Third International Computer Programming Education Conference

ICPEC 2022, June 2–3, 2022, Polytechnic Institute of Cávado and Ave (IPCA), Barcelos, Portugal

Edited by

Alberto Simões

João Carlos Silva

OASICS

Editors

Alberto Simões
Polytechnic Institute of Cávado and Ave (IPCA), Barcelos, Portugal
asimoes@ipca.pt

João Carlos Silva
Polytechnic Institute of Cávado and Ave (IPCA), Barcelos, Portugal
jcsilva@ipca.pt

ACM Classification 2012
Applied computing → Education

ISBN 978-3-95977-229-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-229-7.

Publication date
July, 2022

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.

Digital Object Identifier: 10.4230/OASIcs.ICPEC.2022.0

OASIcs – OpenAccess Series in Informatics

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Preface

This book includes the articles that were accepted for the Third Edition of the International Computer Programming Education Conference (ICPEC). Born in the COVID era, this is the first edition of the conference that was held physically. The event took place in Barcelos, Portugal, at Instituto Politécnico do Cávado e do Ave, from June, 2$^{\text{nd}}$ to June, 3$^{\text{rd}}$.

Teaching computer science is a challenging task: from the traditional education, with paper and a blackboard, to more computational training, where students are able to test their solutions in a computer, or even with sophisticated approaches, using interactive learning tools, there is still no perfect solution. Computer science, and specifically learning how to program and how to develop algorithms and data structures, are still the subjects with less success in Computer Science Bachelor Degrees.

While a lot of lecturers have their own research interests, apart from their teaching duties, there are some lecturers that continue to debate, study and analyse different teaching approaches, and developing solutions to help in the learning process. ICPEC was created by such a group of lecturers, enthusiastic with teaching computer science and, specifically, programming languages and algorithms. During the last three years, ICPEC has grown in a larger (but still small) community of researchers interested in these topics.

This edition includes fourteen contributions in the area of education of computer science subjects. While most works still focus on the challenge that is the teaching of programming languages and algorithms, there are also contributions for the awareness of cloud and internet security, user-interface testing and even mathematics.

We are sure that the possibility to discuss these subjects in a community, and live without digital means of communication, that we are all tired of, has been productive and that synergies will be created for further study of this conference topics.

Hopefully, with the probable control of COVID and the return to face-to-face conferences, we will be able to increase this community and share experiences in order to be better teachers and researchers in this challenging area of computer programming.

The Steering Committee

Alberto Simões
Filipe Portela
Mario Pinto
Ricardo Queirós

Scientific Committee

Editors

Alberto Simões
2Ai, School of Technology, IPCA
Barcelos, Portugal
asimoes@ipca.pt

João Carlos Silva
School of Technology, IPCA
Barcelos, Portugal
jcsilva@ipca.pt

Committees

Alberto Simões
2Ai, School of Technology, IPCA
Barcelos, Portugal
asimoes@ipca.pt

Filipe Portela
Algoritmi, Universidade do Minho
Guimarães, Portugal
cfp@dsi.uminho.pt

Mário Pinto
ESMAD, Politécnico do Porto
Vila do Conde, Portugal
mariopinto@esmad.ipp.pt

Ricardo Queirós
ESMAD, Politécnico do Porto
Vila do Conde, Portugal
ricardoqueiros@esmad.ipp.pt

Organizing Committee

Alberto Simões
2Ai, School of Technology, IPCA
Barcelos, Portugal
asimoes@ipca.pt

Daniel Miranda
2Ai, School of Technology, IPCA
Barcelos, Portugal
damiranda@ipca.pt

João Carlos Silva
School of Technology, IPCA
Barcelos, Portugal
jcsilva@ipca.pt

Nuno Rodrigues
School of Technology, IPCA
Barcelos, Portugal
nfr@ipca.pt

Patrícia Leite
School of Technology, IPCA
Barcelos, Portugal
patricialeite@ipca.pt

Scientific Committee

Alba Amato
Seconda Università degli Studi di Napoli
Naples, Italy
alba.amato@unina2.it

Alberto Simões
2Ai, School of Technology, IPCA
Barcelos, Portugal
asimoes@ipca.pt

Alexander Paar
Duale Hochschule Schleswig-Holstein
Kiel, Germany
alexpaar@acm.org

Alexandre Braganca
ISEP, Politécnico do Porto
Porto, Portugal
atb@isep.ipp.pt

Ana Azevedo
ISCAP, Politécnico do Porto
S. Mamede de Infesta, Portugal
aazevedo@iscap.ipp.pt

Anabela Gomes
University of Coimbra
Coimbra, Portugal
anabela@isec.pt

João Cordeiro
Universidade da Beira Interior
Bragança, Portugal
jpcc@ubi.pt

J. Ángel Velázquez-Iturbide
Universidad Rey Juan Carlos
Madrid, Spain
angel.velazquez@urjc.es

José Carlos Paiva
Faculdade de Ciências
Universidade do Porto, Portugal
josepaiva94@gmail.com

José Paulo Leal
Faculdade de Ciências
Universidade do Porto, Portugal
zp@dcc.fc.up.pt

Karolina Baras
University of Madeira
Funchal, Portugal
kbaras@uma.pt

Leonel Morgado
INESC TEC/Universidade Aberta
Coimbra, Portugal
leonel.morgado@gmail.com

Marco Temperini
Sapienza University of Rome
Rome, Italy
marte@dis.uniroma1.it

María Ángeles Pérez Juárez
University of Valladolid
Valladolid, Spain
mperez@tel.uva.es

Maria José Marcelino
University of Coimbra
Coimbra, Portugal
zemar@dei.uc.pt

Mário Pinto
ESMAD, Politécnico do Porto
Vila do Conde, Portugal
mariopinto@esmad.ipp.pt

Martinha Piteira
EST, Instituto Politécnico de Setúbal
Setúbal, Portugal
martinha.piteira@estsetubal.ips.pt

Micaela Esteves
Polytechnic Institute of Leiria
Leiria, Portugal
micaela.dinis@ipleiria.pt

Míriam Antón-Rodríguez
University of Valladolid
Valladolid, Spain
mirant@tel.uva.es

Muhammad Younas
Oxford Brookes University
Oxford, United Kingdom
m.younas@brookes.ac.uk

Nikolaos Matsatsinis
Technical University of Crete
Akrotiri, Crete, Greece
nikos@ergasya.tuc.gr

Nuno Rodrigues
School of Technology, IPCA
Barcelos, Portugal
nfr@ipca.pt

Patrícia Leite
School of Technology, IPCA
Barcelos, Portugal
patricialeite@ipca.pt

Paula Morais
Universidade Portucalense
Porto, Portugal
pmorais@upt.pt

Paula Tavares
ISEP, Politécnico do Porto
Porto, Portugal
pct@isep.ipp.pt

Pedro Rangel Henriques
Algoritmi, Universidade do Minho
Braga, Portugal
prh@di.uminho.pt

Pedro Ribeiro
Faculdade de Ciências
Universidade do Porto, Portugal
pribeiro@dcc.fc.up.pt

Raffaele Montella
University of Napoli Parthenope
Napoli, Italy
raffaele.montella@uniparthenope.it

List of Authors

Alberto Sierra
CIFP Carlos III
Cartagena, Spain
alberto.sierra@murciaeduca.es

Álvaro Costa Neto
Instituto Federal de Educação
Ciência e Tecnologia de São Paulo
São Paulo, Brazil
nepheus.br@gmail.com

Ana Breda
University of Aveiro
Aveiro, Portugal
ambreda@ua.pt

Ana Francisca Monteiro
Universidade do Minho
Braga, Portugal
anafmonteiro@gmail.com

André Santos
ISCTE-Instituto Universitário de Lisboa
Lisboa, Portugal
andre.santos@iscte-iul.pt

António Osório
Universidade do Minho
Braga, Portugal
ajosorio@ie.uminho.pt

Cristiana Araújo
Universidade do Minho
Braga, Portugal
decristianaaraujo@hotmail.com

Ece Ata
Siemens AG
Germany
eceatata@gmail.com

Eugénio A.M. Rocha
University of Aveiro
Aveiro, Portugal
eugenio@ua.pt

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

Francini Hak
Algoritmi, University of Minho
Guimarães, Portugal
francini.hak@algoritmi.uminho.pt

Gabryella Rodrigues
Universidade do Minho
Braga, Portugal
gabryella.rocha@gmail.com

Jakub Swacha
University of Szczecin
Szczecin, Poland
jakubs@uoo.univ.szczecin.pl

Jorge Oliveira E Sá
Algoritmi, University of Minho
Guimarães, Portugal
jos@dsi.uminho.pt

José Manuel Dos Santos Dos Santos
Escola Superior de Educação, Politécnico do Porto
Porto, Portugal
dossantosdossantos@gmail.com

José Paulo Leal
CRACS - INESC-Porto LA & DCC - FCUP
Porto, Portugal
zp@dcc.fc.up.pt

Juan V. Carrillo
CIFP Carlos III
Cartagena, Spain
juanvicente.carrillo@murciaeduca.es

Luke Adrian Bayzid Gubbins
Universidade de Aveiro
Aveiro, Portugal
luke.adrian@ua.pt participant Marco Primo
Faculty of Sciences, University of Porto
Porto, Portugal
up201800388@edu.fc.up.pt

Maria João Varanda Pereira
Instituto Politécnico de Bragança
Bragança, Portugal
mjoao@ipb.pt

Maria Medvidova
Technical University in Kosice
Kosice, Slovakia
maria.medvidova@gmail.com

Maria Pinto-Albuquerque
ISCTE-IUL - Instituto Universitário de Lisboa Lisboa, Portugal
maria.albuquerque@iscte-iul.pt

Mario Pinto
ESMAD, Politecnico do Porto
Vila do Conde, Portugal
mariopinto@esmad.ipp.pt

Pedro Ferreirinha
Faculty of Sciences of University of Porto
Porto, Portugal
up201805186@edu.fc.up.pt

Pedro Rangel Henriques
Universidade do Minho
Braga, Portugal
prh@di.uminho.pt

Ricardo Queirós
ESMAD - P.PORTO & CRACS - INESC TEC
Vila do Conde, Portugal
ricardo.queiros@gmail.com

Salvador Pellicer
Entornos de Formación (EdF)
Valencia, Spain
salvador.pellicer@entornosdeformacion.com

Teresa Terroso
ESMAD, Politécnico do Porto
Vila do Conde, Portugal
teresaterroso@esmad.ipp.pt

Thomas Schreck
Munich University of Applied Sciences
Munich, Germany
thomas.schreck@hm.edu

Tiago Gasiba
Siemens AG
Germany
tiago.gasiba@siemens.com

Tiange Zhao
Siemens AG
Germany
zhaotiange123@gmail.com

Tilman Dewes
Munich University of Applied Sciences
Munich, Germany
dewes@hm.edu

Ulrike Lechner
Universität der Bundeswehr München
München, Germany
ulrike.lechner@unibw.de

Value-Focused Investigation into Programming Languages Affinity

Alvaro Costa Neto
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Barretos, Brazil

Cristiana Araújo
Centro ALGORITMI, Departamento de Informática,
University of Minho, Campus Gualtar – Braga, Portugal

Maria João Varanda Pereira
Research Centre in Digitalization and Intelligent Robotics,
Polythechnic Insitute of Bragança, Portugal

Pedro Rangel Henriques
Centro ALGORITMI, Departamento de Informática,
University of Minho, Campus Gualtar – Braga, Portugal

Abstract

The search for better techniques to teach computer programming is paramount in order to improve the students' learning experiences. Several approaches have been proposed throughout the years, usually through technical solutions such as evaluation systems, digital classrooms, interactive lessons and so on. Personal factors, such as affinity, have been largely unexplored due to their qualitative and abstract nature. The results of a preliminary survey on how and why affinity is created between programmers and their favorite languages, conducted on a master's degree class at Universidade do Minho, showed unexpected results as to which languages became favorites and the possible reasons for the students' choices. Aiming at further exploration on this topic and continuation of this research, the Value-Focused Thinking method was applied in order to construct a more complex, in-depth survey. This value-oriented method kept focus under control and even raised a handful of opportunities to improve the research as a whole. This paper describes the Value-Focused Thinking method and how it was applied to construct a new and deeper computer programming education survey to understand affinity with languages.

2012 ACM Subject Classification Social and professional topics → Computing education; Software and its engineering → General programming languages

Keywords and phrases Computer Programming, Programming Languages, Affinity, Education, Learning, Value-Focused Thinking

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.1

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/05757/2020 and UIDB/00319/2020.

1 Introduction

Being an inherently intricate process, learning computer programming is widely accepted as being a complex and difficult task. Technical approaches applied to research on how to teach and learn computer programming date many decades ago [6, 1, 2], and has kept high interest in the academic field [7, 18, 14, 3, 9, 10] ever since. Personal and contextual factors also play important roles in learning and should be considered when teaching computer programming. Pedagogical research has shown for more than a century that these personal factors are influential to the teaching-learning process [12, 17, 8, 4] and should be taken into account at all times.

Affinity to programming languages, as a personal factor, may play an important role in the teaching-learning process, as indicated in a previous survey conducted in 2021 with a class of master's degree students [13]. Those preliminary results showed that affinity is more complex than previously assumed and that a deeper, more structured study should be conducted. A value-focused approach was used in order to better organize the construction of the survey for this new study, yelding a strongly focused questionnaire.

This paper starts with a brief presentation of a former preliminary study on affinity to programming languages, but focuses on the construction of a new survey through the Value-Focused Thinking method. The new survey will be used in the near future, aiming to further understand how affinity to a programming language is created. The overall structure of this article is composed of six sections: the introduction discusses which factors may influence the learning process and how they may be categorized. The second section presents the initial investigation, including a preliminary study about affinity to programming languages and a small previous survey that showed interesting results. The third section gives a general explanation of Value-Focused Thinking (VFT), a decision making method used to structure the new survey. Section four explains how Value-Focused Thinking was directly applied to the construction of the new survey. Section five lists the final structure of the new survey and the expected results. The final section concludes this paper and lists the next steps that shall be taken to apply the new survey on affinity to programming languages.

2 Background and Previous Work

Assuming that personal factors are relevant to students – as previously stated – and focusing on the affinity that is commonly observed among programmers (both students and professionals) towards specific programming languages, it would be reasonable to investigate how it is established and which role it plays on the learning process. In order to initiate this investigation, a preliminary study was conducted.

2.1 Lecture and Preliminary Study

In order to better understand what role affinity takes in the learning process and how it is constructed, a preliminary study was conducted with twenty three students of a Masters' degree class in Computer Engineering at Universidade do Minho [13]. The study consisted of a lecture about teaching and learning computer programming, a quick survey during the lecture, and a small questionnaire with a few questions about programming languages, their learning experiences, and which languages they prefered.

The lecture presented and discussed which factors might be relevant to the learning process and how these factors could be applied to improve the students' experiences. During the lecture, a quick survey was conducted, based on snippets of source code that printed a few numbers of the Fibonacci sequence. These snippets were written in seven different languages – BASIC, Lisp, C, Java, Python, Ruby, and Swift – and were shown sequentially at first, and then simultaneously for comparison purposes.

Since it would be impractical – and probably confusing – to show all possible combinations, only a few comparisons were made. Each combination tried to explicate either differences or similarities in their programming languages, encouraging students to consider unusual characteristics that could influence their affinity to one of the languages. As an example, when comparing two different versions of Lisp code, the goal was to highlight the impact caused by the use of the natural coding style for a certain language. This would allow the students to consider nuances beyond the plain syntax definition of a language.

Students were then promptly asked to vocalize their preferred languages and the answers showed C, Java and Haskell[1] as the three highest ranked choices. This question was proposed and read during the lecture, and students answered directly without any kind of written form.

After the lecture, the students were then asked to answer a small questionnaire containing three questions:

1. In a range of *very low* to *very high*, how important is the language choice for learning computer programming? Justify your answer;
2. Which factors – presented in the lecture – are most relevant and influential to learning computer programming?
3. Which languages would you choose to teach computer programming: BASIC, Lisp, C, Java, Python, Ruby, Swift, or some other language? Justify your answer.

The answers to the first question were interesting, albeit inconclusive. While almost every student agreed that it is crucial to wisely choose an initial programming language, pretty much all of them differed on the justification. Opinions ranged from technical – mainly based on the availability of certain syntax constructs and data structures – to pedagogical – the initial language should have an easy learning curve in order to avoid discouraging students. Other justifications cited documentation availability, prospective employability, and in one case, indifference for the language choice *per se*.

Answers to the second question focused on the obstacles that the initial programming language could impose. Technical aspects of the language were preeminent among the answers, but *affinity* was also considered an important factor. Other personal factors included: the relationship stablished with the teachers and the motivation that is cultivated during the first contact with computer programming.

The last question allowed multiple choices and its main goal was to compare the results of the lecture's question (the one where C, Java and Haskell were three favorite languages), with the selection of languages they would choose to pass on, possibly influencing the affinity other students would develop. This would either support their initial choices – their preferred languages were chosen to continue the learning cycle – or contradict it – their preferred languages and their choices to pass on programming knowledge were different. The final goal for this question was to test for external influences, such as popularity and market share. Being currently a popular programming language [16, 15], with many applications in high demand, such as artificial intelligence, data science and numerical computing, Python was expected to be preferred. Nonetheless, C and Java were the highest ranked in the questionnaire, while Haskell also had a perceptive presence in the results and tied the fourth place with BASIC (figure 1). These results become clearer if the students' affinity to these languages was actually constructed as a *consequence of their learning experiences*, since they had been formally taught Haskell, C and Java as their initial programming languages.

These results implied that more conclusions would arise from a new survey for a deeper investigation on how students learn computer programming and their prefered languages. The complex nature of personal factors that influence this process creates opportunities for continued investigation on *affinity with programming languages and its relation to the learning process.*

[1] Haskell is the first programming language for students of Universidade do Minho.

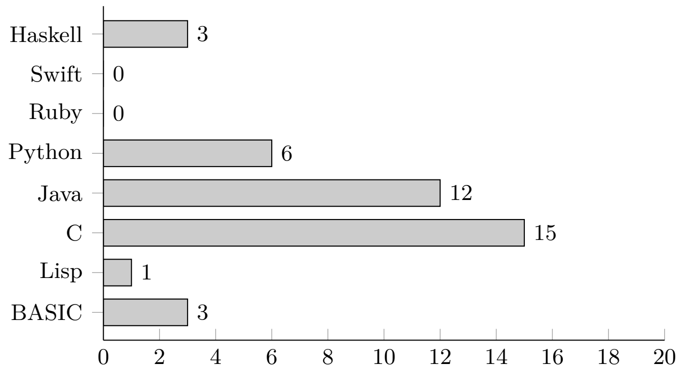


Figure 1 Languages chosen by students when asked which one they would use to teach computer programming. These answers were gathered through a small survey after the lecture. This figure was originaly published in [13].

3 Value-Focused Thinking

Value-Focused Thinking (VFT) is a decision making process proposed by Ralph Kenney [11] and it was chosen as the formal method to plan and guide the construction of the new survey. It is based on the fact that planning alternatives and practical details in the first place tends to diverge the solution from what should be its main focus, concentrating efforts on features that might be discarded, and missing other opportunities that could emerge. In order to avoid this kind of recurrent and short sighted behavior, Kenney proposed a method that would force the definition of the main values and their derived objectives first, driving the whole decision making process with focus. In the author's words:

> Alternative-focused thinking is designed to solve decision problems. Value-focused thinking is designed to identify desirable decision opportunities and create alternatives [11, p. 538].

The method stablishes a well-founded process, based on the premise that the main values, central to the decisions being made, should always be enforced and guarded when thinking about objectives and alternatives. The whole VFT process follows three main stages: definition of values, gathering of objectives and construction of alternatives.

3.1 Definition of Values

The first stage in a Value-Focused Thinking process is to define the values and contextualize the problem to be solved. This step takes into account the desired results, what are the expectations – both for success and failure – and what kind of experience has been observed in the same context and on similar problems. While it is not the most operational part of the process, information gathered at this point is crucial to gather realistic objectives in the next step and to keep the whole decision making process focused.

3.2 Gathering of Objectives

In the second stage of the process, the objectives are gathered, sorted and classified. This stage is crucial for grounded and efficient construction of alternatives – the main goal of the Value-Focused Thinking process. The first to be defined is the *strategic objective* which states the main abstract goal of the decision being made. Other objectives are listed and roughly classified into *fundamental* or *means*:

- Fundamental objectives stablish the main reasons for the decision making in the first place, and are usually directly related to the strategic objective;
- Means objectives define what is necessary to achieve other objectives.

In order to determine if an objective should be considered fundamental or means, Kenney proposed the question "why is this objective important in the decision context?". If the answer is "because it is one of the essential reasons for interest in the situation" then the objective should be considered fundamental. On the other side, if the answer resembles "because of its implications for achieving some other objective", it is a means objective [11].

When the first version of the objectives list is concluded, revisions should be made in order to simplify – by aggregating redundant objectives – and reclassify the list – by applying the question above. There is no definition as to when this revision cycle should end. In summary, as soon as all important objectives are listed and classified, and the revisions no longer change the list, this part of the process is done.

3.3 Construction of Alternatives

The final stage concludes the process by creating alternatives while also identifying opportunities. The alternatives represent the courses of action to be taken and need to directly relate to the previously listed objectives (usually *means objectives*, but not exclusively). The most obvious alternatives usually come from previous experiences and commonly are the first ones to be thought of. Once these have been considered, deep thought about the problem should be carried on, in order to pursue hidden and more unexpected alternatives, always keeping in mind the values defined as per the Subsection 3.1.

Since each alternative should be related to at least one objective, the most straightforward way to undertake this stage is to list the objectives and propose one alternative for each. Once all objectives are evaluated, the construction stage restarts considering two objectives at a time, then three, and so on until all objectives are grouped together to create one alternative. This final state may not be reachable depending on the problem being tackled, but the construction stage should go as far as possible in this direction. Once all alternatives are listed, a review process should try to eliminate redundancies, which usually occurs with the first entries of the list.

Opportunities arise when trying to create alternatives. In some situations, an alternative presents some kind of limitation or necessity that must be fulfilled. Instead of considering it a failed attempt, one should identify these obstacles as opportunities to be further explored, possibly starting entirely new decision making processes.

4 Application of Value-Focused Thinking

The application of Value-Focused Thinking into the development of the survey was based on the fact that, for all intended purposes, *deciding which questions to ask and how to ask them is a decision making situation.* The choice of VFT among other methods was also motivated by its lean and straighforward mechanism that generates highly focused outcomes, and by previous positive experiences using it on similar projects.

The process of constructing the survey through VFT followed the standard course of action for the method: definition of the main values and context, gathering of the objectives, and construction of the alternatives[2].

[2] A read-only copy of the VFT document may be found in the following address: `https://bit.ly/3LrVV2D`.

4.1 Values and Context

The initial part of the VFT method is essential for focusing the rest of the process (as previously mentioned in Subsection 3.1) and it is usually a relatively straightforward part. In this case, though, the method was not being applied to a conventional situation – business related decision making, such as which parts to buy, who to buy them from or which bonds to sell – but to aid in the construction of a research survey. This peculiarity posed an interesting view on the whole process, since the decision being made did not apply directly to the research conducted through the survey, but *to the survey itself.* Being a means to an end – finding out which characteristics are related to programming language affinity – the survey is still part of the research as a whole, but the VFT was applied specifically to support the construction of the survey and its values represented that intent.

The chosen values were:

1. High comprehensability;
2. Focus on the main topics being researched;
3. Maximum coverage of different personal profiles;
4. Easy publishing and completion;
5. Gathering of valid and trustworthy answers.

The list of values clearly states the focus of the survey: to be *efficient* (values 1, 3 and 4) and *reliable* (values 2 and 5). Obvious as it might seem, it is crucial to list – and later abide to – these values. When thinking only on the objectives, as an example, one could easily lose focus and plan too many questions. By clearly stating the values, this unintentional mistake would be immediately identified violating "Easy [...] completion" and properly fixed.

The following step in this part was to list the perfect, average and terrible scenarios that could happen, to serve as guidelines. The perfect scenario pointed to a totally efficient questionnaire, with as many answers as possible, from multiple and varied sources, leading to a clear and encompassing conclusion. The average scenario presented high efficiency to the questionnaire with many answers from many sources, leading to an important and relevant conclusion. The terrible scenario represented an ineficient questionnaire with almost no answers, leading to no conclusion at all.

The final step for the contextualization listed previous experiences in similar endeavours that influenced the results of previous surveys and could possibly happen again. The main occurrences were:

- Low quantity of answers, which diminished the representativeness of the conclusions;
- Discarded answers that pointed to some kind of misunderstanding of what was asked;
- Unexpected and interesting results from open ended questions;
- Direct questions that seemed to be answered randomly (in contrast with other answers).

These experiences, being good or bad, were important warnings of caution to take into consideration for the next part of the VFT method: gathering and listing the objectives.

4.2 Objectives

Based on the values and the context previously stablished, the objectives were listed and categorized. The strategic objective, as explained in Subsection 3.2, represented the main goal of the survey: *to construct and conduct a capable, valid and trustworthy survey to evaluate which factors influence the affinity stablished between programmers – of any level or context – to computer programming languages.*

While the strategic objective was an abstract take on the main values, the following fundamental objectives were a further step into its concretization:

1. To select as wide a range of respondents as possible, including those without prior knowledge of computer programming;
2. To ask questions that allow finding correlations between respondents and their favorite languages;
3. To faithfully characterize both personal aspects of the respondents and technical aspects of the programming languages.

The next step was to stablish the means objectives. They were constructed to support the fundamental objectives, creating the basic ideas of a real survey. Albeit being the most practical step of the gathering of objectives, these should not include implementation details, as these would follow in the construction of alternatives. The mean objectives listed practical needs (an on-line survey system that allows the construction of the intended questionnaire), publishing strategies (educational institutions, social media and professional hubs), which kind of information to gather (personal data, educational history, which languages are known etc.) and interesting results to be obtained (influence of peers on the choices of favorite languages, favorite technical aspects of the programming languages, popularity etc.). These objectives were then directly mapped to an implementation proposal of the survey in the last part of the VFT method: the construction of alternatives.

4.3 Alternatives

The last part of the VFT method created the alternatives for the undergoing decision-making process. In a general sense, the alternatives presented different possibilities to achieve the intended goals. In this specific case, alternatives stablished different ways of constructing the questionnaire, always keeping objectives and values in mind[3].

Each element of the survey definition – which on-line survey tool to use, possible publishing methods, valid and robust user agreement, questions, and desired results – was directly related to the means objectives gathered in the previous part. This meant that the whole survey construction was indirectly guided by all of its values through its previsouly listed objectives.

As an example, one of the means objectives specified that it would be important to gather if the respondent has got a *competitive or cooperative nature*, in order to later verify a possible correlation between this kind of personal characteristic and his or her favorite languages. This objective resulted directly in the following question definition:

- **Subject:** competitive or cooperative profile;
- **Alternatives:**
  1. Indirect observation by asking the number of participations in competitive or cooperative activities;
  2. Direct question of personal preference – competitive or cooperative.

As can be seen, the question itself was not written in this stage, only its specifications were listed. This precaution was taken in order to avoid prematurely fixing the survey before it could be reviewed as a whole. As for the alternatives, they presented themselves as different ways of constructing the questions and which kind of answers would be allowed. The resulting structure of the survey is described in Section 5 with all question definitions that were proposed at this moment.

[3] It is important to realize that the term *alternatives* should be understood as defined in the VFT method, not necessarily options for answering a question. As an example, a compilation of available on-line survey tools could be considered alternatives.

During the definition of the questions, desired statistical results – listed in Subsection 5.4 – were also specified. These served as checkpoints for the questions themselves, revealing possible "blind spots" in the survey through the absence of questions that would be necessary to obtain certain results.

The final step in the creation of alternatives was to list the opportunities that have been identified so far. In total, three opportunities were listed:

1. Since the range of desired results and questions is very wide, it would be possible to create more than one survey, dividing and better focusing the research in specific areas, such as personal factors, technical characteristics of the languages etc.
2. Available free survey tools seem to be incapable of constructing more complex surveys, so it would be beneficial to create a new system;
3. Instead of using real languages to verify the influence of technical characteristics on affinity, it would be better to create a pseudo-language flexible enough to be used in all situations.

It is important to emphasize that *these opportunities were not mandatory courses of action*, but presented different decision-making possibilities that emerged during the process of the survey construction. They may not have been detected if the alternatives (the questions of the survey, in this case) were the first elements to be thought of and, ultimately, this is one of the main advantages of the Value-Focused Thinking method.

5 Structure of the Survey

After applying all the steps of the VFT method, the structure of the survey was completed albeit not implemented. The implementation – i.e. to add the sections, questions and commentary to an on-line tool for publication and participation – is a direct consequence of the planning process, much like in computer programming, and should not be considered a requirement for conclusion of the VFT method. Based on the constructed alternatives, a general structure for the survey has been reached and it was divided into three sections: *personal data*; *background and projections in computer programming*; *affinity to different programming language characteristics.*

5.1 Personal Data

This first section of the survey deals with personal background and is composed of nine direct questions:

1. Age;
2. Gender;
3. Country of residence;
4. Native language;
5. English language level according to the Common European Framework of Reference [5];
6. Formal education level, from "Uneducated" to "Doctorate or beyond";
7. Learning style, with choices for both easiest and hardest to learn: "Mathematical and numerical problems", "Logic exercises", "Memory based questions" and "Practical applications";
8. Household income *per capita*;
9. Current occupation.

The answers to these questions will be correlated to the programming language affinity choices in the third section of the survey (5.3).

5.2 Background and Projections in Computer Programming

This section deals with previous experiences and future goals specifically about computer programming. It is composed of seven questions:
1. Time spent studying computer programming, in years;
2. Time spent working at computer programming, in years;
3. Time spent teaching computer programming, in years;
4. Learning methodologies during studying computer programming;
5. Learning methodologies applied to teach computer programming;
6. Competitive or cooperative preference for group working, measured through number of participations in activities of each kind;
7. List of effectively known programming languages, multiple choices allowed;
8. Which programming language first had contact with;
9. Intended position for future jobs in computer programming.

The answers to these questions in this section will help on drawing a picture of experience with computer programming and its languages. The answers will be compared to choices made in the third section of the survey, in order to procure possible correlations of affinity to the respondents' backgroung with computer programming – meaning that affinity is a result of learning or working with a particular language – and personal foresight – meaning that factors such as popularity and market influence are relevant to affinity.

5.3 Affinity to Different Programming Language Characteristics

The last section of the survey deals directly with affinity, collecting data about which languages lead to affinity and why. It is composed of five questions:
1. Comparisons of source code snippets, written specifically to test common syntax and semantic differences in current programming languages. This question was divided into subquestions which are detailed below;
2. Affinity level to programming languages, measured from "No affinity at all" to "Favorite";
3. Change in affinity to programming languages, which ones lost affinity, which ones gained affinity thoughout the years and what was the perceived cause to the change;
4. Influence of peers in affinity, as a personal observation;
5. Main motivation for affinity to the favorite languages, also as a personal observation.

A list of programming languages has been selected to compose the questions 2 and 3 of this section. This list included both current and former popular languages, aiming also at gathering as varied characteristics as possible, such as syntax, semantics, market share, popularity, paradigm etc.

The snippets of source code shown in question 1 were written carefuly to contrast only one syntactical or semantical characteristic at a time. This question is essential for obtaining insight into which practical characteristics of the programming languages are influential to affinity growth. Also, it represented the most practical and applied questioning of the survey. The following characteristics are queried:
- Variable declaration syntax;
- String representation and basic operation;
- Type inference and conversion;
- Block delimitation;
- Conditional structures;
- Repetition structures;

- Function or method calling convention;
- Presence and use of jumps (as in *goto*);
- End-of-statement syntax;
- General paradigm;
- Default data structures;
- Verbosity.

This list covers most technical characteristics of programming languages that might have some effect on affinity. In order to avoid blurring the respondents' answers by other personal factors, the snippets were written in a neutral algorithmic language that was informally defined[4].

Answers in this section are paramount for any conclusions about affinity, since most of them will be used as the counterpart to the correlation with answers in the previous sections. In the end of the process, since this question would have too many subquestions, it was decided to create another section of the survey dedicated to it.

5.4 Expected Statistical Results

Expected results form an important part of any survey construction. These results were identified during the application of the VFT method:

- Correlation between time spent formally studying, working and teaching computer programming and the languages with most affinity;
- Correlation between the most common technical characteristics of the languages and the affinity level;
- Preferred structural, syntatic and semantic programming languages characteristics;
- Correlation of personal background and affinity;
- Changes in favorite languages and the reasons for the new choice;
- Languages that most frequently lost or gained affinity after a change;
- Frequency at which the first learnt language presents high level of affinity;
- Correlation between career prospection and affinity to the languages with higher market share;
- Correlation between learning style and language affinity;
- Correlation between popularity and language affinity.

With this part done, the Value-Focused Thinking method successfuly helped the construction of a *Beta version* for the desired survey, that aimed at gathering feedback for improvements and validation. This version implemented the result of the VFT method almost entirely without changes, with the exception of characteristics that proved to be unbalanced in practice, such as the number of sections that raised from three to four in order to better separate the types of questions. Finally, feedback questions were added to the end of each section to gather opinions about the questionnaire *per se*. This proved to be of great value for assessing the questionnaire's main values and validate the process of Value-Focused Thinking.

[4] This decision was taken based on an opportunity, as explained in Subsection 4.3, and it might even be relevant in subsequent studies.

5.5 Feedback and Validation

The Beta version[5] of the survey was applied to three different groups of students from Universidade do Minho and Instituto Politécnico de Bragança, in order to test and validate the current implementation. Feedback from the respondents pointed towards a few notes:

- The question about household income were considered too intrusive by a few respondents;
- Some respondents had trouble understanding a few english terms in the questions, which in turn, made the survey harder to answer;
- The grid of possible answers to the questions about syntactic and semantic characeristics confused a few respondents. Some students initially considered that affinity choices were mutually exclusive, allowing only one answer of either "I don't like it", "I like it", or "I prefer it" for each snippet of code. In actuality, each answer could be selected for more than one snippet;
- Although the feedback was essentially positive, both in comprehension and duration of the survey, some notes stating that there were too many questions were collected. This was one of the main concerns about the survey and its values.

With this feedback in mind, the final version of the survey was prepared and will be published for open access in the near future. The changes that will be applied will not be translated back to the Value-Focused Thinking document, since it will be considered a snapshot of the planning process before the first round of feedback. If, otherwise, it is intended to be a live document, changes to the survey should be transcribed back.

6 Conclusion

Strategies to support computer programming education are numerous but still face several and interesting challenges. While many and diverse characteristics have been shown as influential to the teaching-learning process, affinity to a programming language as an influential factor is largely unexplored and might have a positive – or even negative – influence on the whole process.

In order to further understand this topic, a new survey has been constructed, concentrating its focus into realizing which characteristics (personal, technical, contextual etc.) influence affinity between programmers of any level and programming languages. Being a complex and in-depth approach to the continuation of this research, this survey was prepared in a formal manner, using the Value-Focused Thinking method to guide the whole process. This method lead to the definition of the survey's elements based on core values and its derived objectives, creating a highly focused Beta version. The final version of the new survey is now finished – taking into account feedback already gathered – and it is currently open for answers[6]. That version of the survey will be disseminated, in the near future, as much as possible to a broad community of students, teachers and practitioners of programming in order to collect a huge amount of answers; then the collected data will be statistically analyzed and the results will be published.

[5] A copy may be found in the following address: `https://bit.ly/3DDJFtv`.
[6] The survey may be found and fulfilled at the following address: `https://bit.ly/3MdDgIH`.

