based resources)

From these, code reviews stood out as the most promising and interesting task to study. Code reviews are essential in detecting vulnerabilities and other issues in code. They are commonplace in development lifecycles and have been set as requirements in industrial standards such as ISO/IEC 62443.

Eye-tracking technology has been used in software engineering research to study various tasks in the development lifecycle, such as code comprehension, debugging, and code reviews [7]. This technology can record multiple aspects of participant behaviour, including visual focus, attention, interactions, and reading patterns, making it highly useful for research studies.

Generally, the articles related to eye-tracking use in cybersecurity are closely related to education in some form or another. These articles are either directly related to how we can create better pedagogical frameworks to educate individuals on cybersecurity (e.g., [4], [2]), or how we could adapt provided learning materials such as API documentation to further safe cybersecurity practices (e.g., [6]).

Most articles about programming are also relevant to cybersecurity, as analysing how individuals look at code also provides insight into how they deal with specific cybersecurity challenges.

3 Methodology

To answer RQ1, we developed a survey in which participants were asked to evaluate the four tasks mentioned previously: code reviewing, analysis of code review tool outputs, reading the documentation, and researching online resources, in terms of their perceived importance to cybersecurity using a five-point Likert scale. Besides obtaining the participants' opinions on the importance of these tasks, we also acquired some of their background information to help us describe the respondents. All participants are industrial cybersecurity professionals currently actively working in this field.

As for RQ2, we found that the best approach to answering this question lies in creating a study in which participants are presented with several code snippets and, for each one, are tasked with determining any vulnerabilities present in the code. The task we created was designed to simulate a code review.

The survey and the eye-tracking experiments were conducted with the same individuals, with the survey being conducted with participants before the code review experiment. These two parts, on average, took ≈ 8 and ≈ 21 minutes respectively. Our study was conducted in April 2024.

For this study, we used the Gazepoint GP3 Eye-tracking device in conjunction with the Open Gaze and Mouse Analyzer (OGAMA)¹ software which was used to create, record, and analyse the experiments.

Participants were presented with code snippets in C++ representing the five most common vulnerabilities according to the number of registered occurrences on CVEdetails.com². These vulnerabilities' CWE IDs are CWE-79, CWE-119, CWE-89, CWE-20, and CWE-787. The code snippets were ordered in terms of difficulty, from the least to the most complex to analyse:

1. CWE-787 – Out-of-bounds Write
2. CWE-119 – Buffer Overflow
3. CWE-20 – Improper Input Validation
4. CWE-89 – SQL Injection
5. CWE-79 – Cross-site Scripting

4 Results

Participants

As mentioned previously, all participants in our study are industrial cybersecurity professionals currently working in this field. Besides answering RQ1, the survey we created allowed us to obtain some background information on the participants, which, in turn, would help us draw some conclusions from the phenomena we observed during the experiments.

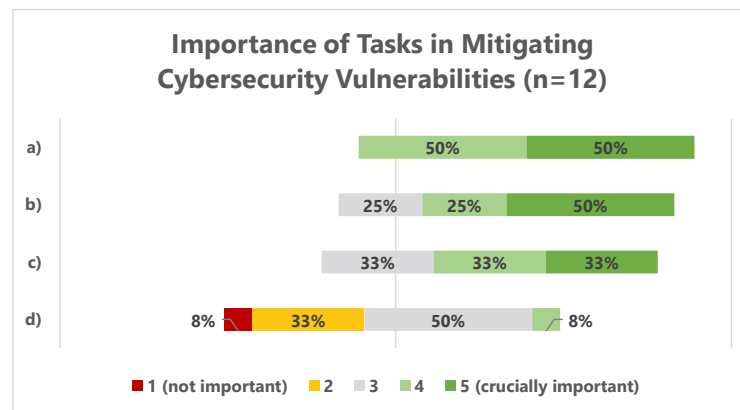
A total of 12 individuals participated in the study. All were above the age of 24, with eight being between 25 and 34 years of age, and only one over 55. Two participants were female, and the remaining nine were male.

Regarding education, the participants have varying degrees, including one bachelor's degree, four doctorates, and seven master's degrees. Naturally, participants with higher degrees of education also had, generally, more years of work experience in cybersecurity. The participants with the least work experience in this field stated they had three years' worth of experience, and the most out of all participants was 25 years.

¹ <http://www.ogama.net/>

² <https://www.cvedetails.com/>

17:4 Improving Industrial Cybersecurity Training



■ **Figure 1** Importance given by participants to tasks in mitigating cybersecurity vulnerabilities.

RQ1 – What tasks in the software development life cycle do industrial cybersecurity professionals consider to be the most crucial in mitigating cybersecurity vulnerabilities?

Participants were asked to evaluate the following tasks on a scale from one (not important) to five (crucially important) regarding their assigned importance in mitigating cybersecurity vulnerabilities.

- a) Code reviewing
- b) Analysis of code review tool outputs
- c) Reading documentation
- d) Researching online resources (e.g. Stack Overflow or other community-based resources)

From this question, we obtained the results in Figure 1.

Professionals consider task a), code reviews, among the most critical tasks when mitigating cybersecurity vulnerabilities. In our survey, code reviewing was given an importance of four or five out of five by all of our survey participants, and many considered it the most important task.

Both b) and c) got similar results being considered, generally, less critical than a) but also having a quite positive average rating of ≈ 4.09 and ≈ 4.01 respectively.

One observation we made was that task d), which involves researching online resources, was considered relatively unimportant by our participants, with an average rating of approximately 2.4. Participants explained that community-based online resources are important for software developers when solving programming issues, but not ideal for addressing cybersecurity vulnerabilities.

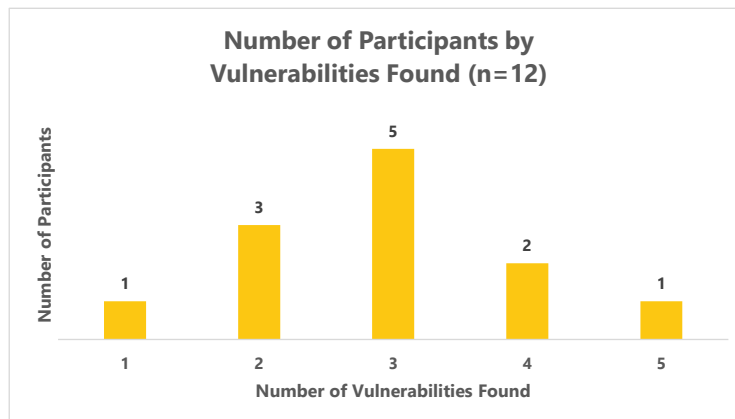
An open-ended question was also included in the survey, asking the participants if there were additional tasks they considered important in mitigating cybersecurity vulnerabilities. Many chose to answer this question, with the most common answers including code testing, penetration testing, and secure coding training/workshops, all significant activities and tasks in cybersecurity.

RQ2 a) – How successful are industrial cybersecurity professionals at finding cybersecurity vulnerabilities while performing code reviews?

For our analysis, we created an AOI, referred to as the target, around the lines of code in these snippets which we considered to contain the main vulnerability to be discovered by the participants.

■ **Table 1** Results on the analysis of code snippets with cybersecurity vulnerabilities.

Vulnerability	CWE-787	CWE-119	CWE-20	CWE-89	CWE-79
Discovery Rate	17%	50%	42%	100%	83%
Avg. Time Analysed (s)	234	289	259	214	242
Avg. Time to Find Vuln. (s)	205	168	137	91	189
Avg. Time Analysing Target (s)	51	100	51	23	36



■ **Figure 2** Number of Participants by Vulnerabilities Found.

The accuracy of responses varied considerably between the code snippets. For instance, in our first snippet of CWE-787, only two users identified the vulnerability in the code, while all users identified the vulnerability in the CWE-89 code snippet. Table 1 shows an overview of the results.

CWE-89 and CWE-79 stand out by being the most easily identified, by a considerable margin. These two vulnerabilities correspond to SQL injection and cross-site scripting (XSS), respectively, and are likely the most commonly discussed code vulnerabilities. This leads to the conclusion that the fact that these vulnerabilities are so well-known by professionals made them stand out and be easily identifiable.

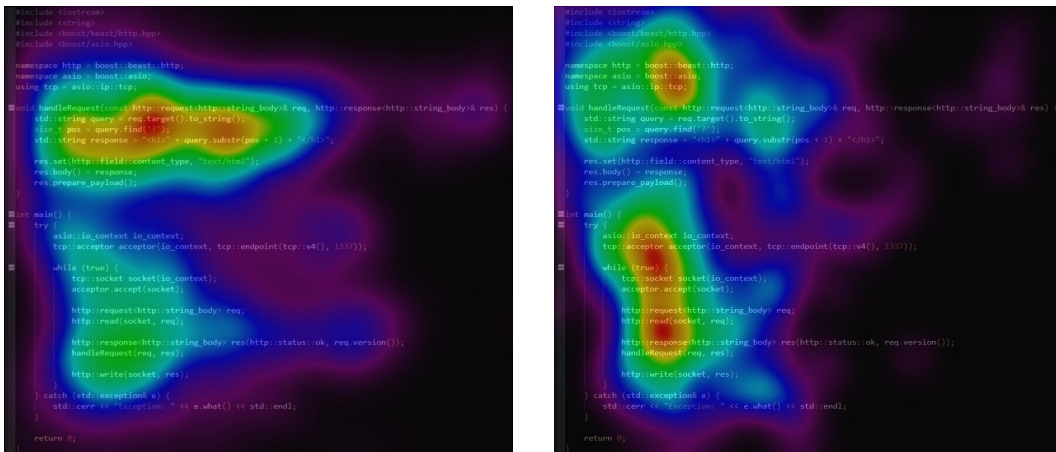
CWE-89 was the fastest to be found by our participants, while CWE-79 was one of the code snippets in which they took the longest to find vulnerabilities, even though these programs were quite similar in size. By watching the recordings made with the eye-tracking software, we see that our participants followed the code's execution path, which, for the code snippet on CWE-79, took fairly long before encountering the vulnerable code.

RQ2 b) – Is there a relation between the patterns revealed using eye-tracking technology and the code reviewers' success in spotting the vulnerabilities?

The average number of vulnerabilities found per participant is ≈ 3 , Figure 2 shows the distribution of the participants by the number of vulnerabilities they detected out of the five considered.

When comparing the number of vulnerabilities found with the background information provided by participants, we found no strong correlations. The years of experience and educational degree showed correlation values of about 0.08 and 0.31, respectively.

Next, we compared the results of participants who discovered the vulnerability, with those who did not, for each vulnerability. From this, we found that, on average, participants who didn't find the vulnerabilities looked at the code snippets and targets longer than those who



(a) CWE-89 Heatmap – Participants who discovered the vulnerability.

(b) CWE-89 Heatmap – Participants who did not discover the vulnerability.

■ **Figure 3** Heatmaps for Code Snippet of CWE-89.

■ **Table 2** Average fixation rates of participants (fixations per second).

Average	Standard Deviation	Minimum	Maximum
3.86	1.43	1.73	5.96

did. We also found moderate correlations which indicate that participants who discovered more vulnerabilities looked at the code and targets for less time with values ≈ -0.4 and ≈ -0.5 , respectively.

Heatmaps were created which helped us compare the fixation times on different parts of the code and targets, revealing potential differences between participants who discovered vulnerabilities and those who did not; an example of this can be seen in Figure 3.

Between the various heatmaps we compared, we noticed that, for most code snippets, participants who correctly identified vulnerabilities spent considerably more time looking at the parts of the programs with vulnerable code. Furthermore, in some snippets, such as the one seen in the images above, the high performers had a more focused approach to analysing the code, concentrating on a few key program elements.

We also examined our participants’ gaze paths to determine if their code-reading strategies somehow impacted their performance. However, we are not able to draw any definitive conclusions on this since all types of participants followed some recognisable patterns, such as following the instructions from the main method and stepping into the functions that are called.

We then looked at the fixation rates of our participants, finding a meaningful correlation of ≈ -0.59 between the time spent analysing the snippets and their fixations per second or fixation rate. This correlation indicates that, on average, users with a higher fixation rate spent less time reading the code. We also found that the fixation rates were fairly consistent for each individual but varied quite significantly between the participants. Statistics on the fixation rates of our participants can be seen in Table 2.

When comparing the number of vulnerabilities found to each participant’s average fixation rate, we see that a moderate correlation of ≈ -0.36 exists between the number of vulnerabilities discovered and the fixation rate of participants. This value indicates that participants who discovered more vulnerabilities had, on average, a lower average fixation rate.

5 Discussion

From the survey we created, our participants indicated that, out of the four tasks they were presented with, code reviewing was the most important in mitigating cybersecurity vulnerabilities. This demonstrates this task's significance in industrial cybersecurity and underscores the need for robust and effective code review practices.

Researching online resources was seen as the least important task from the list we presented. Participants indicated that they gave this task low importance despite being commonplace in program development because, specifically from the perspective of mitigating cybersecurity vulnerabilities, community resources can be unreliable and industry-followed resources such as standards and official documentation are preferable. Additionally, multiple participants referenced tasks such as penetration testing and secure coding training as being very important.

Our code review experiments revealed some interesting results. Firstly, SQL Injection and XSS vulnerabilities were detected at a much higher rate than the other vulnerabilities related to issues with memory allocation and buffer over/underflows. We believe, participants identified these issues quickly because the patterns for SQL Injection and XSS were very well-known. This indicates a need for increased awareness and training on other cybersecurity vulnerabilities, particularly those related to memory management and buffer handling, which may not be as widely recognised or understood.