References

1 M. V. P. Almeida, L. M. Alves, M. J. V. Pereira, and G. A. R. Barbosa. EasyCoding - Methodology to Support Programming Learning. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:8, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2020.1.

2 M.V.P. Almeida. Easycoding: Methodology to support programming learning. Master's thesis, Instituto Politécnico de Bragança, 2020.

3 A.G. Applin. Second language acquisition and cs1. *SIGCSE Bull.*, 33(1):174–178, February 2001. doi:10.1145/366413.364579.

4 D.R. Barbosa. Adequacy Analysis of Learning Resources in Adult Education. Master's thesis, Minho University, Braga, Portugal, October 2021.

5 Council of Europe. *Common European Framework of Reference for Languages: Learning, teaching, assessment – Companion volume.* Council of Europe Publishing, Strasbourg, France, 2020. URL: https://www.coe.int/lang-cefr.

6 R.R. Fenichel, J. Weizenbaum, and J.C. Yochelson. A program to teach programming. *Communications of the ACM*, 13(3):141–146, March 1970. doi:10.1145/362052.362053.

7 J. Figueiredo and F.J. García-Peñalvo. Building skills in introductory programming. In *Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM'18, pages 46–50, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3284179.3284190.

8 P. Freire. *Pedagogia da Autonomia: Saberes necessários à prática educativa.* Paz e Terra, 2011.

9 A. Gomes and A.J. Mendes. Learning to program: difficulties and solutions. In *Proceedings of the 2007 ICEE International Conference on Engineering and Education*, ICEE '07. International Network on Engineering Education and Research, 2007.

10 P.J. Guo. Non-native english speakers learning computer programming: Barriers, desires, and design opportunities. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 1–14, New York, NY, USA, 2018. Association for Computing Machinery.

11 Ralph L. Keeney. Value-focused thinking: Identifying decision opportunities and creating alternatives. *European Journal of Operational Research*, 92(3):537–549, 1996. doi:10.1016/0377-2217(96)00004-5.

12 J. Piaget, M. Piercy, and D.E. Berlyne. *The Psychology of Intelligence.* Routledge classics. Routledge, 2001.

13 Redacted. Redacted for blind review purposes. In *Redacted for blind review purposes*, Redacted.

14 S.A. Robertson and M.P. Lee. The application of second natural language acquisition pedagogy to the teaching of programming languages—a research agenda. *SIGCSE Bulletin*, 27(4):9–12, December 1995. doi:10.1145/216511.216517.

15 Stack Overflow. Stack overflow developer survey, 2021. URL: https://insights.stackoverflow.com/survey/2021.

16 StatisticsTimes.com. Top computer languages, 2020. URL: http://statisticstimes.com/tech/top-computer-languages.php.

17 L.S. Vygotsky, E. Hanfmann, G. Vakar, and A. Kozulin. *Thought and Language.* The MIT Press. MIT Press, 2012.

18 B.C. Wilson and S. Shrock. Contributing to success in an introductory computer science course: A study of twelve factors. In *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '01, pages 184–188, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/364447.364581.

Sprinter: A Didactic Linter for Structured Programming

Francisco Alfredo
Visor.ai Portugal, S.A., Lisbon, Portugal

André L. Santos
ISTAR-IUL, University Institute of Lisbon, Portugal

Nuno Garrido
ISCTE-IUL, IT-IUL, University Institute of Lisbon, Portugal

Abstract

Code linters are tools for detecting improper uses of programming constructs and violations of style issues. Despite that professional linters are available for numerous languages, they are not targeted to introductory programming, given their prescriptive nature that does not take into consideration a didactic viewpoint of learning programming fundamentals. We present Sprinter, a didactic code linter for structured programming supporting Java whose novelty aspects are twofold: (a) providing formative feedback on code with comprehensive explanatory messages (rather then prescriptive); (b) capability of detecting some control-flow issues to a deeper extent than professional linters. We review Sprinter features against popular tools, namely IntelliJ IDEA and Sonarlint.

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

Keywords and phrases structured programming, code quality, code linter

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.2

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]. We thank the anonymous reviewers for their valuable feedback.

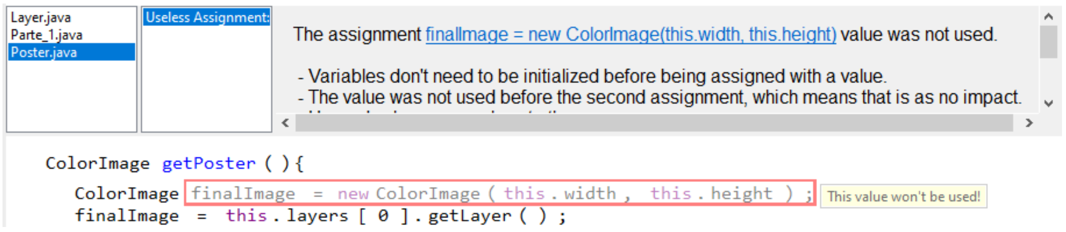
1 Introduction

Code quality has not been widely researched by the programming education community, namely with respect to tooling. Recent studies have identified that quality issues have a significant presence in the code of students [9]. Given the importance of code quality in the software industry, we believe that it is important to bring up the topic early on in the curricula.

From teaching experience, we frequently observe that code quality issues in student code are in part due to their fragile skills with respect to programming constructs. Ruvo et. al. [5] referred to such issues as *semantic style indicators*, and discovered that these are found in the code of students throughout the degree curriculum.

The fact that students develop implementations that work, does not necessarily imply that they fully understand the underlying concepts [8, 14]. Certain aspects of non-optimal student code may have resulted from trial-and-error attempts, combined with unsatisfactory mastery of programming constructs. We speculate that some code quality issues may relate to misconceptions [13], but that investigation is out of the scope of this paper.

In this paper we present Sprinter, a didactic code linter that comprehensively explains code quality issues in terms of what is expressed in the user program, evidencing the problem, rather than the fix (see Figure 1). The perceptions of code quality are not consensual among

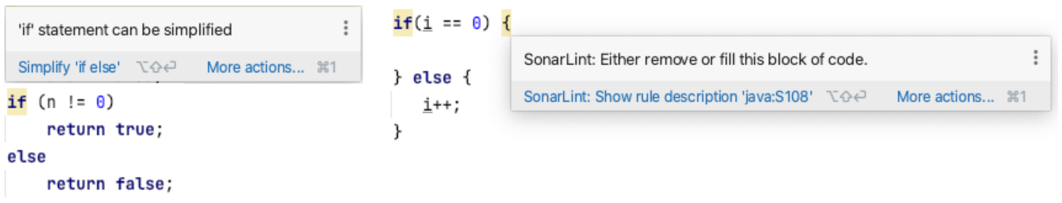


The assignment finalImage = new ColorImage(this.width, this.height) value was not used.

- Variables don't need to be initialized before being assigned with a value.
- The value was not used before the second assignment, which means that is as no impact.

```
ColorImage getPoster ( ){
   ColorImage finalImage  =  new ColorImage ( this . width ,  this . height ) ;   This value won't be used!
   finalImage  =  this . layers [ 0 ] . getLayer ( ) ;
```

Figure 1 Sprinter: student code is analized and annotated with a comprehensive explanation of an encountered issue.


'if' statement can be simplified
Simplify 'if else'
More actions...
SonarLint: Either remove or fill this block of code.
SonarLint: Show rule description 'java:S108'
More actions...

```
if (n != 0)
    return true;
else
    return false;
```

```
if(i == 0) {

} else {
  i++;
}
```

Figure 2 Prescriptive warnings on a code quality issues given by IntelliJ IDEA and Sonarlint.

educators, students, and developers [3]. We focus on quality issues that stem from lack of mastery of elementary programming constructs, as opposed to purely stylistic issues, such as naming and spacing conventions.

Software quality is recommended as a learning goal of Software Engineering curricula [12]. Therefore, didactic tools that raise awareness to code quality issues are an advantage towards this aim. In our contribution we address quality issues within the scope of optimal usage of structured programming constructs, which fundamentally involve sequence, branching, and looping. We address characteristics of programs that in isolation or simultaneously: (a) lead to useless computations, (b) exhibit redundancy in their instructions (duplication), or (c) are unnecessarily verbose. We aim at providing code quality feedback targeting novice programmers, differing from professional linters in terms of message format and improvements in issue detection.

Sprinter is a proof of concept for Java. However, the issues are not Java-specific, and our approach is applicable to languages that have similar structured programming constructs. So far, we have developed 14 detectors for structured programming issues, mostly relying on control-flow analysis. In addition to presenting the issues centered on the problem rather than the fix, our tool detects a few issues that some professional linters are not able to detect. A tool such as Sprinter could be useful in programming courses, given that it can raise awareness of knowledge gaps and deliver hints for students to improve their skills.

2 Related work

Professional code linters may be used in teaching, as they typically provide plugins for widely-used IDEs (e.g., Sonarlint[1]) or are integrated in the IDEs themselves (e.g., IntelliJ[2]). These industrial linters have a *prescriptive* nature in their interaction, since they simply tell *what to do* to fix an issue, typically with an accompanying *quick fix* option for performing the

[1] https://www.sonarlint.org
[2] https://www.jetbrains.com/idea

operation automatically. For didactic purposes, we believe that a tool should explain *why* some aspect of the program is not as good as it could be, elucidating the user as much as possible, while not providing a "blind fix" that can be applied without understanding. Our goal is that students can understand an issue, and ideally, fix it autonomously without the aid of automatic features. Otherwise, one may fix without understanding, which we argue is not ideal for the learning process.

The code linter concept has been applied to early stages of programming learning. LitterBox [6] is a linter for Scratch programs that is capable of providing hints to users based on a catalog of patterns, which are used to analyze the abstract syntax tree. One of the main motivations for this approach is aligned with ours. The fact that learners produce working code with correct outputs does not imply that they are using programming constructs in the best way and have no misconceptions regarding their application.

Keuning et al. [9] have analyzed a large amount of Java code from the BlackBox dataset [4], finding that quality issues have a significant presence in student code. Further, they observed that some quality issues tend to remain as a code file evolves. The same authors have conducted a study with programming teachers [10] in order to investigate the type of quality feedback they would provide to students. They observed that teacher feedback consists of aspects pertaining to reducing algorithmic complexity that is not covered by professional tools, such as simplifications of `if-else` statements. We address several of those issues in Sprinter.

Keuning et al. [11] also investigated if students were able to address quality issues using a refactoring tutor, where instructors develop exercises that start with working code with quality issues, and further provide hints for students to fix, while checking progress when correct steps are taken. Sprinter is similar to such a tutor, but it supports detection of quality issues in arbitrary student code. Hence, students become aware of quality issues by means of their own code, rather than through instructor-designed cases.

FrenchPress [2] is a diagnosing system for Java programs that focuses on issues related to object-orientation and use of variables. Sprinter has only a small overlap with FrenchPress with respect to boolean expressions. Our focus is on structured programming constructs, and nearly to practically all the issues we detect are not handled by FrenchPress. Hence, the diagnoses of both tools could be combined into a broader tool.

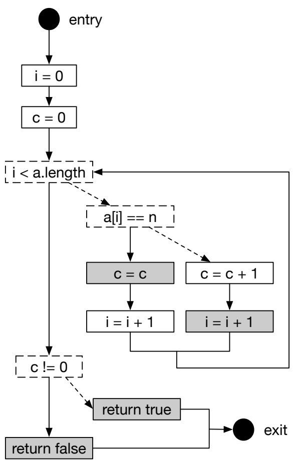
The CompareCFG tool [7] provides students with a control-flow graph (CFG) for their submitted solution, side by side with a CFG of another submitted solution that is less complex. The goal is that students can improve their code autonomously by comparing the solutions and reading additional actionable feedback provided by the tool. This approach is similar to ours, as we also make use of a CFG to detect issues and derive actionable feedback, but we do not present it to students.

Hyperstyle [1] is a tool for automated evaluation of code quality that relies on reusing the functionality provided by professional linters. The messages that explain the issues are not those provided by linters, but rather custom designed by the tool. However, no details are available so far regarding how issues are presented. Hyperstyle supports several languages, but with respect to Java, it relies on the linters Checkstyle[3] and PMD[4]. The latter address mostly syntactic stylistic issues, whereas issues related to control-flow are not detected. In our work we focus on issues related control-flow and structured programming constructs.

[3] `https://checkstyle.sourceforge.io`
[4] `https://pmd.github.io`

```
boolean contains(int[] a,
    int n) {
  int i = 0;
  int c = 0;
  while(i < a.length) {
    if(a[i] == n) {
      c = c + 1;
      i = i + 1;
    }
    else {
      c = c;
      i = i + 1;
    }
  }
  if(c != 0)
    return true;
  else
    return false;
}
```

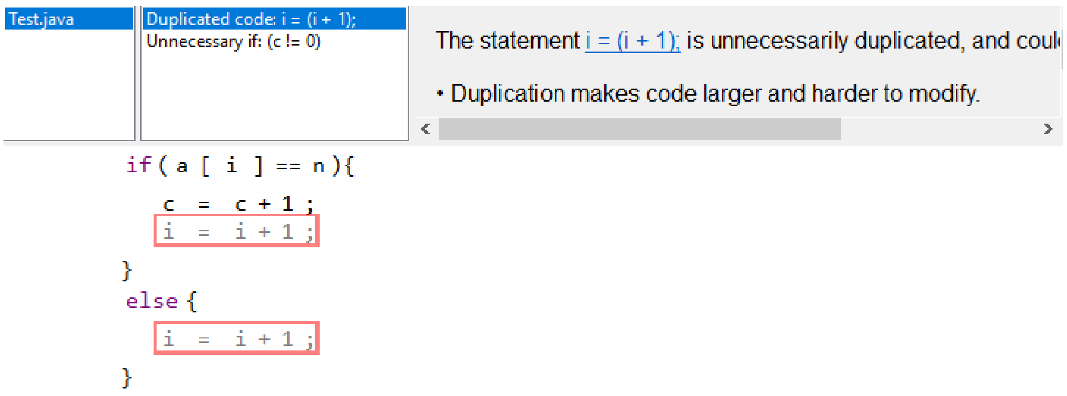
(a) Code example with quality issues.



(b) Control-Flow Graph of (a).

```
boolean contains(int[] a,
    int n) {
  int i = 0;
  int c = 0;
  while(i < a.length) {
    if(a[i] == n)
      c = c + 1;
    i = i + 1;
  }
  return c != 0;
}
```

(c) Version of (a) with improved quality.



Test.java

Duplicated code: i = (i + 1);
Unnecessary if: (c != 0)

The statement i = (i + 1); is unnecessarily duplicated, and coul

• Duplication makes code larger and harder to modify.

```
if(a[i] == n){
   c = c + 1;
   i = i + 1;
}
else {
   i = i + 1;
}
```

(d) Presentation of the quality issues in Sprinter (when `c=c` was already addressed).

Figure 3 Quality issue detection based on control-flow analysis.

3 Control-flow analysis

With the exception of the more trivial cases, most of the issues detected by Sprinter rely on control-flow analysis. We derive a CFG for each procedure (i.e., method, using Java's terminology) of the code under analysis. A CFG models a procedure with an *entry* and *exit* point, statements (nodes), and transitions (edges). Each node is either a statement, or a branching point that determines the following statement according to the evaluation of an expression (control structures). The detection of code issues is performed by querying the CFG for "anomalies".

Figure 3a presents a code example which, despite its contrived appearance, combines several aspects reported in previous studies [5, 9] that any programming education teacher with some experience has seen. For illustration purposes, we combine three code issues in a single example. Notice that in addition to the code issues, algorithmic-wise, the solution is also not optimal (the array iteration does not have to be complete, and no counter is strictly necessary). However, our approach is not concerned with algorithmic strategies, but rather in how those are expressed. Figure 3b presents a CFG that models the code of Figure 3a. The gray instructions are "superfluous" and could be removed through the process of reaching the equivalent solution presented in Figure 3c.

The most trivial issue in the code is the self assignment `c=c`. Given its redundancy, it could be eliminated without any elaborate analysis. Further, by analyzing branches to check that all starting/ending have the same statements (`i=i+1`), we detect that they could be factored out from their branch. As so, the `else` branch becomes empty, and in turn, may be removed as well. Finally, the last `if` is an identity map of the expression `c!=0` to the `return` expression, and hence, it may be simplified to `return c!=0`.

Another aspect that we rely on for detecting code issues is the nature of procedures, namely if they are either *pure functions* without side-effects, or *procedures* that modify state. We determine this through static analysis. When there are no references passed to a procedure, or when all the referenced arrays/objects are not modified nor passed to a procedure, we classify it as pure function. We also distinguish between constant-time and non-constant-time function, by analyzing the function body.

4 Sprinter

We developed the Sprinterprototype supporting Java. Although it is an object-oriented language, in this work we are only focusing on structured programming constructs, which are available across many programming languages with equivalent or similar semantics (e.g., C, Python, Matlab, R). The tool is currently standalone, but it would certainly make sense to integrate it in an IDE in the future (e.g., Eclipse, IntelliJ, VS Code). We also aim at

Sprinter takes as input Java files, which are checked against issue detectors. Each encountered issue is presented to the user in isolation, using annotations in the code (see Figure 3d). We highlight the parts of the code that are involved in the issue and we may attach a small warning sentence about the problem (see Figure 1). In a separate panel, we provide an accompanying explanation of the issue and related concerns. The text may hold links to the code to aid the user in relating the explanation to the code. We do not provide any quick fix options to solve the issues automatically.

So far, we managed to successfully implement detectors for 14 issues, of which we present the ten-most significant in Table 1. We have omitted the more trivial cases such as self-assigning a variable or "tautology if-guards" consisting of the literal `true`. Each case is illustrated with a sketch-example, and we indicate if IntelliJ (version 2022.1.1) and Sonarlint (version 6.4.3) are capable of fully or partially detecting the issue. The former can be considered to be one of the most advanced IDEs for Java, whereas the the latter is a leading professional linter. Notice that in order to have the coverage of issue detection of Sprinter one has to use a combination of two widely used professional linters. Using multiple (professional) tools in educational settings creates undesired overhead.

In addition to the form in which code issues are presented, Sprinter detects some issues that are not flagged by some industry-strength tools. Notice that there are issues not (fully) detected by either IntelliJ or Sonarlint. We are able to achieve this in part because we reduce the scope of code analysis, currently by not delving into Java's library classes. That is, the classification of pure functions is not performed in these cases, as required by some detectors.

The issue of Magic Numbers is prone to some subjectivity. We compromise by flagging cases when the same literal is found twice or more within the same procedure (excluding literals 0, 1, and 2). However, the solution is not perfect, and achieving one is not trivial. A same number may refer to different unrelated things, and that is hard to determine with precision. On the other hand, the often-used 0, 1, and 2, may also be used in situations where a constant would make sense.

Table 1 Code issues detected by Sprinter with illustrative examples. We mark the issues for which there is an equivalent detection available in IntelliJ IDEA (IJ) and Sonarlint (SL). Full support: ●; partial support: ◐; [a] Does not take into account semantically equivalent duplication (see code example); [b] Empty `if` is signaled, but `else` is not take into consideration in the explanations.

Name	Description	Example (Java)	IJ	SL

Useless Assignment	Assignment of a value that is not used.	`int[] array = new int[100];`
		`array = newRandomArray(100);`
			●	●

Useless Call	A call to a function that has no side-effects without making use of the returned value.	`copy(array);`	●	

Useless Return	Return at the end of `void` methods.	`void doSomething() {`
		`  ...`
		`  return;`
		`}`
			●	

Identity Return	Mapping the evaluation to separate return statements.	`if(bool) return true;`
		`else return false;`
			●	●

Redundant Equality	Comparing a boolean expression to a boolean literal.	`if(boolExpression == true) {`
		`  ...`
		`}`
			●	●

Redundant Call	A call to a non-constant pure function with the same arguments.	`int m = max(list);`
		`list.remove(max(list));`		

Redundant Guard	A guard that is checking a condition that is known to be true (given the enclosing control structure).	`while(i > 0) {`
		`  if(i > 0) { ... }`
		`}`
			●	

Duplication in Branches	Common behavior in alternative branches that could be factored out.	`if(v[i] > 0) {`
		`  c++;`
		`  i++;`
		`} else {`
		`  i+=1;`
		`}`
			◐[a]	

Counter Guard Branching	Empty block in `if` with desired behavior in `else`.	`if(guard) {`
		
		`} else {`
		`  // do something`
		`}`
			◐[b]	◐[b]

Magic Number	Numeric literal that is used without an explanation.	`if(c > 255)`
		`  c = 255;`
				●

5 Conclusions

We presented a novel form of explaining code quality issues related to structured programming constructs embodied in the Sprinter tool. Some of these issues are not, or are only partially addressed by professional tools. As future work, we plan to address other issues, such as code duplication at the expression level, as well as other issues related to branching.

A tool such as Sprinter, if properly integrated in the programming practice workflow, could be beneficial to raise awareness to code quality, while helping students to improve their code autonomously. Tool support becomes even more relevant in the context of large-scale course where human tutoring is scarce or not even available. As future work, we plan to carry out a user study with programming beginners to investigate how they can cope with the issues raised by Sprinter.

References

1 Anastasiia Birillo, Ilya Vlasov, Artyom Burylov, Vitalii Selishchev, Artyom Goncharov, Elena Tikhomirova, Nikolay Vyahhi, and Timofey Bryksin. Hyperstyle: A tool for assessing the code quality of solutions to programming assignments. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2022, pages 307–313, New York, NY, USA, 2022. Association for Computing Machinery. `doi:10.1145/3478431.3499294`.

2 Hannah Blau and J. Eliot B. Moss. Frenchpress gives students automated feedback on java program flaws. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, pages 15–20, New York, NY, USA, 2015. Association for Computing Machinery. `doi:10.1145/2729094.2742622`.

3 Jürgen Börstler, Harald Störrle, Daniel Toll, Jelle van Assema, Rodrigo Duran, Sara Hooshangi, Johan Jeuring, Hieke Keuning, Carsten Kleiner, and Bonnie MacKellar. "I know it when I see it" perceptions of code quality: ITiCSE '17 working group report. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*, ITiCSE-WGR '17, pages 70–85, New York, NY, USA, 2018. Association for Computing Machinery. `doi:10.1145/3174781.3174785`.

4 Neil Christopher Charles Brown, Michael Kölling, Davin McCall, and Ian Utting. Blackbox: A large scale repository of novice programmers' activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 223–228, New York, NY, USA, 2014. Association for Computing Machinery. `doi:10.1145/2538862.2538924`.

5 Giuseppe De Ruvo, Ewan Tempero, Andrew Luxton-Reilly, Gerard B. Rowe, and Nasser Giacaman. Understanding semantic style by analysing student code. In *Proceedings of the 20th Australasian Computing Education Conference*, ACE '18, pages 73–82, New York, NY, USA, 2018. Association for Computing Machinery. `doi:10.1145/3160489.3160500`.

6 Gordon Fraser, Ute Heuer, Nina Körber, Florian Obermüller, and Ewald Wasmeier. Litterbox: A linter for scratch programs. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 183–188, 2021. `doi:10.1109/ICSE-SEET52601.2021.00028`.

7 Lucy Jiang, Robert Rewcastle, Paul Denny, and Ewan Tempero. Comparecfg: Providing visual feedback on code quality using control flow graphs. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, pages 493–499, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3341525.3387362`.

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

9 Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. Code quality issues in student programs. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '17, pages 110–115, New York, NY, USA, 2017. Association for Computing Machinery. `doi:10.1145/3059009.3059061`.

10 Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. How teachers would help students to improve their code. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, pages 119–125, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3304221.3319780`.

11 Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. Student refactoring behaviour in a programming tutor. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, Koli Calling '20, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3428029.3428043`.

12 T. C. Lethbridge, R. J. Leblanc Jr, A. E. Kelley Sobel, T. B. Hilburn, and J. L. Diaz-Herrera. Se2004: Recommendations for undergraduate software engineering curricula. *IEEE Software*, 23(6):19–25, 2006. `doi:10.1109/MS.2006.171`.

13 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`.

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

Understanding the Usage of IT-Security Games in the Industry and Its Mapping to Job Profiles

Tilman Dewes
Siemens AG, München, Germany

Tiago Gasiba
Siemens AG, München, Germany

Thomas Schreck
Hochschule für angewandte Wissenschaften München, Germany

Abstract

Due to the increasing dependency on IT systems in both the private and industrial sectors, IT security training is becoming increasingly important. One way to teach IT security topics is through serious games, which besides being fun to play, impart knowledge on certain topics. As these games are more and more used in the industrial environment, this paper aims to develop a mapping between industrial roles and the games to show which game fits how well for the training of an industrial role. In doing so, an evaluation of the games was established that allows for comparability across the different roles. Thus, the research question which serious games is suitable for which industrial role could be addressed. Further results of the work are an ontology, which contains the essential characteristics of serious games for this work, a collection of industrial roles with their required IT-skills and a collection of serious games with an evaluation of the level of support of IT-skills.

2012 ACM Subject Classification Applied computing → Learning management systems; Security and privacy → Software security engineering; Applied computing → Distance learning; Applied computing → E-learning

Keywords and phrases Serious Games, IT-Security, Industrial Roles, Mapping, Ontology

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.3

1 Introduction

Obtaining practical IT security skills takes much effort. To acquire the required level of skills, it often takes "a long journey of discovery, trial and error, and optimization seeking through a broad range of programming activities that learners must perform themselves." [12]. In order to ensure secure systems in today's world, where the dependency of companies on IT systems continues to grow, programmers must be adequately trained. Especially because in the last few years, there has been an increase in IT attacks of all kinds, as the current version of the report on the state of IT security in Germany [2] shows.

One possibility to impart knowledge in the area of IT security offers *Serious Games* [4]. These games with a pedagogical learning background usually impart knowledge in a particular topic through gameplay. As the work of Lui et al. [10] and Švábensk et al. [15] shows, game-based learning offers an effective way of teaching security-related scenarios. However, the field of Serious Games in the IT security is diverse and ranges from the conventional board and card games to Capture the Flag- (CTF) and other Cyber Security Challenges (CSC) [6].

Mainly, these games are produced by many developers, which is why a general disorder of games prevails, also with variations in terms of quality as shown in the work of Caserman et al. [3]. Although the work of Katsantonis et al. [9] shows that frameworks for Serious Games in the field of IT-Security are available, these are only aimed at the development of the games, not at publication or description. Consequently, the games are scattered distributed throughout the world wide web, often with inadequate descriptions and unclear educational goals. As a result, it is often unclear which game is suitable for which situation, especially in an industrial environment. In particular, it is unclear which games can be used to promote the know-how needed for specific industrial roles. Considering all these circumstances, it becomes clear that there is a need for a clearer structure in this area, especially since the games are often not focused on the industry and its requirements (Gasiba et al. [5]).

The contribution of this work is to present a framework for the selection of Serious Games for people in industrial environments. It highlights how well a game is suitable for the training of an industrial role compared to other games. Therefore, a mapping process for existing Serious Games to industrial roles is presented in this work. The methodology of this work is adaptable to non-industrial educational activities and students and pupils. By contributing a proposal on the selection of games when using them for education, we believe that this work contributes to a better structure and clearer arrangement of the topic of Serious Games.

The next section provides an overview of relevant related work on this topic. Section 3 describes the methodology that was used to achieve the goals of this work. Section 4 then states the achieved results of the approach. These are then discussed in section 5, put into context, and consequences drawn from them. In the 6th section, all points of the work are briefly summarized, and an outline of further work is given.

2 Related Work

In order to gather the necessary information, the topic of serious games, in general, was considered first. The paper of Stephen Tang and Martin Hanneghan [14] served as a first introduction to the topic of serious games. It describes a broad ontology that describes serious games in detail with all their characteristics. However, the level of detail is relatively high, and therefore much information is superfluous for the goal of this work. Furthermore, the ontology presented by the authors does not aim at mapping games to roles. Nevertheless, their work serves as a basis for building our ontology. This work was considered relevant, although it is older than 2016, since the information of the paper is up to date and no comparable more recent work could be found regarding ontology in the field of serious games.

In order to determine relevant industrial roles, the company internally published job descriptions were considered. These roles were derived from the Product Solution Security (PSS) Curriculum. The PSS Curriculum is a pictorial representation of the organization within the company to ensure product security. Eight different job profiles were considered relevant through additional expert interviews and internal research. A distinction was made between two types of product solution officers and six different types of product solution experts. The only difference between the two types of officers is the scope of the systems to be supported. The classic Product Solution Security Officer (PSSO) usually operates on a hierarchically lower level and reports to the Principal Product Solution Security Officer (PPSSO). Even though the scope of their tasks is similar, their skillsets differ, as different skills are required at different hierarchical levels. In addition, six different areas were identified in which the Product Solution Security Experts (PSSE) specialize. The following areas of expertise were considered: Expert in Secure Architecture, Implementation, Testing, Manufacturing, Service, and Project Integration. These job profiles contained information about which skills are in demand at what level in which role and served as an essential basis for mapping the Serious Games to them.

Adam Shostack describes himself on his homepage [13] as one of the leading experts in threat modeling and a consultant, expert witness, author, and game designer. He has designed games such as Elevation of Privilege and researched serious games. He presents a collection of physical board and card games that have an IT security learning objective on his website. This collection serves as the basis for the mapping. However, it had to be expanded with information to guarantee a successful mapping. Furthermore, some games were outdated, are no longer sold, or are so similar to other games that they had to be trimmed from the list.

Hill et al. [8] provide a survey of serious games used in cybersecurity education and training. A categorization of the games was carried out into four types based on the topics they cover and the purposes of the games: security awareness, network, and web security, cryptography, and secure software development. The paper then offers a catalog of serious games for different target audiences. However, their work does not consider industrial roles but mainly targets groups in the school environment. Also, no evaluation was performed, but the games were recommended only on the authors' assessment. Furthermore, the categorization for recommendations in the industrial environment is too superficial and does not sufficiently reflect the required skills of the roles.

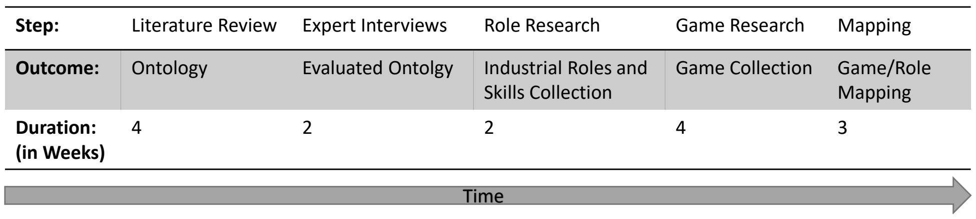
Gasiba et al. [5] investigate the requirements for Serious Games geared towards software developers in the industry, with the focus on CTFs. It was found that although there are a lot of games available, they are mainly developed for pen-testers and white hackers. Software developers receive little attention; hence there is little information about the challenge design requirements in the secure coding area, especially in the industry. This paper was already an essential indication for this work that industrial roles are neglected in the development of serious games.

Hendrix et al. [7] investigate whether Serious Games are suitable for cybersecurity training. The authors examined several serious games in the IT security context. It was found that games are effective cyber security training tools. Nevertheless, some quality deficiencies could be identified. For example, some of the games were not evaluated, or the overall topic of the game was not clearly communicated. In the end, they conclude that "there is a clear gap in target audience with almost all products and studies targeting the general public and very little attention given to IT professionals and managers"[7]. This work clarified that the subject area of serious games in IT security needs to be more clearly structured and better aligned with the industry.

3 Methodology

In order to develop a suitable mapping process of Serious Games to industrial roles, the procedure was divided into five major steps, which are illustrated by the following graphic:

Table 1 Mapping Methodology and Outcome.



Step:	Literature Review	Expert Interviews	Role Research	Game Research	Mapping
Outcome:	Ontology	Evaluated Ontolgy	Industrial Roles and Skills Collection	Game Collection	Game/Role Mapping
Duration: (in Weeks)	4	2	2	4	3

Time

As can be seen, the steps were performed sequentially and produced different results. The results are presented in detail in Chapter 4. In this chapter the methodology is clarified by describing the steps in detail.

3.1 Literature Review

A literature review was conducted as an initial introduction to the topic. Primarily the google scholar database, other search engines like researchgate or springer were used. These were searched for the keywords Serious Games, IT security, ontology, and efficiency. In addition, the work of Adam Shostack [13] was considered, who broadly gives information about Serious Games on his homepage. In the literature review, only works no older than 2016 were consulted, with two exceptions. The work of Stephen Tang and Martin Hanneghan [14], and from N. Noy and Deborah Mcguinness [11] are from the year 2011, and 2001. However, it was considered relevant for this work because the characteristics of Serious Games and ontologies have not changed in time, and this work contributed essential insights, especially for ontology development. The most important sources used are described in more detail in the Chapter 2. The literature review gained insights into how Serious Games can be used in the IT security context and what characterizes Serious Games in general. The results of the literature review are presented in detail in Chapter 4.

3.2 Expert-Interviews

After the literature review, the collected knowledge was evaluated in several expert interviews. A total of five security experts were interviewed. The interviews were conducted in January 2022 with experts from Germany who work in the company's Product Security Lifecycle. In all cases, an interview lasted between 30 and 60 minutes. In this process, the interviewees were first informed about the work's general aim, and then the knowledge gained through the literature review was presented in the form of an ontology. Based on the feedback from the experts, the ontology was shortened in superfluous places and supplemented in missing places. However, not only was the ontology evaluated, but also knowledge about contact points where know-how about internal/industrial roles, their skillset and Serious Games can be found was collected. Thus, the expert interviews were an essential step for the subsequent role research.

3.3 Role Research

The expert discussions described above determined that the focus should be on the Product Solution Security (PSS) Curriculum with its roles and respective skillset. The PSS curriculum represents and details the company's organization established to ensure product security. It maps job roles with their associated training. For the roles mentioned therein, detailed job profiles could be found, describing the roles with their functions and their required skills. Each skill has been assigned a skill level between Basic, Advanced, Expert, or none. Since the role descriptions were too confusing for quick comparison and evaluation, the relevant information was filtered out and transferred to a spreadsheet. The different skill levels were assigned numbers representing the skill level instead of the written word (Basic =1, Advanced =2, Expert =3). These numbers allow for comparing different roles and more easily present the differences and commonalities between them. A total of 40 different skills in four different skill categories were identified with the previously mentioned skill levels.

3.4 Game Research

After roles and skills were defined, the next step was to create a detailed collection of Serious Games that target the area of IT security. The collection of Adam Shostack [13] described in the related work chapter was primarily used for this purpose. Since this collection offers only rather superficial descriptions, it had to be supplemented with some information. In addition, a few of the games were no longer available, so they had to be removed from the collection. In order to obtain all the necessary information, all the games were inspected individually, and the relevant data was again recorded in the form of an spreadsheet, with the following information: Game name, overall topic, costs, duration, number of players, availability, further link, and a short game description.

3.5 Skillset Mapping

For the mapping process, the first step was to shorten the skillset of the industrial roles to those skills that do not have an IT security background. Thus, only eleven of the 40 skills were considered relevant to IT security. Then, an assessment was performed for each of the games: Each game got a rating between the skill level 1 (Basic), 2 (Advanced), 3 (Expert), or - (none) on the previously defined skill, depending on how strongly the game promotes the respective IT security skill. This resulted in an overview of the games which showed which IT security skill they promote at which level (see Table 4).
In the second step, an assessment was made on how well a game fits a role and its skillset. In each case, the distance (d) between the skill level of the role and the skill level of the game was considered. The distance was calculated according to the following formula:

$$d = (Skill\,level\,Role - Skill\,level\,Game) \tag{1}$$

Points were then awarded for each of the eleven skills according to the following scheme:

Table 2 Scoring in the Mapping Process.

d	-2	-1	0	1	2
points	-50	-25	100	50	25

As can be seen, a maximum distance between -2 and 2 could be reached (Skill level Game 1 $-$ Skill level Role 3 or Skill level Game 3 $-$ Skill level Game 1). For a negative distance, either 25 or 50 negative points were assigned; the higher the distance in the negative range, the higher the negative points. This calculation is based on the fact that if a game promotes a higher skill level on a certain skill than is required in a role, it is not suitable for this role. This is especially true if the game targets a topic not required for an expert role in another area, for example. On the other hand, if the skill level of the role is higher than that of the game, positive points are still awarded. Because even if the required skill level of the role is higher, the required skill is still promoted to a certain extent. The highest points were awarded when the distance was zero. In this case, the game maps the respective skill exactly to the role level and fits the role accordingly. All points awarded for each of the eleven skills were added up, so the maximum score was 1100 points (11×100 points). This score ultimately shows how well the game fits the industrial role. Through Excel, an automated evaluation in the form of formulas was possible. The results of the mapping are listed in the next chapter.

4 Results

After the steps of Table 1 were explained in the last chapter, the outcome of the steps will be described in more detail in the following.

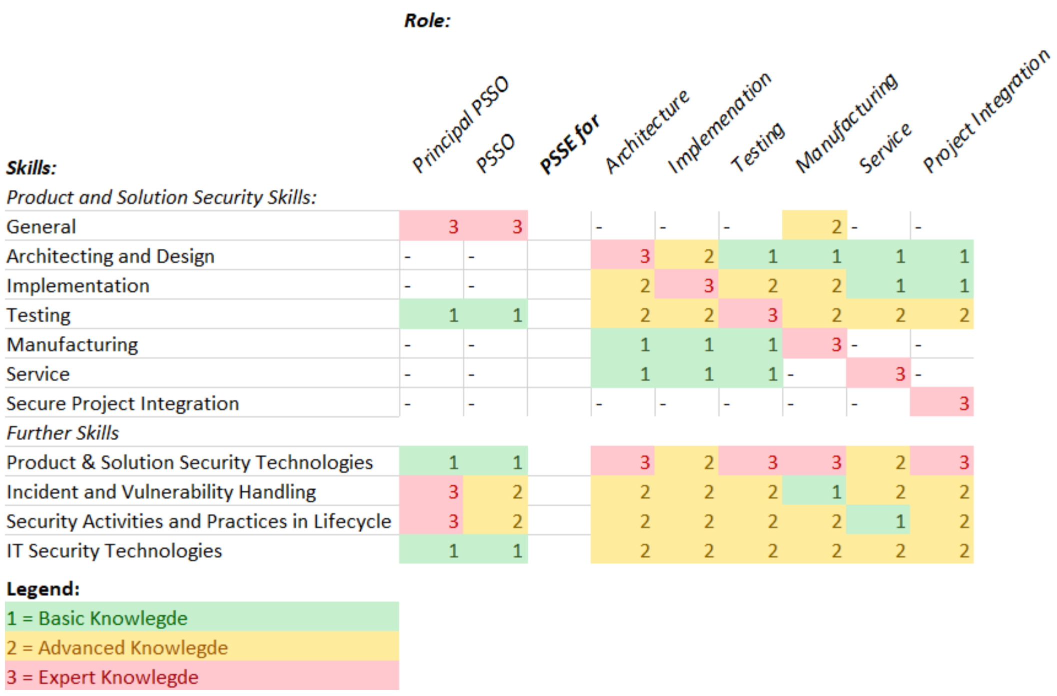
4.1 Ontology

The general knowledge about Serious Games gathered through the literature review and the expert interviews was captured in the form of an ontology. It is provided in the appendix. The upper left part of the ontology refers to Serious Games. Explaining the characteristics contained therein would go beyond the scope of this paper, which is why only the aspects relevant to this paper will be discussed here. It can be seen that an essential part of Serious Games is the game player. He/she has personal characteristics determined by his role and the resulting skills with their skill level. The second part important for this work is the pedagogic learning factor. This has a specific goal and a specific topic. This topic can be IT security. This work aims to match the IT security topics of the Serious Game with the IT security skills of the game player. In the ontology, the matching is represented by the big arrow.

4.2 Industrial Role and IT security Skill Collection

The second achievement of the work was to get a collection of industrial roles with their IT security skillset.

Table 3 Role Collection with Skillset Assessment.



	Role:								
Skills:	Principal PSSO	PSSO	**PSSE for**	Architecture	Implementation	Testing	Manufacturing	Service	Project Integration
Product and Solution Security Skills:									
General	3	3		-	-	-	2	-	-
Architecting and Design	-	-		3	2	1	1	1	1
Implementation	-	-		2	3	2	2	1	1
Testing	1	1		2	2	3	2	2	2
Manufacturing	-	-		1	1	1	3	-	-
Service	-	-		1	1	1	-	3	-
Secure Project Integration	-	-		-	-	-	-	-	3
Further Skills									
Product & Solution Security Technologies	1	1		3	2	3	3	2	3
Incident and Vulnerability Handling	3	2		2	2	2	1	2	2
Security Activities and Practices in Lifecycle	3	2		2	2	2	2	1	2
IT Security Technologies	1	1		2	2	2	2	2	2

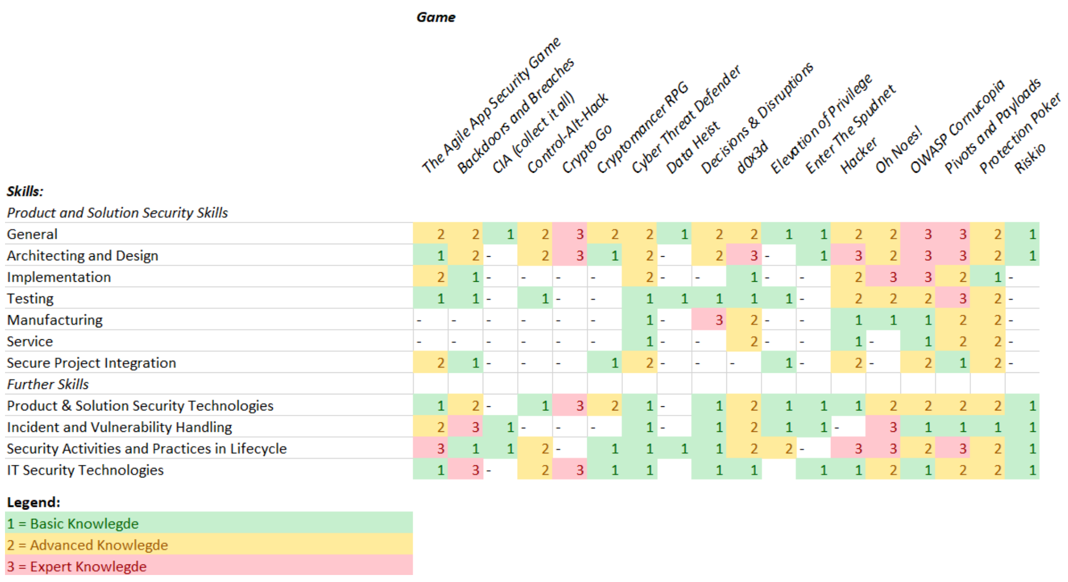
Legend:
- 🟩 1 = Basic Knowlegde
- 🟨 2 = Advanced Knowlegde
- 🟥 3 = Expert Knowlegde

As can be seen, eleven IT security skills were defined. Each of the eight different industrial roles was assigned a skill level between 1 (Basic), 2 (Advanced) , 3 (Expert), or – (none). This table was essential for the subsequent mapping of the games to the roles.

4.3 Game Collection

Another partial result of the work was a collection of 18 board and card games with IT security references. The collection contains information about game-name, overall topic, costs, duration, number of players, availability, further links, and a short game description. In addition, all games in this collection contain a rating between –, 1, 2, and 3 on each of the skills mentioned in Chapter 4.2, depending on how strongly the game promotes the respective skill.

Table 4 Game Collection with Skillset Assessment.



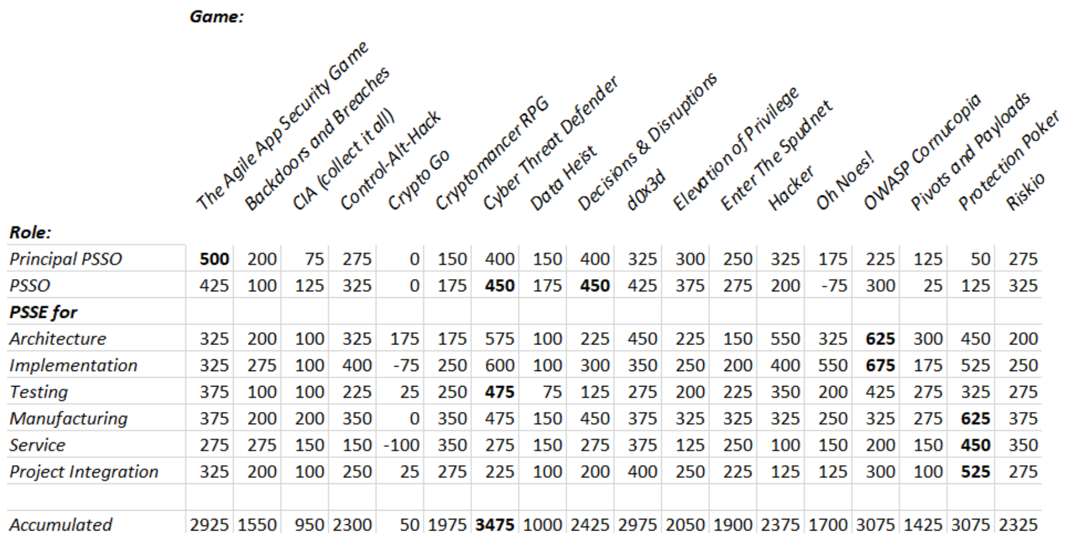
Skills:	The Agile App Security Game	Backdoors and Breaches	CIA (collect it all)	Control-Alt-Hack	Crypto Go	Cryptomancer RPG	Cyber Threat Defender	Data Heist	Decisions & Disruptions	d0x3d	Elevation of Privilege	Enter The Spudnet	Hacker	Oh Noes!	OWASP Cornucopia	Pivots and Payloads	Protection Poker	Riskio
Product and Solution Security Skills																		
General	2	2	1	2	3	2	2	1	2	2	1	1	2	2	3	3	2	1
Architecting and Design	1	2	-	2	3	1	2	-	2	3	-	1	3	2	3	3	2	1
Implementation	2	1	-	-	-	-	2	-	-	1	-	-	2	3	3	2	1	-
Testing	1	1	-	1	-	-	1	1	1	1	1	-	2	2	2	3	2	-
Manufacturing	-	-	-	-	-	-	1	-	3	2	-	-	1	1	1	2	2	-
Service	-	-	-	-	-	-	1	-	-	2	-	-	1	-	1	2	2	-
Secure Project Integration	2	1	-	-	-	1	2	-	-	-	1	-	2	-	2	1	2	-
Further Skills																		
Product & Solution Security Technologies	1	2	-	1	3	2	1	-	1	2	1	1	1	2	2	2	2	1
Incident and Vulnerability Handling	2	3	1	-	-	-	1	-	1	2	1	1	-	3	1	1	1	1
Security Activities and Practices in Lifecycle	3	1	1	2	-	1	1	1	1	2	2	-	3	3	2	3	2	1
IT Security Technologies	1	3	-	2	3	1	1		1	1		1	1	2	1	2	2	1

Legend:
1 = Basic Knowlegde
2 = Advanced Knowlegde
3 = Expert Knowlegde

4.4 Game/Role Mapping

The main result of the work is a mapping between the previously described Game Collection and the Role Collection. The data mentioned in Tables 3 and 4 were used to evaluate through the procedure mentioned in Chapter 3. As a result, a table was created to record how well a game fits the respective industrial role. The higher the score, the better the game supports the required skills mentioned in the job profiles. As shown in Table 5, scores between 675 and −100 were achieved. The games in bold show which games achieve the highest score in each role and thus best match it. The accumulated value is the added value of the games across the roles. From this, it can be derived how well a game fits the industrial roles in general. The highest value was achieved by Cyber Threat Defender, followed by OWASP Cornucopia and Protection Poker. So these are particularly well suited to training in the industrial sector. The game Crypto Go is the least suitable, with a score of only 50, mainly because it hardly promotes skills in demand, as shown in Table 4. The highest score for a specific role was achieved by the game OWASP Cornucopia in combination with the PSSE for Implementation. Besides this, only the game Protection Poker and Cyber Threat Defender on the roles PSSE for Implementation and PSSE for Manufacturing could collect points of 600 or more. The Agile App Security Game, Decisions and Disruptions, d0x3d, Hacker, and Oh Noes! were also able to collect a score of over 400 in certain roles and thus achieve comparatively high values. The table also shows that the score achieved in a game can vary

Table 5 Results of Game Role Mapping.



Role: \ Game:	The Agile App Security Game	Backdoors and Breaches	CIA (collect it all)	Control-Alt-Hack	Crypto Go	Cryptomancer RPG	Cyber Threat Defender	Data Heist	Decisions & Disruptions	d0x3d	Elevation of Privilege	Enter The Spudnet	Hacker	Oh Noes!	OWASP Cornucopia	Pivots and Payloads	Protection Poker	Riskio
Principal PSSO	**500**	200	75	275	0	150	400	150	400	325	300	250	325	175	225	125	50	275
PSSO	425	100	125	325	0	175	**450**	175	**450**	425	375	275	200	-75	300	25	125	325
PSSE for																		
Architecture	325	200	100	325	175	175	575	100	225	450	225	150	550	325	**625**	300	450	200
Implementation	325	275	100	400	-75	250	600	100	300	350	250	200	400	550	**675**	175	525	250
Testing	375	100	100	225	25	250	**475**	75	125	275	200	225	350	200	425	275	325	275
Manufacturing	375	200	200	350	0	350	475	150	450	375	325	325	325	250	325	275	**625**	375
Service	275	275	150	150	-100	350	275	150	275	375	125	250	100	150	200	150	**450**	350
Project Integration	325	200	100	250	25	275	225	100	200	400	250	225	125	125	300	100	**525**	275
Accumulated	2925	1550	950	2300	50	1975	**3475**	1000	2425	2975	2050	1900	2375	1700	3075	1425	3075	2325

greatly from role to role, for example in the game Oh Noes! where values between −75 and 550 were achieved. This shows that the different skillsets of the role have a substantial impact on the rating. Other games, such as Crypto go or Data Heist, failed to score 200 or more on any reel. This shows that these games are unsuitable for use in the industrial sector. This table shows how well a game fits one of the defined industrial roles, and the Accumulated Value shows how well it is generally applicable in the industrial environment.

5 Discussion

The previously described results are discussed and put into context in this chapter. For all of the results, it should be noted that the majority knowledge come from internal company sources are therefore limited.

5.1 Discussion on Literature Review

In the course of the review, it became apparent that a large number of literature on the topic of Serious Games is available, including literature on IT security, which demonstrates a high degree of research in this area. Nevertheless, the quality of the games is mostly not on the expected level. This does not mean the creativity or the structure of the games, but rather that the learning objective and the required and promoted skills are not clearly recognizable. Even if new works like from Zhao et al. [16] or Beckers and Pape [1] show that there is a stronger focus on suitability in the industrial environment, the majority of available games neglect this factor. It would be desirable to establish a procedure in which the developers of the games define the goal of the games and the associated skills. These could then be easily understood with the overall topic as a unified game description. Furthermore, it was noticeable that there are hardly any games specifically designed for use in the industrial sector for professionals in the cybersecurity workforce. Although some of the games covered knowledge areas that are important for specific roles, no game specifically designed for one of the roles could be found.

5.2 Discussion on Ontology

In the case of ontology, it is important to note that it has been adapted to the purpose of the work. Especially subcategories of aspects of Serious Games that do not contribute to the mapping or are essential to the understanding of Serious Games have been shortened. For example, there are different types of rules, such as Interaction Rules or Scoring Rules, that determine the events within the game. Also, the game objects were described in much more detail in the initial ontology by attributes such as Vital, Position, or Solidity State, but these did not contribute to the mapping or the general basic understanding of Serious Game. However, changes and additions were also made through the expert discussions and our reflections. Two experts mentioned that it would also be important to map the game's internal role, game master or player, in the ontology. The division into game-specific and personal characteristics were based on the experts' considerations. This way, a good differentiation could occur because the game-specific characteristics serve mainly the basic understanding for Serious Games while the personal characteristics were essential for mapping the games. Also, the wording was adapted in some places to provide a better understanding by the players; for example, the Pedagogic Event Indicator became a pedagogic learning factor, or the Game Scenario became a Level. Overall, the ontology could make the purpose of this work more understandable and can be used for research on similar topics, but it could be that for another goal, the ontology contains too little information, or the added information is superfluous. The ontology aims to support research and help understand what the industrial requirements are for games. Thus, it can support the preparation and selection of games.

5.3 Discussion on the Mapping Process

The mapping process must also be considered from a certain point of view. Although the games were examined in as much detail as possible, the evaluation of the games in their respective skill levels was still done from a subjective point of view. In addition, there was simply no time to play each game from start to finish, which is why there could be reasons for distortions in the evaluation. In order to ensure the most accurate rating possible, an evaluation of the required skills directly by the developer or publisher would be desirable. For this purpose, a framework could be developed. For example, the developer or publisher of the game could choose from a pool of skills how strongly they are promoted. This could easily create comparability of the games, and mapping to industrial roles would be much easier, and it would generally contribute to standardization in the field of Serious Games.

Nevertheless, the mapping process created here can also be used for roles and games not covered in this paper. When adding new games, you only have to evaluate how strongly the game promotes the eleven defined skills, according to the principle outlined in the chapter methodology. The same applies to adding new roles, as long as they contain the same IT security skills.

A role with new skills would also be conceivable. But then the new skills would have to be evaluated for the PSSO and PSSE roles as well, or they would have to be deleted, and only the new roles are considered. Practically speaking, one could define what skills children of a certain grade level need to have and thus select the most appropriate game to complement traditional teaching methods.

A finer detailing of the skill levels is also discussable. In this work, a distinction is only made between the Basic, Advanced, Expert, and none levels. This can be extended as desired. Then a new evaluation of all games and roles must take place; thereby, a more exact mapping evaluation could be done. However, the amount of work must be considered because this should be in proportion to the benefits.

6 Conclusion

In summary, through research, feedback from experts, and an evaluation system, it was possible to determine which Serious Games are relevant for IT security roles in the industry. In this paper, two different roles for general Product Security (PSSO) and six different types of Expert roles (PSSE) in the industry were considered relevant. A collection of board and card games by Adam Shostack was used to conduct a mapping for these roles. Some useful information was added to this collection, and a few outdated or no longer available games were removed from the list. The mapping result was that the games were given a score that shows how well a game fits an IT security role. The mapping assessment considered how much a game promoted the required skills of one of the PSSO or Expert roles. Eleven IT security skills were taken into account. Thereby comparability could be established based on how well a game represents the required IT security skills of the respective role. The basis for this scoring was that each game received a rating between none, Basic, Advanced, and Expert, which showed how a particular skill was promoted in one of the games. The roles also received a rating, as described, considering what level of skill is required in that role. A score could then be derived based on a rating system, which describes how well a game fits a role.

During the process, an ontology was developed, which only contains the essential aspects of Serious Games and mainly aims at developing the mapping process. The level of detail of the general characteristics of Serious Games is not very high, but the ontology shows which aspects can be used to map the games to the personal characteristics determined by the industrial role and the resulting skills. This knowledge can be used to apply the mapping process to other circumstances, such as companies, schools, or universities with other roles with other skillsets or to the general audience. Also, the created collection of games with skills ratings can be used for other mapping processes, either by adding new games to the games collection with a skills rating or by adding new roles with a rating of the eleven skills.

We will transfer the obtained results into a database for easy querying for future work. The results can also be presented in a graphical user interface. It would be conceivable to expand the interface with additional information about Serious Games so that a platform is established that serves as a central point of contact for Serious Games in the context of IT security. Furthermore, a recommendation process could be created here, which concludes the interests and skills of the user based on specific questions. Based on the answers, extra points can be given to the games, and the ones with the highest score will be recommended. In conclusion a process could be developed to run a mapping between industrial roles and IT security Serious Games. As a result, different games are now available with an evaluation that describes how much the required skills of the industrial roles are mapped. The result of the work also shows that there is an unused potential for Serious Games in IT security. First, the games' content and goals can be better adapted to the industrial roles and skills, and second, the content and the goal could be communicated more transparently, for example, by a standardized description of which skills the respective game promotes.

References

1 Kristian Beckers and Sebastian Pape. A Serious Game for Eliciting Social Engineering Security Requirements. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 16–25, 2016. doi:10.1109/RE.2016.39.

2 Bundesamt für Sicherheit in der Informationstechnik. *Die Lage der IT-Sicherheit in Deutschland 2021*. BSI, 2021.

3 Polona Caserman, Katrin Hoffmann, Philipp Müller, Marcel Schaub, Katharina Straßburg, Josef Wiemeyer, Regina Bruder, and Stefan Göbel. Quality criteria for serious games: Serious part, game part, and balance. *JMIR Serious Games*, 8(3):e19037, July 2020. `doi:10.2196/19037`.

4 Ralf Dörner, Stefan Göbel, Wolfgang Effelsberg, and Josef Wiemeyer. *Serious Games: Foundations, Concepts and Practice*. Springer International Publishing, 1. Ed, Switzerland, 2016. `doi:10.1007/978-3-319-40612-1`.

5 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`.

6 Tiago Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. CyberSecurity Challenges for Software Developer Awareness Training in Industrial Environments. In *International Conference on Wirtschaftsinformatik*, pages 370–387. Springer, 2021.

7 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, March 2016. `doi:10.17083/ijsg.v3i1.107`.

8 Hill Jr, Mesafint Fanuel, Xiaohong Yuan, Jinghua Zhang, and Sajad Sajad. A Survey of Serious Games for Cybersecurity Education and Training. *KSU Conference on Cybersecurity Education, Research and Practice*, October 2020.

9 Menelaos Katsantonis, Isabella Kotini, Panayotis Fouliras, and Ioannis Mavridis. Conceptual Framework for Developing Cyber Security Serious Games. In *2019 IEEE Global Engineering Education Conference (EDUCON)*, pages 872–881, April 2019. `doi:10.1109/EDUCON.2019.8725061`.

10 Lin Liu, Affan Yasin Chouhan, Tong Li, Rubia Fatima, and Jianmin Wang. Improving Software Security Awareness Using A Serious Game. *IET Software*, 13, July 2018. `doi:10.1049/iet-sen.2018.5095`.

11 Natalia. Noy and Deborah Mcguinness. Ontology Development 101: A Guide to Creating Your First Ontology. *Knowledge Systems Laboratory*, 32, January 2001.

12 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.*, January 2022. Just Accepted. `doi:10.1145/3513140`.

13 Adam Shostack. Threat Modeling Expertise, Training, Coaching, July 2022. URL: `https://shostack.org/`.

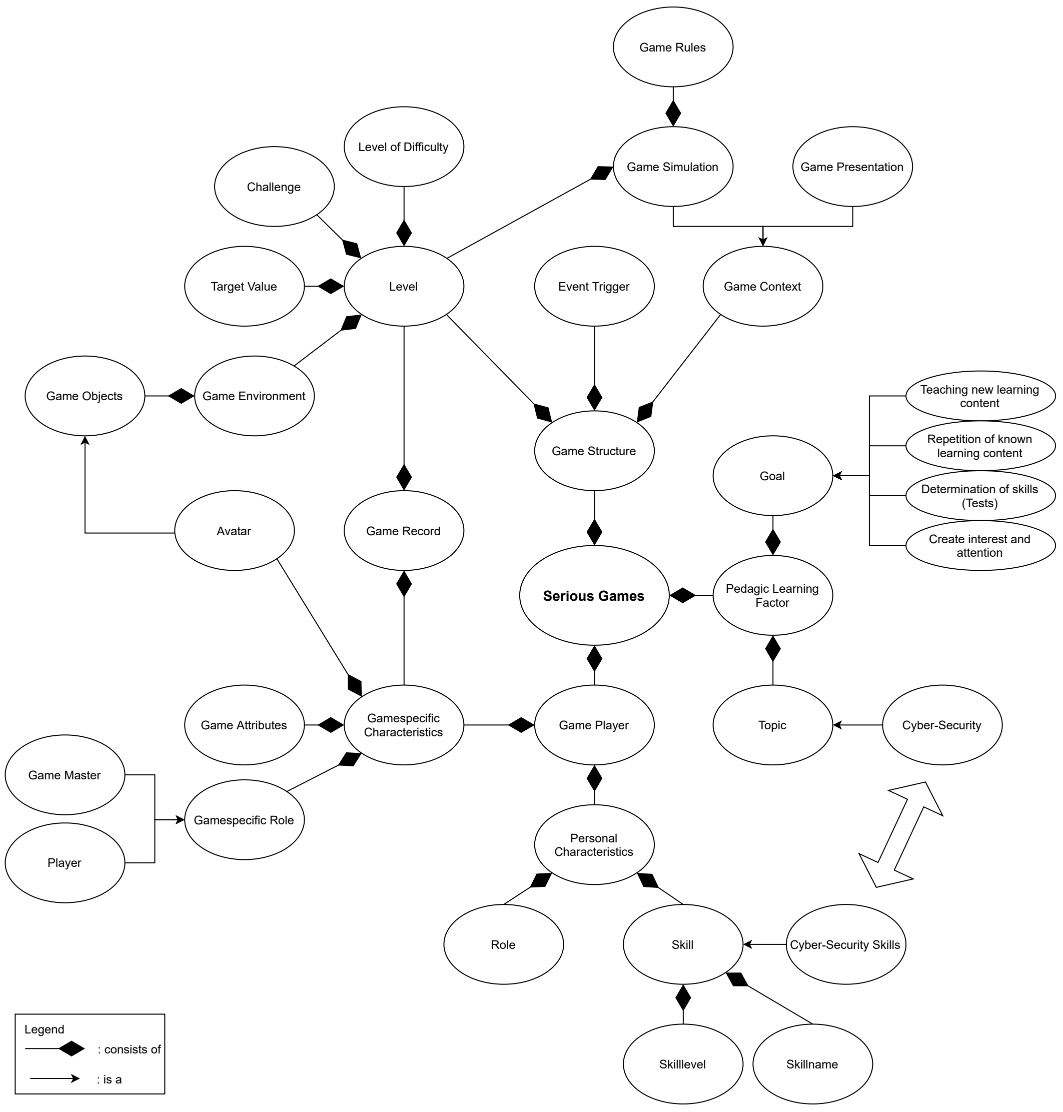
14 Stephen Tang and Martin Hanneghan. Game Content Model: An Ontology for Documenting Serious Game Design. In *2011 Developments in E-systems Engineering*, pages 431–436, 2011. `doi:10.1109/DeSE.2011.68`.

15 Valdemar Švábenský, Jan Vykopal, Milan Cermak, and Martin Laštovička. Enhancing Cybersecurity Skills by Creating Serious Games. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018, pages 194–199, New York, NY, USA, 2018. Association for Computing Machinery. `doi:10.1145/3197091.3197123`.

16 Tiange Zhao, Tiago Espinha Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. Exploring a Board Game to Improve Cloud Security Training in Industry (Short Paper). In *ICPEC*, 2021.

A Appendix

Table 6 Ontology of Serious Games regarding Mapping to Industrial Roles.



Introductory Programming in Higher Education: A Systematic Literature Review

Gabryella Rodrigues
Research Centre on Education (CIEd), Institute of Education, University of Minho, Braga, Portugal

Ana Francisca Monteiro
Research Centre on Education (CIEd), Institute of Education, University of Minho, Braga, Portugal

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

Abstract

A systematic literature review was performed on 33 papers obtained from the ACM, IEEE and Sciencedirect databases, in order to understand in depth, the introductory programming discipline (CS1) in higher education. Recently published works have been covered, providing an overview of the teaching-learning process of introductory programming and enabling to find out whether the research developed by universities worldwide is in line with the proposals made by ACM/IEEE-CS group for computer courses, regarding the transition to the competency-based model. The results show that the new techniques/technologies currently used in software development, as an example of agile methodology, has influenced the teaching-learning process of CS1 together with methods such as visual programming and e-learning. The analyzed papers discuss the importance of developing not only technical, but also social skills, corroborating that methodologies used in introductory programming courses need to focus on preparing students for an increasingly competitive market, associating new skills with technical aspects.

2012 ACM Subject Classification General and reference → Surveys and overviews

Keywords and phrases systematic literature review, CS1, introductory programming, teaching programming, learning programming

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.4

Funding This work is supported by CIEd – Research Centre on Education, Institute of Education, University of Minho, and funded by FCT – Foundation for Science and Technology, scholarship nº2021.07850.BD.

1 Introduction

The Association for Computing Machinery (ACM) guides and recommends higher education institutions worldwide in the context of analyzing the characteristics of graduation students and in the construction of curricula in the field of computing, since 1960. The most recent document – published in partnership with the Computer Society of the Institute of Electrical and Electronics Engineers (IEEE-CS) – the Computing Curricula 2020: Paradigms for Global Computing Education [19], referred to as CC2020, records significant updates.

The main change from previous documents focuses on the transition from a traditional teaching model to a competency-based model. The first one is described through areas of knowledge, units of knowledge and learning outcomes. However, this paradigm has been shown to be inefficient through two new challenges: the new ways of acquiring knowledge and the gap between graduates' skills and the skills expected in professional activities, known as "the skill gap" [19].

Therefore, with the aim of promoting the success of learning and the effective reduction of the skill gap, the ACM/IEEE group made use of previous experiences and incorporated to CC2020 the concept of competence as a primary characteristic in the construction of the computer science curriculum. The curriculum guidelines for undergraduate degree programs in Information Technology [45] identified as IT2017 was the initial inspiration.

This research focuses on the introductory programming discipline, specifically in algorithms, which according to CC2013 [44] are fundamental to the development of any software system and present in all areas of computing. The study of algorithms provides an insight into the nature of the problem as well as possible solution techniques, independently of a programming language and programming paradigm, computer hardware or any other aspect of implementation.

A systematic literature review was conducted in June 2021 in order to systematically collect recent data available in scientific databases on: the course identity, expected skills, methodologies, programming languages and tools used in teaching and learning of introductory programming.

2 Previous Studies

Most of the recent systematic literature review on introductory programming focus on a specific teaching/learning method [32, 35, 53, 36] or on the assessment of tools, programming languages or programming paradigms [27, 24, 3]. In general, studies that address the teaching and learning programming on novice, do not cover results after 2018 [5, 37, 40] and centers only on dropout, failure rates, problems faced by students and technical skills expected before and at the end of the course.

Despite the large volume of work on introductory programming, there is a lack of studies that focus on competency instead of knowledge expectations [45]. To gain a different perspective in this review, we aiming in studies that also include a competence model.

3 Research Questions

Firstly, given the recommendation to parameterize the course´s terminologies and classifications and the areas of knowledge proposed in CC2020, this research seeks to identify how the introductory programming discipline is identified and structured and in which computing degree program is targeted (Q1). In addition to pointing out, or not, a pattern in the discipline´s identity, this research seeks to register what contents are covered, in order to recognize the presence of the algorithm subject. It also seeks to examine which undergraduate courses in computing place more emphasis on the development of algorithms, comparing these results with the panorama suggested in CC2020, in which it is quantified the importance that each area of knowledge offers to courses.

Secondly, driven by the transition to the CC2020 competency model, this research investigated whether universities are still based on the classical teaching model or are already concerned about the transition to the new model proposed and what skill set are classified as essential for computer science students taking the introductory programming course and whether these skills are explored within a real-world context. For that it is important to note which technical and social skills are involved in the process of creating and analyzing an algorithm (Q2).

After the analysis of the included texts, a list of methodologies, tools, languages and programming paradigms used in the teaching and learning of introductory programming course was compiled, in order to analyze the influence of the technology in this process and observe how close are the tools used in universities of those required by job market, identified through the question: What are the methodologies, tools, languages and programming paradigms used in the teaching and learning process of introductory programming (Q3)?

The results of this work will define which terminologies will be used to reference this course in future work; the level of importance that the discipline has in each computer course will be used to define which curriculum recommendations documents (currently there are seven different reports regulating computer courses) will be used as a basis in the construction of a conceptual framework of a doctoral research project in the context of learning the introductory programming course in higher education. In addition to providing information to justify the choice of the courses that will be observed in the field work.

The data related to skills, technical skills, social skills and methodologies described in the collected documents will be used as a basis for the construction of a observation guide in a case study that will be developed in the future.

4 Method

The systematic literature review seeks to employ a research methodology with scientific rigor and great transparency. To ensure such rigor and transparency Elisa Nakagawa [42] affirm that all systematic review must contain a protocol of investigation, which describes the entire process in detail. This study follows the guidelines for systematic literature review presented by [29] and [42] and consists of the following stages: formulation of the research questions to be answered; strategies adopted in the search and selection process of the studies that will be included in the review; the procedure for data extraction and classification and finally the data synthesis strategies and analysis of results.

4.1 Search Strategy

To define the string used in this systematic review, the research questions were broken down into keywords and synonyms were searched for each term [29]. For the first research question (Q1), in which the objective is to reveal the identity of the discipline, we opted to use a manual search (test search) in computing databases with the terminology CS1.

The CS1 (Computer Science 1) acronym was originally created in 1978 by the ACM and refers to the first computing course that introduces the programming basics [18]. Although the contents have changed over the past four decades, the name and general principles have remained [23].

The result of this preliminary search showed us that the researches carried out on this field also make references to the keywords: "Introductory Programming Course" and "Introduction to Programming".

Similarly to the second research question (Q2), the word "skill" was chosen to identify which skills are described as necessary to students who are attending and/or have attended the fundamentals of programming. The choice of not distinguishing social skills from technical skills in this process was based on the results obtained in this preliminary exercise, which showed that with the exclusive use of the word Skill researches involving both capabilities were returned.

For the third research question (Q3), the chosen words that proved to be timely in identifying the methodologies, tools, languages and programming paradigms used by teachers and students in the discipline were: method and tools. When the word "method" was associated with "CS1" it returned works that involve the use of new teaching methods in the teaching of introductory programming. Using the keyword "tools" combined with "CS1", we obtained papers that cited the use of tools and programming languages. For the last item of Q3: programming paradigms, it was not necessary to use a specific keyword, because programming paradigms can be revealed from the identification of the programming language used, as [11] explains.

Since this work seeks to research both teaching and learning processes of introductory programming, the identified keywords were associated to the terms: “teaching programming” and “learning programming” and the search strategy, described in Table 01, was built.

The use of a wide search string, provided its use in several databases [29] and its validation was made by an expert in the area, who reproduced the final protocol.

4.2 Database

After the process described above, an automatic search was performed in Sciencedirect, IEEE Xplore and ACM Digital Library, using the previously constructed research strategy. The first two sources, according to the classification of Felizardo et al. [16] are identified as bibliographic databases, I.e only return studies published by their own publisher. And in order to mitigate possible limitations in the searches, we also used a hybrid database: ACM Digital Library, which indexes studies published by the publisher and studies from other sources [16].

In addition to the characteristics mentioned above, the ACM and IEEE associations, which respectively maintain the ACM Digital Library and IEEE Xplore database, are responsible for producing and publishing all the reports that guide computer courses. Also, they are classified as two of the main indexing services in the area of computing, electricity and electronics [9].

In order to collect all the evidence used to answer the research questions, we have made use of a third source: Sciencedirect. A multidisciplinary database made available by Elsevier, which brought together papers published by researchers from various countries and a collection covering several thematic areas in the field of Science and Technology.