As for the time it took participants to discover each of the vulnerabilities, the code snippets we selected are not ideal for this comparison, as several factors have to be considered to compare the vulnerability detection time, other than the vulnerabilities themselves. To determine what kinds of vulnerabilities take longer to be found, an experiment must be created which addresses the following two problems: first, different types of vulnerabilities require different program structures giving context to the vulnerable code which may vary in their ease of interpretation. Second, the participants' code scan path must be carefully considered as, ideally, the time it takes a user to read any code snippet before reaching the vulnerable code should be the same; this point has been discussed in other publications (e.g., [3]).

Through heatmaps, we determined that participants who correctly identified vulnerabilities had a seemingly more focused approach in looking at the code, usually focusing on the parts of the code snippets containing the vulnerabilities. This leads us to believe that individuals with more knowledge of the vulnerabilities are quicker to find them as they look at the program more efficiently. These results warrant further investigation as we can use the insight into how successful code reviews are conducted to teach people how to replicate this success.

The scan paths of our participants were also analysed; this was accomplished by reviewing the eye-tracking recordings of our participants. Some publications have found patterns in the code analysis process of experts when compared to that of novices[3], however, at this time we are not able to determine any specific strategies which led to better results during the experiment.

The eye-tracking data we obtained also included information on the fixation rates of our participants. Some authors have linked higher fixation rates with increased effort, interest, and exploration, while lower fixation rates may indicate a lower efficiency in tasks such as finding vulnerabilities in code [7][8]. In our experiment, participants who had spent less time looking at the code snippets were found to have higher fixation rates possibly indicating higher involvement in reading the code. However, we also found that participants who discovered fewer vulnerabilities had higher average fixation rates, which may indicate an increased effort in interpreting these code snippets, possibly leading to worse performance.

A limitation of our survey and study was the small sample size, this was mostly a product of the selection criteria we applied when choosing our participants. This restricts our confidence in the results and the analysis we can conduct with the data.

As for the future direction of this work, a more in-depth data analysis will be conducted, additional AOI will be created to enhance our analysis, and other code snippets, besides those analysed here, will be considered.

6 Conclusions

We conducted a survey and study on code reviews with industrial cybersecurity professionals. Our main objectives included determining how important these experts consider code reviews, determining how well they perform during code reviews, and analysing their performance with the help of eye-tracking technologies.

We conclude that industry professionals consider code reviews critical in mitigating cybersecurity vulnerabilities. Tasks such as the consultation of community-based resources (e.g., Stack Overflow), are considered less than ideal for cybersecurity as the information may be unreliable, and better sources, such as industry standards and official documentation, are more appropriate in this field.

By a considerable margin, SQL Injections and Cross-site Scripting (XSS) vulnerabilities were the most commonly detected vulnerabilities. This result can be explained by the fact that these two vulnerabilities are some of the most well-known and frequently discussed cybersecurity vulnerabilities.

When comparing the performances of those who discovered vulnerabilities and those who did not, we found that those who correctly identified vulnerabilities had a more focused approach to analysing the code and a slightly lower fixation rate, previously linked to lower efficiency in tasks such as code reviews. This insight into how successful code reviews are conducted warrants further investigation as it may help us teach people how to replicate this success.

In our future work, we plan to analyse data with different methods and explore other code snippets included in the study which weren't discussed here. This will allow us to provide further insights into how cybersecurity education should be adapted to improve performance during code reviews.

In summary, our study emphasises the crucial role of code reviews in cybersecurity. We also saw the importance of following certain code analysis patterns, and that exposure to different vulnerabilities is invaluable for code reviewers as commonly discussed issues were easily recognised.

References

- 1 Federal Cyber Security Authority. The state of it security in germany in 2023. *Federal Office for Information Security*, 2023.
- 2 Leon Bernard, Sagar Raina, Blair Taylor, and Siddharth Kaza. Minimizing cognitive load in cyber learning materials – an eye tracking study. In *ACM Symposium on Eye Tracking Research and Applications*, volume PartF169257. Association for Computing Machinery, May 2021. doi:10.1145/3448018.3458617.
- 3 Teresa Busjahn, Simon, and James H. Paterson. Looking at the main method - an educator's perspective. In Otto Seppälä and Andrew Petersen, editors, *Koli Calling '21: 21st Koli Calling International Conference on Computing Education Research, Joensuu, Finland, November 18 - 21, 2021*. Association for Computing Machinery, November 2021. doi:10.1145/3488042.3488068.

- 4 Daniel Kyle Davis and Feng Zhu. Understanding and improving secure coding behavior with eye tracking methodologies. In J. Morris Chang, Dan Lo, and Eric Gamess, editors, *Proceedings of the 2020 ACM Southeast Conference, ACM SE '20, Tampa, FL, USA, April 2-4, 2020*, ACM SE '20, pages 107–114, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3374135.3385293.
- 5 Daniel Kyle Davis and Feng Zhu. Analysis of software developers' coding behavior: A survey of visualization analysis techniques using eye trackers. *Computers in Human Behavior Reports*, 7, August 2022. doi:10.1016/j.chbr.2022.100213.
- 6 Peter Leo Gorski, Sebastian Möller, Stephan Wiefeling, and Luigi Lo Iacono. 'i just looked for the solution!' on integrating security-relevant information in non-security api documentation to support secure coding practices. *IEEE Transactions on Software Engineering*, 48:3467–3484, September 2022. doi:10.1109/TSE.2021.3094171.
- 7 Zohreh Sharafi, Yu Huang, Kevin Leach, and Westley Weimer. Toward an objective measure of developers' cognitive activities. *ACM Transactions on Software Engineering and Methodology*, 30, May 2021. doi:10.1145/3434643.
- 8 Zohreh Sharafi, Bonita Sharif, Yann Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha Crosby. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering*, 25:3128–3174, September 2020. doi:10.1007/s10664-020-09829-4.

Using ChatGPT During Implementation of Programs in Education

Norbert Baláž 

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

Jaroslav Porubán¹  

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

Marek Horváth  

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

Tomáš Kormaník  

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

Abstract

This paper examines the impact of ChatGPT on programming education by conducting an empirical study with computer science students at the Department of Computers and Informatics at the Technical University in Košice. The study involves an experiment where students in a Component Programming course use ChatGPT to solve a programming task involving linked lists, comparing their performance and understanding with a control group that does not use the AI (artificial intelligence) tool. The task necessitated the implementation of a function to add two numbers represented as linked lists in reverse order. Our findings indicate that while ChatGPT significantly enhances the speed of task completion – students using it were nearly three times quicker on average – it may also detract from deep understanding and critical thinking, as evidenced by the uniformity and superficial engagement in solutions among the ChatGPT group. On the other hand, the group working independently displayed a broader variety of solutions and deeper interaction with the problem, despite slower completion times and occasional inaccuracies. The results highlight a dual-edged impact of AI tools in education: while they enhance efficiency, they may undermine the development of critical thinking and problem-solving skills. We discuss the implications of these findings for educational practices, emphasizing the need for a balanced approach that integrates AI tools without compromising the depth of learning and understanding in students.

2012 ACM Subject Classification Software and its engineering → Software creation and management

Keywords and phrases generative artificial intelligence, chatbot, ChatGPT, prompt engineering, source code generation

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.18

Funding This work was supported by project VEGA No. 1/0630/22 Lowering Programmers Cognitive Load Using Context-Dependent Dialogs.

1 Introduction

In recent years, AI (artificial intelligence) has begun to develop at a tremendous speed, while its applications have become an integral part of our daily lives. Among its newest and most exciting areas is its use for code generation, where machine learning models such as OpenAI's ChatGPT show significant potential for automating and streamlining software development. This technology impacts both students [1] and educators [2] in the realm of software-related education.

¹ Corresponding author



Since its creation, ChatGPT has been used by students around the world to help with a variety of tasks, from writing school papers to implementing term projects [16]. Multiple research papers are exploring ChatGPT, its benefits, and its drawbacks when used in education. The paper [9] highlighted ChatGPT's varied performance across different subject domains and its potential benefits when serving as an assistant for instructors and as a virtual tutor for students. The paper [6] provides an examination of issues and explores the potential use of ChatGPT in educational contexts. The paper [5] explores the potential and problems associated with applying advanced AI models in education. A systematic review of the literature and an analysis of the impact of the application of the ChatGPT tool in education are presented in [10]. Authors in the [11] assess the efficacy of employing the ChatGPT language model to generate solutions for coding exercises within an undergraduate Java programming course. In this paper, we will explore the impact of ChatGPT on education, specifically focusing on its role in learning programming.

A study detailed in [16] involved 41 university students (33 males and 8 females) aged 19–25, examining the impact of ChatGPT on programming education. Throughout an eight-week period, students utilized ChatGPT for their weekly object-oriented programming projects. The study concluded with a questionnaire where students shared their insights, providing an evaluation of ChatGPT's advantages and disadvantages based on their experiences.

The study highlights the benefits of using ChatGPT in programming education, including its ability to provide quick and largely accurate responses and boost confidence in coding. However, it also points out some significant drawbacks, such as promoting laziness, producing occasional inaccuracies, and raising concerns about future job security. The findings suggest that while ChatGPT can be an effective tool in programming education, its utilization needs careful moderation to address both its positive aspects and potential pitfalls.

The reliability of ChatGPT's responses remains a contentious issue, especially in educational settings. The impact of ChatGPT on programming education continues to be an area of active research, focusing on its practical uses, benefits, drawbacks, and ethical considerations [1]. Initially, some educational bodies responded to the rise of AI tools like ChatGPT by banning them, as occurred in early 2023 [14]. As attitudes changed within months, bans were lifted and AI resources were recommended for educational use.

For the correct implementation of ChatGPT in education, it is important to choose the right ways of using it. AI can support creative thinking when solving problems, but it is also simple to use it just to generate the necessary code for a given task, often without any deeper understanding of the problem.

In our study conducted at the Department of Computers and Informatics at the Technical University in Košice, we designed an experiment involving computer science students. The experiment required students enrolled in the Component Programming course to develop and code a solution to a specified task using Java, followed by completing a questionnaire about their understanding of the task, their evaluation, and their opinions on using ChatGPT. We divided the students into two groups, allowing one to use ChatGPT and prohibiting the other.

We hypothesize that while ChatGPT might improve task completion speed and productivity, it may detract from students' understanding of the task, critical thinking, and problem-solving creativity. The questionnaire aimed to gather students' perspectives on the use of ChatGPT and how they personally utilized the tool. We anticipated that the group using ChatGPT, despite potentially completing tasks correctly, might exhibit a poorer grasp of the task. Conversely, the group without ChatGPT might demonstrate a deeper understanding of the problem or proposed solutions, even if their implementations were not fully functional.

2 Experiment

In this section we present the experiment of using ChatGPT by students in a Component Programming course to solve the programming task. The task concerns the addition of two numbers, which are represented as a linked list, where each node represents one digit of the whole number. This task complies with our department's standards [4].

Text of the task: We have entered 2 integers represented as the linked list format so that each node represents a digit in the number. Nodes form a number in reverse order, e.g. 1 -> 2 -> 3 -> 4 -> 5 is the number 54321. Complete the provided code so that it returns the sum of the two provided numbers in this format. For example: 9 -> 9 and 5 -> 2 returns 124 (99 + 25) represented as 4 -> 2 -> 1. Provided code:

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
    }
}

class AddLinkedLists {

    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        // TODO: implement this function for adding 2 numbers
    }

    public static void main(String[] args) {
        ListNode l1 = new ListNode(0);
        l1.next = new ListNode(1);

        ListNode l2 = new ListNode(3);
        l2.next = new ListNode(2);