Another point to consider about the databases chosen was the fact that they index new studies regularly and with peer-reviewed. They have a search engine that allow an easy adaptation of the string, versatility in exporting the results and integration with a reference management software.

4.3 Selection Criteria

In order to include relevant documents and ensure that no important study was excluded, selection criteria were defined (inclusion and exclusion) in the search protocol before starting the automatic searching process in ScienceDirect, IEEE Xplore and ACM Digital Library databases.

4.3.1 Inclusion Criteria

The condition of having been published in the period from June 2017 to June 2021 (five-year period), was the inclusion criteria used to select documents in this systematic review, allowing the presentation of a current scenario of what has been investigated on “introductory programming”. In particular, it can provide insights into what new technologies are influencing the teaching and learning process in this discipline.

At the same time, the first document of the ACM/IEEE that embraced the concept of competencies as the main characteristic of a computing curriculum was the Information Technology (IT2017) report [45], published in 2017. This led to the adherence of CC2020 to the competence-based learning model.

Although the mentioned document was published only in December 2017, this review moved the observation window back to June 2017 with the intention to identify whether the literature already indicate this concern even before any publication by the ACM/IEEE.

We highlight that the selection criteria was not limited to a specific languages, the only criteria for including documents in this systematic review is limited to time.

4.3.2 Exclusion Criteria

And with the aim of eliminating texts considered irrelevant [29], a set of five exclusion criteria was adopted:

- Duplicate Records

According to Kitchenham and Charters [29], it is important not to consider repeated evidence from the same study, to avoid create distortions in the conclusions. Thus, this criteria considers the most recent study, in cases of papers written by the same author or group of authors that address the same subject. However the study will only be deleted if the most recent one deals with all the contents of the older version. If the study contains an intersection, it will not be excluded.

- Papers providing only the abstract or unfinished research

Detailed information on the methodological aspects, such as type of research, data collection method, target audience, instruments, programming languages used, as well as details of the results, may not be included in the abstract or in unfinished research. However, it is a type of information that needs to be collected to assist the data analysis phase. Therefore, works that do not provide access to the full text or are ongoing research will not be included in the analysis phase.

- Papers that do not result from a scientific research

A systematic literature review aims to collect evidence from primary studies [42]. Thus, studies that are not the result of a research will not be considered. This case includes panel, journal column, tutorial, editorial and other systematic literature reviews.

- Works carried out with students of different levels

The interest in programming is not exclusive to higher education courses and much less to courses in the exact sciences area. There are many researches carried out also with students of all levels and modalities of teaching. However the interest of this review is to focus only on how to learn and teach in introductory programming in undergraduate courses.

- Works that do not address identity, basic learning conditions and the teaching methods of the discipline of "introductory programming"

The quality of a systematic review is linked to the papers chosen for the analysis. Works that do not answer at least one of the research questions of this review were not classified for the next phase.

After the definition of the inclusion and exclusion criteria, a research protocol, used in the selection process of the studies in this work was defined and is described in Table 01.

This protocol summarizes information on research questions, keywords and boolean operators used in the search process, databases, inclusion and exclusion criteria, methodological validation criteria and analysis of results.

Table 1 Research Protocol.

Research Questions
Q1 – How the introductory programming discipline is identified and structured and in which computer courses are the researches carried out?? Q2 – What are the technical and social skills involved in the process of creating and analyzing an algorithm? Q3 – What are the methodologies, tools, languages and programming paradigms used in the process of teaching and learning introductory programming?
Search Strategy
All: (“introductory programming course” OR “introduction to programming” OR “cs1”) AND All: (“teaching programming” OR “learning programming”) AND Abstract: (“skill” OR “knowledge” OR “method” OR “tools”) AND (publication date: 01/01/2016 TO 12/31/2021)
Database
ACM Digital Library, IEEE Xplore e ScienceDirect
Inclusion Criteria
1. Papers published between June 2017 and June 2021
Exclusion Criteria
1. Duplicate Records 2. Papers providing only the abstract or unfinished research 3. Papers that do not result from a scientific research 4. Works carried out with students of different levels 5. Works that do not address identity, basic learning conditions and the teaching methods of the discipline of “introductory programming”
Methodological validation criteria
Replication of the search process by another researcher

4.3.3 Selecting Studies

The first step was to conduct an automatic search in the databases, using the string and inclusion criteria defined in the search protocol. In total, 136 papers have been provided of which 90 from the ACM Digital Library, 21 from IEEE Xplore and finally 25 from Science Direct.

The identification and organization of all studies was done. The metadata were exported to a reference management software, which enabled the proper organization and automatic detection of duplicate articles. Eleven documents were excluded for duplicity, resulting 125 selected to the next stage of the process. After reading the respective titles and abstracts, the exclusion criteria 2, 3 and 4 have been applied.

In the end, 70 documents were selected for the full reading phase. Then, the last exclusion criterion has been applied, generating a set of 33 eligible studies (24,6% of the total) to data extraction and analysis step.

4.3.4 Data Classification

In addition to documenting the search and selection strategy used, the authors Kitchenham and Charters [29] indicate the need to also document the strategy used to extract the data contained in the selected studies. Pointing to the importance of constructing a data extraction form, which contains fields that record all documents uniformly – built in parallel with the search protocol – providing the foundation for appraising, analysing, summarising and interpreting a body of evidence.

The first data was collected by the reference management software. Items such as: title, authors, institution, country of publication, type of document, year of publication, abstract, keywords and database were inserted in the data extraction form.

Information such as research questions, data collection technique, type of study and results obtained were also identified in the studies to assist in the final analysis. All information extracted from the primary studies were managed by a spreadsheet program.

In order to standardize the extracted data, a data dictionary [29] was created to restrict the possibilities of categorizing certain form items, establishing a set of permissible values in each field, in order to facilitate the classification process.

5 Results

5.1 Identity of the Introductory Programming discipline (Q1)

The first research question launched by this study aims to determine how researchers, who investigate this topic, refer to this discipline. Thus, analyzing the collected data it was possible to realize that 22 papers refers to the name “introductory program course”. However, similar nomenclatures have also been identified as “introduction to programming” and “introductory computer programming”. In spite of being similar terminologies, all studies use the acronym CS1 to quote the discipline. That term, as previously identified, refers to Computer Science I, the first discipline created for the computer science course, in which the fundamentals of programming are addressed [30]. However, it is still used in the curriculum of all courses in the computing field, aiming to facilitate the transfer of students between educational institutions and also as a keyword for research developed in this field [23].

As can be seen in Table 02, other denominations were also identified and a new proposal is reported in [21]: “Computational Thinking Course”, being considered a lighter version of CS1 designed to engage and stimulate future CS1 students. And training skills such as: problem solving, logical thinking, abstraction, decomposition and pattern recognition, recommendations already foreseen in the ACM/IEEE-CS 2013 Report [44], that arise mainly to address students outside of computer or engineering courses.

This proposal can also be found in specific courses in the computing field, but is classified as an optional discipline and often offered before the beginning of the regular academic period. Identified by the terminology CS0. It is responsible for presenting computer science to students without previous skill, promoting mainly the development of problem solving and mathematical skills, in order to reduce retention rates and withdrawals in the future [14].

Of the papers that identified the audience involved in the study, 73% engaged students of computer courses, namely: computer science, computer engineering, information systems, information technology and software engineering. Also 9% of the research included students from other STEM courses like: electrical, electronics, civil, industrial and telecommunications engineering or bachelor’s degree in mathematics and 18% developed activities with graduates from other courses, that is, outside the computing field, which are also identified in the literature as non-Computer Science (non-CS) [11].

Offer introductory programming courses for audiences with different backgrounds, expectations and with a thematic focus became popular [44]. The great interest in programming in non-CS courses is related to initiatives that seek to make computing accessible to all [12], enabling students to interact consciously with new technologies, allowing the acquisition of the ability to read, write, analyze and modify program codes.

Table 2 Introductory Programming Identity.

Term	References
Algorithms and Problem Solving	[22]
Algorithms and Programming I	[52]
Computational Thinking Course	[21]
Introductory Computer Programming	[51]
Introductory Programming Course	[63, 33, 25, 47, 1, 2, 55, 58, 50], [13, 7, 38, 46, 12, 28, 4, 59, 56], [57, 6, 41, 48]
Introduction to Programming	[49, 20, 34, 17]
Programming 1	[15]
Programming Course	[39]

With regard to the approaches, most of the research focused on the use of a certain programming language. Leaving aside the emphasis on coding, 33% ([33, 39, 47, 2, 58, 34, 21, 52, 22, 46, 41]) provided a broader introduction to the concepts of programming, using algorithmic as an alternative approach.

This last perspective is analogous to the "Algorithms-first" model, proposed in CC2001 [18], which emphasizes the importance of students working with a variety of data and control structures, without having to deal with the specificities that programming languages inevitably introduce.

The research studies that used this approach were developed in the courses of: computer science [33, 2, 34] and [46], computer engineering [52, 41] and [58], information systems [58] and software engineering [58]. This information corroborates with the data published in CC2020 about the degree of importance that algorithms and data structure have in each course.

5.2 Technical and Social Skills (Q2)

Considering the 33 articles analyzed, 29 identified one or more previous skills needed or expected at the end of an introductory programming course. They are identified in table 03 and were gathered in two groups: technical skills (related to programming) and general skills.

Problem solving was the most cited skill, which it is identified by the computer science – CS2013 ([44]) and computer engineering – CE2016 ([43]) reports and defined by [34] as being the ability to understand a given context, identify key information and build a plan to solve it. To plan a solution, the programmer needs to divide the problem into smaller parts, analyze input and output data, and formulate the necessary steps for resolution [62]. The ability to divide a large and complex problem into parts that are manageable to solve, test, and maintain is also known as decomposition. Another skill identified in three papers as a prerequisite for learning programming. [25] states that learn to decompose a computational problem facilitates the software development process, but [13] emphasize that it is not a determining factor of success.

Besides decomposition, another skill, also related to problem solving, is algorithmic thinking. It is defined by [34] as a set of skills connected to building and analyzing algorithms; problem analysis; detailed specification/description of the problem; definition of the necessary actions and construction of an algorithm to solve the problem.

Table 3 Technical and General Skills.

Skill	References	Category
Abstraction	[25, 20, 34, 7, 22, 46]	Technical Skill
Algorithmic Thinking	[25, 47, 20, 15]	Technical Skill
Decomposition	[33, 25, 58, 20, 34, 52, 57]	Technical Skill
Debugging	[51, 2, 55, 58, 7, 59]	Technical Skill
Mathematical Skill	[46, 28]	Technical Skill
Problem Solving	[51, 33, 25, 2, 58, 20, 34, 50, 52, 56, 57]	Technical Skill
Critical Thinking	[51, 20, 21, 52, 59]	General Skill
Communication, creativity, persistence, voluntary participation, perseverance and trust	[51, 28, 41, 10]	General Skill
Spatial Visualization	[17]	General Skill
Team Work and collaboration	[52, 22, 28, 59]	General Skill

In six publications the importance of abstraction was discussed. [61] claims that it is the process used in setting patterns, generalized from specific instances and parameterization in order to capture essential properties that are common to a set of objects. The author also describes that an algorithm is an example of a process abstraction, that starts by receiving input values (input), running a sequence of steps (algorithm), and producing a result (output) to satisfy a given objective (problem).

These skills (abstraction, decomposition, algorithmic thinking and problem resolution) are the key concepts of computational thinking, proposed by [60] and defined as a set of skills that allows us to recognize aspects of computing in everyday life. Beyond computational thinking, another feature developed throughout the discipline and evidenced by [51, 2, 55, 58, 7] and [59] is the debugging. Defined as the process of finding and reducing errors in a code [51], it is identified as the most neglected topic in teaching introductory programming. [59] have identified that software developers tend to debug codes using a scientific method that requires cycles of generations and hypothesis testing. This process, according to the authors, promotes a deeper learning and provides an insight into how critical the individual is and the strategies used. However, the curriculum tends not to focus on teaching strategies that promote critical thinking [59]. Authors such as [20] and [52] claim that this ability is usually noticeable only in experienced programmers, as it is related to the individual's level of maturity.

Mathematical knowledge is cited as necessary and closely related to problem solving, decomposition, and abstraction skills. The lack of this ability can influence programming learning [46] and [28]. However, [34] rejects the hypothesis of correlation between grades obtained by high school students in mathematics and learning programming. Nevertheless, mathematical skills can be used to identify cognitive problems in students as incapable of connecting algorithmic thinking to other mathematical concepts [13]. Associated with mathematical skills, geometry and spatial visualization is also present in the collected data, which influences the understanding and the ability to mentally manipulate a two/three-dimensional figure in space [54]. [17] exposes an example in which a 3D object is placed on a table, and the student needs to imagine how the object will be rotated, without having any physical/real interaction or change of perspective.

And finally, teamwork, communication, creativity, persistence, voluntary participation, perseverance and trust are social skills identified in eight different papers. The authors [51] and [52] emphasized several aspects related to the attitude of students and [22] sustain that knowledge is built from the participation and collaboration between peers. CC2020 corroborates with this statement, and clarifies that all computer courses emphasize the professional knowledge required of each professional, including problem solving, critical thinking, communication, and teamwork [19].

In summary, two areas of computational thinking were outlined in the collected data: Computational practices (algorithm, decomposition, abstraction, etc.) and computer perspectives, i.e. the understanding that students have of themselves and their interaction with others and with technology [8]. This idea broadens the original definition of computational thinking proposed by [60], changing the individualistic conception of programming to a vision that includes focusing on social dimensions, called computational participation [26]. This new conception explore computational practices and perspectives that together enable insight into sociological and cultural dimensions, with an emphasis on learning to code so that learners are able to meaningfully participate as critical thinkers, as well as producers, consumers, and distributors of technology [26].

5.3 Methodologies, Tools, Languages and Programming Paradigms(Q3)

Many teaching strategies have been used and reported. However, a trend towards the use of traditional approach was identified: where the teacher reviews the content, explain the terms and concepts followed by paper-based programming exercises ([10, 50] and [52]). The practical activities are developed later in a laboratory with the support of software/tools ([63, 51, 39, 47, 55, 20, 7, 52, 15, 4, 59, 56, 57, 6] and [48]) or hardware ([10] and [50]). Facing the constraints of using the traditional teaching method, or in order to increase students' motivation ([21, 39, 47, 1, 56]), reduce failure rates in CS1 ([34, 56, 25, 20]), attend classes with large numbers of students ([55, 50] and [41]), active methods have been incorporated as innovative teaching strategy and are cited in Table 4.

Table 4 Methodologies.

Teaching Strategy	**References**
Active Learning/Peer learning	[21, 22]
Blended Learning	[38]
E-learning	[25, 1, 12, 48]
Flipped Classroom	[25, 2]
Gamification	[39, 38]
Peer Programming	[49, 21, 22]
Problem Based Learning	[52]
Project Based Learning	[10, 22, 46]
Storytelling/Storyboard	[58, 38]
Tradicional Classes	[63, 51, 47, 55, 20, 50, 7, 52, 15, 4, 59, 56, 57, 6, 48]

This scenario, in which traditional classes are enriched with laboratory practice are foreseen in the reports published in 2013 (computer science), 2014 (software engineering) and 2016 (computer engineering). The suggestion for change is clear and incisive from the CC2020, which suggests that exploring new methods of learning can augment the learning of knowledge and allow students to interact with each other to develop new skill sets as well as to develop both communication and teamwork skills by studying with others [19].

About programming languages and paradigms, 57,5% of the authors used a programming language in the discipline under investigation. The summary of these data is identified in Table 5 and from this information it was possible to correlate which programming paradigms the courses chose to use.

A programming paradigm, according to [31] is a way to classify programming languages according to their functionalities, it will determine how the program will be structured and executed. Most of the research studies opted for the use of the object-oriented paradigm, followed by five papers that addressed the structured paradigm and two multi-paradigm. The courses that used one of the programming languages, also associated the use of an Integrated Development Environment (IDE), a software for building applications that combines common developer tools into a single graphical user interface (GUI). As an example were mentioned: Spyder, Visual Studio, ArduinoStudio and Eclipse.

Table 5 Programming Languages used in CS1.

Programming Languages	**Paradigms**	**References**
C	Structured	[63, 1, 7, 52, 46]
C++	Object-oriented	[39, 47, 10]
Java	Object-oriented	[25, 49, 13, 12]
Phyton	Multi-paradigm	[20, 21]
C#	Object-oriented	[15]

Research such as [20] and [10] also integrated the use of an Arduino, an electronic prototyping platform which allows the development and control of interactive systems of low cost. Another particularity was the adoption of a code review tools used by [25, 49, 1, 55, 7, 59] e [48] in order to facilitate the work done by the teachers in correcting coding exercises, consequence of a large number of students per class.

Despite the facilities offered by these tools, [7] and [6] proved the importance of a qualitative feedback, so that students can identify not only where the syntax error is, but also what cause the error. Others ([33, 58, 22, 15]) have made the option to use block-based programming environments, such as: Alice, Inventior app, Blockly games, code.org, gameblox, Pencil code, microsoft makecode and Scratch. These platforms allow more accessible programming environments than programming languages, as they use graphical interfaces that enable programs by dragging and dropping blocks.

Studies ([33, 34, 13, 28]) in which tools or code evaluation techniques have been developed, did not mention methods of teaching or learning introductory programming. In contrast, [21, 22] and [38] cited the use of one or more strategies in their research.

Although 54% of studies referenced the use of innovative methods, but most described conducting uniquely a single experiment. Only [10] narrates a four-year experience with students of computer engineering, software engineering and information systems courses.

6 Conclusion

A systematic review was performed with the analysis of 33 recently published studies obtained from ACM, IEEE and ScienceDirect databases in order to understand in depth, the introductory programming discipline in higher education.

For the first research question, the data revealed that most studies mentioned the name "Introductory Program Course". In addition to this, the acronym CS1 was also widely used as a synonym for "introductory program course". Thus, in future work, the authors will opt to use the term and the acronym to indicate the first course in which students have the first contact with the fundamentals of programming.

Throughout the history of computing, the structure of the CS1 discipline has been the subject of intense debate. Many strategies have been proposed and numerous discussions have been raised around. As explained, the analyzed papers present several approaches.

Some have focused on the core concepts of software development associated with a particular programming language, others put aside the emphasis on programming and provide a broader introduction to the concepts, with an emphasis on algorithmic. And the courses that were involved in the development of these reflections were: Computer Science, Computer Engineering, Software Engineering and Information System. This corroborates with the CC2020 report about the level of importance that this discipline has in each course.

For Q2, there was a concern in stimulating technical skills linked to programming. On the other hand, skills such as critical thinking, communication, creativity, persistence, voluntary participation, perseverance, trust and teamwork emerged in twelve studies. 69% of the papers that identified the use of some active method in teaching-learning CS1, identified one or more social skills developed in programming practice. These data lead us to believe that knowledge and skills, be they technical or social are clearly identified, transferred and achieved through the practices described. However, research has not yet identified this triad as part of a competency-based model.

Finally, in response to Q3, it was noted that instead of a particular programming language or paradigm being favored, a list of programming languages and paradigms are successfully used in the courses. As this systematic review collected data that refers only to CS1, it was not possible to observe whether the analyzed courses address more than one paradigm in their programs. But, restricting the students' experience to just one paradigm can make the transition to the next more complicated [64]. Shifting the focus from the programming languages and paradigms to the tools, it was observed that the courses used different platforms, that can bring the classroom learning closer to professional contexts, as is the case of IDE.

The same applies for teaching and learning strategies. Methods like pair programming, a practice derived from the agile software (EXtreme programming - XP) and other methods like problem-based and design-based learning, are techniques that provide the development of the triad: knowledge, technical and social skills, in a practical context.

In conclusion, this systematic literature review in the area of introductory programming education over the past 5 years, have explored a variety of themes in the computing field, making at least the following contributions: identifying a standard identity for introductory programming discipline; identifying the aspects of CS1 have been focus of publication; summarizing the strategies, tools and technology used in teaching and learning programming; and highlighting the evidence of the use of a competency-based model in learning-teaching process.

This work is part of a PhD thesis and as future work the authors intend to further explore these issues and to focus on how the dimensions of computational participation may influence the learning processes of introductory programming, promoting the skills revealed in this review.

7 Limitation of Our Systematic Review

This work shares the most limitations of systematic review method: the bias in selecting articles and in data extraction due to our choices of eligibility criteria. Furthermore, other limitation lies to the fact that the research was not supplemented with a complementary process or made use of a quality evaluation to selecting papers. These limitations were addressed developing a strong protocol that answer the search problem and using a combined manual and automatic search. For ensuring the quality we selected papers based on the characteristics of the studies, described in item "Data Classification" and used different types of databases.

References

1 Ella Albrecht, Fabian Gumz, and Jens Grabowski. (R08) Experiences in Introducing Blended Learning in an Introductory Programming Course. In *Proceedings of the 3rd European Conference of Software Engineering Education*, ECSEE'18, pages 93–101, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Seeon/ Bavaria, Germany. doi:10.1145/3209087.3209101.

2 Saleh Alhazbi and Osama Halabi. (R09) Flipping Introductory Programming Class: Potentials, Challenges, and Research Gaps. In *Proceedings of the 10th International Conference on Education Technology and Computers*, ICETC '18, pages 27–32, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Tokyo, Japan. doi:10.1145/3290511.3290552.

3 Mike Barkmin and Torsten Brinda. Analysis of programming assessments — building an open repository for measuring competencies. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, Koli Calling '20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3428029.3428039.

4 Brett A. Becker, Catherine Mooney, Amruth N. Kumar, and Sean Russell. (R26) A Simple, Language-Independent Approach to Identifying Potentially At-Risk Introductory Programming Students. In *Australasian Computing Education Conference*, ACE '21, pages 168–175, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual, SA, Australia. doi:10.1145/3441636.3442318.

5 Brett A. Becker and Keith Quille. 50 years of cs1 at sigcse: A review of the evolution of introductory programming education research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 338–344, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3287324.3287432.

6 Jeremiah Blanchard, Christina Gardner-McCune, and Lisa Anthony. (R30) Dual Modality Instruction &amp; Programming Environments: Student Usage &amp; Perceptions. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, pages 481–487, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual Event, USA. doi:10.1145/3408877.3432434.

7 Yorah Bosse, David Redmiles, and Marco A. Gerosa. (R18) Pedagogical Content for Professors of Introductory Programming Courses. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 429–435. Association for Computing Machinery, New York, NY, USA, 2019. doi:10.1145/3304221.3319776.

8 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.

9 Pearl Brereton, Barbara Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571–583, 2007. doi:10.1016/j.jss.2006.07.009.

10 David W. Brown, Sheikh K. Ghafoor, and Stephen Canfield. (R13) Instruction of Introductory Programming Course Using Multiple Contexts. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018, pages 147–152, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Larnaca, Cyprus. doi:10.1145/3197091.3197105.

11 Parmit Chilana, Celena Alcock, Shruti Dembla, Anson Ho, Ada Hurst, Brett Armstrong, and Philip Guo. Perceptions of non-CS majors in intro programming: The rise of the conversational programmer. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 251–259, 2015. doi:10.1109/VLHCC.2015.7357224.

12 Rodrigo Silva Duran, Jan-Mikael Rybicki, Arto Hellas, and Sanna Suoranta. (R24) Towards a Common Instrument for Measuring Prior Programming Knowledge. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 443–449. Association for Computing Machinery, New York, NY, USA, 2019. doi:10.1145/3304221.3319755.

13 Nikita Dümmel, Bernhard Westfechtel, and Matthias Ehmann. (R17) MuLE: A Multiparadigm Language for Education. The Object-Oriented Part of the Language. In *Proceedings of the 4th European Conference on Software Engineering Education*, ECSEE '20, pages 32–41, New York, NY, USA, 2020. Association for Computing Machinery. event-place: Seeon/Bavaria, Germany. doi:10.1145/3396802.3396806.

14 Cenk Erdil and Darcy Ronan. Implementing CS0 with Computer Science Principles Curriculum. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 1272, New York, NY, USA, 2019. Association for Computing Machinery. event-place: Minneapolis, MN, USA. doi:10.1145/3287324.3293791.

15 Osman Erol and Adile Aşkım Kurt. (R22) The effects of teaching programming with scratch on pre-service information technology teachers' motivation and achievement. *Computers in Human Behavior*, 77:11–18, 2017. doi:10.1016/j.chb.2017.08.017.

16 Katia Romero Felizardo, Emilia Mendes, Marcos Kalinowski, Érica Ferreira Souza, and Nandamudi L. Vijaykumar. Using forward snowballing to update systematic reviews in software engineering. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '16, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2961111.2962630.

17 José Figueiredo and Francisco García-Peñalvo. (R32) Teaching and Learning Tools for Introductory Programming in University Courses. In *2021 International Symposium on Computers in Education (SIIE)*, pages 1–6, 2021. doi:10.1109/SIIE53363.2021.9583623.

18 The Joint Task Force for Computing Curricula 2001. Computing curricula 2001. *Journal on Educational Resources in Computing (JERIC)*, 1(3ª), 2001. Publisher: ACM New York, NY, USA.

19 The Joint Task Force for Computing Curricula 2020. *Computing Curricula 2020: Paradigms for Global Computing Education.* Association for Computing Machinery, New York, NY, USA, 2020.

20 G. Cooper, R. Walker, E. Hill, and N. Waksmanski. (R12) Incorporating IoT and Data Analytics in an Introductory Programming Course. In *2020 15th International Conference on Computer Science & Education (ICCSE)*, pages 169–175, August 2020. Journal Abbreviation: 2020 15th International Conference on Computer Science & Education (ICCSE). doi:10.1109/ICCSE49874.2020.9201863.

21 Luke Gusukuma, Austin Cory Bart, Dennis Kafura, and Jeremy Ernst. (R16) 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. event-place: Espoo, Finland. doi:10.1145/3230977.3231002.

22 H. Amer and S. Harous. (R20) Smart-Learning Course Transformation for an Introductory Programming Course. In *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*, pages 463–465, July 2017. Journal Abbreviation: 2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT). doi:10.1109/ICALT.2017.91.

23 Matthew Hertz. What Do "CS1" and "CS2" Mean? Investigating Differences in the Early Courses. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 199–203, New York, NY, USA, 2010. Association for Computing Machinery. event-place: Milwaukee, Wisconsin, USA. doi:10.1145/1734263.1734335.

24 Theresia Devi Indriasari, Andrew Luxton-Reilly, and Paul Denny. Improving student peer code review using gamification. In *Australasian Computing Education Conference*, ACE '21, pages 80–87, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3441636.3442308.

25 J. Skalka, M. Drlík, and J. Obonya. (R05) Automated Assessment in Learning and Teaching Programming Languages using Virtual Learning Environment. In *2019 IEEE Global Engineering Education Conference (EDUCON)*, pages 689–697, April 2019. Journal Abbreviation: 2019 IEEE Global Engineering Education Conference (EDUCON). doi:10.1109/EDUCON.2019.8725127.

26 Yasmin B. Kafai and Quinn Burke. Computational Participation: Teaching Kids to Create and Connect Through Code. In Peter J. Rich and Charles B. Hodges, editors, *Emerging Research, Practice, and Policy on Computational Thinking*, pages 393–405. Springer International Publishing, Cham, 2017. `doi:10.1007/978-3-319-52691-1_24`.

27 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`.

28 Natalie Kiesler. (R25) Towards a Competence Model for the Novice Programmer Using Bloom's Revised Taxonomy - An Empirical Approach. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, pages 459–465, New York, NY, USA, 2020. Association for Computing Machinery. event-place: Trondheim, Norway. `doi:10.1145/3341525.3387419`.

29 Barbara Kitchenham and Stuart Charters. *Guidelines for performing systematic literature reviews in software engineering.* Citeseer, 2007.

30 Elliot Koffman, Philip Miller, and Caroline Wardle. Recommended Curriculum for CS1, 1984. *Commun. ACM*, 27(10):998–1001, October 1984. Place: New York, NY, USA Publisher: Association for Computing Machinery. `doi:10.1145/358274.358279`.

31 Sirojiddin Komolov, Nursultan Askarbekuly, and Manuel Mazzara. An Empirical Study of Multi-Threading Paradigms Reactive Programming vs Continuation-Passing Style. In *2020 the 3rd International Conference on Computing and Big Data*, ICCBD '20, pages 37–41, New York, NY, USA, 2020. Association for Computing Machinery. event-place: Taichung, Taiwan. `doi:10.1145/3418688.3418695`.

32 Patrick Korber and Renate Motschnig. The effects of pair-programming in introductory programming courses with visual and text-based languages. In *2021 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, 2021. `doi:10.1109/FIE49875.2021.9637186`.

33 L. Carlos Begosso, L. Ricardo Begosso, and N. Aragao Christ. (R03) An analysis of block-based programming environments for CS1. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, October 2020. Journal Abbreviation: 2020 IEEE Frontiers in Education Conference (FIE). `doi:10.1109/FIE44824.2020.9273982`.

34 L. M. de Souza, B. M. Ferreira, I. M. Félix, L. de Oliveira Brandão, A. A. F. Brandão, and P. A. Pereira. (R14) Mathematics and programming: marriage or divorce? In *2019 IEEE World Conference on Engineering Education (EDUNINE)*, pages 1–5, March 2019. Journal Abbreviation: 2019 IEEE World Conference on Engineering Education (EDUNINE). `doi:10.1109/EDUNINE.2019.8875849`.

35 Philip I.S. Lei and António José Mendes. A systematic literature review on knowledge tracing in learning programming. In *2021 IEEE Frontiers in Education Conference (FIE)*, pages 1–7, 2021. `doi:10.1109/FIE49875.2021.9637323`.

36 Madeleine Lorås, Guttorm Sindre, Hallvard Trætteberg, and Trond Aalberg. Study behavior in computing education – a systematic literature review. *ACM Trans. Comput. Educ.*, 22(1), October 2021. `doi:10.1145/3469129`.

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

38 M. F. Ercan and D. Sale. (R21) Teaching programming: An evidence based and reflective approach. In *2020 IEEE REGION 10 CONFERENCE (TENCON)*, pages 997–1001, November 2020. Journal Abbreviation: 2020 IEEE REGION 10 CONFERENCE (TENCON). `doi:10.1109/TENCON50793.2020.9293812`.

39 B. Marín, J. Frez, J. Cruz-Lemus, and M. Genero. (R04) An Empirical Investigation on the Benefits of Gamification in Programming Courses. *ACM Trans. Comput. Educ.*, 19(1), November 2018. Place: New York, NY, USA Publisher: Association for Computing Machinery. doi:10.1145/3231709.

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

41 Nathan Mills, Allen Wang, and Nasser Giacaman. (R31) Visual Analogy for Understanding Polymorphism Types. In *Australasian Computing Education Conference*, ACE '21, pages 48–57, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual, SA, Australia. doi:10.1145/3441636.3442304.

42 Elisa Yumi Nakagawa, Kátia Romero Felizardo Scannavino, Sandra Camargo Pinto Ferraz Fabbri, and Fabiano Cutigi Ferrari. *Revisão sistemática da literatura em engenharia de software: teoria e prática.* Elsevier Brasil, 2017.

43 Task Group on Computer Engineering Curricula. Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering. Technical report, Association for Computing Machinery, New York, NY, USA, 2016.

44 Task Group on Computer Science Curricula. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* Association for Computing Machinery, New York, NY, USA, 2013.

45 Task Group on Information Technology Curricula. *Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology.* Association for Computing Machinery, New York, NY, USA, 2017.

46 P. E. Martínez López, D. Ciolek, G. Arévalo, and D. Pari. (R23) The GOBSTONES method for teaching computer programming. In *2017 XLIII Latin American Computer Conference (CLEI)*, pages 1–9, September 2017. Journal Abbreviation: 2017 XLIII Latin American Computer Conference (CLEI). doi:10.1109/CLEI.2017.8226428.

47 Fredi E. Palominos, Seomara K. Palominos, Claudia A. Durán, Felisa M. Córdova, and Hernán Díaz. (R06) Challenges in the use of a support tool with automated review in student learning of programming courses. *Procedia Computer Science*, 139:424–431, 2018. doi:10.1016/j.procs.2018.10.260.

48 Filipe Dwan Pereira, Samuel C. Fonseca, Elaine H. T. Oliveira, Alexandra I. Cristea, Henrik Bellhäuser, Luiz Rodrigues, David B. F. Oliveira, Seiji Isotani, and Leandro S. G. Carvalho. (R33) Explaining Individual and Collective Programming Students' Behavior by Interpreting a Black-Box Predictive Model. *IEEE Access*, 9:117097–117119, 2021. doi:10.1109/ACCESS.2021.3105956.

49 Reinhold Plösch and Cornelia Neumüller. (R07) Does Static Analysis Help Software Engineering Students? In *Proceedings of the 2020 9th International Conference on Educational and Information Technology*, ICEIT 2020, pages 247–253, New York, NY, USA, 2020. Association for Computing Machinery. event-place: Oxford, United Kingdom. doi:10.1145/3383923.3383957.

50 James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. (R15) Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ICER '18, pages 41–50, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Espoo, Finland. doi:10.1145/3230977.3230981.

51 Vijayalakshmi Ramasamy, Hakam W. Alomari, James D. Kiper, and Geoffrey Potvin. (R02) A Minimally Disruptive Approach of Integrating Testing into Computer Programming Courses. In *Proceedings of the 2nd International Workshop on Software Engineering Education for Millennials*, SEEM '18, pages 1–7, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Gothenburg, Sweden. doi:10.1145/3194779.3194790.

52 S. M. Souza and R. A. Bittencourt. (R19) Report of a CS1 Course for Computer Engineering Majors Based on PBL. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 837–846, April 2020. Journal Abbreviation: 2020 IEEE Global Engineering Education Conference (EDUCON). doi:10.1109/EDUCON45650.2020.9125121.

53 Leonardo Silva, António José Mendes, and Anabela Gomes. Computer-supported collaborative learning in programming education: A systematic literature review. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 1086–1095, 2020. doi:10.1109/EDUCON45650.2020.9125237.

54 Sheryl Sorby. A Course in Spatial Visualization and its Impact on the Retention of Female Engineering Students. *Journal of Women and Minorities in Science and Engineering*, 7:50, January 2001. doi:10.1615/JWomenMinorScienEng.v7.i2.50.

55 Kristin Stephens-Martinez and Armando Fox. (R10) Giving Hints is Complicated: Understanding the Challenges of an Automated Hint System Based on Frequent Wrong Answers. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018, pages 45–50, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Larnaca, Cyprus. doi:10.1145/3197091.3197102.

56 Shelsey Sullivan, Hillary Swanson, and John Edwards. (R28) Student Attitudes Toward Syntax Exercises in CS1. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, pages 782–788, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual Event, USA. doi:10.1145/3408877.3432399.

57 Lasang Jimba Tamang, Zeyad Alshaikh, Nisrine Ait Khayi, Priti Oli, and Vasile Rus. (R29) A Comparative Study of Free Self-Explanations and Socratic Tutoring Explanations for Source Code Comprehension. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, pages 219–225, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual Event, USA. doi:10.1145/3408877.3432423.

58 Damla Topalli and Nergiz Ercil Cagiltay. (R11) Improving programming skills in engineering education through problem-based game projects with Scratch. *Computers & Education*, 120:64–74, 2018. doi:10.1016/j.compedu.2018.01.011.

59 Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. (R27) Novice Reflections on Debugging. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, pages 73–79, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual Event, USA. doi:10.1145/3408877.3432374.

60 Jeannette Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006. Publisher: ACM New York, NY, USA.

61 Jeannette Wing. Computational Thinking. *J. Comput. Sci. Coll.*, 24(6):6–7, June 2009. Place: Evansville, IN, USA Publisher: Consortium for Computing Sciences in Colleges.