        AddLinkedLists solution = new AddLinkedLists();
        ListNode result = solution.addTwoNumbers(l1, l2);

        while (result != null) {
            System.out.print(result.val + " ");
            result = result.next;
        }
    }
}
```

In the designated coding exercise, numbers are encapsulated within linked lists with the least significant digit at the head of the list. Students are tasked with implementing the `addTwoNumbers` function, which takes two such linked lists, `l1` and `l2`, as inputs. This function adds the numbers represented by these lists and returns a new linked list that encapsulates the sum, with digits again in reverse order, starting from the least to the most significant.

The challenge lies in correctly iterating through both lists, summing corresponding digits, and managing any carry-over that occurs when digits sum to 10 or more. This process begins with the head of each list, ensuring that the addition mirrors the operation of adding numbers from their least significant digits upwards.

This task, while straightforward for ChatGPT with its ability to quickly generate the correct solution, presents a significant challenge for introductory programming course students without AI assistance. These students must not only develop the solution independently, but they must also deeply understand the underlying problem and manage the complexities of linked list manipulation and digit-wise addition. The experiment familiarized the students

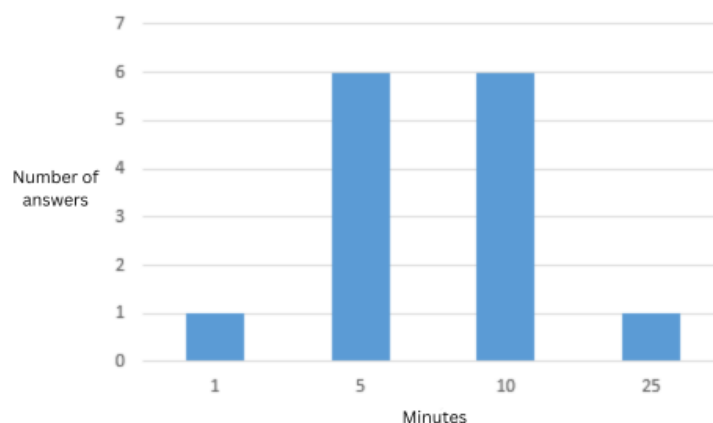
18:4 Using ChatGPT During Implementation of Programs in Education

not only with the task but also with the problem, laying the groundwork for a comparative analysis of problem-solving approaches and the depth of understanding between the two groups.

A total of 23 students worked on the assignment. There were 14 students in the first group who worked with ChatGPT, and 9 in the second group who worked without ChatGPT support. Students were in different study groups, which caused a significant disproportion in group size due to absences. Students solved the task and filled out the questionnaire during the exercise from the Component programming course. Their solutions can be further used in analysis or comparison with previously created relevant datasets [15].

2.1 ChatGPT Group

In the group where students used ChatGPT to solve the task, all participants successfully implemented the correct solution. The times taken to complete the task varied, with an average duration of 8 minutes and 17 seconds. The quickest completion time recorded was 1 minute, and the slowest was 25 minutes, as depicted in the Fig. 1.



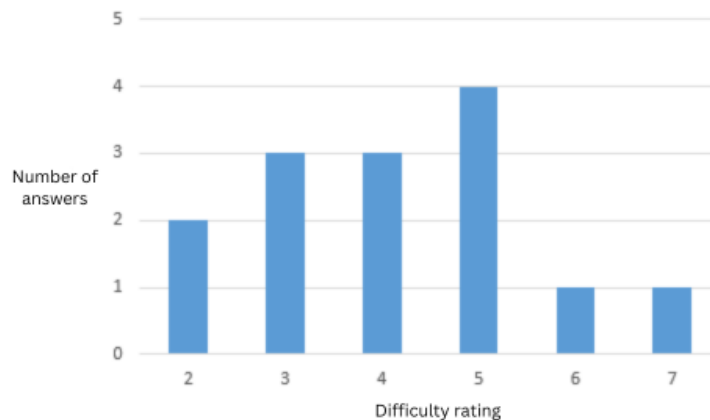
■ **Figure 1** ChatGPT group task solving time.

Analysis of the interactions between students and ChatGPT revealed that the average number of prompts (i.e., questions or commands given to ChatGPT) per student was 3.7, with the range being from 1 to 9 prompts. The transcripts of these interactions revealed highly uniform solutions, primarily generated by ChatGPT with minimal modification by the students.

This uniformity in the solutions and reliance on ChatGPT's outputs could explain why students perceived the task as relatively easy, rating its difficulty as 4 out of 10 on average. This rating is visualized in Fig. 2, which illustrates the students' perceived challenge of the task. This perception highlights the impact of AI tools like ChatGPT in simplifying complex tasks, but also raises concerns about the depth of understanding and engagement in problem-solving when using such tools.

2.1.1 Understanding the problem

The questionnaire, which included specific questions like "Why are numbers in a linked list represented in reverse?", assessed students' understanding of problem-solving. The first group had access to ChatGPT for assistance. Responses varied, with some students clearly recognizing benefits like simplified manipulation and more efficient operations, while many answers were vague or off-topic, indicating a lack of engagement with the material.



■ **Figure 2** ChatGPT group difficulty rating.

Further questions asked students to explain the logic behind their solutions, even if incorrect. Many demonstrated an understanding of adding numbers via linked lists, with responses varying from detailed technical explanations involving carry transfers and pointer movements to concise summaries. However, some responses were unclear or irrelevant, indicating confusion or disinterest.

The final question probed whether students gained new insights or improved their mastery of the data structure. Responses were split, with half reporting new knowledge or enhanced understanding and the other half noting no significant learning. This variation underscores the differing levels of comprehension and engagement with the task.

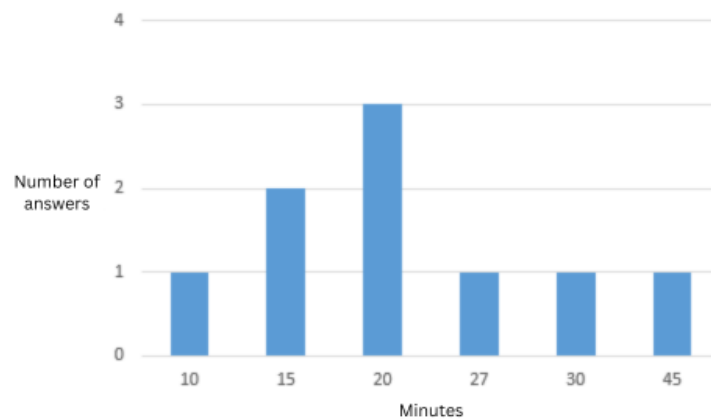
2.2 Independent Group

In the independent group, 9 students attempted to implement the task without ChatGPT assistance, and 6 of them managed to solve it correctly. However, a closer look at the code revealed some discrepancies. Although one student implemented the solution correctly, they failed to convert the result back into a list format. Another student's code only outputs the tenth digit of the result. This latter error, though minor and quickly rectifiable, seemed to stem from a lack of careful code testing before submission, especially since this student took the longest to complete the task. Despite these issues, since the primary focus of this experiment was not on code perfection but on solving the problem, these solutions were deemed correct.

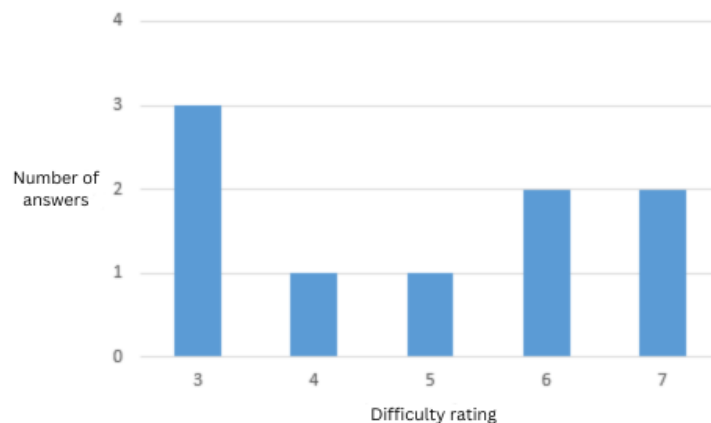
The average time taken by this group to complete the task was 22 minutes and 26 seconds, as illustrated in Fig. 3. Students who solved the problem correctly had an average completion time of 23 minutes and 40 seconds, with the quickest solution taking 15 minutes and the slowest taking 45 minutes. Those who did not solve the task spent an average of 20 minutes, with times ranging from 10 to 30 minutes.

Overall, this group rated the task's difficulty as 5 out of 10, with individual evaluations depicted in the Fig. 4. This suggests a moderate level of challenge perceived by the students who tackled the task independently.

18:6 Using ChatGPT During Implementation of Programs in Education



■ **Figure 3** Independent group task solving time.



■ **Figure 4** Independent group difficulty rating.

2.2.1 Correct solutions

The analysis of correct solutions using verified methods [12] has shown several diverse approaches. Some students converted linked lists into integers, summed them, and converted the sum back into a linked list. This method demonstrates a straightforward application of mathematical operations and transitions between number formats, but it poses a risk of precision issues or memory overflow with large numbers. Meanwhile, a different approach involved the use of *StringBuilder* to reverse and manipulate the numbers, showcasing creativity and originality, even if this method might be less efficient in some scenarios.

The most accurate solution to the task involved simultaneously traversing both lists, summing corresponding nodes, and properly managing carryover from one digit to the next. This approach was particularly effective as it minimized type conversions and directly manipulated the nodes of the linked lists, aligning closely with the task's objectives.

2.2.2 Understanding the problem

Responses to the analysis of why linked lists often store numbers in reverse showed a wide range of understanding levels. Most were concise; some addressed the key point directly, others came close, and a few admitted complete ignorance.

The majority of students demonstrated at least a basic grasp of the rationale behind storing numbers in reverse order, particularly highlighting how this facilitates the addition process. Notably, even students who struggled with the task recognized the purpose of this data structure orientation.

The responses to questions regarding the logic behind their solutions revealed a diversity of approaches and depths of understanding. This variability provides insights into the students' strategic thinking and problem-solving skills, even though not all solutions were optimal or adhered strictly to best practices.

In their explanations, most students described converting the numbers represented by the lists into whole numbers, adding these, and then converting the result back into a linked list. This method aligns with the intuitive logic commonly used in everyday number manipulation and was the predominant approach even among those who did not complete the task successfully. Other students, who briefly described their working solutions, mentioned techniques like using *StringBuilder* for manipulating strings and numbers.

In response to the question about gaining new knowledge or a better understanding of the given data structure, the group's feedback was predominantly negative, with seven indicating no new insights and only two reporting positive learning outcomes. This suggests that most students were already quite familiar with the data structure in question.

3 Conclusion

In this paper, we investigate the effect of using ChatGPT on programming education by conducting a limited-scale empirical study with introductory programming course students. After analyzing both groups of students and their responses to the programming task, we can draw several conclusions about the impact of using ChatGPT versus working independently:

- **Speed of Completion:** Students who used ChatGPT finished their tasks much faster, nearly three times quicker on average. This efficiency demonstrates the capability of AI to streamline problem-solving processes.
- **Depth of Understanding:** The speed advantage for the ChatGPT group came at a cost. Most of their solutions were very similar, suggesting heavy reliance on AI without much alteration. Many students accepted the generated solutions without deeply engaging with their content or understanding their functionality, leading to a superficial grasp of the tasks. However, about a third of these students showed they might have the potential to solve the problems independently, indicating some retained problem-solving abilities.
- **Diversity of Solutions and Critical Thinking:** On the other hand, the independent group, while slower, displayed a wider variety of solutions and tended to describe their methods simply and effectively. This not only shows a broader scope of creativity but also suggests a deeper interaction with the task, which can enrich the learning experience.
- **Approach to Problem Solving:** Independent students often used approaches similar to traditional paper-and-pencil methods, reflecting an intuitive and straightforward way of thinking. This method suggests that they relied on fundamental problem-solving and mathematical reasoning skills rather than automated processes.

These findings underscore the need for a balanced approach in educational settings that thoughtfully integrates the use of AI tools like ChatGPT with traditional learning techniques. This strategy ensures that students not only achieve quick solutions but also deeply understand the processes and principles involved, thereby cultivating their critical thinking and problem-solving skills. Their solutions should be examined carefully, since overuse of ChatGPT without actual understanding of the created solution can be considered

plagiarism, which can create issues for the evaluator [3] and students of its own. Naturally, source codes created by ChatGPT can be somewhat different in individual instances, but their overall similarity can be easily detected, as is confirmed in previous research conducted on our department [8]. We also observed that multiple results that utilized ChatGPT were not the best possible ones but rather the most common ones, which is also proven by related work [7].

One notable observation is that students who worked without ChatGPT frequently gravitated towards more traditional methods of solving the task. This choice suggests a deeper level of engagement with the problem and a more inventive approach to finding solutions. Even though not all these attempts were successful, the students' ability to devise potential strategies on their own demonstrates their capacity for critical thinking and independent analysis. This contrast with the AI-assisted group highlights how reliance on technology can sometimes bypass the deeper learning processes involved in problem-solving.

When interpreting the results and conclusions of this experiment, it is necessary to consider validation risks that may affect the generality and accuracy of our findings:

- Sample size – With a total of 23 participants, the sample is relatively small, which may limit the statistical significance of the findings and their applicability to a broader student population.
- Group heterogeneity – Dividing students into two groups may introduce hidden differences beyond just access to ChatGPT (such as prior programming experience, motivation, or personal preferences), which could bias the results.
- Limited scope of tasks – The experiment focuses on only one specific programming task, which may limit the ability to apply the findings to other types of tasks or subjects.

Using ChatGPT and similar tools to solve school assignments and projects can significantly increase the speed and productivity of students, but at the same time, it can have a potentially negative effect on their understanding of the subject matter, their ability to think critically, and their creativity. Conducted experiment thus confirmed our assumption. The findings support the results presented by Savelka et al. [13] about students and by Balse et al. [2] regarding mentors.

A suitable approach could be the combined use of ChatGPT as a tool for obtaining quick information or solution proposals while simultaneously ensuring a deeper study of the material and independent problem solving.

References

- 1 Christos-Nikolaos Anagnostopoulos. Anagnostopoulos CN. ChatGPT impacts in programming education: A recent literature overview that debates ChatGPT responses [version 1; peer review: 1 approved with reservations]. *F1000Research*, 2023. doi:10.12688/f1000research.141958.1.
- 2 Rishabh Balse, Prajish Prasad, and Jayakrishnan Madathil Warriem. Exploring the potential of GPT-4 in automated mentoring for programming courses. In *ACM Conference on Global Computing Education Vol 2*, page 191, 2023. doi:10.1145/3617650.3624946.
- 3 Miroslav Binas. Affecting students behavior with plagiarism detection in evaluation process. In *International Conference on Overcoming the Challenges and Barriers in Open Education*, pages 441–441, 2018.
- 4 Miroslav Binas and Emilia Pietrikova. Useful recommendations for successful implementation of programming courses. In *IEEE International Conference of Emerging Elearning Technologies and Applications*, pages 397–401, 2014. doi:10.1109/ICETA.2014.7107618.
- 5 Simone Grassini. Shaping the future of education: exploring the potential and consequences of AI and ChatGPT in educational settings. *Education Sciences*, 13(7):692, 2023. doi:10.3390/educsci13070692.

- 6 Mohanad Halaweh. ChatGPT in education: Strategies for responsible implementation. *CONTEMPORARY TECHNOLOGY*, 2023. doi:10.30935/cedtech/13036.
- 7 Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. Exploring the responses of large language models to beginner programmers' help requests. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, ICER '23, pages 93–105, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3568813.3600139.
- 8 Marek Horvath and Emilia Pietrikova. An experimental comparison of three code similarity tools on over 1,000 student projects. In *IEEE World Symposium on Applied Machine Intelligence and Informatics*, pages 423–428, 2024. doi:10.1109/SAMI60510.2024.10432863.
- 9 Chung Kwan Lo. What is the impact of ChatGPT on education? a rapid review of the literature. *Education Sciences*, 13(4):410, 2023. doi:10.3390/educsci13040410.
- 10 Marta Montenegro-Rueda, José Fernández-Cerero, José María Fernández-Batanero, and Eloy López-Meneses. Impact of the implementation of ChatGPT in education: A systematic review. *Computers*, 12(8):153, 2023. doi:10.3390/computers12080153.
- 11 Eng Lieh Ouh, Benjamin Kok Siew Gan, Kyong Jin Shim, and Swavek Wlodkowski. ChatGPT, can you generate solutions for my coding exercises? an evaluation on its effectiveness in an undergraduate java programming course. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, pages 54–60, 2023. doi:10.1145/3587102.3588794.
- 12 Emilia Pietrikova and Sergej Chodarev. Profile-driven source code exploration. In *IEEE Federated Conference on Computer Science and Information Systems*, pages 929–934, 2015. doi:10.15439/2015F238.
- 13 Jaromir Savelka, Arav Agarwal, Christopher Bogart, Yifan Song, and Majd Sakr. Can generative pre-trained transformers (GPT) pass assessments in higher education programming courses? In *Conference on Innovation and Technology in Computer Science Education V. 1*, pages 117–123. ACM, 2023. doi:10.1145/3587102.3588792.
- 14 Jody Serrano. New york city schools lift ban on ChatGPT, say initial fear “overlooked the potential” of AI. URL: <https://gizmodo.com/new-york-city-public-schools-lift-ban-chatgpt-ai-1850453424>.
- 15 Matus Sulir, Michaela Bacikova, Matej Madeja, Sergej Chodarev, and Jan Juhar. Large-scale dataset of local java software build results. *Data*, 5(3):86:1–86:11, 2020. doi:10.3390/data5030086.
- 16 Ramazan Yilmaz and Fatma Gizem Karaoglan Yilmaz. Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning. *Computers in Human Behavior: Artificial Humans*, 1(2):100005, 2023. doi:10.1016/j.chbah.2023.100005.

Exercisify: An AI-Powered Statement Evaluator

Ricardo Queirós   

School of Media Arts and Design & CRACS – INESC TEC,
Polytechnic University of Porto, Portugal

Abstract

A growing concern with current teaching approaches underscores the need for innovative paradigms and tools in computer programming education, aiming to address disparate user profiles, enhance engagement, and cultivate deeper understanding among learners. This article proposes an innovative approach to teaching programming, where students are challenged to write statements for solutions automatically generated. With this approach, rather than simply solving exercises, students are encouraged to develop code analysis and problem formulation skills. For this purpose, a Web application was developed to materialize these ideas, using the OpenAI API to generate exercises and evaluate statements written by the students. The transformation of this application in H5P and its integration in a LMS gamified workflow is explored for wider and more effective adoption.

2012 ACM Subject Classification Social and professional topics → Computer science education

Keywords and phrases Code generation, Computer Programming, Gamification

Digital Object Identifier 10.4230/OASICS.ICPEC.2024.19

Funding This work is co-funded by the Erasmus+ Programme of the European Union within the project FGPEPlusPlus, with Agreement Number 2023-1-PL01-KA220-HED-000164696.

1 Introduction

Learning to code can be tough. Understanding programming languages, algorithms, and problem-solving can be overwhelming, specially for novice students [3].

This work explores a different way of teaching programming. Instead of students solving problems, they're given a completed JavaScript solution and asked to write a clear problem statement based on it. This way, rather than focusing solely on implementing solutions, students are challenged to think at a higher level by articulating problem statements. This requires them to analyze, synthesize, and communicate complex ideas, fostering critical thinking and problem-solving skills which are crucial in programming.

By working backward from a provided solution to create a problem statement, students gain a deeper understanding of programming concepts and logic embedded within the solution code. Students are challenged to think critically and analytically as they articulate the problem requirements based on the provided solution, fostering problem-solving skills essential in programming.

For this purpose, Exercisify was created. The workflow is straightforward: 1) a complete source code of a programming exercise is generated and presented to the student; 2) students write a clear problem statement and test case based on the solution given and 3) the problem statement and test cases are automatically evaluated and a score is delivered. Both the exercise generation and the statement evaluation is supported by the OpenAI API.

The evaluation includes specific criteria to assess various aspects from relatedness of the statement to the solution code, clarity and coherence of the statement, explanation of input parameters return values, test cases correctness and English writing.

The rest of the article is structured in three sections: the second section presents related work on computer learning teaching environments. The following section presents Exercisify and all its components. Finally, the contributions of this article to the scientific community are presented as well as the future work.



© Ricardo Queirós;

licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 19; pp. 19:1–19:6

OpenAccess Series in Informatics



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

2 Literature review

Currently, AI tools play a crucial role in improving the teaching and learning process of computer programming by offering personalized and adaptive learning experiences. In programming education, AI assists educators and learners in various aspects, including providing automated feedback and code review, offering adaptive learning paths, facilitating programming tutoring systems, automating assessment and grading tasks, and aiding in code summarization and documentation.

Another way AI can be used is by generating programming exercises. There is a notable interest in such generators, as creating diversified and challenging programming exercises can be time-consuming for educators. These generated exercises can be tailored to cater to different skill levels, ensuring a dynamic learning environment that adapts to individual learner needs. Several tools have been developed to address the challenges of programming exercise generation.

Kurdi [2] conducted a systematic review of automatic exercise generation in various domains, highlighting the need for tools that offer exercises of controlled difficulty and provide features like enriching question forms and structures, automating template construction, improving presentation, and generating feedback.

Zavala [6] presented a tool that uses Automatic Item Generation (AIG) to create consistent programming exercises using pre-defined templates, ensuring uniformity in testing.

ExGen [5] generates ready-to-use exercises tailored to specific difficulty levels and concepts, leveraging advances in large language models (LLMs) to autogenerate novel exercises and filter them to suit students.

Agni [1] is a code playground tailored for learning JavaScript which includes a back end with an exercise generation component powered by the ChatGPT API. This integration automates the exercise creation process by generating statements, solution code, and test cases. The tool supports the IMS LTI specification, allowing seamless integration with Learning Management Systems (LMS) such as Moodle, Blackboard, or Canvas.

TESTed [4] is an educational testing framework that supports the creation of programming exercises with automated assessment capabilities in a programming-language-independent manner.

Recent work has leveraged pre-trained LLMs for automatic exercise generation using novel AI technologies. However, challenges such as model biases and system brittleness need to be addressed when applying LLMs to education.

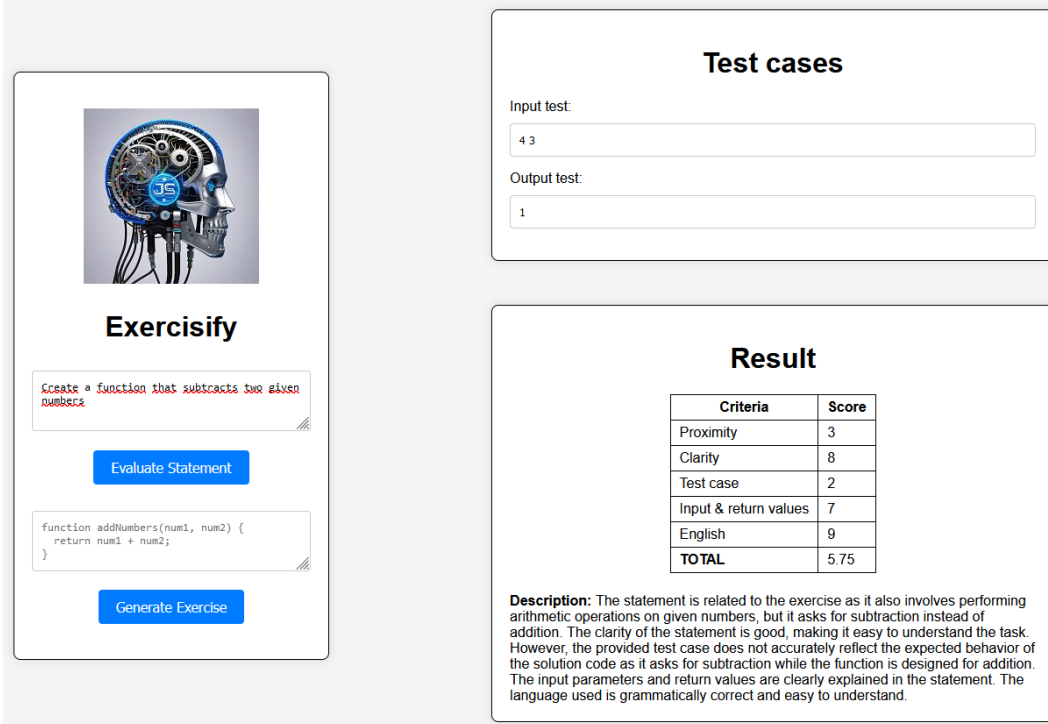
While several tools exist for programming exercise generation, there is a gap in tools capable of offering different approaches as the one presented in this article.

3 Implementation

This work introduces a Web application called Exercisify, designed to enhance programming knowledge through a unique approach. In Exercisify, students are tasked with creating exercise statements based on generated solutions. This approach offers a novel methodology that not only challenges their understanding of programming concepts but also fosters critical thinking, problem-solving, and communication skills crucial for real-world software development.

The Exercisify graphical user interface (GUI) is structured into three main areas: the statement and generated solution display on the left, the test case creation area at the top right, and the result display at the bottom right. The left area showcases the generated solution and allows students to compose the exercise statement and initiate evaluation. The

top right section enables students to define test cases comprising input and output data. Finally, the bottom right area presents the evaluation results after the evaluation button is pressed.



The screenshot displays the Exercisify GUI interface. On the left, a box titled "Exercisify" contains a prompt: "Create a function that subtracts two given numbers". Below the prompt is a code editor with the following JavaScript code:

```
function addNumbers(num1, num2) {
  return num1 + num2;
}
```

Two buttons are visible: "Evaluate Statement" and "Generate Exercise". On the right, a "Test cases" section shows input and output fields. The input field contains "4 3" and the output field contains "1". Below this is a "Result" section featuring a table with evaluation criteria and scores:

Criteria	Score
Proximity	3
Clarity	8
Test case	2
Input & return values	7
English	9
TOTAL	5.75

Below the table, a description states: "Description: The statement is related to the exercise as it also involves performing arithmetic operations on given numbers, but it asks for subtraction instead of addition. The clarity of the statement is good, making it easy to understand the task. However, the provided test case does not accurately reflect the expected behavior of the solution code as it asks for subtraction while the function is designed for addition. The input parameters and return values are clearly explained in the statement. The language used is grammatically correct and easy to understand."

■ **Figure 1** Exercisify GUI.

The Exercisify architecture comprises two primary components: the Exercise Generator and the Statement Evaluator.

3.1 Exercise Generator

The Exercise Generator component is responsible for generating programming solutions using the OpenAI API. The code below starts by defining an asynchronous JavaScript function which is responsible for generating exercise statements using the ChatGPT API from OpenAI. The prompt variable contains the instruction provided to the ChatGPT model. In this case, it instructs the model to generate a very simple function in JavaScript. The apiKey variable holds the authentication key required to access the OpenAI API. It's essential for authenticating requests to the API. The apiUrl variable specifies the endpoint of the OpenAI API that will be used to send requests for text generation. The requestData object contains the data to be sent in the request body to the API. It includes the model to use for text generation (gpt-3.5-turbo), the user's prompt message, and parameters like max_tokens (maximum number of words to generate), temperature (controls randomness), and stop (criteria to stop text generation). Then the fetch function is used to send a POST request to the specified API endpoint with the request headers, such as content type and authorization. The request body contains the requestData object, which is converted to JSON format using JSON.stringify().

19:4 Exercisify: An AI-Powered Statement Evaluator

■ **Listing 1** Programming solution generation.