62 Lan Wu, Yang Liu, Axi Wang, YuanLi Gong, and ShengQuan Yu. An analysis of Interaction of Cognitive and Social Aspects during Collaborative Problem Solving. In *2021 International Conference on Advanced Learning Technologies (ICALT)*, pages 105–107, 2021. doi:10.1109/ICALT52272.2021.00039.

63 Jooyong Yi, Umair Z. Ahmed, Amey Karkare, Shin Hwei Tan, and Abhik Roychoudhury. (R01) A Feasibility Study of Using Automated Program Repair for Introductory Programming Assignments. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 740–751, New York, NY, USA, 2017. Association for Computing Machinery. event-place: Paderborn, Germany. doi:10.1145/3106237.3106262.

64 Daeng Zuhud. Some Prospective Approaches for the Shift of Programming Paradigms. In *Proceedings of the 2013 International Conference on Information Systems and Design of Communication*, ISDOC '13, pages 87–93, New York, NY, USA, 2013. Association for Computing Machinery. event-place: Lisboa, Portugal. doi:10.1145/2503859.2503873.

Feedback Systems for Students Solving Problems Related to Polynomials of Degree Two or Lower

Luke Adrian Gubbins Bayzid
Campus Universitário de Santiago, University of Aveiro, Portugal
Thematic Line GEOMETRIX, University of Aveiro, Portugal

Ana Maria Reis D'Azevedo Breda
Campus Universitário de Santiago, University of Aveiro, Portugal
Center for Research & Development in Mathematics and Applications, University of Aveiro, Portugal

Eugénio Alexandre Miguel Rocha
Campus Universitário de Santiago, University of Aveiro, Portugal
Center for Research & Development in Mathematics and Applications, University of Aveiro, Portugal

José Manuel Dos Santos Dos Santos [1]
Centre for Research and Innovation in Education (inED), Escola Superior de Educação – Politechnic of Porto, Portugal

Abstract

In this paper, we present the first attempts to design and implement an algorithm that effectively responds to errors in a student's resolution in problems related to polynomials of degree two or lower. The algorithm analyzes the student's input by comparing pre-established resolution patterns. The obtained results of the implementation show that the algorithm is effective at the classes of problems created within the project's scope. Future work will concern the expansion of the number of classes to other common types of problems, such as higher-degree polynomials, and its use at a large scale using open-source software with CAS capabilities.

2012 ACM Subject Classification Mathematics of computing; Computing methodologies

Keywords and phrases Automatic feedback, Algorithms, Algebraic systems

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.5

Funding This research was supported by the Center for Research and Development in Mathematics and Applications (CIDMA) through the Portuguese Foundation for Science and Technology (FCT – Fundação para a Ciência e a Tecnologia), and within the scope of the research conducted by Thematic Line GEOMETRIX, references UIDB/04106/2020 and UIDP/04106/2020; The Centre for Research and Innovation in Education (inED), through the FCT – Fundação para a Ciência e a Tecnologia, I.P., under the scope of the project UIDB/05198/2020; and Organization of Ibero-American States for Education, Science and Culture (OEI).

1 Introduction

Feedback is crucial in the development of a student's ability to validly reason in any subject of study [2, 3, 1]. Consequently, the detection of mistakes made by students while solving a problem is of extreme importance, as it permits the personalization of feedback, highlighting aspects of the problem that the student should pay more attention to. Once a student knows which issues they should focus on, studying goes from a task about haphazardly improving at a set of problems into recognizing and formulating specific ideas associated to the material at hand, facilitating the identification and communication of issues.

[1] Corresponding author.

Given the number of students assigned to a class, it can be difficult or even impossible for a teacher to provide each student with the advice needed to improve at the subject at hand, and this situation becomes even more complex for students with difficulties in expressing their doubts, some of them choosing to focus only on autonomous study as a consequence.

Unlike many other fields, mathematics deals with structured forms that have definite rules, allowing anyone, even computers, to modify previous structures into others. While generating interesting structures continues to be an issue, verifying them automatically is something that computers excel at.

In this article, we propose a general method to generate feedback based on the input of an ordered list of structured equations provided by a student, presenting a particular case of using this procedure for problems involving polynomials of degree two or lower.

2 Feedback

In general, *feedback* can be defined as a piece of information related to the previous results of a particular task that reflects what ought to be improved. For example, information detailing only the subject's achieved score would be considered as a form of feedback [7].

Given that definition, one aspect to focus on would be the quality of the information provided. In particular, the information that we decided to tackle in this article concerns only the quality of the result or the method used by the student; this restriction was made to ensure the feasibility of the project within its intended scope. An example of a type of feedback that falls outside of those two categories would be recommendations regarding the mindsets that a student should consider while studying.

With regard to education, feedback seeks to increase the long-term score of a student's ability to perform the tasks associated to the subject being taught. From various experiments performed over the years using a variety of methods, the positive effects of feedback within education have been empirically validated with a high degree of statistical certainty [8]. However, different categories of feedback and subsets thereof have also proven to be more influential than others [5, 6], necessitating a more particular analysis of the chosen categories for the project.

Using our categorizations of feedback, we were interested in knowing how much feedback regarding the student's methodology would benefit them over merely providing an aptitude score. In accordance with relevant research [4], while the results aren't drastic enough to cause a sudden shift in the education system, the measurable increase in providing both types of feedback was sufficient enough for us to deem it as a worthwhile endeavor.

Though effective feedback can take many forms [2], only the lesser of them tend to get implemented into software that's meant to support the education of students. In particular, one commonly observed detail about many of those programs is that they can only discern whether or not an answer is correct; some of them might also provide a solution written out by a professor. While having a curated solution can definitely help students struggling to answer questions aptly, it might not help them with understanding the subject at hand, only to memorize a list of steps that lead to a correct answer. Though a proper understanding is not the goal of all students, having such might help them generalize what they're taught, improving performance in similar kinds of questions over a variety of circumstances.

3 Motivation and Problem

The difficulty in solving this particular kind of problem primarily stems from automating the process of identifying a mistake and appropriately associating it to its respective feedback. While the former problem can be solved by using a sufficiently powerful CAS, the fact that a problem can be solved through a variety of means and with varying amounts of detail makes the latter problem a bit less trivial to solve. For example, the two mistaken resolutions $x + 1 = 0 \iff x = 1$ and $x + 1 = 0 \iff x + 1 - 1 = 0 + 1 \iff x = 1$ have different lengths and vary in detail but still display the misuse of signs being the source of the issue. As such, the feedback returned by the algorithm should be invariant to such changes.

Our motivation to implement a system based on axiom schemata and tree substitution came from the necessity to generate structured data using simple rules. Originally, we believed that it would be too computationally inefficient, but further tests showed that a naive implementation would suffice for the sample of rules necessary to represent a subset of the types of mistakes performed by a student in the given task. Later on, different ideas for optimization were considered but deemed unnecessary for the algorithm's first implementation; more research will be made in that regard.

Given the initial goal of managing polynomials of degree two or lower, a previous attempt used the roots of the provided polynomial to construct a canonical representation by expanding its factored form and dividing all coefficients by the value of the leading coefficient. From there, comparisons would be made between that representation and the one of the step at which the student made a mistake. However, due to its lack of generality beyond polynomials, this algorithm was dropped in favor of the one detailed in this article.

An idea that came from that previous attempt was to automatically encode every step in a canonical representation; in particular, the canonical representations would be constructed in a way that would facilitate the detection of mistaken symbols, such as inserting a minus instead of a plus. This idea was also abandoned due to the lack of a general pattern regarding what might facilitate the detection of mistakes for more complicated kinds of equations.

4 Algorithm

4.1 Overview

The algorithm described in this article consists in four steps: parsing and cleaning text, validating the equality of the solution sets of adjacent steps, generating paths that match the student's mistake, and providing adequate feedback. Following the presented order, each of the following paragraphs will detail the overview of the process.

The parsing algorithm that we implemented consists of a collection of metrics that decide whether or not to include a character of the input based on surrounding characters and if it belongs to a curated set of expected characters. After that, a structured tree is formed using the order of operations that are either implicit or explicit through the use of parentheses and common syntax.

The goal of a validation system is to ensure that each step is entailed from those that came before it. A property that facilitates this goal is that equations can be marked as equivalent if they have equal solution sets. Knowing that, our method uses a CAS to search for the first step in which a mistake was made by testing the equality of the solution sets of each ordered pair of consecutive steps. For a formalization thereof, see Algorithm 1; it should be noted that Ω is a function that returns the solution set associated to a step and that *Steps* is a tuple representing each equation written down by the student in order.

Once the first two equations in which a mistake happened have been identified, the system will take the first of them and generate all of the possible expressions from it using the supplied patterns; it will repeat that process for a finite number of steps. Finally, it will aggregate all of the paths with solution sets that are equal to the solution set of the mistaken step.

Algorithm 1 Determine First Erroneous Step.

```
1: for $i = 1, \ldots, |Steps| - 1$ do
2:     if $\Omega(Steps_i) \neq \Omega(Steps_{i+1})$ then
3:         return $(Steps_i, Steps_{i+1}, i + 1)$
4:     end if
5: end for
6: return ()
```

Given that each pattern used in the previous step has an associated flag, the final aspect of the algorithm is to return the feedback associated to each of the generated paths in natural language.

Figure 1 shows an overview of the algorithm's logic.

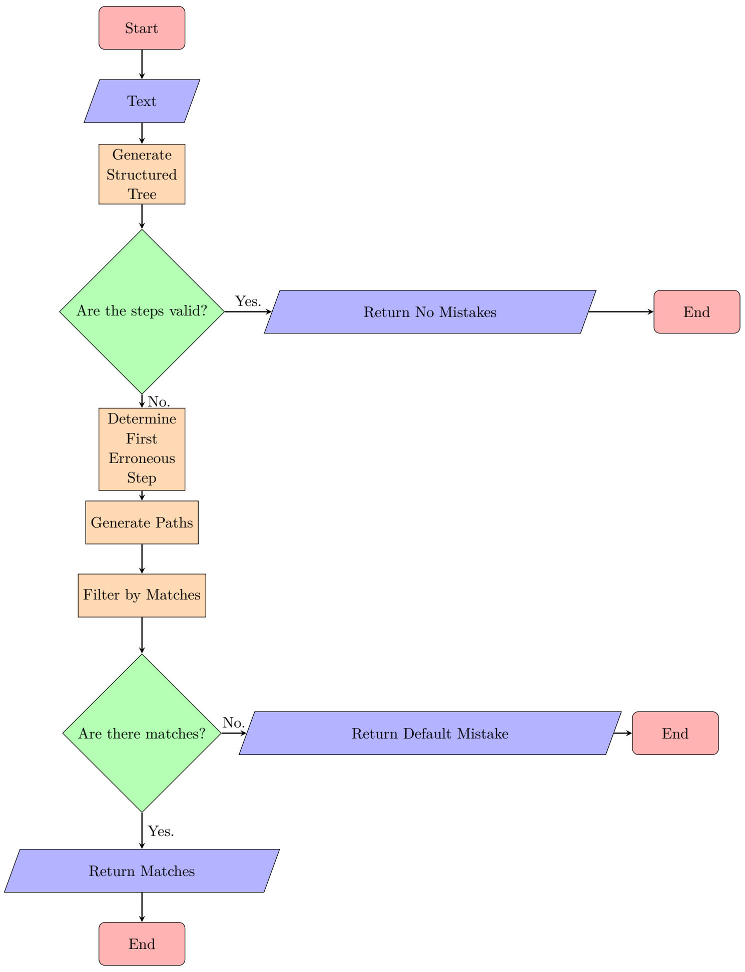


Figure 1 Flowchart of the Algorithm's Logic.

4.2 Patterns

Within our implementation, expressions are represented as trees with subtrees that are determined by the order of operations presented within the mathematical expression. In the case of polynomials, only functions of arity two or lower need to be considered, so each node in a polynomial expression need only have two or three children, where those children can be strings or other trees.

Each pattern consists of an axiom schema that reflects a particular deformation that can be applied to an expression. Our implementation represents axiom schemata as pairs of trees consisted only of constants, schematic variables, and subtrees. Constants are strings that must be exactly matched to have the pattern apply, whereas schematic variables are strings that are uniquely denoted using a dollar symbol before their declaration and can be substituted by a constant or a tree; subtrees are used to encode the order of operations within an expression. For example, $((\text{"\$1"}, \text{"}+\text{"}, \text{"\$2"}), (\text{"\$2"}, \text{"}+\text{"}, \text{"\$1"}))$ would represent the commutativity of addition, and $((\text{"\$1"}, \text{"}*\text{"}, (\text{"\$2"}, \text{"}+\text{"}, \text{"\$3"})), ((\text{"\$1"}, \text{"}*\text{"}, \text{"\$2"}), \text{"}+\text{"}, (\text{"\$1"}, \text{"}*\text{"}, \text{"\$3"})))$ would represent the distributive property of multiplication over addition.

4.3 Applying Patterns

Similar to many previous approaches regarding applying rules to structured data, using an ordered list of strings denoting schematic variables, constants, and other lists of the same type to represent a tree, applying a pattern merely requires verifying several conditions before performing a substitution.

A necessary property to apply patterns is to detect when they can be applied. This can be achieved by having the tree representing the pattern be a generalization of the structure that we want to deform, which primarily pertains to the order of its subtrees and schematic variables. We say that X is a generalization of Y, which we denote as $X \geq Y$, if and only if

$$|X| = |Y| \land \forall i \in [|X|], X_i \geq Y_i \lor (X_i \in V_X \land Y_i \notin E),$$

where V_X is the set of schematic variables of X and of all of the subtrees thereof; should X be a string, V_X will return a set containing only X if and only if X is a schematic variable, returning an empty set otherwise. Another aspect of importance is that if X and Y are strings, then $X \geq Y$ if and only if X is a schematic variable or syntactically equal to Y. Finally, E represents the set of strings that cannot be considered for substitutions; our implementation disregarded operators, functions, and predicates.

Let S be the set of all strings. Given that $X \geq Y$, the next step is to generate a set containing all of the differences between them. This can be done by defining

$$\Delta(X,Y) = \begin{cases} \{(X,Y)\} & X \in S \land X \neq Y \\ \bigcup\limits_{i\in[|X|]} \Delta(X_i, Y_i) & X \notin S \land X \neq Y \\ \emptyset & X = Y \end{cases}.$$

We say that a set of differences is consistent if and only if it can be constructed into a function. In other words, $\forall x, y \in \Delta(X,Y), x_1 = y_1 \implies x_2 = y_2$, which we represent as $\phi(\Delta(X,Y))$.

Under the assumption that it's consistent, we need a function to apply the set of differences provided by Δ to a particular tree. Denoting it by α, we can define it as

$$\alpha(X, \Delta(X,Y)) = \begin{cases} \Delta(X,Y)_{i,2} & \exists i \in [|\Delta(X,Y)|], X = \Delta(X,Y)_{i,1} \\ X & \forall i \in [|\Delta(X,Y)|], X \neq \Delta(X,Y)_{i,1} \land X \in S \\ (\alpha(X_1, \Delta(X,Y)), \ldots, \alpha(X_{|X|}, \Delta(X,Y))) & \forall i \in [|\Delta(X,Y)|], X \neq \Delta(X,Y)_{i,1} \land X \notin S \end{cases}.$$

Having defined all of the above, the substitution function can, therefore, be defined as

$$\sigma(X,(Y,Y')) = \begin{cases} \alpha(Y',\Delta(Y,X)) & Y \geq X \wedge \phi(\Delta(Y,X)) \\ X & Y \not\geq X \vee \neg\phi(\Delta(Y,X)) \end{cases},$$

where (Y,Y') represents a pattern.

4.4 Generating Paths

A path consists of an ordered list of trees. Given a particular axiom schema, our implementation generates a path by starting from the top node, creating one tree where the axiom schema is not applied and one where it is applied, and applying this algorithm recursively to each of the possibilities generated by the previous step. Finally, once all of the axiom schemata have been applied, all of the generated paths are collected and taken into consideration for the next application of the same set of axiom schemata, and this process repeats itself for a number of steps chosen by the user. More precisely, letting

$$\Sigma(X,(Y,Y')) = \begin{cases} \{X,\sigma(X,(Y,Y'))\} \cup \bigcup\limits_{i\in[|X|]} \{(\ldots,X_{i-1},x,X_{i+1},\ldots) \mid x \in \Sigma(X_i,(Y,Y'))\} & X \notin S \\ \{\sigma(X,(Y,Y'))\} & X \in S \end{cases},$$

we can generate all possibilities of the application of a pattern to a particular tree. From that, denoting A as the last step before the first mistake, we can start with an initial set T_0, where $T_0 = \{(A)\}$. As such, to generate the paths, we need only state that

$$T_i = \bigcup_{((x_1,\ldots,x_m),Y)\in T_{i-1}\times P} \{(x_1,\ldots,x_m,y) \mid y \in \Sigma(x_m,Y)\},$$

where P represents the set of all specified patterns for the algorithm. This process of iterating T_i happens until i reaches a desired value.

For an overview of this section in pseudocode, see Algorithm 2.

Algorithm 2 Generate Paths.

```
1: $Paths \leftarrow \{(Steps_k, ())\}$
2: for $i = 1,\ldots,n$ do
3:     $Copy \leftarrow Paths$
4:     for $((x_1,p_1),\ldots,(x_i,p_i)) \in Copy$ do
5:         for $j \in Patterns$ do
6:             $Paths \leftarrow Paths \cup \{((x_1,p_1),\ldots,(x_i,p_i),(x,j)) \mid x \in \Sigma(x_i,j)\}$
7:         end for
8:     end for
9: end for
10: return Paths
```

4.5 Returning Feedback

Once all paths have been generated, the algorithm picks only the ones whose final step has a solution set that's equal to the solution set of the student's mistaken step.

It is important to note that since only the solution sets need to be equal, which is not as strict as requiring syntactic equality between a generated step and the subsequent step, the algorithm is robust against not having all steps provided or a general lack of detail. While this might come at the cost of potentially returning irrelevant feedback when the algorithm is equipped with patterns that conditionally return the same solution sets without having the same sort of feedback associated, we believe that certain heuristics or statistical models could be used to manage conflicts of that sort by ranking each piece of returned feedback. To that end, under the assumption that the probability that a student will perform many mistakes is low, we chose to implement a heuristic that sorts the list of matches by the number of steps associated to each match in ascending order, putting at the forefront the matches that are considered more common. While many other heuristics were considered to rank the matches, they were not a part of what was researched.

After the matches have been filtered, a function that maps each used pattern to its associated feedback is used. For a complete picture of the algorithm, see Algorithm 3.

Algorithm 3 Feedback Algorithm.

```
1: $Error \leftarrow Determine\ First\ Erroneous\ Step(Steps)$
2: if $Error \neq ()$ then
3:     $Paths \leftarrow \{p \in Generate\ Paths(Error_1, Patterns) \mid \Omega(p_{|p|,1}) = \Omega(Error_2)\}$
4:     if $|Paths| > 0$ then
5:         return $\{(Feedback(p_1), \ldots, Feedback(p_n), Error_3) \mid ((x_1, p_1), \ldots, (x_n, p_n)) \in Paths\}$
6:     else
7:         return $\{(Error_3)\}$
8:     end if
9: else
10:     return $\emptyset$
11: end if
```

4.6 Complexity

One aspect of concern within this project regarding its generality is how the computational complexity of algorithm grows with an increase in the number of patterns and the maximum path length. In particular, this information can help make guided decisions on how the algorithm might be made more efficient in future works.

While the complexity varies significantly with the patterns and input being used, an approximation for the upper bound can trivially be found. Under the assumption that n patterns that are different from the identity pattern and result in k expressions can be applied during each step, the algorithm can generate $(n * k + 1)^m$ paths at most from a single expression, where m represents the maximum path length; the addition of unity comes as a consequence of the necessity of at least one pattern being the identity pattern. As such, an increase in the number of patterns causes a polynomial growth in complexity, whereas the algorithm's complexity scales exponentially with the maximum path length.

Taking the approximation for an upper bound of the algorithm's complexity into account, we can conclude that it is more efficient to add more patterns and reduce the maximum path length. However, while decreasing the maximum path length and increasing the number of patterns, care must be taken not to reduce the algorithm's ability to properly discern the necessary information to provide feedback.

Finally, given that each pattern can be applied independently from the others during each step, parallelization can be implemented trivially, opening avenues for hardware that supports it. While implementing it might not be prove to be too beneficial for a large maximum path length, it might be helpful for time-sensitive tasks, such as a server hosting an implementation of this algorithm responding to requests from clients.

5 Preliminary Results

5.1 Example

Before ascertaining the results, let's manually perform the algorithm. Using only $(”-”, ”+”)$ as a pattern and a maximum path length of one, we'll run through an informal example of how the algorithm ought to be performed.

Given that $x-1=0 \iff x=-1$ is the student's input, let's attempt to provide feedback. First of all, we need to find the first pair of adjacent steps whose solution sets are different; since $\{1\} \neq \{-1\}$, we've found a mistake on the second step. Secondly, each equation within the mistaken step provided must be formatted as a tree, leading $x-1=0$ and $x=-1$ to be formatted as $((”x”, ”-”, ”1”), ”=”, ”0”)$ and $(”x”, ”=”, (”-”, ”1”))$ respectively. Next, we need to exhaustively apply our patterns to the first of those expressions:
$\Sigma(((”x”, ”-”, ”1”), ”=”, ”0”), (”-”, ”+”)) =$

$\{((”x”, ”-”, ”1”), ”=”, ”0”), \sigma(((”x”, ”-”, ”1”), ”=”, ”0”), (”-”, ”+”))\} \cup \{(x, ”=”, ”0”) \mid x \in \Sigma((”x”, ”-”, ”1”), (”-”, ”+”))\} \cup \{((”x”, ”-”, ”1”), x, ”0”) \mid x \in \Sigma(”=”, (”-”, ”+”))\} \cup \{((”x”, ”-”, ”1”), ”=”, x) \mid x \in \Sigma(”0”, (”-”, ”+”))\} =$

$\{((”x”, ”-”, ”1”), ”=”, ”0”)\} \cup \{(x, ”=”, ”0”) \mid x \in \Sigma((”x”, ”-”, ”1”), (”-”, ”+”))\} =$

$\{((”x”, ”-”, ”1”), ”=”, ”0”)\} \cup \{(x, ”=”, ”0”) \mid x \in \{(”x”, ”-”, ”1”), (”x”, ”+”, ”1”)\}\} =$

$\{((”x”, ”-”, ”1”), ”=”, ”0”), ((”x”, ”+”, ”1”), ”=”, ”0”)\}$.

Finally, by checking the solution sets of our acquired paths, we can conclude that $((”x”, ”+”, ”1”), ”=”, ”0”)$ has the same solution set as $(”x”, ”=”, (”-”, ”1”))$; as such, since what led to that path was changing a sign, we may return feedback stating that the student erred on the second step by mistaking a sign.

5.2 Results From a Python Implementation

To perform a preliminary test of the algorithm's ability to provide meaningful feedback, twenty representative examples were chosen and tested with an implementation of this algorithm in Python using SymPy. These examples were manually sampled from what we considered to be common mistakes, such as mistaking signs or factoring incorrectly; our future works will go over results aggregated on a larger scale and under more realistic conditions.

The following is a list of the patterns used for the examples:
$(”+”, ”-”)$, $(”-”, ”+”)$,
$(((”\$1”, ”+”, ”\$2”), ”**”, ”2”), ((”\$1”, ”**”, ”2”), ”+”, (”\$2”, ”**”, ”2”)))$,
$(((”\$1”, ”-”, ”\$2”), ”**”, ”2”), ((”\$1”, ”**”, ”2”), ”-”, (”\$2”, ”**”, ”2”)))$,
$(((”\$1”, ”**”, ”2”), ”+”, (”\$2”, ”**”, ”2”)), ((”\$1”, ”+”, ”\$2”), ”**”, ”2”))$, and
$(((”\$1”, ”**”, ”2”), ”-”, (”\$2”, ”**”, ”2”)), ((”\$1”, ”-”, ”\$2”), ”**”, ”2”))$.

Before interpreting the results, a metric of success ought to be defined. Given the easy way of finding the first step in which a mistake occurred, we consider a result to be successful if and only if one of the found matches returns feedback that would've prevented the mistake. That being said, even if it can only detect which step was mistaken, the algorithm might still prove to be useful to many students.

Given that definition, despite the good results with the limited patterns used, more of them might need to be added to correctly detect certain categories of mistakes. For example, given $49-x^2=0 \iff (7-x)^2=0$ as the input, the equipped patterns would not be sufficient; however, the addition of more patterns, such as $(”\$1”, ((”\$1”, ”**”, (”1”, ”/”, ”2”)), ”**”, ”2”))$, would permit the algorithm to respond aptly.

Table 1 Results From a Python Implementation of the Misuse of Signs.

Input	Erroneous Step	Output	Expected Output
((”x”, ” − ”, ”1”), ” = ”, ”0”), (”x”, ” = ”, (” − ”, ”1”))	Second Step	Sign Error	Sign Error
((”2”, ” + ”, ”x”), ” = ”, ”0”), (”x”, ” = ”, ”2”)	Second Step	Sign Error	Sign Error
((” − ”, ”10”), ” = ”, (”5”, ” ∗ ”, ”x”)), (((”5”, ” ∗ ”, ”x”), ” − ”, ”10”), ” = ”, ”0”), ((”5”, ” ∗ ”, ”x”), ” = ”, ”10”), (”x”, ” = ”, ”2”)	Second Step	Sign Error	Sign Error
(((”7”, ” ∗ ”, ”x”), ” − ”, ”8”), ” = ”, ”0”), ((”7”, ” ∗ ”, ”x”), ” = ”, (” − ”, ”8”)), (”x”, ” = ”, (” − ”, (”8”, ”/”, ”7”)))	Second Step	Sign Error	Sign Error
((”x”, ” − ”, (”x”, ” ∗∗”, ”2”)), ” = ”, ”0”), ((”x”, ” ∗∗”, ”2”), ” = ”, (” − ”, ”x”))	Second Step	Sign Error	Sign Error
(((”5”, ” ∗ ”, (”x”, ” ∗∗”, ”2”)), ” + ”, (” − ”, (”10”, ” ∗ ”, ”x”))), ” = ”, ”0”), (((”x”, ” ∗∗”, ”2”), ” + ”, (” − ”, (”2”, ” ∗ ”, ”x”))), ” = ”, ”0”), ((”x”, ” ∗∗”, ”2”), ” = ”, (” − ”, (”2”, ” ∗ ”, ”x”)))	Third Step	Sign Error	Sign Error
(((”5”, ” ∗ ”, ”x”), ” + ”, ”1”), ” = ”, ”0”), ((”5”, ” ∗ ”, ”x”), ” = ”, ”1”), (”x”, ” = ”, (”1”, ”/”, ”5”))	Second Step	Sign Error	Sign Error
((((”2”, ” ∗ ”, (”x”, ” ∗∗”, ”2”)), ” − ”, ”x”), ” + ”, ”1”), ” = ”, ”0”), ((”2”, ” ∗ ”, (”x”, ” ∗∗”, ”2”)), ” = ”, (”1”, ” − ”, ”x”)), ((”x”, ” = ”, (” − ”, ”1”)), ”|”, (”x”, ” = ”, (”1”, ”/”, ”2”)))	Second Step	Sign Error	Sign Error
(((”x”, ” ∗∗”, ”2”), ” + ”, ”5”), ” = ”, ”0”), ((”x”, ” ∗∗”, ”2”), ” = ”, ”5”), ((”x”, ” = ”, (” − ”, (”5”, ” ∗∗”, (”1”, ”/”, ”2”)))), ”|”, (”x”, ” = ”, (”5”, ” ∗∗”, (”1”, ”/”, ”2”))))	Second Step	Sign Error	Sign Error

Table 2 Results From a Python Implementation of the Misuse of Factoring.

Input	Erroneous Step	Output	Expected Output
(((”x”, ” − ”, ”1”), ” ∗∗”, ”2”), ” = ”, ”0”), (((”x”, ” ∗∗”, ”2”), ” − ”, (”1”, ” ∗∗”, ”2”)), ” = ”, ”0”), ((”x”, ” = ”, (” − ”, ”1”)), ”|”, (”x”, ” = ”, ”1”))	Second Step	Factoring Error	Factoring Error
(((”x”, ” ∗∗”, ”2”), ” + ”, ((”x”, ” − ”, ”10”), ” ∗∗”, ”2”)), ” = ”, ”0”), (((”2”, ” ∗ ”, (”x”, ” ∗∗”, ”2”)), ” − ”, ”100”), ” = ”, ”0”), ((”x”, ” = ”, (” − ”, (”50”, ” ∗∗”, (”1”, ”/”, ”2”)))), ”|”, (”x”, ” = ”, (”50”, ” ∗∗”, (”1”, ”/”, ”2”))))	Second Step	Factoring Error	Factoring Error
((”x”, ” ∗∗”, ”2”), ” = ”, (” − ”, ((”x”, ” − ”, ”1”), ” ∗∗”, ”2”))), ((”x”, ” ∗∗”, ”2”), ” = ”, (”1”, ” − ”, (”x”, ” ∗∗”, ”2”))), (”x”, ” = ”, ((”1”, ”/”, ”2”), ” ∗∗”, (”1”, ”/”, ”2”)))	Second Step	Factoring Error	Factoring Error
((((”x”, ” + ”, ”1”), ” ∗∗”, ”2”), ” + ”, (” − ”, (”2”, ” ∗ ”, ”x”))), ” = ”, ”0”), ((((”x”, ” ∗∗”, ”2”), ” + ”, ”1”), ” + ”, (” − ”, (”2”, ” ∗ ”, ”x”))), ” = ”, ”0”), (”x”, ” = ”, ”1”)	Second Step	Factoring Error	Factoring Error
(((”x”, ” ∗∗”, ”2”), ” + ”, (”1”, ” ∗∗”, ”2”)), ” = ”, ”0”), (((”x”, ” + ”, ”1”), ” ∗∗”, ”2”), ” = ”, ”0”), (”x”, ” = ”, (” − ”, ”1”))	Second Step	Factoring Error	Factoring Error
(((”x”, ” − ”, ”5”), ” ∗∗”, ”2”), ” = ”, ”0”), (((”x”, ” ∗∗”, ”2”), ” − ”, (”5”, ” ∗∗”, ”2”)), ” = ”, ”0”), ((”x”, ” = ”, (” − ”, ”5”)), ”|”, (”x”, ” = ”, ”5”))	Second Step	Factoring Error	Factoring Error
((((”x”, ” − ”, ”6”), ” ∗∗”, ”2”), ” + ”, (”2”, ” ∗ ”, (”x”, ” ∗∗”, ”2”))), ” = ”, ”0”), (((”3”, ” ∗ ”, (”x”, ” ∗∗”, ”2”)), ” − ”, ”36”), ” = ”, ”0”), ((”x”, ” = ”, (” − ”, (”12”, ” ∗∗”, (”1”, ”/”, ”2”)))), ”|”, (”x”, ” = ”, (”12”, ” ∗∗”, (”1”, ”/”, ”2”))))	Second Step	Factoring Error	Factoring Error
((”7”, ” ∗ ”, (”x”, ” ∗∗”, ”2”)), ” = ”, (” − ”, ((”x”, ” − ”, ”2”), ” ∗∗”, ”2”))), ((”7”, ” ∗ ”, (”x”, ” ∗∗”, ”2”)), ” = ”, (”4”, ” − ”, (”x”, ” ∗∗”, ”2”))), (”x”, ” = ”, ((”1”, ”/”, ”2”), ” ∗∗”, (”1”, ”/”, ”2”)))	Second Step	Factoring Error	Factoring Error
((((”5”, ” − ”, ”x”), ” ∗∗”, ”2”), ” + ”, (” − ”, (”24”, ” ∗ ”, ”x”))), ” = ”, ”0”), (((”25”, ” − ”, (”x”, ” ∗∗”, ”2”)), ” + ”, (” − ”, (”24”, ” ∗ ”, ”x”))), ” = ”, ”0”), ((”x”, ” = ”, (” − ”, ”25”)), ”|”, (”x”, ” = ”, ”1”))	Second Step	Factoring Error	Factoring Error
((”49”, ” − ”, (”x”, ” ∗∗”, ”2”)), ” = ”, ”0”), (((”7”, ” ∗∗”, ”2”), ” − ”, (”x”, ” ∗∗”, ”2”)), ” = ”, ”0”), (((”7”, ” − ”, ”x”), ” ∗∗”, ”2”), ” = ”, ”0”), ((”7”, ” − ”, ”x”), ” = ”, ”0”), (”x”, ” = ”, ”7”)	Third Step	Factoring Error	Factoring Error

Another issue with this approach is that it requires the solution sets of all of the expressions to be computable by the CAS. Given the task of solving it for polynomials of degree two or lower, this was not a problem, but it is possible that complications may arise from generalizations of this algorithm.

6 Future Work

Given what was discussed in the section about the algorithm's complexity, one avenue of research is finding which patterns should be used to reduce the number of steps needed to provide apt feedback. In particular, the plan is to develop a method of automatically generating useful patterns and manually label them with appropriate feedback; currently, we plan to research how machine learning could be used to do such.

Beyond that, as mentioned in our section about preliminary results, experiments will be performed on a larger scale with open-source software with CAS capabilities. The hope is to be able to find software that can serve as a front-end interface for the presented algorithm and test its capabilities on a larger scale and with a greater variety of inputs and patterns. Should the results show promise, other avenues of improvement will be researched.

References

1 Anderson Pinheiro Cavalcanti, Arthur Barbosa, Ruan Carvalho, Fred Freitas, Yi-Shan Tsai, Dragan Gašević, and Rafael Ferreira Mello. Automatic feedback in online learning environments: A systematic literature review. *Computers and Education: Artificial Intelligence*, 2:100027, 2021. doi:10.1016/j.caeai.2021.100027.

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

3 Mark Jellicoe and Alex Forsythe. The development and validation of the feedback in learning scale (fls). *Frontiers in Education*, 4, 2019. doi:10.3389/feduc.2019.00084.

4 Raymond W. Kulhavy and William A. Stock. Feedback in written instruction: The place of response certitude. *Educational Psychology Review*, 1(4):279–308, 1989. doi:10.1007/BF01320096.

5 Susanne Narciss. Feedback strategies for interactive learning tasks. *Handbook of research on educational communications and technology*, 3:125–144, 2008. URL: https://www.routledgehandbooks.com/doi/10.4324/9780203880869.ch11.

6 Susanne Narciss, Elsa Hammer, Gregor Damnik, Kerstin Kisielski, and Hermann Körndle. Promoting prospective teacher competencies for designing, implementing, evaluating, and adapting interactive formative feedback strategies. *Psychology Learning & Teaching*, 20(2):261–278, 2021. doi:10.1177/1475725720971887.

7 Ernesto Panadero and Anastasiya A. Lipnevich. A review of feedback models and typologies: Towards an integrative model of feedback elements. *Educational Research Review*, 35:100416, 2022. doi:10.1016/j.edurev.2021.100416.

8 Scott A. Schartel. Giving feedback – an integral part of education. *Best Practice & Research Clinical Anaesthesiology*, 26(1):77–87, 2012. Challenges in Anaesthesia Education. doi:10.1016/j.bpa.2012.02.003.

Cloud of Assets and Threats: A Playful Method to Raise Awareness for Cloud Security in Industry

Tiange Zhao
Siemens AG, München, Germany
Universität der Bundeswehr München, Germany

Ulrike Lechner
Universität der Bundeswehr München, Germany

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

Ece Ata
Siemens AG, München, Germany
Technische Universität München, Germany

Abstract

Cloud computing has become a convenient technology widely used in industry, providing profit and flexibility to companies. Many enterprises embrace cloud service by migrating their products and solutions from on-premise to cloud environments. Cloud assets and applications are vulnerable to security challenges if not adequately protected. Regulations, standards and guidelines aim to enforce cloud security controls in the industry and practitioners need training to raise awareness of cloud security issues and learn about the defense mechanisms and controls. We propose a serious game Cloud of Assets and Threats (CAT) for enhancing cloud security awareness of industrial practitioners. This study extends first results of applying such a serious game in industry [25] and refines its design in two iterations. In the first design iteration, we implemented a digital game platform with six attack scenarios and developed a new player versus environment gaming mode. In the second design iteration, we adjusted the attack scenarios and introduced different difficulty levels for the scenarios. We present, analyse, and discuss the game events. We conclude that CAT is a promising method to raise awareness for cloud security in the industry.

2012 ACM Subject Classification Computer systems organization → Cloud computing; Social and professional topics → Computer and information systems training

Keywords and phrases Cloud security, Cloud control matrix, Shared-responsibility model, Industry, Training, Gamification

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.6

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.

1 Introduction

Nowadays, companies are moving their products and solutions from on-premises to cloud deployment. The size of the cloud computing and hosting market grows at a steady speed, as shown statistically [22]. On the one hand, the convenience and flexibility of cloud services contribute to the market growth. On the other hand, cloud deployment could expose industry systems to serious cybersecurity threats. The traditional vulnerabilities become more dangerous as cloud deployment increases system exposure and new types of cloud specific threats, e.g. breach of cloud storage object, are emerging. Several stakeholders, including cloud asset managers, cloud asset owners and cloud service providers, are involved in the design and development life cycle of cloud services. Each stakeholder needs to understand its role and responsibility to secure a cloud asset. Standards and white papers aim to regulate the industry field and provide the basis for a shared understanding of roles and responsibilities in securing cloud assets among all stakeholders involved. However, security training is important to convey concepts and strategy to decision makers, developers, system architects and cloud service users. In cybersecurity, serious games can be a method to raise awareness and enable learning by doing in a playful, safe environment. This research contributes to enrich and improve the existing training methods in cloud security through a serious game. Our goal is to increase the level of cloud security by increasing cloud security awareness with a special focus on shared roles and responsibilities in securing cloud assets.

We have designed a board game, Cloud of Assets and Threats(CAT), which can be used for cybersecurity educational purposes for industry practitioners. We address different roles and their shared responsibilities to secure cloud asset as one of the topics of our serious game. This study presents a refinement of our board game dedicated to tackling different aspects of cloud security and facilitating awareness and improving cloud security for the industry. This initial prototype is described in our previous work [25]. It is a game in which attacker and defender develop strategies to attack or defend the cloud assets. This study presents a refinement: single players or teams play defenders for cloud-based systems in various attack scenarios and a evaluator component assesses the effectiveness of defense. This refinement of the game logic allows for a more effective training as defense is what professionals in industry need to understand. Two contexts of the game are used in the two design iterations: in the first design iteration, the game is part of a larger serious game experience, in the second it is a stand-alone game used to bring contents from a training session into practical application. These two different contexts allow to understand how to tailor the game to specific user groups and insights on raising awareness. This paper describes the design, implementation, and evaluation of such a game performed in first and second design iterations. We reflect on the effectiveness of the game to raise awareness and the understanding of how to tailor the game to various user groups.

This article is structured in the following sections: section 2 introduces the related work in the areas of information security standards relevant for cloud security, and serious games in information security. Section 3 gives an overview of the design science research method we followed in our research. Section 4 describes the framework of our game and the two design iterations we went through. Section 5 shares the game dynamic data and results we collected from our game events and our thinking upon them. Section 6 concludes this work and gives an outlook into the future research direction.

2 Related work

Existing information security standards in the industry describe the requirements and necessary controls on cloud security. The cloud controls matrix (CCM) [1] from cloud security alliance (CSA) provides mapping and comparison of various control specifications and 44 relevant industrial standards. Best known is the ISO/IEC 2700X family [8, 18, 19, 20], which provides information security requirements in general, or specifically address information security controls for cloud service. In addition to the standards mentioned above, MITRE ATT&CK cloud matrix categorizes possible attacks towards cloud systems based on real-world observation and groups them in different techniques and tactics [5]. For each possible attack, the defense mechanisms are listed. Our work uses the MITRE framework as a reference for the attack and defense models. This helps to bring industry standard requirements to the practitioners by introducing the cloud security concepts through a serious game.

Dörner et al. [7] presents seminal concepts for designing serious games. They describe that serious games are designed with a goal instead of pure entertainment. It guides us in the design and instantiation of our game. The goal of the serious game is to raise the cybersecurity awareness of the participants.

There are numerous examples of serious games application in information security that help raise awareness and serves educational purposes directly or indirectly. Frey et al. [9] focus on the human factor and use a serious game as a novel method to study the decision-making process in the field of information security. Shostack lists more than forty tabletop serious games for cybersecurity [23], and the list is still growing. One of them is the game Riskio [14], which is successful in increasing cybersecurity awareness for people without technical backgrounds working in organizations. Apart from tabletop games, the work of Gasiba et al. [10, 11, 13, 12] present a serious game inspired by the capture-the-flag genre, yet dedicated to software developers in the industry to raise their awareness on secure programming and improve their secure coding skills in a variety of programming languages.

However, none of the games mentioned above specifically addresses the issues in cloud security, e.g., the shared-responsibility model and the newborn types of threats that are unique in the cloud environment. This article is based on our design and evaluation of a cloud security game firstly introduced in our previous research [25, 24]. This work extends and improves the existing game prototype by implementing a digital platform, designing game mode variations, and refining the game to address the feedback collected.

3 Method

Our research is guided by the design science paradigm proposed by Hevner et al. [16, 15]. They describe the design of a useful artefact as a creative search process. Hevner et al. describe the cycle of design & implement and justify & evaluate to be the core of design science. We use explanatory design theory proposed by Baskerville et al. [6] to guide design and evaluation. Baskerville et al. present an approach which differentiates between the process to design the artefact and the evaluation of the artefact and emphasises on the design process to be a creative endeavor. This approach describes a general design solution as a class of problems that relates a set of general components to a set of general requirements [17].

We identify the requirements and relate them to components in our design. General requirements (GR) for the game are listed as the following:

- **GR1**: The game artefact should reflect facts or characteristics about cloud security.
- **GR2**: The game artefact should be understandable and straightforward to the players.
- **GR3**: Players' cloud security awareness should increase by playing the game.
- **GR4**: The game should be fun and interactive in that it motivates the participants to learn about cloud security.

General components (GC) are also identified and could be mapped to the general requirements:

- **GC1, mapped to GR1**: Attack scenarios developed are based on actual attacking activities observed from real-world as well as standards as the MITRE attack vectors.
- **GC2, mapped to GR2**: Tutorials and different difficulty levels are available in the game.
- **GC3, mapped to GR3**: The game concept encourages players to act as defenders against the game´s attack scenarios. This addresses the specific skills needed in the everyday job.
- **GC4, mapped to GR4**: Participants see hints and suggestions from the platform to strengthen their defense strategy until the threshold of defense success rate is reached.

Table 1 Overview of the game events in iteration 1 and 2.

	Iteration 1 (I1)	Iteration 2 (I2)
Number of participants	17	14
Date	January 26, 2022	March 15, 2022
Teams / single players	4 Teams	14 Single players
Game time	~60 mins	30 mins
Number of submissions	175	656
Avg. submission per scenario per team/player	7.3	7.8

This study builds on a game idea and a prototype with its evaluation presented in [25]. This study continues the creative search process for a solution that is more effective in raising the level of cloud security awareness and to a deeper understanding of players needs, and players level of cloud security knowledge. Our game design is developed in two design iterations. In both iterations we organized game events to collect feedback for evaluation of the artefact. Table 1 provides an overview of the two iterations and the serious game event conducted as empirical basis for the evaluation. All players are professionals in industry and they are invited to game event after awareness training or during a CyberSecurity Challenges (CSC) event [13, 12], which normally has a group size of 10 to 20 trainees or participants. The table provides information about the number of participants, whether the game was played in single player or team mode, the number of solution submissions done in the game and the average number of submissions per team or per single player.

4 Game design

In this section, we describe the design of our game. Firstly, we briefly introduce our game prototype as a base line. Then we describe our game design in the first iteration. At last, we explain the improvement and adjustment we made in the second iteration.

4.1 Baseline – the Game prototype

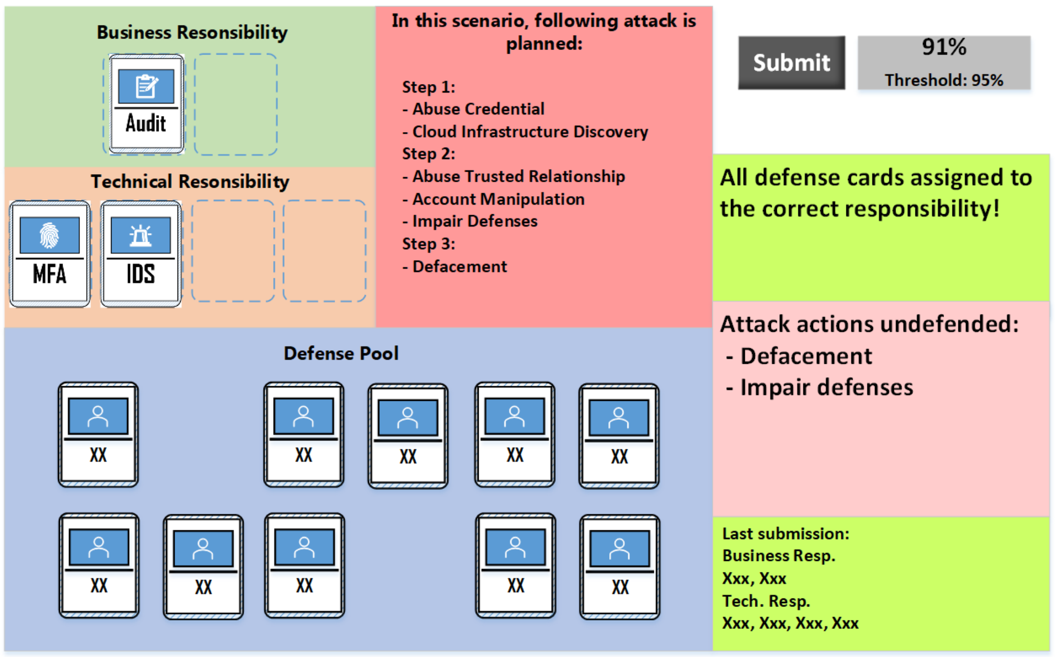
The original game artefact is an online board game. There is a game master (GM), and players play either as defender team or as attacker team against each other. The defender team uses the defense action cards to build a defense-in-depth defense plan within a given time by selecting cards and assigning the cards to the responsible roles. The attacker team uses the attack action cards to build a step-by-step attack plan. When time is up, an automated evaluator takes both the attack and defense plans as input and calculates the defense plan's probability of withstanding the attack plan. The higher the probability, the more likely the defender team would win. A wheel-of-fortune takes the probabilities calculated by the evalulator components and determines the winner. A more detailed description of the game prototype can be found in our previous publication [25]. This game was evaluated positively on the basis of a limited empirical basis. The game logic is validated and the trial run participants found the game engaging and they learned about how to build an effective defense for cloud assets in real world. Notwithstanding this first positive evaluation, we did an analysis to see whether there is room for improvement. In our previous work, we conducted online trial runs with the game prototype, where players played versus players (PvP) integrated either in defender team or attacker team. There are some disadvantages of such a game mode:

- The learning effect for the attacker team is not well aligned with the goal, since professionals need skills in defense, not in attack strategies.
- This game mode requires a game master to coordinate and the game master cannot take care of the training activities and questions while managing the game.
- There must be at least two players to build two teams, and the size of a team cannot be too big for effective communications and a good sense of involvement. So the game does not scale well in an industrial context.

4.2 First design iteration

In the first design iteration (I1), the previously-mentioned disadvantages are addressed. We implement a digital platform with a player versus environment (PvE) game mode and as a part of the environment, we build six attack scenarios derived from real-world cloud security attack activities. The implementation is based on the game board we designed as baseline in the prototype and we developed the digital platform using konva[4]. Konva is HTML5 2d canvas JavaScript library for desktop and mobile applications, which provides our the necessary features for the development. We introduce each element one by one subsequently and share the evaluation of the first iteration.

4.2.1 Digital platform



In this scenario, following attack is planned:

Step 1:
- Abuse Credential
- Cloud Infrastructure Discovery

Step 2:
- Abuse Trusted Relationship
- Account Manipulation
- Impair Defenses

Step 3:
- Defacement

All defense cards assigned to the correct responsibility!

Attack actions undefended:
- Defacement
- Impair defenses

Last submission:
Business Resp.
Xxx, Xxx
Tech. Resp.
Xxx, Xxx, Xxx, Xxx

Figure 1 Mock-up of the cloud security game board.

The digital platform is a single-page web application, on which the players play the game. When a player accesses the game homepage, the game interface depicted in figure 1 will be displayed. The web application has a back-end and a front-end. The evaluator algorithm runs in the back-end calculating the quality of the defense plan against the given attack. The front-end displays the cards and game interface, players can drag and drop cards. The defense plan area is designed with magnetic effect. When the player drags the cards near to the area, the card would be pulled to one of the reserved space. Magnetic effect assures the player that this is the place where the cards should be placed. If cards are placed outside this area, it would be sent to its original position in the defense pool. The The game interface has seven areas with specific purpose and functionality. Clockwise from the top left are:

- Defense Plan Area – The defense plan area consists of the business and technical responsibility zones. The players are supposed to pick cards from these sub-areas and place them in the correct responsibility zone.
- Attack Plan Description Area – This area describes the hypothetical attackers plan, step-by-step. In the mock-up of figure 1, we show the attack scenario I1S1. The players can search for technical terms that the attackers use in each step to build an effective defense plan.
- Submit Button and Threshold – The player can send the selected defense plan to the back-end for evaluation by hitting the Submit button. The calculated probability of the submitted defense plan withstanding the hypothetical attack is displayed in the top right corner. The player completes the challenge if the threshold is reached.
- Hint Area for Responsibility – Hint Area for Responsibility is the first hint area. It shows in the last submission if all the cards are assigned to the correct responsibility. The wrongly assigned ones will be listed here, and the players can improve their defense plan based on the hint.
- Hint Area for Undefended Attacks – Hint Area for Undefended Attacks is the second hint area. Based on the last defense plan submission, it shows the undefended cards in the given attack plan. The players can improve their defense success rate by trying to cover as many attack actions as possible.
- Last Submission Area – The defense plan in the last submission is shown here. It gives the player a reference as they are trying to improve their Defense.
- Defense Pool – The defense pool shows a wide range of defense cards from which the players can choose. There are 24 cards in total. The details regarding each defense card can be found in our previous work [25, 24].

In the game event of the first design iteration, CAT was a part of a full-day CyberSecurity Challenges (CSC) event [13, 12]. The game engine was hosted in a docker container in a cloud environment, and the backend logged the player submissions and access. After the end of the event, data were downloaded from the cloud virtual machine instance, and then the instance was cleaned up.

4.2.2 Player-vs-environment game mode

The player-vs-environment (PvE) mode allows all game participants to address cloud security from the security or defense perspective. We defined six different scenarios from real-world hacking activities as described in the section 4.2.3. In game, all players build defense plans to stop the attack.

Note that the PvE mode allows more flexibility with little overhead - there is no limitation on the number of players, and actions from a game master are not required. Participants can join as single players or play in teams. Since the task is to defend themselves against pre-defined attack scenarios, players dedicate the time spent in the event to learn about how to be a good defender for their cloud assets instead of learning to be strategic hackers. We argue that this new mode is more efficient in raising awareness for cloud security than the baseline game prototype. This is in line with design decisions we took in the design of the CyberSecurity Challenges and we argue that such a defense-only perspective might be successful in optimizing the outcome for the players gain.

Table 2 The attack scenarios Scenario 1 (S1) – Scenario 6 (S6) used in iteration 1 and 2.

	Attack Action Card	Scenario						
		I1S1	I1S2	I1S3	I1S4	I1S5	I1S6	I2S6
Step 1: Initial Access	Exploit Public-Facing Application			x	x		x	
	Abuse Credential	x				x		
	Cloud Infrastructure Discovery	x	x			x		x
	Network Service Discovery		x		x			x
	Brute Force			x			x	
Step 2: Launch Attack	Abuse Trusted Relationship	x	x	x		x	x	x
	Cloud Storage Breach			x	x			x
	Account Manipulation	x					x	
	Exploit Unused Region							x
	Impair Defenses	x	x	x	x	x	x	
	Infrastructure Manipulation		x					
	Monitoring Escaping				x	x		
Step 3: Make Impact	Defacement	x					x	
	Resource Hijacking		x		x	x		x
	Denial of Service			x				

4.2.3 Attack scenarios

As mentioned above, the game includes six attack scenarios in the PvE game mode. In the first design iteration, we presented attack scenarios I1S1–6 as shown in Tab. 2. Each scenario consists of three attack steps: Initial Access, Launch Attack and Make Impact. That reflects the attack kill chain [2]. Each step in the kill chain uses different attack actions, each represented in an attack card. Each step may consist of several attack actions.

Players are instructed to defend their cloud assets against these kill chains. Their task is to defend themselves by selecting helpful defense cards and assigning the cards to the responsible organizational role.

I1S1–4 are based on the examples listed in MITRE ATT&CK Cloud Matrix [5]. For instance, I1S1 is derived from the hacking activity of threat group Lazarus Group [3] and Sandworm Team [3]. In the first step, to gain initial access, the attackers have two attack actions in parallel: Abuse Credential and Cloud infrastructure Discovery. Abuse Credential means the attacker tries to obtain and abuse account credentials to access the system [21]. "Cloud Infrastructure Discover" means the attackers discover resources that are available within the environment [21]. In the second step to launch the attack, the attackers have three attack actions in parallel: Abuse Trusted Relationship, Account Manipulation, and Impair Defense. Abuse Trusted Relationship means the attackers breach the organization to access the protected resource in the environment [21]. Account Manipulation refers to the attack that they manipulate the stolen account to open a backdoor and maintain access to victim systems [21]. In the meantime, Impair Defenses symbolizes that they maliciously modify the components of a victim environment in order to hinder defensive mechanisms [21]. In the last step, Make Impact, the attackers choose Defacement, which means they modify the homepage of the enterprise homepage to cause a panic and gain fame [21]. I1S5 is a collection of historically attackers' most-chosen cards from early trial runs of PvP mode. I1S6 is a collection of attack cards that can be defended by the least number of defense cards, which are supposed to be difficult to defend. The table 2 gives an overview of the pre-defined attack cards. Every scenario consists of 3 steps, and each step has 2, 3, or 1 card(s) chosen. In the table, "X" means this attack action card is chosen for the given scenario.

In all the scenarios, the players are required to build a complete defense plan from the beginning. The defense plan is assessed by the evaluator and a probability of withstanding the given attack scenario will be calculated. Players solve the challenge if the calculated probability is higher than 90%. Additionally, during the game time, a brief description of each scenario is distributed to the players. In the description, each attack action on the three attack steps are explained. The players can use the description as assistance material or background information.

4.2.4 Evaluation of first design iteration

As shown in table 1, we conducted the game event on January 26, 2022. In the game event, 17 participants formed four teams with 4 or 5 players per team. One player in each team shared the screen with other teammates and was in charge of submissions. Submissions were first discussed within the team and agreed upon by all the team members. The six scenarios of I1 were included in a full day CyberSecurity Challenge [13, 12] event as six challenges one after another. The teams were free to choose when to work on these challenges within the full day event. Therefore, we cannot precisely determine the exact game time. According to our observation, each team spent about 60 minutes on the six scenarios. We collected 175 valid submissions from the four teams over six scenarios. On average, each team makes 7.3 submissions per scenario. A submission was captured when player hit the “Submit” button. It included the defense cards the team has chosen in a certain scenario. Feedback from the players were collected and analyzed as evaluation of the first design iteration. The collected feedback and observation can be summarized as following:

- **S1:** Players enjoyed the game and found it helpful for understanding cloud security.
- **S2:** Players liked the interactive game as a hands-on exercise.
- **S3:** For some, it is challenging to build a full defense plan from scratch.
- **S4:** For some, the feedback of incorrectly assigned cards is not clear.
- **S5:** Some teams discovered some cards, e.g. “Account Management” was useful in all the scenarios and has an strong positive effect on the success rate.
- **S6:** All teams solved all scenarios in approximately one hour.

S1 and **S2** are positive feedback that implies the game logic is correct and the game itself is interesting and helpful for player. So we kept organizing game events and maintained the game logic in the second design iteration.

4.3 Second design iteration

We aim to improve our game artefact by addressing the issues exposed in the first design iteration. After adapting the game artefact, we conducted another game event to validate the improvement on March 15, 2022, as shown in table 1. In this iteration, the game event was a standalone game event following a full-day security awareness training. Recall, in the first design iteration, the cloud game was part of a full-day CyberSecurity Challenge. In this section, we introduce the changes we made and summarize the second design iteration (I2).

4.3.1 Design changes in second design iteration

To address the issues exposed in the first design iteration, we made the following changes in the second iteration:

- The first two scenarios were used as tutorials. We pre-selected cards on I1S1 and I1S2 and guided the players through the scenarios as a tutorial. This was meant to reduce difficulty as reflected in **S3**.

Table 3 Comparison of the presented scenarios in I1 and I2.

	Iteration 1 (I1)	Iteration 2 (I2)
I1S1	Full scenario	Tutorial scenario w. pre-selected cards
I1S2	Full scenario	Tutorial scenario w. pre-selected cards
I1S3	Full scenario, same for I1 and I2	
I1S4	Full scenario, threshold = 90%	Full scenario, threshold = 95%
I1S5	Full scenario, threshold = 90%	Full scenario, threshold = 95%
I1S6	Full scenario, only presented in I1	
I2S6	Full scenario, only presented in I2	

- We gave more detailed information to the players: We demonstrated in I1S1 that the success rate can be improved by covering more attack cards. In I1S2, we demonstrated that the success rate can be improved by correctly assigning a misplaced card to the proper role to cover **S4**.
- Transforming I1S1 and I2S2 into a tutorial reduced the difficulty of the game. We increased the success rate threshold to 95% from 90% for I1S4 and I1S5 to make the game more challenging. Note that **S6** implies that there is room for higher difficulty levels.
- As mentioned in **S5**, there are some overlaps in the attack scenarios. On one hand, repetition enhances learning. On the other hand, we could use the opportunity to show the importance of other defense mechanisms. So we decided to replace I1S6 with a new scenario I2S6 in which the defense card "Account Management" is not helpful. The last column of table 2 shows the exact attack actions in I2S6, which includes a new attack card, "Exploit Unused Region." That card means that the attacker creates cloud instances in new geographic service regions to evade detection [21]. Note that this type of attack is not covered in previous scenarios and it is an attack not often seen in practice.

Table 3 shows a summary of the differences in each scenario presented in iteration 1 (I1) and iteration 2 (I2).

4.3.2 Evaluation of second design iteration

Based on the feedback we collected in the first iteration, we made improvements in second design iteration. As shown in table 1, in the game event there were 14 participants, and everyone interacted with the platform as single players. The game was meant to deepen the understanding of cloud security topics covered in a "classic" training the day before the game.

Since the defense plans in the first two scenarios were partially pre-selected, we assigned 30 minutes for the players to solve all the six scenarios. We collected 656 valid submissions from the 14 participants over the six scenarios. On average, one player had 7.8 submissions on each scenario, slightly more than in Iteration 1. One of the reasons contributing to this might be that the single players cannot discuss with the teammates, so they sometimes choose the strategy to figure out the solution by trial and error.

As expected, with the help of the tutorials, we did not receive any feedback like **S3** or **S4** after the second game event. Even though we increased the threshold for I1S4 and I1S5, participants understood the tasks well and some managed to solve all the scenarios. Additionally, we continued to receive positive feedback as **S1** and **S2**. Some participants mentioned: "It would be nice to have more exercises like this one." and "Last exercise was quite good, quite interactive. The game was very good!". It seems interactive hands-on exercises are highly welcomed by the participants and add fun to the overall training experience. Also, the participants learned about the cloud security concept by building defense plans that are more solid and have better coverage. One of the participants shared the experience of reaching 99% of success rate for all six scenarios and spending quite some time trying to get 100%. It was not mentioned at the beginning that 100% security exists neither in the real world nor in the game. It was very well reflected in the game. Our lesson learned is that this needs to be better explained when introducing the game to the participants to avoid confusion.

5 Result and analysis

In the game event of both iterations, we captured the submissions of the teams and players. In this section, we will share our observations in the captured data.

5.1 Growth of success rate

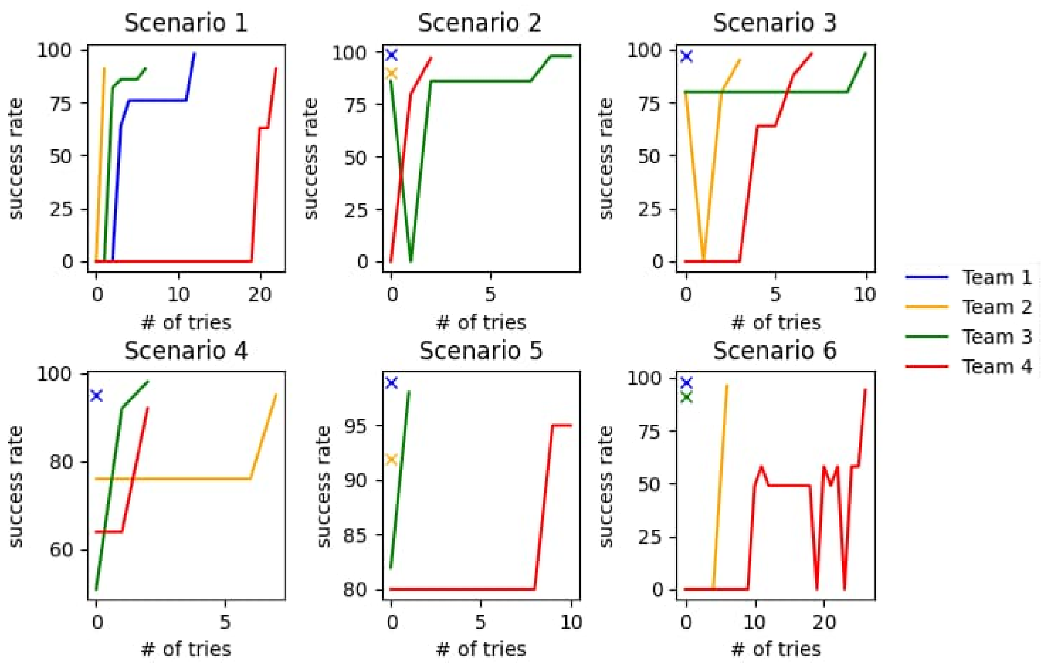
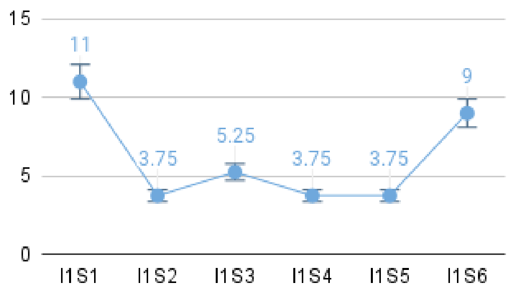


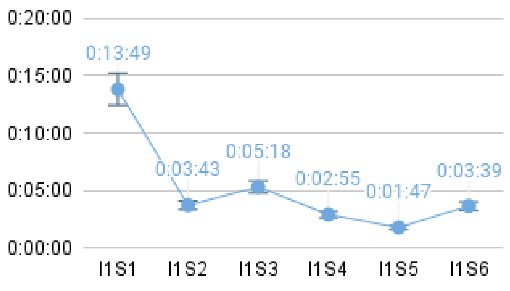
Figure 2 Success rate rising during gaming process per scenario.

In I1, all the teams are asked to build full defense plans for six attack scenarios. Figure 2 illustrates the growth of success rate across the submissions in six scenarios. The X-axis is the order of submission, and the Y-axis is the success rate of each submission. Despite some turbulence, the success rate increases as the team continues to try. The observed tendency in submission data shows that the game logic is understandable to the players. It implies that the participants learned from each submission and used the hint to improve their defense plan until the threshold was reached for them to solve the task. As in real life, 100% security does not exist. In our game, we set the threshold for a success defence to be 90%. The drops of success rate in scenario 2, 3 and 6 suggest that the teams could also make mistakes and weaken the defense plan, but finally every team manage to correct the mistakes and reach the threshold. The "x" symbol on the figure means that the team solved the task on the first try. We observed that Team 1 was having some quick success in solving the scenarios. It could be a sign that the difficulty level should be increased.

5.2 Average number of attempts and duration to complete each scenario



(a) Average submission numbers per scenarios.



(b) Time spent on each scenario.

Figure 3 Average data collected from trial runs.

Figure 3a illustrates the average number of attempts the teams made in six scenarios in I1. Teams made more attempts in I1S1 since it was the first scenario. As players became more familiar with the platform, the attempts number reduces in I1S2–5. We observe no significant difference among I1S2–5, since they are designed to be equally difficult. Besides, figure 3b presents the average time spent for each scenario, which shares the same tendency with figure on the left. In the last scenario I1S6, the teams made more attempts. However, if we compare it with right figure in 3a, there is no significant difference in the spent time. This might suggest the players entered the gaming mode, i.e. the players try to beat the game engine without considering the learning goal. We need to investigate further this observation and understand the reason behind it.

6 Conclusion and future work

In this paper, we presented CAT, an online board game with attack scenarios in different difficulty levels designed for industrial practitioners. We implemented and improved CAT in two iterations starting from a prototype, with a limited yet positive evaluation. Based on the feedback collected in the first iteration, we refined and validated the game in the second iteration. CAT provides an interactive and enjoyable way to convey critical concepts in cloud security. By engaging in attack scenarios derived from actual attacks, CAT enables a proactive thinking. Through building and strengthening a defense plan, the participants can gain a straightforward impression of cloud security roles and responsibilities and raise awareness about cloud security in the industry. In the future, we would like to collect further feedback for improvement in additional game events.

References

1 Cloud Security Alliance. Cloud controls matrix v4. `https://cloudsecurityalliance.org/artifacts/cloud-controls-matrix-v4/`, 2021.

2 Michael J Assante and Robert M Lee. The industrial control system cyber kill chain. *SANS Institute InfoSec Reading Room*, 1, 2015.

3 MITRE ATT&CK. Hacking group. `https://attack.mitre.org/groups/`, May 2017.

4 MITRE ATT&CK. Techniques. `https://attack.mitre.org/techniques/`, May 2017.

5 MITRE ATT&CK. Mitre att&ck cloud matrix. `https://attack.mitre.org/versions/v8/matrices/enterprise/cloud/`, 2020.

6 Richard L. Baskerville and Jan Pries-Heje. Explanatory design theory. *Business & Information Systems Engineering*, 2:271–282, 2010. URL: `https://aisel.aisnet.org/bise/vol2/iss5/2`.

7 Ralf Dörner, Stefan Göbel, Wolfgang Effelsberg, and Josef Wiemeyer. *Serious Games: Foundations, Concepts and Practice*. Springer, 2016.

8 International Organization for Standardization. Iso/iec 27001 information security management. `https://www.iso.org/isoiec-27001-information-security.html`, 2017.

9 Sylvain Frey, Awais Rashid, Pauline Anthonysamy, Maria Pinto-Albuquerque, and Syed Asad Naqvi. The good, the bad and the ugly: a study of security decisions in a cyber-physical systems game. *IEEE Transactions on Software Engineering*, 2017.

10 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. IEEE, 2019.

11 Tiago Espinha Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. Sifu-a cybersecurity awareness platform with challenge assessment and intelligent coach. *Cybersecurity*, 3(1):1–23, 2020.

12 Tiago Espinha Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. Cybersecurity challenges for software developer awareness training in industrial environments. *Innovation Through Information Systems. WI 2021. Lecture Notes in Information Systems and Organisation*, 47, 2021. `doi:https://doi.org/10.1007/978-3-030-86797-3_25`.

13 Tiago Espinha Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. Cybersecurity challenges: Serious games for awareness training in industrial environments. *Federal Office for Information Security (ed.): Germany. Digital. Secure. 30 Years BSI - Proceedings of the 17th German IT Security Congress 2021*, February 2021.

14 Stephen Hart, Andrea Margheri, Federica Paci, and Vladimiro Sassone. Riskio: A serious game for cyber security awareness and education. *Computers & Security*, 95:101827, 2020. `doi:10.1016/j.cose.2020.101827`.

15 Alan Hevner. A three cycle view of design science research. *Scandinavian Journal of Information Systems*, 19:4, January 2007.

16 Alan Hevner, Salvatore March, and Jinsoo Park. Design science in information systems research. *Management Information Systems Quarterly*, 28:75–105, 2004.

17 IEEE. IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990. `doi:10.1109/IEEESTD.1990.101064`.

18 ISO27002. Iso/iec 27002:2013information technology – security techniques – code of practice for information security controls. `https://www.iso.org/standard/54533.html`, 2013.

19 ISO27017. Iso/iec 27017:2015 information technology – security techniques – code of practice for information security controls based on iso/iec 27002 for cloud services. `https://www.iso.org/standard/43757.html`, 2015.

20 ISO27018. Iso/iec 27018:2019information technology – security techniques – code of practice for protection of personally identifiable information (pii) in public clouds acting as pii processors. `https://www.iso.org/standard/76559.html`, 2019.

21 konva. Konva.js - html5 2d canvas js library for desktop and mobile applications. `https://konvajs.org/`, May 2022.

22 Kimberly Mlitz. Size of the cloud computing and hosting market market worldwide from 2010 to 2020 (in billion u.s. dollars). `https://www.statista.com/statistics/500541/worldwide-hosting-and-cloud-computing-market/`, January 2021. Accessed: 2021-05-08.

23 Adam Shostack. Tabletop security games & cards. `https://https://shostack.org/games.html`, 2021.

24 Tiange Zhao, Tiago Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. Raising awareness about cloud security in industry through a board game. *Information*, 12(11), 2021. `doi:10.3390/info12110482`.

25 Tiange Zhao, Tiago Espinha Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. Exploring a Board Game to Improve Cloud Security Training in Industry. In Pedro Rangel Henriques, Filipe Portela, Ricardo Queirós, and Alberto Simões, editors, *Second International Computer Programming Education Conference (ICPEC 2021)*, volume 91 of *Open Access Series in Informatics (OASIcs)*, pages 11:1–11:8, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/opus/volltexte/2021/14227`, `doi:10.4230/OASIcs.ICPEC.2021.11`.

Python Programming Topics That Pose a Challenge for Students

Justyna Szydłowska
University of Szczecin, Poland

Filip Miernik
University of Szczecin, Poland

Marzena Sylwia Ignasiak
University of Szczecin, Poland

Jakub Swacha
University of Szczecin, Poland

Abstract

Learning programming is often considered as difficult, but it would be wrong to assume that all programming topics are equally tough to learn. In this paper, we make use of a gamified programming learning environment submission repository containing records of over 9000 attempts of solving Python exercises to identify topics which pose the largest challenge for students. By comparing students' effort and progress among sets of exercises assigned to respective topics, two topics emerged as especially difficult (Object-oriented programming and Classic algorithms). Also interesting are the identified differences between genders (indicating female students to fare better than male at the initial topics, and the opposite for the most advanced topics), and the scale of effort some students put to succeed with the most difficult exercises (sometimes solved only after tens of failed attempts).