```
async function generateExercise() {
  const prompt = "Generate a JavaScript function...";
  const apiKey = "API-KEY";
  const apiUrl = "https://api.openai.com/v1/chat/completions";

  // Data to be sent in the request body
  const requestData = {
    model: "gpt-3.5-turbo",
    messages: [{"role": "user", "content": prompt}],
    max_tokens: 100,
    temperature: 0.7,
    stop: ["\\n"],
  };

  try {
    const response = await fetch(apiUrl, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        "Authorization": `Bearer ${apiKey}`
      },
      body: JSON.stringify(requestData)
    });
    ...
  } catch (error) {
    ...
  }
}
```

Once the solution is generated, it is presented to the student, who is tasked with creating a statement for the exercise based on the solution provided. This manual process encourages students to think critically about the problem and articulate their understanding in writing.

3.2 Statement Evaluator

The Statement Evaluator component assesses the quality and relevance of the exercise statement created by the student in relation to the provided solution. It leverages the ChatGPT API from OpenAI once again to evaluate the relatedness of the statement and the solution based on specific criteria. The key elements of the Statement Evaluator include

1. **Evaluation Prompt Creation:** The web app generates a prompt for the ChatGPT API, incorporating both the exercise statement created by the student and the provided solution. This prompt guides the API in evaluating the alignment of the statement with the solution based on predefined criteria.
2. **ChatGPT API Integration:** The Statement Evaluator interacts with the ChatGPT API to assess the exercise statement. The API processes the prompt and offers feedback on the statement's coherence with the solution.

The evaluation criteria used in the Statement Evaluator component is depicted in the following table.

The first criterion prioritizes the alignment between the exercise statement and the provided solution code, ensuring it effectively describes the programming task, algorithms used (if any), and guides students towards the intended solution.

Criteria	Description	Weight
Proximity to solution	How closely does the exercise statement align with the provided solution code?	35%
Clarity of Explanation	How clear and understandable are the instructions provided in the exercise statement?	25%
Test Case Correctness	How accurately do the provided test cases reflect the expected behavior of the solution code?	25%
Input and return values	How effectively does the exercise statement clarify input parameters and expected return values?	10%
English Proficiency	How grammatically correct and fluent is the English language used in the exercise statement?	5%

The clarity of explanation criterion underscores the importance of clear communication in the exercise statement, ensuring students can readily comprehend the task and its requirements.

Including test cases is crucial for validating the correctness of the solution code. By assigning significant weight to this criterion, the evaluation process ensures that the provided test cases accurately reflect the expected behavior of the solution.

Additionally, clarifying input parameters and expected return values helps students grasp the function's purpose and behavior. While not as heavily weighted as other criteria, this aspect still contributes to the overall effectiveness of the exercise statement.

Finally, ensuring grammatical correctness and fluency in the exercise statement is vital for clear communication. While English proficiency is essential, it's appropriately given a lower weight compared to other criteria, as long as the statement remains understandable.

3.3 H5P transformation

Integrating Exercisify into an LMS workflow can significantly enhance the teaching and learning experience in programming education. Two common methods for seamless integration are through Learning Tools Interoperability (LTI) and H5P (HTML5 Package).

Learning Tools Interoperability (LTI) is a standard protocol that allows learning systems, such as LMS platforms, to integrate with external tools and content.

H5P (HTML5 Package) is a content creation tool that allows educators to develop and share interactive content across several platforms.

LTI integration facilitates standardized access within LMS platforms, enabling single sign-on, centralized management, and data exchange for features like grade synchronization. H5P integration, on the other hand, offers customization flexibility, content embedding, reusability, and support for interactive elements. The choice depends on factors such as desired integration level, instructional goals, and LMS platform capabilities.

By integrating Exercisify with the LMS, student performance scores from exercise evaluations can be automatically transported to the LMS gradebook. This eliminates the need for manual score entry by instructors, saving time and ensuring accuracy in grading. At the same time, students receive real-time feedback on their exercise submissions, including scores and performance metrics. This immediate feedback loop promotes continuous learning and allows students to track their progress throughout the course.

One of the benefits of the LMS integration is the capability of including Exercisify in the LMS gamification workflow. Here are some examples:

- **Unlocking Content:** Gamification elements such as unlocking content based on performance scores can be implemented within the LMS. Students can progress through course materials and access additional resources or levels as they achieve certain performance milestones in Exercisify exercises.

- Earning Badges and Achievements: Students can earn badges or achievements within the LMS for achieving specific scores or completing exercises in Exercisify. These gamified incentives incentivize engagement, encourage mastery of programming concepts, and add an element of fun to the learning process.
- Competition and Leaderboards: the LMS can enable the creation of leaderboards based on Exercisify performance scores. Students can compete with peers, track their rankings, and strive to improve their standings, fostering a sense of healthy competition.

4 Conclusion and Future Work

In conclusion, the development of the Exercisify web app and the integration of AI-based exercise generation and statement evaluation can empower students to actively engage in learning, fostering critical thinking and problem-solving skills.

In regard of future directions, the most obvious is to validate the Exercisify tool with real users. This validation process could involve conducting user testing sessions with students and educators to gather feedback on usability, effectiveness, and overall user experience. Additionally, collecting data on user performance and learning outcomes through the tool can provide insights into its impact on programming education.

Additional future directions:

- Enhanced AI Integration: continuously update and refine the AI models used for exercise generation and evaluation to improve accuracy.
- Expanded Exercise Types: diversify the range of supported exercise types to cover several programming languages and problem-solving scenarios.
- H5P support: finalize the transformation of the Web app to the H5P package.

References

- 1 Yannik Bauer, José Paulo Leal, and Ricardo Queirós. Can a Content Management System Provide a Good User Experience to Teachers? In *4th International Computer Programming Education Conference (ICPEC 2023)*, volume 112 of *Open Access Series in Informatics (OASISs)*, pages 4:1–4:8, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- 2 Ghader Kurdi, Jared Leo, Bijan Parsia, Uli Sattler, and Salam Al-Emari. A systematic review of automatic question generation for educational purposes. *International Journal of Artificial Intelligence in Education*, 30, 2019.
- 3 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, Jakub Swacha, and Filip Miernik. Managing gamified programming courses with the FGPE platform. *Information*, 13(2):45, 2022.
- 4 Niko Strijbol, Charlotte Van Petegem, Rien Maertens, Boris Sels, Christophe Scholliers, Peter Dawyndt, and Bart Mesuere. Tested – an educational testing framework with language-agnostic test suites for programming exercises. *SoftwareX*, 22:101404, 2023.
- 5 Nguyen Binh Duong Ta, Hua Gia Phuc Nguyen, and Gottipati Swapna. Exgen: Ready-to-use exercise generation in introductory programming courses. In *Proceedings of the 31st International Conference on Computers in Education Conference*, pages 1–10, Matsue, Shimane, Japan, december 4-8 2023.
- 6 Laura Zavala and Benito Mendoza. On the use of semantic-based aig to automatically generate programming exercises. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education, SIGCSE '18*, pages 14–19, New York, NY, USA, 2018. Association for Computing Machinery.

Use of Programming Aids in Undergraduate Courses

Ana Rita Peixoto ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

André Glória ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), Instituto de Telecomunicações, Portugal

José Luís Silva ✉ 

ITI/LARSyS, Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Maria Pinto-Albuquerque ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Tomás Brandão ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Luís Nunes ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Abstract

The use of external tips and applications to help with programming assignments, by novice programmers, is a double-edged sword, it can help by showing examples of problem-solving strategies, but it can also prevent learning because recognizing a good solution is not the same skill as creating one. A study was conducted during the 2nd semester of 23/24 in the course of Object Oriented Programming to help understand the impact of the programming aids in learning. The main questions that drove this study were: Which type(s) of assistance do students use when learning to program? When / where do they use it? Does it affect grades? Results, even though with a relatively small sample, seem to indicate that students who used aids have a perception of improved learning when using advice from *Colleagues*, *Copilot*-style tools, and *Large Language Models*. Results of correlating average grades with the usage of tools suggest that experience in using these tools is key for its successful use, but, contrary to students' perceptions, learning gains are marginal in the end result.

2012 ACM Subject Classification Social and professional topics → Computing education

Keywords and phrases Teaching Programming, Programming aids

Digital Object Identifier 10.4230/OASICS.ICPEEC.2024.20

Funding This work was partially supported by Fundação para a Ciência e a Tecnologia, I.P. (FCT) ISTAR Projects: UIDB/04466/2020 and UIDP/04466/2020.

1 Introduction

The possibility of using programming aids in undergraduate courses has increased gradually, but steadily, throughout the last decades, and recently was broadened with the introduction of Large Language Models (LLM). From the onset of Integrated Development Environments (IDEs), one of the available tools was the introduction of standard snippets with a hotkey, the immediate syntax highlighting of errors, or the code advisors that tried to guess the following tokens necessary to end a construct - some more successfully than others.

The use of these tools raises the question of whether students should be evaluated in their programming skills with or without these aids and also if their evaluation should include assessing their proficiency in using the available aids. On the one side, it is important to teach students to create their own solutions, on the other hand, in their future jobs, they will most certainly be able to use these and other tools to help in programming, and so it seems logical that they gather experience in using them.



© Ana Rita Peixoto, André Glória, José Luís Silva, Maria Pinto-Albuquerque, Tomás Brandão, and Luís Nunes;

licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 20; pp. 20:1–20:9

OpenAccess Series in Informatics



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

20:2 Use of Programming Aids in Undergraduate Courses

The consequences reported seem to be: the change in the skills necessary to train a good programmer and the increased productivity provided by these new tools [4]. It is no longer necessary to be as meticulous and focused as before and to memorize syntax in such detail since the development environment helps with that. Also, most current languages come with extensive libraries that make algorithmic development a skill that current programmers use less frequently than a few years ago.

Code completion tools have grown more and more accurate and are evolving into code generators (as some “copilots” that have both functions integrated) [1], and Large Language Models, have contributed to increasing the sense that most programming tasks (indeed, most project development tasks) can be automated [9].

The main question, to which we will try to contribute, is whether these tools are beneficial for students learning to program or if they indeed prevent learning.

The research questions are:

- Q1: Which type(s) of assistance do students use when learning to program?
- Q2: When / where do they use it?
- Q3: Does it affect grades?

The paper contains a summarized Literature Review, a description of the experiments’ Methodology and context, followed by a Results section and the corresponding Discussion, and ends with Conclusions and Future Work.

2 Literature Review

Studies regarding the use of programming aids in learning programming have been frequent in the last few years and mostly tend to conclude that the impact of code generation tools on novice programmers is significant [6]. Given the length of this work we were forced to select a few of the most obviously related.

Burak et al. [11] assessed the code generation capabilities of several code-generating tools using the benchmark HumanEval Dataset and evaluated the proposed code quality metrics. This study reveals that current tools have very different capabilities, advocating an advantage for Github Copilot.

Code completion tools assist programmers by suggesting completions for partially typed code snippets. These tools can significantly improve programming efficiency and reduce syntax errors for novice programmers. Studies such as [10] found that code completion can enhance productivity and code quality by reducing the cognitive load associated with remembering syntax and identifiers.

Kazemitabaar et al. [3] and Prather et al. [8] defend that code generators like OpenAI Codex and Github Copilot can enhance code-authoring performance and completion rates. Additionally, automated programming hints, such as next-step code hints with textual explanations, have been found to improve immediate programming performance and learning [7].

Other authors advocate that the use of code completion and code generation tools on novice programmers has significant pedagogical implications. Research by Lister et al. [5] emphasizes the importance of incorporating these tools into programming education to provide support for novices and enhance their learning experience. [2] analyses the impact of LLM generated “worked examples,” concluding that students find them useful for their learning.

Automated programming hints, particularly with textual explanations and self-explanation prompts, significantly improve novice programmers’ immediate performance and learning outcomes in similar subsequent tasks, according to [7]. This work reports that code hints with

textual explanations significantly improved immediate programming performance. However, these hints only improved performance in a subsequent post-test task with similar objectives when they were combined with self-explanation prompts.

Code generation and completion tools automate the process of writing code. These tools can help novice programmers as they provide scaffolding and reduce the complexity of programming tasks. But, do code generation and completion tools (programming aids) enhance novice programmers' understanding of code structure and promote learning by allowing them to focus on problem-solving rather than syntax details? How do different advising strategies compare? Those are the questions we will be trying to shed some light on in the following sections.

3 Methodology

To contribute to answering the questions posed in the previous sections, we have prepared an experiment during the Object Oriented Programming (OOP) course in the first semester of 23/24 (sep-23 to dec-23). In this course, students are evaluated in class by the exercises they complete once a week in 8 of the 12 laboratories (30%), by a mid-term test (20%, a grade of less than 7.5/20 would result in immediate *Fail* in the course), and a final project with presentation and discussion (50%). Exercises and the project can be done individually or in pairs. At the beginning of the course, students were told that they could use whatever tools available to do the exercises as long as they could explain every single line of the code they delivered for evaluation, except in the mid-term test, an individual test, where no aids were allowed. During exercise evaluation and project discussion, students were frequently asked to explain and change the solutions presented so that their knowledge of what they were delivering was tested. Situations, where there were doubts concerning the students' understanding of the solution they presented, were extremely rare (2 detected cases in approximately 8 exercises \times 278 students).

The students selected for this study are from three different graduation programs: Computer Science and Engineering (LEI), Computer Science and Management (LIGE), and Computer Science and Telecommunications (LETI). The first two have day and night-shifts. Night-shifts are termed “-PL” (*pós-laboral*): LEI-PL and LIGE-PL and are usually frequented by working-students. All students have a previous similar background in terms of the programming courses offered in the three programs. All programs have Introduction to Programming (one semester, 6 ECTS, mandatory course, approval is required to enroll for OOP) and an Algorithms and Data Structures course (one semester, 6 ECTS). Students enrolled in LEI have a significantly higher entrance grade.

An inquiry on the aids used was done after the grades were published to ensure that fear of influencing the grade would not be a factor. Still, the response was lower than expected (92/298 students responded, roughly 1/3 of the students enrolled in OOP).

The inquiry (originally in Portuguese, the native language of nearly all of these students) was composed of an introduction and 9 questions (the text in English of the introduction of the inquiry is in annex). The questions were the following:

1. What is your student number (optional)?
2. What is your age group? 18-23, 24-35, >35
3. Have you ever used (before OOP) tools / strategies to support learning programming (see examples in the next question)? Yes/No
4. If you answered “Yes” to the previous question, which ones? (multiple answer possible)
 - Direct advice from *colleagues*, family or friends
 - Paid *tutor* or mentoring

20:4 Use of Programming Aids in Undergraduate Courses

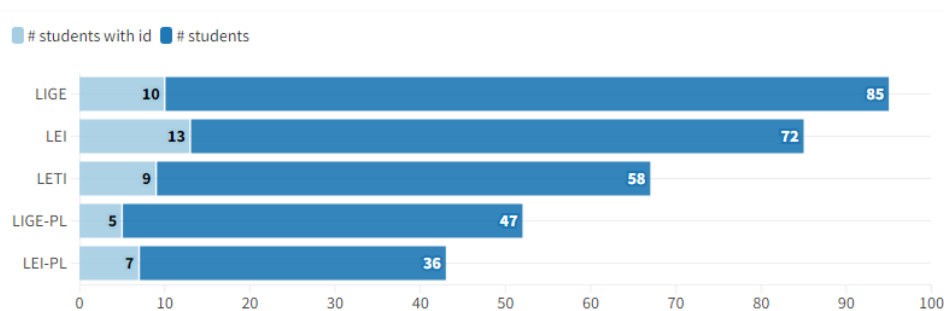
- Developer *communities* and online forums (StackOverflow, W3Schools...)
 - Eclipse *auto-complete* (or other development environment integrated tools - IDE)
 - External applications or plugins based on *Copilot* (on GitHub, IntelliJ, PyCharm, ...)
 - *Large Language Models*, i.e. artificial intelligence technologies for conversation (ChatGPT, Bard, ...)
 - *Other*
5. Have you used any programming support tools / strategies this year in the OOP course?
 - No
 - Only in exercises
 - Only in project
 - In project and exercises
 6. If you answered yes to the previous question, which ones? (options equal to question 4, multiple answer possible)
 7. Which ones did you find most effective in helping to find appropriate / correct solutions? from 1 (least effective) to 5 (most effective). (options equal to question 4, but reply in [1..5] for each)
 8. In your opinion, has the use of these tools improved your learning of programming? Yes/No/Don't know/Didn't use
 9. If you used them, which ones did you find most useful in helping you to learn OOP? from 1 (least useful) to 5 (most useful). (options equal to question 7)

Of the 92 responses to the inquiry (out of a universe of 298 students), 44 of those have valid student identifications (question #1, inserted voluntarily), and are relatable to the students' courses and grades.

The part of the students that inserted an *id* is (naturally) biased towards more successful students, the course had a 81% success-rate but only one of the 44 responders that inserted an *id* did not pass the course (2% of the sample).

The age group of responders (question #2) was 90.2% 18-25, and 91% (84/92) claimed to have used previously some of the learning aids mentioned in the inquiry (question #3), the same number, 91%, are convinced that using these tools improved their learning ability (question #8). Only 2% (2) are convinced of the contrary.

Concerning the questions of having used aids in OOP and for what purpose (question #4), only a minority claims not to have used (14/92) (table 3).

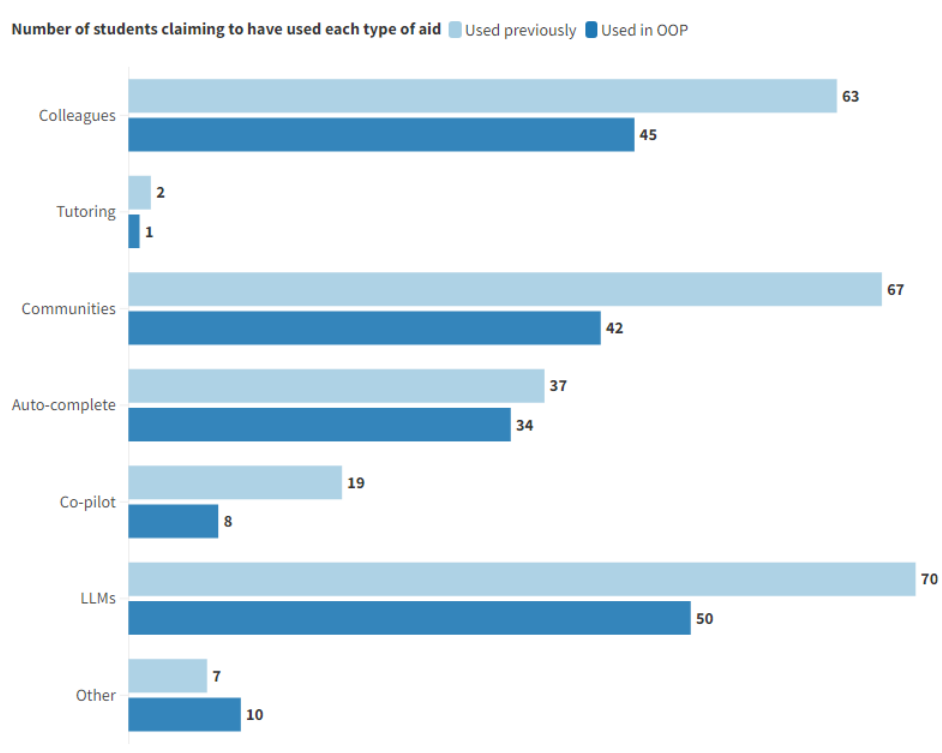


■ **Figure 1** Number of students that responded with id and the number of students enrolled from each program.

The courses of the 44 identifiable students are: LIGE – 10/85, LETI – 9/58, LEI – 13/72, LEI-PL – 7/36, LIGE-PL – 5/47. The denominator represents the total number of students in each program enrolled in OOP. Between 10% and 20% of the population for each course responded and with a valid an *id* (figure 1).

4 Results

All types of aid seem to have been used less during OOP than previously (questions #4 and #6), according to the responses of the 92 students (figure 2). *Large Language Models (LLMs)*, *Colleagues* and *Communities* are the most admittedly used, in that order, followed by *Auto-complete*. Use of *Copilot* is low and *Tutoring* is residual (only one student in this sample admitted to have paid tutoring for OOP). Also, a low usage of *Other* (non-specified) resources. *Tutoring* and *Other* will not be considered in most of the remaining analysis due to lack of support.



■ **Figure 2** Histogram of declarations of the types of aid used before and during OOP.

The perceived utilities to help find solutions and learning (tables 1 and 2, questions #7 and #9, respectively) tend to be better for those that used each tool than for those that did not (with the exception of the utility of using *Copilot* to solve problems).

Favoured tools (both to find good solutions – Table 1 – and to help in the learning process – Table 2) – are unclear, although *Colleagues* have a slight advantage for solving problems, in the groups of students that actually use these tools. Also, in the group of users of the tools, *Copilot*, *Colleagues*, and *LLMs* seem to be perceived as more helpful for learning.