2012 ACM Subject Classification Applied computing → Interactive learning environments; Applied computing → E-learning

Keywords and phrases learning programming, programming exercises, gamified learning environment, learning Python

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.7

Funding The work described in this paper was achieved within two projects supported by the European Union's Erasmus Plus programme: the Framework for Gamified Programming Education (2018-1-PL01-KA203-050803) and FGPE Plus: Learning tools interoperability for gamified programming education (2020-1-PL01-KA226-HE-095786).

1 Introduction

Students that are trying to learn programming face a variety of problems, which concern many different fields. Obstacles and challenges encountered on the way relate to almost every aspect of the area. A question arises, which of them are the most difficult, or posing the largest challenge for students. Knowing that can be helpful in deciding where the focus of programming courses should be made, assigning sufficient time frames for learning respective topics, and selecting problems of adequate difficulty for the final exam.

In this paper, we investigate this matter on the case of the "Introduction to Python 3 programming" exercise set developed at University of Szczecin by a team led by the last co-author of this paper, as a part of the effort on the Framework for Gamified Programming Education project realized under the Erasmus+ international programme [6]. The exercise set covers all key aspects of Python, from the basics of the syntax to the Python implementation of classic algorithms, arranged in 12 thematic sections (called lessons). The exercise set is released as open source as a part of a larger collection, and can be freely reused and modified [8]. What is important, the whole exercise set is gamified in order to constantly encourage students to keep learning. While solving the exercises, writing their own code, finding bugs and improving previous submissions, students receive various virtual rewards, and compete against each other for the top ranks in the leaderboard.

The exercises were provided on the FGPE PLE interactive learning web platform [13] to two groups of students (counting together 39 students, including 30 male and 9 female) attending introductory Python courses in the winter semester 2021/2022 at two business schools in Szczecin, Poland. FGPE PLE consequently records students' submissions at the server, including the result of their evaluation. During the whole semester, the students made over 9000 code submissions.

We decided to capitalize on the collected data to identify the programming topics (i.e., lessons, or thematic sections of the exercise set) which posed the largest challenge for the students, in belief that results obtained this way were more precise and reliable than those obtained by interviewing the students on the difficulty of respective topics – especially, if the latter is performed at the end of a course, when students' memory is already blurred.

In order to identify the topics which pose the most challenge to the students, two measures were employed:

- the average number of code submissions made for an exercise, which reflects how much effort was put by those who eventually solved an exercise,
- the lesson completion ratio, or the percentage of students who completed all exercises from a lesson out of all who started it, which shows for how many students the challenge was big enough to resign from completing a lesson.

The paper is organized as follows. In the subsequent section, we describe the context of our research and briefly review existing reports of similar kind. Next, we describe the structure and the contents of the exercise set, and present student progress data revealing which topics were the most challenging, and which were not. We then discuss the obtained results and conclude the paper.

2 Background and Related Work

Learning programming is difficult even to the extent of being called "IS student's worst nightmare" [21]. Having been diagnosed already a long time ago (see [1] and works cited therein), this problem has attained wide research interest ever since. One of the results of this interest were various attempts made to reduce difficulty of learning programming, by making use of, e.g., program execution visualization [2], automatic evaluation [9], gamification [13], or full-fledged educational games [5]. Despite all such efforts and improvements in educational process and technology, the problem continues to be significant as confirmed by a recent international survey [19].

An important research direction is focused at finding causes of this difficulty, which may be of various character [16]. Some researchers look for them within students, pointing to their prior knowledge [4] or individual patterns of learning [14]; some blame the teachers' own poor content knowledge [18] or the outdated teaching methods [7]; some indicate the differences in difficulty of learning various programming languages [17].

This paper contributes to yet another vein of research on programming learning difficulty, which aims at identifying the programming course topics that pose the largest challenge for students. The largest research of this kind that we are aware of was done on a cohort of around 1500 students at the Open University, United Kingdom. The data source was an online 'Python help forum' where students were recording and discussing encountered challenges. The forum contained 178 discussions with a total of 1430 posts of which 29 were Python-related, 15 focusing on module-specific questions or issue reports and 19 on problem solving and general programming skills. Among the Python-related topics, IDE was the most frequently brought up (in 40 discussions), followed by Collections (in 21 discussions), Functions (16 discussions) and Error messages (15 discussions), whereas if we look at the median number of posts in a discussion, which may indicate the depth of a problem, the highest were noted for Functions (12) followed by Imports (11) and Error messages (10) [15].

The second largest study was done at Helsinki University of Technology, Finland, in a form of a survey administered to programming course participants in two subsequent years (2006–7). Data were collected from 459 students who passed the course and 119 who dropped out. Among the programming concepts, inheritance and abstract class, handling files, object-oriented concepts, and exceptions turned out to be the most difficult, while statements, loops and methods were the least difficult.

Another study in this vein was conducted in Chennai, India, involving 195 undergraduate students (56.41% female and 43.59% male) of Engineering and Technology who were taught Java Programming in the first semester of 2011. The most difficult topics to learn were noted to be “Concurrent programming”, “UI components with Swing” and “Generic programming”, whereas “Exceptions and Assertions”, “Event Handling” and “Interfaces and inner classes” were marked as moderately difficult; topics “Graphics programming”, “OO Access controls” and “Object Orientation” were perceived as less difficult, and “Fundamental programming structure in Java” as not to be difficult at all [20].

The results obtained from 145 students (120 female and 25 male) attending a one-semester introductory programming course at Buraimi University College, Oman, in 2013–14 indicate that the most difficult topics for students beginning their journey with programming are repetition structures, arrays and functions, the easiest being input/output statements, selection structure, parameters and operators [11].

Among 105 students (49.5% male and 50.5% female) of Sultan Idris Education University in Malaysia, the multidimensional array turned out to be the least understood, succeeded by looping statements, functions and the array data structure. Variables, constants and data types as well as selection statements were rated as mostly understood and input/output statements were best understood [3]. Another research involved 66 respondents, comprising programming students from the University of Dundee, and lecturers and teachers of programming from various universities in the UK, who were asked to indicate concepts and topics they found to be most difficult to cope with. The highest difficulty was given to copy constructors, operator overloading, templates, dynamic allocation of memory, pointers and other data structures (trees, linked-lists). In the middle, there were topics such as virtual functions, polymorphism, constructors/destructors, input/output and file handling, passing by reference/passing by value, and inheritance. Assessed as the easiest were looping operations, conditional operations, operators and precedence, basic function calling/program flow, and variable/function declarations [12].

A much smaller survey from central Anatolia, Turkey, involved 12 undergraduate students participating in the Internet Programming course in the fall of 2013. Among topics posing a challenge to learn, “syntax” was reported most often, followed by “functions and parameters”, “concepts, principles”, “assign variable” and “decision structures and loop” [22].

3 Challenging Topics in an Introductory Python course

3.1 Structure and Contents of the Exercise Set

The “Introduction to Python 3 programming” exercise set consists of 94 gamified exercises grouped in 12 consecutive lessons (see Table 1).

Table 1 Content of the “Introduction to Python 3 programming” course.

#	Lesson 1: Basics	Lesson 2: Strings	Lesson 3: Variables
1	Arithmetic operators	Strings	Creating variables
2	Nested parentheses	Apostrophes	Setting variable values
3	The order of arithmetic operators	Special characters	Modifying the value of a variable
4	Exponentiation	Raw strings	Setting several variables at once
5	Alternative number systems	Multi-line string literals	Changing the values of many variables
6	Real numbers and integers	String concatenation	Naming variables
7	Scientific notation	Strings vs. numbers	-
8	Division remainder	String repetitions	-

#	Lesson 4: Conditionals	Lesson 5: Loops	Lesson 6: Sets
1	Comparisons	Introduction to loops	Introduction to sets
2	Equality check	Ranges	Set initialization and modification
3	The else block	Loop counter	Loop over a set
4	Parity checking	Two parameters of the range function	Functions operating on set elements
5	Many cases	Three parameters of the range function	Subsets and operations on sets
6	Combining comparisons	The continue instruction	Turning a string into a set
7	Nesting comparisons	Nested loops	Operations on sets made from strings
8	-	Aggregates	-
9	-	Breaking a loop and the else block	-

#	Lesson 7: Lists	Lesson 8: String processing	Lesson 9: Dictionaries
1	Lists	ASCII codes	Dictionaries
2	Create and combine lists	String immutability	Accessing items in a dictionary
3	Extending a list	Searching for a substring	Dictionaries as a collection of keys
4	Shortening a list	Determining the substring position	Operations on dictionary elements
5	Checking the contents of a list	Replacing a substring	Operations on a whole dictionary
6	Accessing list items by index or value	Changing the letter case	Default values of dictionary elements
7	Inserting and deleting elements	Centering strings	Using a dictionary to translate words
8	List slicing	Splitting and combining strings	-
9	Counting items in a list	-	-
10	Converting strings or sets to lists	-	-

#	Lesson 10: Functions	Lesson 11: Object-oriented programming	Lesson 12: Classic algorithms
1	Defining a function	Introduction to classes	Binary search
2	Function result	Parameterized constructor	Quicksort
3	Function parameters	Displaying object contents	Fibonacci sequence
4	Parameter names	Defining a class	The problem of 8 queens
5	Default values of parameters	Inheritance	The knight’s tour problem
6	Variable number of parameters	Method overloading and base class access	The change-making problem
7	Unpacking parameters from the list	Accessing object’s class	-
8	Many results of the function	-	-
9	Local variables	-	-
10	Nested functions	-	-
11	Global variables	-	-

3.2 Topics’ Difficulty According to Students’ Effort and Progress

Figure 1 compares the two difficulty measures visually, with each dot representing an individual exercise, and its color denoting the lesson it belongs to. A reader is reminded that both the source of the presented data and the procedure of its processing were described in the Introduction.

As we can observe, generally, the further the lesson an exercise belongs to is placed in the set, the smaller is the share of students who solved it (note that the students were given an access to all exercises regardless of its topic from the very beginning, i.e., they were not required to complete the earlier lessons to access the later ones). Exercise Arithmetic operators was the one solved by most students (92.31%), whereas exercises The knight’s tour problem (2.56%), then The problem of 8 queens and The change-making problem (both

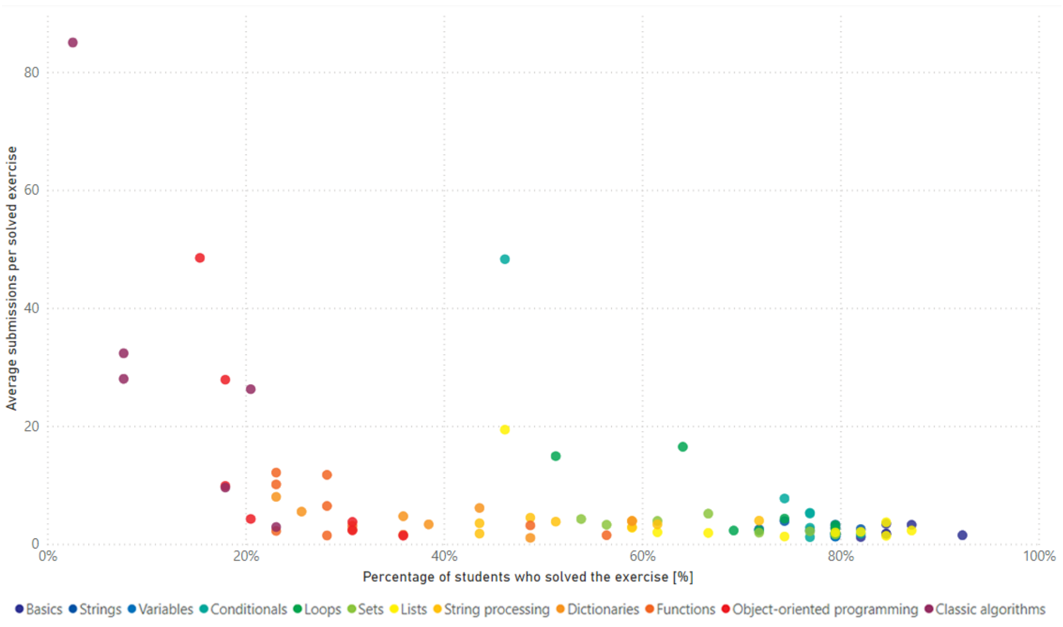


Figure 1 Average submissions per solved exercise by percentage of students who solved it.

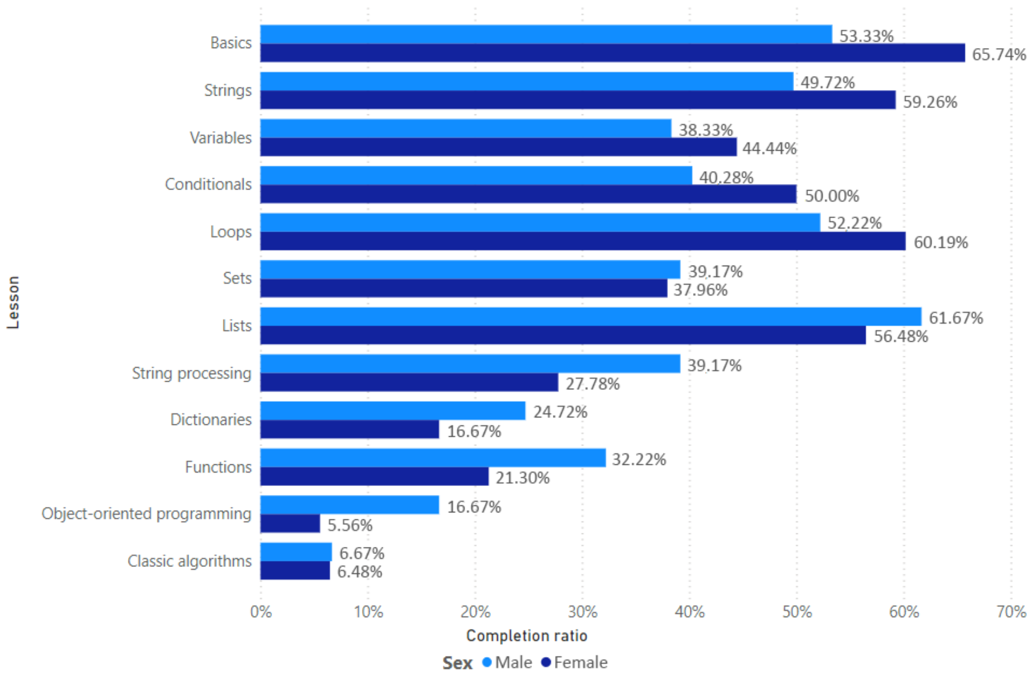


Figure 2 Completion ratio of every exercise (male and female).

7.69%) were the ones solved by least students. As for the average number of tries students made to solve an exercise, the picture is more complicated: even in the later lessons, there are exercises solved in the first attempt by most students, and most exercises were solved after few submissions at most. However, we have some outliers: the most notable is exercise The knight's tour problem from lesson Classic algorithms which took on average over 80 tries to solve. The similar cases are exercises Defining a class from lesson Object-oriented programming and Nesting comparisons from lesson Conditionals which both took about 50 submissions on average.

In Figure 2, we present the average ratios of students who completed exercises from a given lesson, depending on their gender. For females, the average completion ratio ranged from about 2/3 for the introductory first lesson to about 1/18 for the Object-oriented programming lesson (which was the one before the last; interestingly, for the last topic, Classic algorithms, a bit higher ratio of about 1/16 has been achieved among female students). To a lesser degree, females also struggled with two other lessons: Dictionaries (completion ratio of 1/6) and Functions (completion ratio near 1/5). For males, unexpectedly, the first lesson was not the easiest (completion ratio of about 1/2), but the lesson on Lists (completion ratio of about 3/5). Males struggled the most with the last lesson (Classic algorithms, completion ratio of about 1/16), then with the topics of Object-oriented programming (completion ratio of 1/6) and Dictionaries (completion ratio of about 1/4).

In total, female students fared better in the first five lessons, and male students in the remaining seven. Such a result indicates that females more often than males grasped the concepts of programming from the very beginning, whereas males were better at dealing with more complex topics further on. Looking at the differences between the two genders, while for some lessons it was small, for some it was large. In particular, female students achieved a 23% higher completion ratio for lesson Basics, 19% for Strings, and 24% for Conditionals, whereas male students achieved a three times higher completion ratio for Object-oriented programming, 51% for Functions, 41% for String processing, and 48% for Dictionaries.

The number of failed submissions made by students who eventually solved an exercise averaged for every lesson and disaggregated by gender has been presented in Figure 3.

For females, the average number of failed submissions ranged from about 0.2 (the vast majority of submissions were accepted on the first try) for the Variables lesson to over 4 per exercise for the Classic algorithms lesson. Females have also sent a significantly high number of submissions in lesson Loops (3.2). For the remaining lessons, the average numbers ranged from 1.4 to 2.7. In the case of males, the average number of failed submissions ranged from about 0.4 for the Variables lesson to 5.9 for the Conditionals lesson. Male students also needed a small number of tries before passing in lessons Basics (1.0), Strings (1.1) and Functions (1.4). For the remaining lessons, the average numbers ranged from about 1 to 3.

The average number of failed submissions per solved exercise for both male and female students is very similar in most lessons. The greatest difference between the two genders can be seen for lesson Conditionals, where males have sent about 3.7 more failed submissions per exercise on average than females. The opposite situation happened for lessons: Classic algorithms (1.4 less submissions sent by male students on average), Functions (1.3 less), and Object-oriented programming (1.1 less).

4 Discussion

The prior work focused at surveys reporting students' own estimation of difficulty, whereas the results presented here were based on objective data measuring students' ability to solve particular exercises and the effort they made to accomplish that. While the latter also has

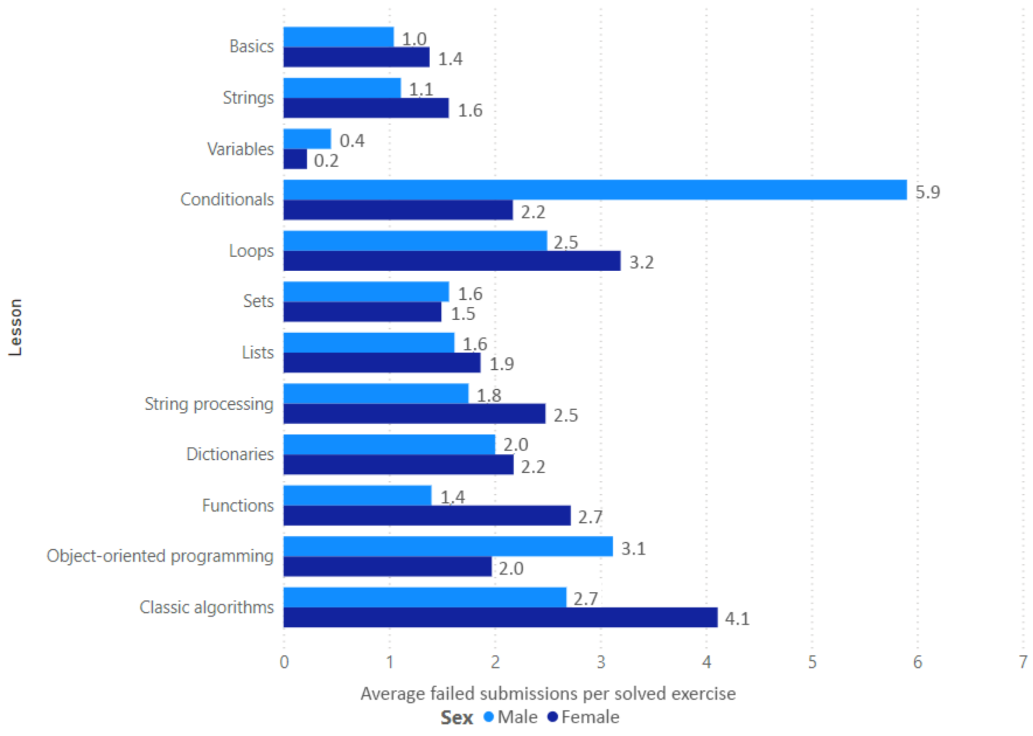


Figure 3 Average failed submissions per solved exercise for every lesson (by gender).

its limitations – students may discontinue a course at any time for reasons other than its difficulty, and the number of failed submissions may be underestimated as students can develop and test their code in another environment, and paste it into the learning environment only after they believe it is correct – we still consider it a more reliable estimator of topic difficulty than students' opinion, especially declared long after solving the exercises actually took place.

Comparing the presented results to those reported in prior work, no clean pattern emerges. Our results, similar to [12] and [10], point to object-oriented programming as one of the most difficult topics, and loops as one of the easier ones, whereas works [20] and [3] reported almost opposite results regarding these two topics. The latter study, however, agrees on syntax not posing a learning challenge; this is in contrast to [22], which on the other hand is consistent with our results with regard to considering functions as difficult, alike also [11]. This may stem from differences in both programming languages taught and the educational context.

The reviewed literature did not give sufficient attention to gender differences. The results of our study show that female students fare better at handling the basics of programming, but struggle at later topics, whereas those male students who pass the early barriers, are more capable of passing the exercises belonging to more difficult topics as well.

5 Conclusion

In this paper, we extended the knowledge of which programming learning topics pose the largest challenge for students with new results obtained from two groups of students playing with a set of gamified Python exercises.

Unlike the most previous research in this vein, instead of surveying the students about the perceived difficulty of respective topics, we based our analysis on objective data comprising the share of students who completed all exercises belonging to a lesson on a given topic, and the average number of submissions the students made before their solution was accepted.

While, in general, the measured difficulty increases the further in the exercise set we proceed, the most notable observation is the very sharp rise of the difficulty for the two final lessons, devoted to Object-oriented programming and Classic algorithms, respectively. Possibly, these two topics should not belong to an introductory programming exercise set.

By distinguishing the gender of students, we were able to reveal in our data that female students had an easier start with learning programming, whereas the male students were more inclined to continue solving exercises till the end, even in spite of numerous failed attempts.

A very interesting observation was made thanks to measuring the number of submissions before one was accepted: for some exercises, a number of students kept trying and eventually succeeded even after failing tens of times. We link this observation with the fact the exercises were gamified, which most probably helped to keep the engagement and motivation high for at least a part of the students. Proving that, however, would require a comparison with a group learning with non-gamified exercises. This indicates the direction of our future work.

References

1 Yorah Bosse and Marco Aurélio Gerosa. Why is programming so difficult to learn? patterns of difficulties related to programming learning mid-stage. *SIGSOFT Softw. Eng. Notes*, 41(6):1–6, 2017. `doi:10.1145/3011286.3011301`.

2 Michael P. Bruce-Lockhart and Theodore S. Norvell. Lifting the hood of the computer: program animation with the teaching machine. In *2000 Canadian Conference on Electrical and Computer Engineering. Conference Proceedings. Navigating to a New Era (Cat. No.00TH8492)*, volume 2, pages 831–835, 2000. `doi:10.1109/CCECE.2000.849582`.

3 SitiRosminah MD Derus and Ahmad Zamzuri. Difficulties in learning programming: Views of students. In *Proc. 1st International Conference on Current Issues in Education*, pages 74–78, Yogyakarta, Indonesia, 2019. `doi:10.13140/2.1.1055.7441`.

4 Dimitrios Doukakis, Maria Grigoriadou, and Grammatiki Tsaganou. Understanding the programming variable concept with animated interactive analogies. In *Proceedings of the The 8th Hellenic European Research on Computer Mathematics & Its Applications Conference (HERCMA'07)*, 2007. URL: `http://users.sch.gr/adamopou/docs/syn_HERCMA2007_doukakis.pdf`.

5 Michael Eagle and Tiffany Barnes. Experimental evaluation of an educational game for improved learning in introductory computing. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 321–325, New York, NY, USA, 2009. Association for Computing Machinery. `doi:10.1145/1508865.1508980`.

6 FGPE Consortium. Framework for Gamified Programming Education, 2020. accessed on 22 April 2022. URL: `http://fgpe.usz.edu.pl`.

7 Mark Guzdial and Elliot Soloway. Teaching the nintendo generation to program. *Commun. ACM*, 45(4):17–21, 2002. `doi:10.1145/505248.505261`.

8 Jakub Swacha, Thomas Naprawski, Ricardo Queirós, José Carlos Paiva, José Paulo Leal, Ciro Giuseppe De Vita, Gennaro Mellone, Raffaele Montella, Davor Ljubenkov, Sokol Kosta. Open Source Collection of Gamified Programming Exercises. In *Proceedings of the thirty-seventh Information Systems Education Conference ISECON 2021*, pages 120–123, Oak Creek, 2021. Foundation for IT education. URL: `http://proceedings.isecon.org/download/co8h5jrbkvipjznly7dp`.

9 Mike Joy, Nathan Griffiths, and Russell Boyatt. The Boss online submission and assessment system. *J. Educ. Resour. Comput.*, 5(3):2–es, 2005. doi:10.1145/1163405.1163407.

10 Päivi Kinnunen. *Challenges of teaching and studying programming at a university of technology – viewpoints of students, teachers and the university.* Doctoral thesis, Helsinki University of Technology, Espoo, Finland, 2009. URL: https://aaltodoc.aalto.fi/handle/123456789/4710.

11 Sohail Iqbal Malik and Jo Coldwell-Neilson. A model for teaching an introductory programming course using ADRI. *Education and Information Technologies*, 22(3), 2017. doi:10.1007/s10639-016-9474-0.

12 Iain Milne and Glenn Rowe. Difficulties in learning and teaching programming – views of students and tutors. *Education and Information Technologies*, 7(1):55–66, 2002. doi:10.1023/A:1015362608943.

13 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, Jakub Swacha, and Filip Miernik. An open-source gamified programming learning environment. In *Second International Computer Programming Education Conference (ICPEC 2021)*, volume 91 of *OASICS*, pages 5.1–5.8, Saarbrücken, Wadern, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2021.5.

14 D. N. Perkins, Chris Hancock, Renee Hobbs, Fay Martin, and Rebecca Simmons. Conditions of learning in novice programmers. *Journal of Educational Computing Research*, 2(1):37–55, 1986. doi:10.2190/GUJT-JCBJ-Q6QU-Q9PL.

15 Paul Piwek and Simon Savage. Challenges with learning to program and problem solve: An analysis of student online discussions. In *SIGCSE '20: Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 494–499, New York, 2020. ACM. URL: http://oro.open.ac.uk/68074/.

16 Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.*, 18(1), 2017. doi:10.1145/3077618.

17 Łukasz Radliński and Jakub Swacha. C# or Java? – analysis of student preferences. *Studies & Proceedings of Polish Association for Knowledge Management*, 58:101–113, 2012.

18 Philip Sadler, Gerhard Sonnert, Harold Coyle, Nancy Cook-Smith, Jaimie Miller-Friedmann, and Harvard-Smithsonian Center. The influence of teachers' knowledge on student learning in middle school physical science classrooms. *American Educational Research Journal*, 50(5):1020–1049, 2013. doi:10.3102/0002831213477680.

19 Simon, Andrew Luxton-Reilly, Vangel V. Ajanovski, Eric Fouh, Christabel Gonsalvez, Juho Leinonen, Jack Parkinson, Matthew Poole, and Neena Thota. Pass Rates in Introductory Programming and in other STEM Disciplines. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, pages 53–71, Aberdeen, 2019. ACM. doi:10.1145/3344429.3372502.

20 M. Sivasakthi and R. Rajendran. Learning difficulties of 'object-oriented programming paradigm using Java': students' perspective. *Indian Journal of Science and Technology*, 4(8):983–985, 2011. doi:10.17485/ijst/2011/v4i8.9.

21 Amy B. Woszczynski, Hisham M. Haddad, and Anita F. Zgambo. An IS student's worst nightmare: Programming courses. In *SAIS 2005 Proceedings*, 2005. URL: https://aisel.aisnet.org/sais2005/24/.

22 Büşra Özmen and Arif Altun. Undergraduate Students' Experiences in Programming: Difficulties and Obstacles. *Turkish Online Journal of Qualitative Inquiry*, 5(3):9–27, 2014. doi:10.17569/tojqi.20328.

Thoughts of a Post-Pandemic Higher Education in Information Systems and Technologies

Francini Hak[1]
Algoritmi Research Center, University of Minho, Braga, Portugal

Jorge Oliveira e Sá
Algoritmi Research Center, University of Minho, Braga, Portugal

Filipe Portela
Algoritmi Research Center, University of Minho, Braga, Portugal

Abstract

In late 2019, a new class of coronavirus appeared in China that triggered a worldwide pandemic declared by the World Health Organization. Several businesses were affected and people had to adapt their social life to a virtual mode. Higher education institutions suffered from this sudden change, and had to adapt without any preparation or planning. After almost two years of carrying out activities in an online format, face-to-face activities in higher education have returned. This case study aims to analyze the performance of students in a new curricular unit of the Engineering and Management of Information Systems course at the School of Engineering of the University of Minho in Portugal. The study is applied to 142 students who entered the 1st year of the 2019/2020 school year and returned to face-to-face activities in the 3rd year of the 2021/2022 school year. Two questionnaires were applied, one at the beginning of the semester with 71 answers and another at the end of the semester with 39 answers. The main objective was to understand the students' feedback regarding the functioning of the classes, in which a great difficulty in teamwork was highlighted.

2012 ACM Subject Classification Information systems → Evaluation of retrieval results

Keywords and phrases Emergency Remote Teaching, E-learning, Higher Education, Covid-19 Pandemic, Information Systems

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.8

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

Since the World Health Organization (WHO) declared COVID-19 a global pandemic on 11 March 2020, the crisis hit higher education worldwide, including in Portugal. The Coronavirus catalysed changes, seemingly overnight, from face-to-face to remote classes. Learning online can be challenging in general, for both teachers and students and remote classes requires careful preparation and planning [7].

However, in March 2020, there was no time to do this preparation and planning work, causing a sudden Emergency Remote Teaching (ERT), temporary move to distance education in response to a crisis that prevents in-person class meeting, may be considerably more challenging, particularly because of the variation in delivery. Instructors varied in how they introduced remote teaching. Some approaches focused on classes were limited to set times of the day or week, with synchronous online teleconference meetings, other classes were completely asynchronous, with pre-recorded lectures and/or written materials in lieu of live meetings. However, this affected the students who met a learning experience that for many was a new experience.

[1] Corresponding author

This case study was applied in students who started their studies at the University of Minho in the 2019/2020 academic year, that is, students who were in the 1st year at that time. In the following academic year 2020/2021, learning should have taken place in a blended format, that is, a mix between face-to-face and remote; however, most classes took place in a remote format. What happened is that the theoretical and theoretical-practical classes are recommended to be in a remote format and the practical classes should have taken place in a "face-to-face" format; in fact, this did not happen, as the teachers chose to teach these classes in a remote format. In the actual academic year 2021/2022, classes are mandatory taught in a face-to-face format.

When the universities closed their campuses in March 2020, the 1st year students of the 2019/2020 academic year had not yet acclimated to the university context [6], so these students stayed since March 2020 to September 2021 practically working remotely. This distance meant that students were deprived of socializing with colleagues (from the academic year, as well as from later years). Because in addition to the remote classes, all socializing events between students (new and older) have been cancelled.

At the time of writing this study, these students are in the 3rd year of their study cycle, in the academic year 2021/2022, and the classes are mandatory in face-to-face format. In the 3rd year of the study cycle, most courses require students to work in a team and the number of students can vary from three to eight elements per team. Thus, students currently attending the 3rd year were faced with the need for face-to-face classes, which for them is almost a novelty and with the need to work as a team with colleagues that are strangers to them.

This study focuses on students who attend the last year, 3rd year, of the first cycle of Engineering and Management of Information Systems (EMIS), of the School of Engineering of the University of Minho. The course subject of this study is called Data Engineering for Decision Making Support (DEDMS) which has 162 students enrolled and is a new new curricular unit created from the restructuring of the EMIS course. This course provides students with first contact with data engineering to support decision making in organizational context, providing fundamental knowledge to students to understand the importance of data and how information technologies enable data collection, processing and analysis. In this case study, two questionnaires were applied in order to understand the expectations and opinions of students regarding the DEDMS subject.

This article is structured in five sections, starting with an introductory part. A background on the pandemic in higher education is presented. The third section address the course structure and the applied subject. Section four describes the methods used and the results obtained. Finally, conclusions are made.

2 Background

The Covid-19 pandemic declared in March 2020 affected the entire world, forcing people to close their businesses and working from home. Higher education was also affected, as universities were forced to shut down campuses by interrupting classroom education. Universities were not prepared to suspend their activities, nor were students and teachers prepared for distance learning. In the beginning, a lot of time was spent choosing the right methods and resources, such as online platforms for video conferencing classes or platforms for recording classes.

It can be said that this caused a crisis called Emergency Remote Teaching (ERT) which translates into a temporary change in the way of teaching due to the circumstances of the crisis, involving the use of solutions for distance learning. The main objective was to provide fast and reliable access available to perform daily tasks. However, the robustness and quality of teaching/learning have been compromised [3], as online learning can be a challenge in general, both for teachers and especially for students who expected a face-to-face experience [7]. Higher education teachers have approached remote education in different ways. On the one hand, classes were limited to fixed times of the day or week, with synchronous online video conferencing meetings. On the other hand, the classes were asynchronous with pre-recorded classes and with materials available on some platforms. All students, however, were faced with a learning experience they did not anticipate and that many would not have chosen for themselves [3].

ERT differs from online education that already existed. Teaching in ERT was a rapid response to the emergency health crisis that delivered content in a varied, unplanned and difficult way, without predicting the consequences of these actions. The online teaching or e-learning method does not compromise the structure of classes and learning content. Although both methods use evidence-based teaching, this difference suggests that ERT may be associated with a different learning experience. Some studies focused on the involvement and motivation of students during the ERT period [5, 2], but did not bother with objectively measuring the learning [3]. Much discussion around ERT between teachers and administrators is correctly related to how well students are learning during ERT, and how learning outcomes we have the knowledge, skills, attitudes and habits that students acquire [3].

However, no distinction was made in the application of ERT education among students who had just entered university in the 2019/2020 school year, that is, they were attending their first year, and students who already attended university, that is, who were already in the 2nd, 3rd, 4th or even 5th grade. This distinction is important, because this group of students, in the school year 2020/2021, had all their classes remotely and only in 2021/2022 do they attend face-to-face classes.

From our knowledge we did not identify any study directed at these students, we found that there are studies that report that students who attend the most advanced years of study tend to be more favourable to the online learning environment than younger students, evident a tendency to a more favourable engagement in online learning environments with increased specialization, on the other hand, an important factor is lost which is student-student interaction [8] and that ERT caused a loss in personal relationships and even loneliness and depression [1, 4].

Thus, ERT affected the success of teaching and it is urgent to study how universities implemented organizational, pedagogical and educational concepts to minimize the loss of learning time due to the closure of schools [9]. In our opinion, this study should include other factors, such as that proposed in this study, which is the year attended by the student when the ERT occurred.

3 Course Structure

3.1 EMIS Scope

The Engineering and Management of Information Systems (EMIS) is a high school course offered by the School of Engineering of the University of Minho in Portugal. The professional profile associated to the EMIS course combines competences from informatics engineering and Information Technology (IT) management. The role played by Information Systems (IS) engineers and managers is to use IT and its applications to the benefit of organizations. IT is a means to the improvement of organizations and not an end in itself. Therefore, Information Systems and Technology (IST) professionals are expected to intervene in the adoption of IT and to manage the organizational and work engineering processes.

IST graduates should also possess competences for building IT applications and for getting involved in the activities related to the organization IT infrastructure. The scientific areas present in the course program reveal the combination of competences mentioned above. Higher education in IST involves characteristics typical from informatics engineering together with aspects that enables understanding organizations and their workings, their processes and management activities. The informatics component of the program emphasizes the configuration and customization of existing IT products and platforms either for operational or managerial work.