Students did not seem to make a difference between finding the solutions and learning with the most favoured tools. Correlations (Pearson) between votes in these categories in questions #7 and #9 are: 0.87 for *Colleagues*, 0.82 for *Copilot*, 0.73 for *Communities* and 0.73 for *LLMs*. The use of *Auto-complete* has the lowest correlation (0.59) between the perceived value for learning and for finding solutions. Other correlations (between 33 numerical variables) have no highlights apart from the obvious correlations between previous use and use in OOP of the same tools and grades of different evaluations. The variables

20:6 Use of Programming Aids in Undergraduate Courses

■ **Table 1** Perceived utility for solving problems from those that used each tool vs those that didn't.

Type of aid	Used	Didn't use
Colleagues	3.98 +/- 0.84	3.14 +/- 1.09
Communities	3.66 +/- 0.76	3.57 +/- 1.03
Auto-complete	3.66 +/- 0.93	3.20 +/- 1.21
Copilot	3.29 +/- 1.37	4.00 +/- 1.41
LLMs	3.59 +/- 1.02	3.60 +/- 0.89

■ **Table 2** Perceived utility for learning OOP from those that used each tool vs those that didn't.

Type of aid	Used	Didn't use
Colleagues	4.05 +/- 0.82	2.82 +/- 1.22
Communities	3.54 +/- 0.92	3.47 +/- 0.76
Auto-complete	3.47 +/- 1.07	2.72 +/- 1.19
Copilot	4.13 +/- 1.13	2.29 +/- 1.20
LLMs	3.94 +/- 0.79	2.91 +/- 1.45

correlated were the previous use of aids, the use in OOP, the types of tools used in each case, the score for effectiveness in helping to solve problems, as well as learning effectiveness and grades for each assessment.

■ **Table 3** Counts per type of usage.

When aid was used	#count	#count students with id
No	14	6
Only in exercises	9	5
Only in final project	26	15
Exercises and final project	43	18

As for the relation between the type and frequency of aid usage with grades (Tables 3 and 4), results are unclear. The best average grade in *Exercises*, but also in the *Test* (recall, without aids) is from students that have admittedly used aids, but *only during exercises* (even though this is a relatively small sample, 5 students). What stands out in these figures is that students who admit to having used aids *only in the final project* have lower average grades in all evaluations. This is an observation with reasonable support: 15/44. Average grades of students that allegedly used *No* aids are the best in *Project* and second-best in other evaluations.

The relationship between the type of aid used and the grades (Table 5, focusing on the classes that have a reasonable number of samples, >10) appears to indicate that the type of aid used is not relevant showing only a slight decay in the average *Test* grades (the only evaluation where aids are not allowed) of students that use advice from *Colleagues* even though they have good grades in exercises.

If the *Test* results measure the true value for learning of the different types of aid, then *Communities* and *LLMs* seem to hold some learning value even though standard deviations are relatively high.

■ **Table 4** Average grades per type of usage. Grades in [0..20].

Type of aid	Exercises	Test	Project
No	19.27 +/- 1.15	15.11 +/- 3.40	16.97 +/- 2.08
Only in exercises	19.75 +/- 0.56	16.34 +/- 3.42	16.80 +/- 2.39
Only in final project	17.16 +/- 4.45	14.45 +/- 3.55	15.07 +/- 3.90
Exercises and final project	19.03 +/- 1.59	15.09 +/- 4.92	16.47 +/- 2.61

■ **Table 5** Average grades per type of aid used and Standard Deviations.

Type of aid	Exercises	Test	Project
Colleagues	18.79 +/- 1.57	14.42 +/- 4.68	16.09 +/- 2.56
Communities	18.35 +/- 3.10	15.86 +/- 4.43	16.52 +/- 2.69
Auto-complete	17.83 +/- 3.62	15.09 +/- 5.06	16.26 +/- 2.57
LLMs	18.02 +/- 3.52	15.77 +/- 3.66	16.03 +/- 3.45

5 Discussion

The experience of the teaching staff during this year was particularly rewarding given the reduction of retained students to < 20%. The new evaluation method (based on exercises, evaluated nearly every week) is likely to have contributed to this success.

Even though we explicitly mentioned the subject at the beginning of the semester, the use of tools external to the IDE (Communities, Copilot, or ChatGPT) was seldom seen in class, where most students still seem to prefer asking colleagues or the teacher. Likely most of the aids were used during class preparation or individual work.

Apart from very few exceptions, students seemed to be quite familiar with the produced code, both in the exercises and in the final project.

The majority of the results point to a marginal difference between using aids regularly (in exercises or always) and not using any, with a slight advantage for those who used aids (the majority) even in evaluations where aids are not allowed.

Students, that used aids only in the final project, had slightly lower scores in all evaluations. The hypothesis, based on this, but that we cannot confirm with the current experiment, is that experience plays a key role in the efficient use of these tools, as in many other technological tools.

Analyzing the differences between the votes for “tools to find good solutions” and “tools that help learning,” the differences, although small, show a tendency to consider *Communities* and *Auto-complete* as tools that are more useful to “find good solutions” than to “help learning,” and *Large Language Models* the reverse.

Still, the only clue to events, that seem actually to affect grades negatively, is the use of advice from *Colleagues* that seems to have a positive effect during exercises but a negative effect during the *Test*. This may be related to the fact that being able to explain a solution is a different skill from actually being able to produce that solution.

Nevertheless, some of the automatic aids seem to affect the ability to generate solutions less than advice from *Colleagues* even though the latter should often imply the need to code the solution themselves.

6 Conclusions and Future Work

Data limitations beyond those that were expected (low participation of students) limit our conclusions, still, we believe this account presents a valid contribution to add to many others.

In this paper, we describe an experiment that took place in the 23/24 edition of the Object Oriented Programming course. In this experiment, we have explicitly liberated the use of any programming aids with the sole demand that students should (at all times) be able to explain what they were doing.

This experiment aimed at understanding which are the most usual tools used as programming aids by novice students (Q1), when / where students apply them (Q2), and how this affects their grades (Q3). The main conclusions based on the analysis of this data are the following:

- Most students use some form of programming aids (74/92, 84.8%)
- The most common types of aids used are: *Large Language Models* (50-70/92), *Colleagues* (45-63/92); *Communities* (42-67/92) and *Auto-complete* (34-37/92), in this order of preference (answering Q1). The two numbers correspond to previous use and use in OOP;
- 47% (43/92) of the students use aids in exercises and on the project, 10% (9/92) only in exercises, 28% (26/92) only use aids in the final project and 15% (14/92) claim to have used no aids (answering Q2);
- The usage of these tools does not seem to have a significant impact on the grades, neither in situations where aids are allowed nor in those where they are not (addressing Q3). However, the sample of students who claim not to have used any aids is small (n=14, 15%), and only 6 of these inserted an id, so the results lack support in this respect;
- The use of these tools for the last part of the evaluation (project) only seems to be related to students with lower grades;
- Using advice from *Colleagues* is the only event relatable to a drop of *Test* grades (the only evaluation where no aids were allowed);
- The students who reported not using any aids (although in small number) have average grades similar to those who reported using aids.

References

- 1 Zhamri Che Ani, Zauridah Abdul Hamid, and Nur Nazifa Zhamri. *The Recent Trends of Research on GitHub Copilot: A Systematic Review*, pages 355–366. Springer, 2024. doi:10.1007/978-981-99-9589-9_27.
- 2 Breanna Jury, Angela Lorusso, Juho Leinonen, Paul Denny, and Andrew Luxton-Reilly. Evaluating llm-generated worked examples in an introductory programming course. In Nicole Herbert and Carolyn Seton, editors, *Proceedings of the 26th Australasian Computing Education Conference, ACE 2024, Sydney, NSW, Australia, 29 January 2024- 2 February 2024*, pages 77–86. ACM, January 2024. doi:10.1145/3636243.3636252.
- 3 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 Albrecht Schmidt, Kaisa Väänänen, Tesh Goyal, Per Ola Kristensson, Anicia Peters, Stefanie Mueller, Julie R. Williamson, and Max L. Wilson, editors, *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, CHI 2023, Hamburg, Germany, April 23-28, 2023*, CHI '23, pages 455:1–455:23, New York, NY, USA, 2023. ACM. doi:10.1145/3544548.3580919.
- 4 Amy J. Ko. More than calculators: Why large language models threaten learning, teaching, and education, December 2023. URL: <https://tinyurl.com/yck47y5s>.

- 5 Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L. Whalley, and Christine Prasad. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In Renzo Davoli, Michael Goldweber, and Paola Salomoni, editors, *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2006, Bologna, Italy, June 26-28, 2006*, volume 38, pages 118–122, New York, NY, USA, June 2006. ACM. doi:10.1145/1140124.1140157.
- 6 Wenhan Lyu, Yimeng Wang, Tingting Chung, Yifan Sun, and Yixuan Zhang. Evaluating the effectiveness of llms in introductory computer science education: A semester-long field study. *CoRR*, abs/2404.13414, April 2024. doi:10.48550/arXiv.2404.13414.
- 7 Samiha Marwan, Joseph Jay Williams, and Thomas W. Price. An evaluation of the impact of automated programming hints on performance and learning. In Robert McCartney, Andrew Petersen, Anthony V. Robins, and Adon Moskal, editors, *Proceedings of the 2019 ACM Conference on International Computing Education Research, ICER 2019, Toronto, ON, Canada, August 12-14, 2019*, ICER '19, pages 61–70, New York, NY, USA, 2019. ACM. doi:10.1145/3291279.3339420.
- 8 James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett B. Powell, James Finnie-Ansley, and Eddie Antonio Santos. "it's weird that it knows what I want": Usability and interactions with copilot for novice programmers. *ACM Trans. Comput. Hum. Interact.*, 31(1):4:1–4:31, November 2024. doi:10.1145/3617367.
- 9 Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *CoRR*, abs/2307.07924, July 2023. doi:10.48550/arXiv.2307.07924.
- 10 Martin P. Robillard, Wesley Coelho, and Gail C. Murphy. How effective developers investigate source code: An exploratory study. *IEEE Trans. Software Eng.*, 30(12):889–903, 2004. doi:10.1109/TSE.2004.101.
- 11 Burak Yetistiren, Isik Özsoy, Miray Ayerdem, and Eray Tüzün. Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. *CoRR*, abs/2304.10778, 2023. doi:10.48550/arXiv.2304.10778.

A Inquiry

With this survey, we want to study the impact of using automatic learning support tools in programming. Since the introduction of tools such as ChatGPT, the use of programming support tools has been discussed, although they have long been used in development environments, they have taken on a different proportion.

It is arguable that their use can benefit or hinder learning. That's why we set out to study their impact. This survey has only been sent out now, after the grades have been published so that there is no doubt about the possibility of this survey influencing the grades. We can also guarantee that nothing will be published that would allow participants to be identified, so I ask everyone to be as honest and thorough as possible in answering all the questions.

This survey should take no more than 5 minutes to complete.

If you have any questions about this survey, please contact: luis.nunes@iscte-iul.pt

– Place for the 9 questions of the Inquiry already presented on pages 3 and 4.

Authoring Programming Exercises for Automated Assessment Assisted by Generative AI

Yannik Bauer   

DCC – FCUP, Porto, Portugal

José Paulo Leal   

CRACS – INESC TEC, Porto, Portugal

DCC – FCUP, Porto, Portugal

Ricardo Queirós   

CRACS – INESC TEC, Porto, Portugal

uniMAD – ESMAD, Polytechnic of Porto, Portugal

Abstract

Generative AI presents both challenges and opportunities for educators. This paper explores its potential for automating the creation of programming exercises designed for automated assessment. Traditionally, creating these exercises is a time-intensive and error-prone task that involves developing exercise statements, solutions, and test cases. This ongoing research analyzes the capabilities of the OpenAI GPT API to automatically create these components. An experiment using the OpenAI GPT API to automatically create 120 programming exercises produced interesting results, such as the difficulties encountered in generating valid JSON formats and creating matching test cases for solution code. Learning from this experiment, an enhanced feature was developed to assist teachers in creating programming exercises and was integrated into Agni, a virtual learning environment (VLE). Despite the challenges in generating entirely correct programming exercises, this approach shows potential for reducing the time required to create exercises, thus significantly aiding teachers. The evaluation of this approach, comparing the efficiency and usefulness of using the OpenAI GPT API or authoring the exercises oneself, is in progress.

2012 ACM Subject Classification Applied computing → Computer-assisted instruction; Computing methodologies → Artificial intelligence; Applied computing → Interactive learning environments

Keywords and phrases ChatGPT, generative AI, programming exercises, automated assessment

Digital Object Identifier 10.4230/OASICS.ICPEEC.2024.21

Funding This work is co-funded by the Erasmus+ Programme of the European Union within the project FGPEPlusPlus, with Agreement Number 2023-1-PL01-KA220-HED-000164696.

1 Introduction

Since its introduction in late November 2022, ChatGPT has produced a range of reactions, from enthusiasm to warnings. In academic and educational contexts, there is legitimate concern about the potential impacts of extensive language models and generative AI. It is important to acknowledge and address these concerns, but it is equally important to recognize the potential of these tools to generate text, code, and data, offering valuable resources and innovative approaches that can positively enhance the educational landscape.

This paper explores the utility of ChatGPT in facilitating the creation of programming exercises, particularly those designed for automated assessment. The process of authoring these exercises is time-consuming and error-prone. It involves generating three distinct components: exercise statements articulated in a natural language, such as English; solutions in a programming language, such as JavaScript; and test data, including input and expected output files. Generative AI is promising for faster and more accurate exercise creation.



© Yannik Bauer, José Paulo Leal, and Ricardo Queirós;
licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 21; pp. 21:1–21:8

OpenAccess Series in Informatics



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

An initial experiment was conducted within the Framework for Gamified Programming Education (FGPE+) project [13] aiming to create 120 programming exercises suited for automated evaluation for AuthorKit, a programming exercise repository [15]. This experiment highlighted ChatGPT's difficulties in generating valid JSON files and creating effective and functional test cases for the proposed code solutions. Learning from this, an enhanced feature was developed for Agni, a virtual learning environment, to assist teachers in creating programming exercises [20]. An ongoing evaluation is comparing the time, quality, and benefits of creating exercises independently versus using ChatGPT's assistance.

2 Related Work

The integration of generative artificial intelligence (AI) in education has gained increasing attention due to its potential to transform teaching and learning practices. Indeed, for teachers, writing good questions/exercises and test cases is a fundamental and time-consuming challenge [11, 24]. New AI tools have already increased productivity in work environments, as measured in a study of issues resolved per hour. They found an increase of 14% on average, including a 34% improvement for novice and low-skilled workers, but with minimal impact on experienced and highly skilled workers [4]. A variety of studies have discussed the possibilities of AI in generating educational content and providing personalized learning experiences, highlighting both the opportunities and challenges of these technologies [3, 8, 2, 1, 14, 18, 23].

A study by Sarsa et al. analyzed the quality of programming exercises generated by OpenAI Codex. It revealed that 84.6% of the exercises generated included a sample solution, with 89.7% of these being executable. Furthermore, 70.8% of these exercises featured tests, although only 30.9% of them passed all tests, achieving a test coverage rate of 98.0% [22]. Similarly, another study showed that 75% of exercises created by generative AI were sensible, 81.8% novel, and between 75.8% to 79.2% aligned with the intended themes and concepts [6]. Another study evaluated that students perceived exercises generated by AI as equal to those created by people. However, the limited variety in AI-generated examples and their close adherence to given prompts raise questions about their adaptability in creating diverse learning resources [5]. These findings suggest that while generative AI can produce good primary educational content, its quality and reliability are still lacking [19].

Other uses, such as automatic feedback, have also shown promise. A web application that leverages GPT-4 to provide feedback on complex exercises demonstrated a high correlation with human feedback and deviated by only 6% from human evaluations [10]. A study on the customization of learning content via generative AI found that AI-generated materials positively impacted lower-performing students without negative effects on accessibility [17]. The findings highlighted the benefit of personalized learning experiences, particularly for students struggling with subjects, by providing materials suited to their specific needs, while more proficient students received more advanced content.

In conclusion, while generative AI holds significant potential for education, it must be integrated carefully, considering its abilities and limitations. Empirical studies and responsible guidelines are essential for harnessing the benefits of AI in educational settings.

3 Early exploration

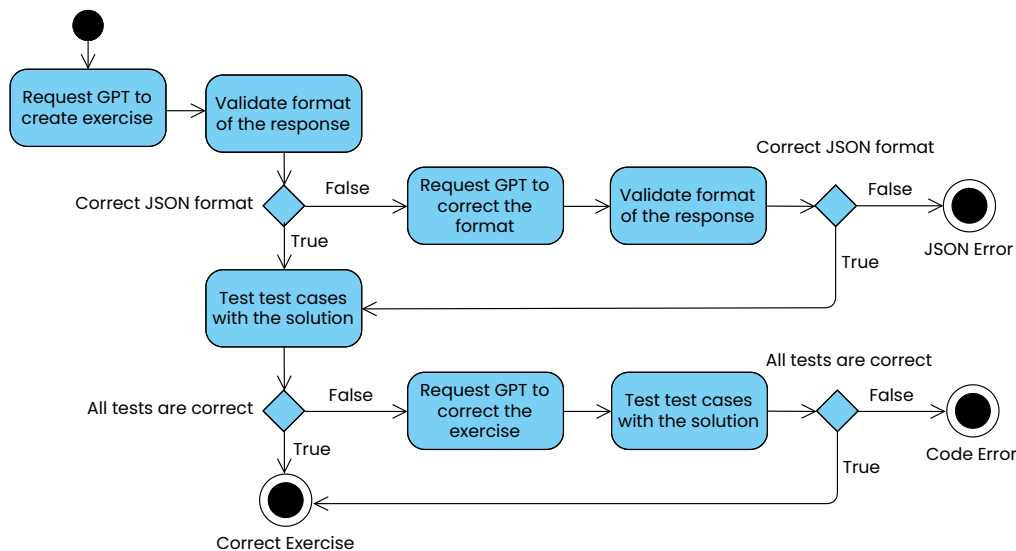
Within the FGPE+ project, an experiment was conducted using OpenAI's GPT-3.5 API to automatically generate 120 programming exercises. These exercises were designed for AuthorKit, a platform for developing programming exercises, that support the definition

■ **Table 1** Results of using GPT to generate exercises.

	Total	% of Total	% of Valid	Resolved	% of Resolved
JSON errors	199	35%		23	12%
Code errors	245	44%	67%	3	1%
Correct exercises	120	21%	33%		
Total	562	100%			

of various parameters. The experiment aimed to generate exercises with several parameters including title, difficulty level, context, task description, input and output details, an example, the language used for the solution, the solution code itself, and five input/output tests. Moreover, parameters such as title, context, etc., were required in multiple languages: Portuguese, English, Italian, and Polish. The objective was to use the GPT API to automatically generate these exercises, verify their correctness, and convert them into the YAPExIL format [16], making them suitable for further integration into AuthorKit.

The main challenges in this process involved getting consistent responses for automatic conversion and testing, as well as ensuring the correctness of the programming exercises. Several different approaches were initially attempted, such as simply asking for the exercise with the desired information or requesting separation with “;”, among others. All these attempts faced the issue of GPT providing inconsistent responses, with changing field keys and sometimes breaking the requested format. Finally, it was decided to request the responses in JSON format, providing an example within the prompt. Thus, each prompt included a description of the required parameters and an example of a response in JSON format. Once the exercises were received, the JSON format was validated. If the format was valid, the solution code was tested with the input and output tests. Therefore in the process, two types of errors could occur: a JSON error, as illustrated in Listing 1, indicating a problem with the format, or a code error, as shown in Listing 2, arising from issues in the solution code or failure of a test case. When one of these errors occurred, a response was sent to GPT describing the error and requesting a fixed exercise. This process is showcased in Figure 1.



■ **Figure 1** Process of the experiment.

21:4 Authoring Programming Exercises Assisted by Generative AI

Table 1 displays the results of achieving 120 correct programming exercises, including the number of JSON and code errors, as well as those that were resolved. To arrive at 120 accurate exercises, a total of 562 were generated, indicating that only 21% of the exercises were correct. 199 exercises, accounting for 35% of the total generated, encountered JSON Errors, of which 12% were rectified upon consulting GPT. Furthermore, 245 exercises, which represent 44% of the total, failed due to errors in a test case or the solution code. Only 3 of these could be corrected. It is important to note that a code error can only be detected if the JSON format is valid. This means that out of the 363 exercises with a valid JSON format, 245 had a code error, translating to 67%. The consistency of parameters such as title, solution, task, etc., was manually verified across most generated exercises, and no significant discrepancies were found.

Some limitations of the experiment included the use of GPT version 3.5; it is possible that version 4.0 might have delivered better results. The reason behind choosing GPT 3.5 was that when the experiment was conducted, GPT 4.0 was relatively new and associated with a higher cost. Furthermore, at that time, there were no other well-known generative AI options with an API to choose from. Additionally, generating the parameters step-by-step could have potentially enhanced GPT's capabilities, but this approach was not chosen due to the API's limitation of three requests per minute [7, 21]. Not assigning a specific role to GPT in the prompts, which is often suggested to improve outcomes, was another constraint. Furthermore, the generation of numerous parameters and the extensive size of the prompts might have negatively impacted the results.

■ **Listing 1** Example of generated exercise with a JSON Error (missing brackets at the end).

```
{
  "title":{"english":"FizzBuzz problem with a twist", ...},
  ...
  "task":{"english":"Write a function that given n, prints the
    FizzBuzzBang output from 1 to n", ...},
  ...
  "solution_code":"...",
  "tests":[..., {"input":"1", "output":"1"}
}
```

Despite these limitations, the results highlight a challenge with GPT in generating valid JSON formats using the chosen method. Similar issues with JSON formats have been reported by other programmers in community forums [9]. The findings also underscore GPT's limitations in creating completely accurate programming exercises. With a general code error rate of 44%, and 67% when considering only exercises with valid formats, the experiment clearly demonstrates these constraints. However, it is worth noting that in many code error instances, the overall exercise was correct; often, only one or two test cases were slightly off, sometimes due to interpretation issues. For instance, in tasks asking for the maximum word length in a string, there were discrepancies in whether a comma was counted to the word before or not. In the solution code, GPT generated a code that counted a comma with the word before, however, in the test case, it interpreted it contrarily.

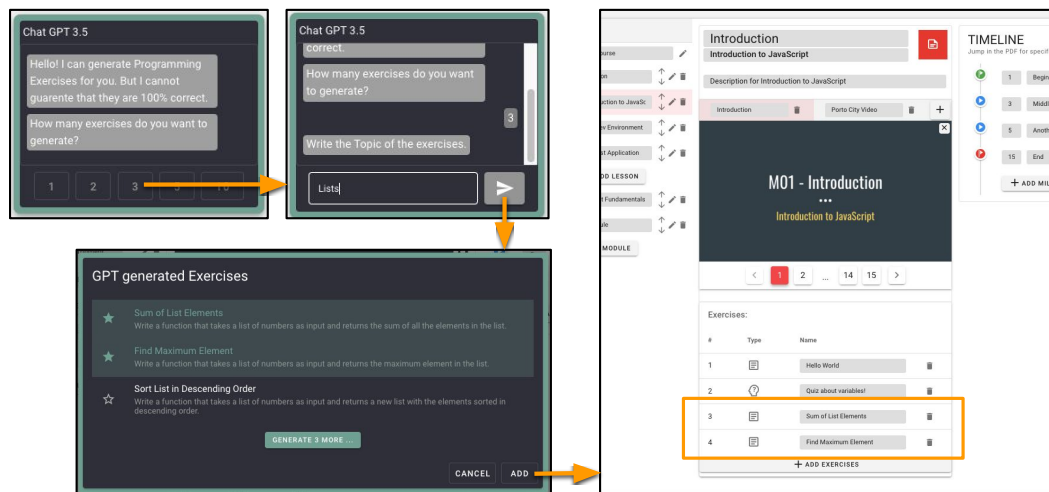
4 Authoring Tools

GPT was integrated into Agni [20], a web-based platform designed for learning and teaching JavaScript, to assist teachers in creating programming exercises. However, some changes were made from the previous experience to improve the results. The approach of Joseph

■ **Listing 2** Example of generated exercise with a Code Error (first test case).

```
{
  "task":{"english":"Create a function that receives a sentence and
    returns the largest word in it. If there are multiple words with
    the same length, return the first one in the sentence.", ...},
  ...
  "solution_code":
    "def largest_word(sentence):
      words = sentence.split()
      largest = ''
      for word in words:
        if len(word) > len(largest):
          largest = word\n return largest",
  "tests":[
    {"input":"Hello, world!", "output": "world"},
    {"input":"What is your profession?", "output": "profession?"},
    {"input":"Mathematics isn't a sport.", "output": "Mathematics"},
    ...]
}
```

Martinez [12] was followed to include a JSON schema as a parameter in the API prompt. This clearly improved the consistency of the responses with no further JSON errors detected. Also, GPT was instructed to take the role of a teacher to help create programming exercises for JavaScript. Another distinction from the experiment was the demand for viewer parameters in Agni, accommodating requests for exercise titles, statements, solution codes, and corner test cases with input and expected output. The version of GPT, namely GPT-3.5, was kept.



■ **Figure 2** Interface of Agni with the Chat GPT Chatbot.

To provide users with an interactive experience and considering broader future use cases with AI, a chatbot-like feature was incorporated, as shown in its usage in Figure 2. In the early stages, this feature does not provide a direct conversation with ChatGPT but serves as a means to input data to generate exercises. This “bot”, accessible during the editing process of a lesson, notifies authors of potential inaccuracies in the exercises, which can occur despite improvements made from previous experiments. It then prompts users to specify the number and topic of exercises they wish to create. A request is sent to the OpenAI API to create the

specified number of programming exercises on the desired topic. The received exercises are then automatically validated in JSON format using the Ajv JSON Schema validator. If valid, they are presented in a pop-up displaying key information such as the title and statement. Within this pop-up, authors can request the generation of three additional exercises and select those to be included in the current lesson. Upon selecting the exercises and clicking the button to add them, they are added to the current lesson. The evaluation of the test cases and the solution is not performed in this feature because, in the previous experiment, many code errors occurred when only one or two out of five test cases failed due to minor issues. Therefore, it was deemed more effective to keep all exercises, as those with errors can be easily corrected. After adding the exercises, when a teacher enters one of the generated ones, the solution code with its test cases is run showcasing visually to the teacher which of the tests fail or pass.

This refined approach resolved the formatting challenges of the initial experiment, offering a robust tool to aid teachers in authoring programming exercises.

5 Ongoing and Future Work

This paper presented an experiment on creating programming exercises using the OpenAI API, highlighting its difficulties in generating valid JSON formats and producing sample solutions with correct test cases. In the experiment, 67% of the correctly formatted exercises contained a code error, indicating that a test case failed when running the provided solution. However, it is important to note that often it was only one out of five test cases that failed, sometimes due to an interpretation issue of the exercise. A refined feature to assist teachers in generating programming exercises was then integrated into Agni. To address the formatting issues, a different approach was adopted, which involved providing a JSON schema in the prompt to the OpenAI API. However, challenges with the solutions and test cases persisted, which is why the feature is considered an assistant rather than an automatic generator.

Despite the stated challenges, the feature still offers potential gains in the speed of creating new programming exercises. Consequently, a questionnaire and survey are being conducted where evaluators have to create programming exercises using Agni, both with and without the help of GPT. The evaluators are recording their time and then responding to follow-up questions about their opinions on the quality and efficiency of using GPT compared to not using it.

After collecting a substantial number of responses, the data will be analyzed to understand the teachers' opinions on this tool and whether a significant reduction in time was observed. We expect these findings to provide insights into how teachers perceive generative AI features and to help to improve current and potentially future features.

References

- 1 Eman A. Alasadi and Carlos R. Baiz. Generative ai in education and research: Opportunities, concerns, and solutions. *Journal of Chemical Education*, 100(8):2965–2971, 2023. doi:10.1021/acs.jchemed.3c00323.
- 2 Brett A Becker, Michelle Craig, Paul Denny, Hieke Keuning, Natalie Kiesler, Juho Leinonen, Andrew Luxton-Reilly, James Prather, and Keith Quille. Generative ai in introductory programming. *Name of Journal*, 2023.
- 3 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*, SIGCSE 2023, pages 500–506, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3545945.3569759.

- 4 Erik Brynjolfsson, Danielle Li, and Lindsey Raymond. Generative ai at work, 2023. arXiv:2304.11771.
- 5 Paul Denny, Hassan Khosravi, Arto Hellas, Juho Leinonen, and Sami Sarsa. Can we trust ai-generated educational content? comparative analysis of human and ai-generated learning resources, 2023. arXiv:2306.10509.
- 6 Paul Denny, Sami Sarsa, Arto Hellas, and Juho Leinonen. Robosourcing educational resources – leveraging large language models for learnersourcing, 2022. arXiv:2211.04715.
- 7 Enterprise DNA Experts. How to use chat gpt: A simple guide for beginners, September 2023. URL: <https://blog.enterprisedna.co/how-to-use-chat-gpt/>.
- 8 Stefan Feuerriegel, Jochen Hartmann, Christian Janiesch, and Patrick Zschech. Generative ai. *SSRN Electronic Journal*, January 2023. doi:10.2139/ssrn.4443189.
- 9 Aaron Issac. Chatgpt functions malformed json, July 2023. URL: <https://community.openai.com/t/chatgpt-functions-malformed-json/306509>.
- 10 Lukas Jürgensmeier and Bernd Skiera. Generative ai for scalable feedback to multimodal exercises in marketing analytics. *Available at SSRN*, 2024.
- 11 Richard Lobb and Jenny Harlow. Coderunner: a tool for assessing computer programming skills. *ACM Inroads*, 7(1):47–51, February 2016. doi:10.1145/2810041.
- 12 Joseph Martinez. Return json from gpt, July 2023. URL: <https://betterprogramming.pub/return-json-from-gpt-65d40bfc2ef6>.
- 13 Rytis Maskeliūnas, Robertas Damaševičius, Tomas Blažauskas, Jakub Swacha, Ricardo Queirós, and José Carlos Paiva. Fgpe+: The mobile fgpe environment and the pareto-optimized gamified programming exercise selection model—an empirical evaluation. *Computers*, 12(7):144, 2023.
- 14 Rosario Michel-Villarreal, Eliseo Vilalta-Perdomo, David Ernesto Salinas-Navarro, Ricardo Thierry-Aguilera, and Flor Silvestre Gerardou. Challenges and opportunities of generative ai for higher education as explained by chatgpt. *Education Sciences*, 13(9), 2023. doi:10.3390/educsci13090856.
- 15 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, and Jakub Swacha. Fgpe authorkit—a tool for authoring gamified programming educational content. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pages 564–564, 2020.
- 16 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, and Jakub Swacha. Yet another programming exercises interoperability language (short paper). In *9th Symposium on Languages, Applications and Technologies (SLATE 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
- 17 Ivica Pesovski, Ricardo Santos, Roberto Henriques, and Vladimir Trajkovik. Generative ai for customizable learning experiences. *Sustainability*, 16(7), 2024. doi:10.3390/su16073034.
- 18 Prajish Prasad and Aamod Sane. A self-regulated learning framework using generative ai and its application in cs educational intervention design. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2024*, pages 1070–1076, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3626252.3630828.
- 19 Junaid Qadir. Engineering education in the era of chatgpt: Promise and pitfalls of generative ai for education. In *2023 IEEE Global Engineering Education Conference (EDUCON)*, pages 1–9, 2023. doi:10.1109/EDUCON54358.2023.10125121.
- 20 Ricardo Alexandre Peixoto de Queirós. Integration of a learning playground into a lms. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 2*, pages 626–626, 2022.
- 21 Laurie Ruettimann. How to interact with chat gpt-4 effectively: Top tips for better questions, September 2023. URL: <https://laurieruettimann.com/chat-gpt-4-ask-questions/>.
- 22 Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1, ICER '22*, pages 27–43, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3501385.3543957.

21:8 Authoring Programming Exercises Assisted by Generative AI

- 23 Jiahong Su and Weipeng Yang. Unlocking the power of chatgpt: A framework for applying generative ai in education. *ECNU Review of Education*, 6:1–12, April 2023. doi:10.1177/20965311231168423.
- 24 John Wrenn, Shriram Krishnamurthi, and Kathi Fisler. Who tests the testers? In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ICER '18, pages 51–59, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3230977.3230999.