It is important to mention that the course was recently reformulated, separating the bachelor (1st cycle – 3 years) and the master (2nd cycle – 2 years). The new program will substitute one already existing programs: an integrated Master (1st cycle and 2nd cycle – 5 years) program, due to the imposition of Portugal Government. This change will have implications on the EMIS professionals, because the competences that it is possible to achieve during the 3 first years of the program correspond to a quite undifferentiated professional profile. Entering in the labour market with such competences has 2 inconveniences: less competitiveness of graduates; and obstacles to the development of a professional profile with higher potential impact in organizations.

In a nutshell, an EMIS professional acts mentioned above demand competences typical of 2nd cycle education, in what concerns: understanding of technology; understanding the context where technology will be deployed; application of techno-scientific knowledge; criticism and judgement capacity; communication of scientific and technological subjects. In order to achieve an effective integration of the program competences, it is understood that the informatics component of the program should be well articulated with the managerial component.

3.2 DEDMS Curricular Unit

Within the scope of Engineering and Management of Information Systems (EMIS) course, the Data Engineering for Decision Making Support (DEDMS) curricular unit is inserted in the first semester of the 3rd year and is a new subject resulting from the reformulation of the course. We selected this curricular unit because it is extremely important for the third year and has a larger dimension. This course is 10 ECTS (European Credit Transfer System) and it is divided into 2 theoretical hours (T class), 2 theoretical-practical hours (TP class) and 2 practical and laboratory hours (PL class) per week, with a teaching team composed of seven professors. In T classes, the teaching learning method is expository, where the relevant concepts are presented and students are encouraged to investigate and expose concepts related to the topics covered. TP classes are an active learning method where activities are planned that involve students' participation, both in the accomplishment of tasks and in the systematization of results and difficulties. In the PL students are involved in the realization of a practical project.

The scope of DEDMS is to develop a Big Data project through a Delta Lake architecture. Distributed technologies such as Apache Spark were adopted. This course provides students with first contact with data engineering to support decision-making in organizational context, providing fundamental knowledge for students to understand the importance of data and how information technologies enable data collection, processing and analysis.

This course is project-oriented having 280 working hours, 90 hours of which relate to teacher and student contact and the remaining to students' self-study or project work. In addition to the permanent involvement of students in this process of continuous interaction, the course provides the involvement of students in a team to the realization of a project. For this project, students dedicate hours of autonomous work. The work is evaluated in stages that are predefined and the delivery date is set at the beginning of the course. The project is followed weekly in PL classes, with the objective of identifying the evolution of each work team and clarifying the doubts that arise as it unfolds.

In general, the main goal is to understand the importance of data in supporting decision-making; identify the different data sources, data types, data quality and data profiling and the necessary transformations in a given analytical context; develop extraction, transformation and loading (ETL) processes whose specification includes mechanisms for optimizing the resources used; Explore data through presentation and visualization technologies such as dashboards, reports or tables,highlighting concerns about the effectiveness and efficiency of the visualization process; Define technology architectures that take advantage of different data storage and processing systems, supporting the needs of analytical information systems in organizations.

4 Results

4.1 Materials and Methods

This article approaches a case study developed with the objective of analyzing and understanding the performance of a group of students in the return of classroom activities. The present case study covered 142 students from DEDMS curricular unit of the EMIS course at the University of Minho in Portugal. The study was applied in the first semester of the 2021/2022 academic year to third-year students, who entered in higher education in the 2019/2020 academic year. The data sample covered 142 students evaluated in DEDMS of 162 enrolled. The evaluation of the curricular unit consisted of an exam and a practical project, in which 136 students were approved. Of the students enrolled in the subject, 109 were male and 52 were female. In addition, the average age of students was 21, with a minimum age of 19 and a maximum age of 44.

Two questionnaires were prepared using Google forms, one applied at the beginning of the semester in order to understand the students' expectations with the new curricular unit, and another at the end of the semester to verify if the students' expectations were met. The structure of the questionnaires was similar, with a total of 8 questions of different types, such as multiple choice and free text. The questionnaire was anonymous and did not aim to identify the students. The questions focused on understanding the students' motivation towards the project, their perception of the communication created between the students and the teaching team, their opinion about the learning contents and about the structure of the practical project, and finally free text for suggestions and comments. The extraction of data from the questionnaires was done manually, using data analysis tools such as Excel and Tableau.

Therefore, it was intended to discuss the evolution of students in DEDMS class in a post-pandemic phase. In addition, it was also intended to observe the performance of students in relation to a new curricular unit that emerged due to the reformulation of the course, which does not contain previous records for comparison.

4.2 Findings

At the beginning of the semester, a questionnaire was made available in order to understand the expectations of students with DEDMS curricular unit. This questionnaire obtained 71 responses. As shown in Figure 1 a), 58% of students responded that their expectations were normal and 32% had high expectations. The second question was related to student communication with teachers, with a rating from 0 to 5. 46% of students rated the work environment created as good (4), 28% as normal (3) and 14% as very good (5). Graph c) corresponds to the students' understanding of the learning content, with 51% saying yes, 9% partially recording and 4% recording no. In general, students at the beginning showed motivation to learn and curiosity about the subject studied.

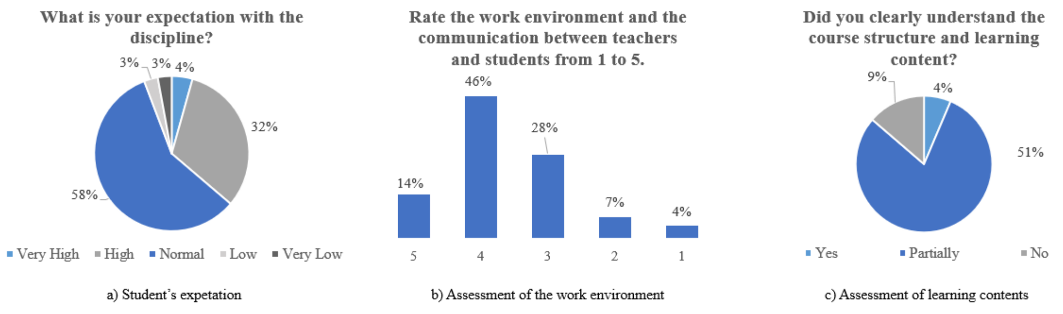


Figure 1 Results obtained from the first questionnaire.

At the end of the semester, a second questionnaire was made available, obtaining 39 responses. Unfortunately, the number of student responses in the second questionnaire decreased by almost 50% compared to the first questionnaire. A specific reason for this reduction was not identified, but we believe that the questionnaire should have been applied a little before the students' assessment, and not at the end of the semester as it was done. The interpretation of results was not greatly affected by this reduction, however a larger sample would be better to compare the results of the two questionnaires. It is also noted that it is not possible to know whether the audience of the two questionnaires was the same, but we believe that a large part was.

The first question in Figure 2 a), refers to the learning contents, where 61% of students reported that the learning contents were clearly presented, 34% classified it as partially and 5% as not. Question b) refers to the functioning of DEDMS, where 37% of students classified it as good (4), 29% classified it as normal (3), and 21% as very good (5). Graph c) corresponds to the general evaluation of DEDMS, where 61% evaluated the course unit as positive, 34% as very positive, 5% negative and no rating very negative.

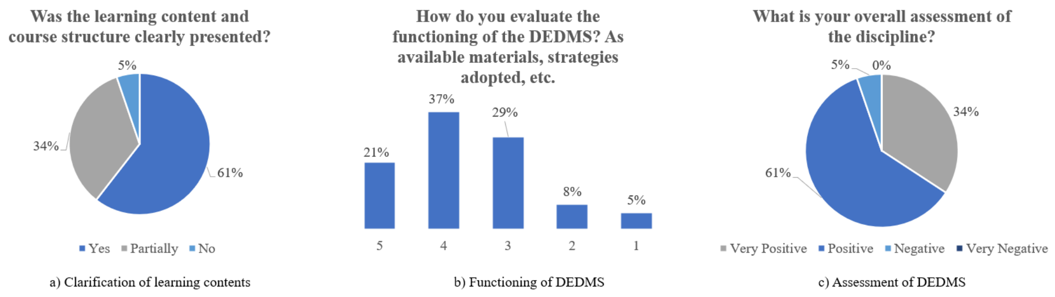


Figure 2 Results obtained from the second questionnaire.

Furthermore, some questions were applied in both questionnaires, which served to compare the results. The graph in Figure 3 a), analyzes the students' motivation towards the DEDMS curricular unit. There was a greater motivation of students at the end of the semester, and a decrease in those who registered "no" in the initial questionnaire. The graph in Figure 3 b) demonstrates the students' opinion towards technologies, where 54% of students at the beginning thought that technologies were a good bet and only 39% at the end. 36% at the beginning fount it partially and 47% at the end. In the first questionnaire 10% think the technologies use were not a good bet and 13% at the second questionnaire. The students' difficulties with the tools used for the development of the practical project were noticed during the classes.

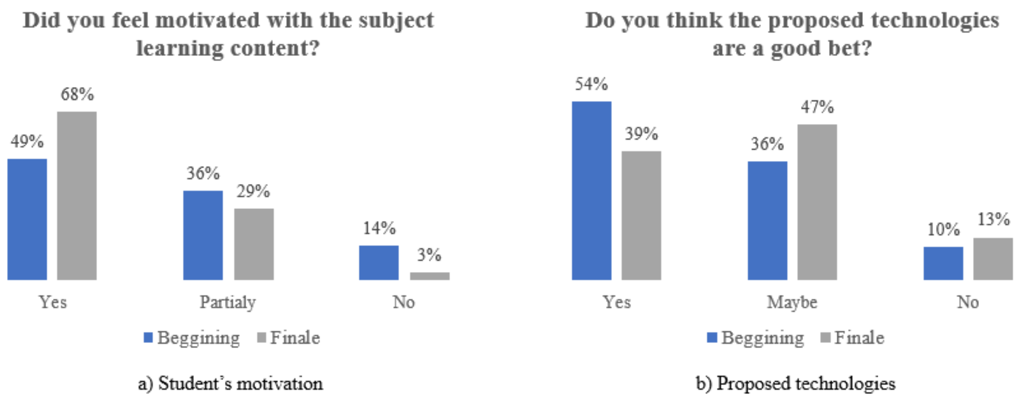


Figure 3 Combination of results obtained from the two forms.

In addition to these questions, the second questionnaire contained a free text field for suggestions and comments that students wished to place. Some comments referred to the tools chosen for the project, where some students pointed out some difficulties in their use. On the other hand, most students highlighted the difficulty of teamwork and stated that the DEDMS project should be more individual and not so much a group. These comments caught the attention of the teachers, because during the classes it was noticed that students had a lot of difficulty in team work, as they did not know how to communicate as a team and ended up creating rivalry between the members of the group itself. In addition to this behavior being perceived during the practical classes, the questionnaires also confirmed the fact that students did not know how to work as a team in the face-to-face format.

As DEDMS is a project-oriented course, teamwork is essential. On the one hand, the results showed that students felt motivated with the DEDMS curricular unit and that they had high expectations regarding the activity contents. On the other hand, the greatest difficulties encountered were in the choice of tools and in the performance of team work. The results of the questionnaires allowed to highlight a difficulty in teamwork on the part of the students, which sometimes generated a certain rivalry between the members of the group itself. The lack of communication between the members made group work a hard task. We believe this was due to students not knowing each other and not being used to working face to face. The social component in an academic environment is essential for students to get to know each other, stimulate conversation and also help each other with academic work.

Based on the difficulties encountered, the teachers needed to take certain measures that promoted continuous monitoring in the exercise of developing the practical project. To this end, the search for answers to questions that were not fully understood or that should be deepened was encouraged; Involvement of students in the assessment process in order to respond to answers from other students; recognition of mistakes made by students; Remembering the evaluation criteria; Promote future performance against established criteria; Promote students' critical thinking; Promote team work and student interaction independently.

5 Conclusion

The Covid-19 pandemic changed format of higher education, contributing to an Emergency Remote Teaching. This case study aims to analyze the performance of students who entered university in the 2019/2020 school year and who are now in the third year after two years of classes in remote format. This school year is in the 3rd year and with face-to-face classes. The study focused on DEDMS, a new curricular unit from the EMIS course. This 10 ECTS curricular unit requires students to work as a team to develop a practical project.

During the practical classes of the DEDMS project, the teaching staff realized that students were not working properly as a team, as they lacked communication, organization, and critical and judgmental capacity. Two questionnaires were applied to understand the general functioning of the DEDMS curricular unit. However, through free answers and in the practical classes, it was found that students had difficulty working as a team because they had not worked face-to-face before. This led to the teaching staff of the practical DEDMS classes having to take steps so that students could work better as a team, namely the definition of well-defined roles for each student, for example, defining a project manager, documentary, responsible for communication, and others; the concrete definition of the expectations of each team member; the preparation of meetings with well-defined objectives; the reporting of meetings; planning and implementation rigorous work to be carried out; among other measures. These recommendations allowed students to be able to bring the DEDMS curriculum unit to a full term. Thus, of the 162 students enrolled, 20 were not evaluated, and of the 142 students evaluated, 136 were approved, making only 6 students not obtained performance and the average of the classifications (0 to 20) in DEDMS was 16 values, which is a good result.

Colleges and universities are trying to decide the ideal way to continue learning. However, although the online format has its advantages, on the other hand it can compromise the social life and can harm the development of personal skills. For students entering university, aged 17 or 18, in addition to the specialization they will get through the course they have chosen, they will also develop and form their social personality. ERT has eliminated this prospect of training. It is therefore appropriate that when choosing the remote format, there is a concern to strengthen the social aspects to overcome the difficulties experienced by these students. Furthermore, it is expected that this sudden change will serve as a lesson for us to be prepared for situations of change and to have the necessary resources to do so.

References

1 Tianhua Chen and Mike Lucock. The mental health of university students during the covid-19 pandemic: An online survey in the uk. *PLoS ONE*, 17, 2022. `doi:10.1371/journal.pone.0262562`.

2 Kristen Fox, Gates Bryant, Nicole Lin, and Nandini Srinivasan. Time for class covid-19 edition: part 1: a national survey of faculty during covid-19 | vocedplus, the international tertiary education and research database. *Tyton Partners and Every Learner Everywhere*, 2020. URL: `https://www.voced.edu.au/content/ngv:88359`.

3 Regan A. R. Gurung and Arianna M. Stone. You can't always get what you want and it hurts: Learning during the pandemic. *Scholarship of Teaching and Learning in Psychology*, October 2020. `doi:10.1037/STL0000236`.

4 Madona Kekelia, Eliso Kereselidze, and Ina Shanava. The covid-19 pandemic and the mental health of students. *Vectors of Social Sciences*, 1, 2021. URL: `https://jlaw.tsu.ge/index.php/vss/article/view/3639`.

5 Barbara Means and Julie Neisler. Suddenly online: A national survey of undergraduates during the covid-19 pandemic, July 2020. `doi:10.51388/20.500.12265/98`.

6 Madeleine Pownall, Richard Harris, and Pam Blundell-Birtill. Supporting students during the transition to university in covid-19: Five key considerations and recommendations for educators:. *Psychology Learning & Teaching*, 21:3–18, July 2021. `doi:10.1177/14757257211032486`.

7 Shannon Riggs, Kathryn E. Linder, and Penny Ralston-Berg. *Thrive online: a new approach to building expertise and confidence as an online educator*. STYLUS PUB LLC, 2019.

8 Katerina Salta, Katerina Paschalidou, Maria Tsetseri, and Dionysios Koulougliotis. Students' engagement and interactions in four university-based science learning communities during a shift from a traditional to a distance learning environment imposed by the covid-19 pandemic. *Science & Education*, 31:93–122, 2022. `doi:10.1007/s11191-021-00234-x`.

9 Klaus Zierer. Effects of pandemic-related school closures on pupils' performance and learning in selected countries: A rapid review. *Education Sciences*, 11:252, May 2021. `doi:10.3390/EDUCSCI11060252`.

Integration of Computer Science Assessment into Learning Management Systems with JuezLTI

Juan V. Carrillo
CIFP Carlos III, Cartagena, Spain

Alberto Sierra
CIFP Carlos III, Cartagena, Spain

José Paulo Leal
CRACS – INESC-Porto LA & DCC – FCUP, Porto, Portugal

Ricardo Queirós
CRACS – INESC-Porto LA & uniMAD, ESMAD/P. Porto, Portugal

Salvador Pellicer
Entornos de Formación (EdF), Valencia, Spain

Marco Primo
Faculty of Sciences, University of Porto, Portugal

Abstract

Computer science is a skill that will continue to be in high demand in the foreseeable future. Despite this trend, automated assessment in computer science is often hampered by the lack of systems supporting a wide range of topics. While there is a number of open software systems and programming exercise collections supporting automated assessment, up to this date, there are few systems that offer a diversity of exercises ranging from computer programming exercises to markup and databases languages. At the same time, most of the best-of-breed solutions force teachers and students to alternate between the Learning Management System – a pivotal piece of the e-learning ecosystem – and the tool providing the exercises.

This issue is addressed by JuezLTI, an international project whose goal is to create an innovative tool to allow the automatic assessment of exercises in a wide range of computer science topics. These topics include different languages used in computer science for programming, markup, and database management.

JuezLTI borrows part of its name from the IMS Learning Tools Interoperability (IMS LTI) standard. With this standard, the tool will interoperate with reference systems such as Moodle, Sakai, Canvas, or Blackboard, among many others. Another contribution of JuezLTI will be a pool of exercises. Interoperability and content are expected to foster the adoption of JuezLTI by many institutions. This paper presents the JuezLTI project, its architecture, and its main components.

2012 ACM Subject Classification Applied computing → Computer-managed instruction; Applied computing → Interactive learning environments; Applied computing → E-learning

Keywords and phrases programming, interoperability, automatic assessment, programming exercises

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.9

Funding JuezLTI is Co-funded by the Erasmus+ Programme of the European Union with ref. 2020-1-ES01-KA226-VET-096004. The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

1 Introduction

Computing skills will be among the most in-demand skills of this new decade [3]. These skills range from building a cloud platform to maintaining it and require solid knowledge in programming, database, and markup languages.

Most Learning Management Systems (LMS) do not support automated assessment in these domains. Hence, academic environments resort to the best-of-breed learning tools. To integrate these tools into the LMS they use several non-standard approaches, ranging from plugins to ad hoc Web services.

The IMS Learning Tools Interoperability (IMS LTI) specification [4] emerged in the last decade to facilitate the integration of the LMS with external tools. Its main goal is to release learning agents from the burden of having to authenticate themselves in multiple tools to benefit from a more engageable learning experience.

Nonetheless, to the best of the authors' knowledge, there is yet no available open tool, seamlessly integrated with the LMS, supporting the automatic assessment of computer science exercises from different domains, including but limited to computer language programming.

This paper presents the current status of a tool for automatic assessment of computer science exercises, under development as part of an international project within the scope of the Erasmus+ programme, key action: *Cooperation for innovation and the exchange of good practices* [8]. The objective of this project is a tool – called JuezLTI – to support the automatic assessment of markup languages, programming, and databases exercises.

As its name suggests, JuezLTI interoperates with online learning environments such as Moodle, Sakai, Canvas, or Blackboard (among many others), thanks to the LTI standard. The improved interoperability and the pool of exercises to be developed will open it to many institutions.The target audience of JuezLTI is computer science instructors and students (also self-education). All the project outputs will be freely available on the Internet under open-source licenses.

The remainder of this paper is organized as follows. Section 2 surveys both CS learning environments and the models they use for LMS integration. Section 3 presents the JuezLTI architecture and its main components. A particular emphasis is placed on the interoperability ensured by the LTI specification. The final section summarizes the current status and identifies opportunities for future developments.

2 Related work

Learning Management Systems (LMS) play a central role in any e-learning ecosystem. These systems were created to deliver course content, collect student assignments, and evolved to provide more versatile and engaging tools, including exercise assessment.

The Computer Science (CS) domain is an apt example of this evolution. In the last decades, due to the high demand of computing skills, a panoply of web-based interactive exercise systems were created to foster coding practice. In that time, most LMS lacked out-of-the-box support for automatic assessment of programming exercises, which is an important feature for a computer science learning environment. This lack of support meant that teachers and students had to switch between tools to enhance programming language learning. In order to solve these issues, some LMS offer integration of external tools as simple modules known as plugins. Some interesting examples are Virtual Programming Lab (VPL)[1], eLiza [2] and Javaunittest[2].

[1] https://vpl.dis.ulpgc.es/
[2] https://moodle.org/plugins/qtype_javaunittest

A Virtual Programming Lab (VPL) is a programming assignment management system where students can edit and execute programs, with automatic and continuous assessment. eLiza [2] is a game-based educational tool, developed and used to support the teaching and learning of programming languages and paradigms related to the development of web applications. Javaunittest is a Java question type that allows teachers to create Java questions and test the knowledge of students about Java programming. The students type source code for a given interface in Java and the response gets graded automatically.

Table 1 CS exercises assessment systems integrated into LMS.

Systems	Module^1	WS^2	LTI v.1^3
ACOS			X
Codeboard			X
CodeCheck			X
CodeHS			X
CodeWorkOut			X
DBQA			X
eLiza	X		
javaunittest	X		
PERE		X	
Virtual Progr. Lab	X		
Web-Cat			X

^{1}Module = implemented as a plugin
^{2}WS = implemented as a web service
^{3}LTI 1 = used the LTI specification 1.0 or 1.1

Despite its apparently success, this approach hinder the modularization and interoperability of tools. In a world where LMS were appearing at a fast pace, other specifications were born to overcome the LMS dependency avoiding the need to create for each tool a module for each LMS supported. One notorious example is the IMS LTI specification which offers seamless integration with most popular LMS in the market such as Moodle, Canvas, Blackboard, Sakai, and others. In this case the LMS takes the role of a platform (tool consumer) whereas the external tool serves as a tool (tool provider). A platform can send (anonymous) student information to the external tool and, in return, the tool reports the grades of students back to the platform (LMS). This way, programming exercises can be seamlessly integrated into every platform that supports LTI [9]. With the evolution and consequent acceptation of the specification a big number of computer programming environments started to support LTI. Most popular tools are CodeWorkOut, DBQA and Web-Cat.

CodeWorkOut[3] is an online exercise system to help people learn a programming language for the first time. It is a free, open-source solution for practicing small programming problems. Web-CAT [15] is a flexible, tailorable automated grading system designed to process computer programming assignments. Database Query Analyzer (DBQA) [7] is a tool that illustrates the effects that clauses and conditions have on an SQL SELECT statement using a visualized, data-oriented approach.

Table 1 presents a few of the most popular solutions and their interoperability features organized in three integration flavours: as a plugin, using ad hoc Web Services (WS) and using LTI specification (v1.0/1). As can be seen, most systems support the LTI v1.1 specification

[3] `https://codeworkout.cs.vt.edu/`

for integration with the LMS. The most recent version of LTI (v1.3) is not yet supported by these systems; this may be due to being a recent specification and with a higher degree of complexity than previous versions.

3 JuezLTI

This section presents JuezLTI – a tool integrated into the LMS to assess exercises in languages used in computing. Examples of such languages are markup languages (a domain where few assessment tools are available), database management languages, and programming languages. Subsection 3.1 provides an overview of JuezLTI, presents its architecture, and introduces its components. The following subsections describe the main components types of this architecture. Finally, Subsection 3.5 discusses the integration of JuezLTI on the LMS.

3.1 General overview

The goal of JuezLTI is to provide assessment of exercises in a wide range of languages used in computing within the LMS. To attain this goal JuezLTI relies on a combination of features:

1. the interoperability with the LMS provided by the TSUGI framework;
2. a modular design allowing the incorporation of evaluators for new domains;
3. a generic feedback system to help students to overcome their difficulties;
4. a centralized repository of exercises from where they can be exported and imported.

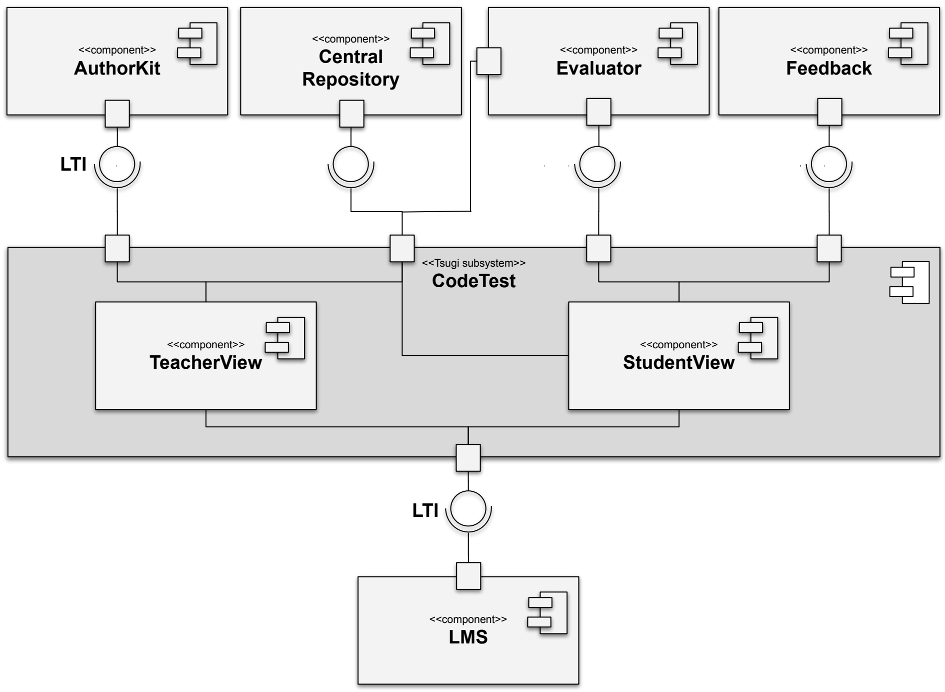


Figure 1 JuezLTI internal and external components.

The architecture of JuezLTI is depicted by the UML component diagram in Figure 1. At the center of this architecture is CodeTest, the main component based on TSUGI [6], the framework that provides LTI support, and its sub-components TeacherView and StudentView.

The former is a web environment where the teacher configures the activity by adding exercises from a central repository. The latter is a web environment where the student attempt to solve the proposed exercises. The TSUGI-based component relies on several kinds of components depicted on the top of the diagram in Figure 1. From left to right they are Autorkit, Central Repository, Evaluator, and Feedback. AuthorKit [12] is an exercise authoring system that can be used by teachers to create exercises for JuezLTI. The central repository acts as a pool of exercises for the system. The evaluator component runs the code of the students and checks the results. Finally, the feedback system provides information about the result of exercise resolution.

JuezLTI is an open-source project distributed under an Apache 2.0 license. It can be installed on the servers of any institution and configured to communicate with the institution's Learning Management System. It is available on GitHub in two different forms: production and development. The former allows the deployment of the entire system (several docker components) on a production server open to the Internet. With the latter, anyone can build the platform locally and contribute to JuezLTI development.

3.2 Exercise resolution

There are several tools described in the literature developed to automatically assess programming exercises, using different approaches such as static or dynamic evaluation of code; some examples are AutoGrader, eGrader, or Kassandra [17]. It's also easy to find tools to grade SQL assessments, such as Learn-SQL, a tool based on the IMS QTI specification [1]; unfortunately, a closed solution. But, despite the massive presence of markup languages in the computer science curriculum, there are few solutions to assess XML exercises.

JuezLTI proposes a tool that supports the evaluation of programming, databases and XML exercises, but that can be extended to other types of exercises given its highly modular architecture.

The module in charge of exercise resolution is called CodeTest. CodeTest relies on other JuezLTI modules for retrieving exercises, evaluating code, and generating feedback. It has web interfaces for two roles: teachers and students. The teacher's role is to create, edit, import, and export a set of exercises. Teachers can also review the results of every single student. Students view exercise statements and have a code editor to introduce their resolution. As JuezLTI supports multiple programming languages, the student can choose the language to solve the exercise, restricted to the context of the exercise. For instance, in a programming language exercise, the student may have the possibility to select among different languages such as Java, Python, or C.

JuezLTI also acts as a central repository of questions and answers. To do that, it stores all the exercises proposed in a non-relational database. External questions can also be imported using the tool AuthorKit.

3.3 Evaluation

The core of JuezLTI is the automated evaluation of different kinds of computer science exercises. According to the kind of exercise, a different module performs the evaluation. Currently, JuezLTI has in development modules to evaluate exercises in programming languages, markup language, and relational databases.

A module responsible for the evaluation of a kind of exercises is called an *evaluator*. Evaluators are micro-services, with their own internal structure, running on their own containers. For instance, a relational databases evaluator can have its own database engine, as a programming language evaluator may have installed several compilers for the languages it supports. These micro-services share a common API used by CodeTest to invoke them. This API specifies the data sent to evaluators – exercises, student attempts – and return by them – evaluation reports.

Student attempts are plain text files. In contrast, exercises are a complex content, including several files – a statement, one or more solutions , several test cases and other auxiliary files – as well as specialized metadata. For instance, a relational database exercise requesting an SQL query may include a database dump and a reference solution; a programming language exercise may include a set of input files and corresponding expected output. Exercises sent to evaluators are encoded in the YAPExIL [13].

An evaluator returns a report detailing the assessment performed on the student's attempt using the data provided by the exercise. This report includes possible compilation errors, including resource usage - memory and time. It may also include hints associated with test cases if these were provided by the exercise. Evaluation reports are encoded as a JSON document in the Programming Exercise Evaluation Assessment Report Language (PEARL). This specification is an evolution of similar XML-based specification [10] developed specifically for JuezLTI.

3.4 Feedback

The role of the feedback manager is to mediate between evaluators and students to distill effective feedback from evaluation reports. Evaluators produce reports with perfusion of details. This information needs to be summarized to make it understandable to students.

Students typically submit several attempts before solving an exercise. Hence, feedback messages must avoid being repetitive and strive to be increasingly more informative. For instance, a first feedback message may simply acknowledge that most students have difficulty in solving the exercise in their first attempt; a second message may provide a hint associated with a failed test case in the evaluation report; a third may provide an input that causes the program to fail.

On the other hand, automated feedback may also be detrimental to some students if they use it as a sort of oracle that constantly solves their difficulties. The feedback manager must also ensure that the information it provides does not scale up too quickly in reaction to submissions that are too similar to the previous ones.

3.5 Interoperability

JuezLTI uses version 1.0 of the Learning Tools Interoperability standard (LTI) for integration into the LMS. This interoperability standard was proposed by the IMS Global Learning Consortium [14] and is supported by reference LMS vendors such as Moodle, Canvas, or Sakai.

With LTI a set of educational services can be used to extend the functionality of the LMS [11] using a Service Oriented Architecture (SOA) [14]. The LMS becomes a marketplace where the teacher can select the resources from different providers to improve the learning experience.

JuezLTI is based on TSUGI, a framework that simplifies the development and deployment of LTI applications [6]. The first application of TSUGI was a self-contained MOOC for training in Python – py4e.com. It was deployed by one of its authors, Dr. Charles Severance, founder of the Open Source LMS Sakai and currently working for IMS. LTI has been widely used since 2010, not only for code testing purposes, as in [16], but also for providing complex assessments not directly supported by the LMS, such as Dig4E, a tool to learn about standards for creating high-quality digital still images [5].

To use JuezLTI the teacher has to request a key/secret pair on the project's website. With those credentials and the supplied URL, the teacher adds an external activity in the LMS. The students are directly identified in JuezLTI using the information provided by the LMS. Whenever a student or teacher opens the activity, the LMS communicates with JuezLTI using LTI (via the TSUGI framework) to identify the user and present the appropriate interface. Then, the LMS sends the `resource_link_id` property identifying the relevant activity. This way, different activities can be added using the same key. Finally, JuezLTI reports grades of solved exercises back to the LMS using an LTI service.

In short, there is bidirectional communication between JuezLTI and the LMS. JuezLTI stores the information of students and their results and sends the grades obtained back to the LMS.

4 Conclusion

The main contribution of the research reported in this paper is JuezLTI – a platform to assess computer science exercises. JuezLTI is an interoperable, open-source, and modular platform. Usability, easiness of use, and extensibility were its main design goals.

JuezLTI supports exercises in different kinds of languages, including programming, database management, and markup. Due to its modular structure, evaluators for new domains can be easily added. It will have access to a centralized repository of exercises adapted to EQF levels, which will promote its adoption by teaching institutions. The TSUGI framework, upon which JuezLTI is based, supports LTI version 1.0, and future versions of the framework will gain access to more advanced versions of the standard with limited recodification. JuezLTI is a work-in-progress. It is currently under beta testing, and some of its components are still being improved, reflecting the suggestions and bugs reported by early adopters. This project intends to contribute with a collection of exercises compatible with its evaluators. Collecting and generating these exercises is also part of our immediate future work.

References

1 Alberto Abelló, M. Elena Rodríguez, Toni Urpí, Xavier Burgués, M. José Casany, Carme Martín, and Carme Quer. LEARN-SQL: Automatic assessment of SQL based on IMS QTI specification. In *2008 Eighth IEEE International Conference on Advanced Learning Technologies*, pages 592–593, 2008. `doi:10.1109/ICALT.2008.27`.

2 Míriam Antón-Rodríguez, María Ángeles Pérez-Juárez, Francisco Javier Díaz-Pernas, David González-Ortega, Mario Martínez-Zarzuela, and Javier Manuel Aguiar-Pérez. An Experience of Game-Based Learning in Web Applications Development Courses. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASIcs)*, pages 3:1–3:11, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ICPEC.2020.3`.

3 European Commission. Coding – the 21st century skill, 2018. accessed on 20 Jan 2020. URL: `https://ec.europa.eu/digital-single-market/en/coding-21st-century-skill`.

4 IMS Global Learning Consortium. Learning tools interoperability core specification, 2019. accessed on 14 Apr 2022. URL: `http://www.imsglobal.org/spec/lti/v1p3/`.

5 Paul Conway and Ann Arbor. Digitization for everybody (Dig4E). *Archiving Conference*, 2020:12–16, April 2020. `doi:10.2352/issn.2168-3204.2020.1.0.12`.

6 Nikolas Galanis, Marc Alier, María José Casany, Enric Mayol, and Charles Severance. Tsugi: A framework for building PHP-based learning tools. In *Proceedings of the Second International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM '14, pages 409–413, New York, NY, USA, 2014. Association for Computing Machinery. `doi:10.1145/2669711.2669932`.

7 Ryan Hardt and Esther Gutzmer. Database query analyzer (DBQA): A data-oriented SQL clause visualization tool. In *Proceedings of the 18th Annual Conference on Information Technology Education*, SIGITE '17, pages 147–152, New York, NY, USA, 2017. Association for Computing Machinery. `doi:10.1145/3125659.3125688`.

8 JuezLTI Project Consortium. JuezLTI - automatic assessment of computing exercises using LTI standard, 2021. accessed on 12 Apr 2022. URL: `https://juezlti.eu`.

9 José Paulo Leal and Ricardo Queirós. Using the learning tools interoperability framework for LMS integration in service oriented architectures. *Technology Enhanced Learning TECH-EDUCATION'11*, 2011.

10 José Paulo Leal, Ricardo Queirós, and Duarte Ferreira. Specifying a programming exercises evaluation service on the e-framework. In Xiangfeng Luo, Marc Spaniol, Lizhe Wang, Qing Li, Wolfgang Nejdl, and Wu Zhang, editors, *Advances in Web-Based Learning - ICWL 2010 - 9th International Conference, Shanghai, China, December 8-10, 2010. Proceedings*, volume 6483 of *Lecture Notes in Computer Science*, pages 141–150. Springer, 2010. `doi:10.1007/978-3-642-17407-0_15`.

11 Jeff Merriman, Tom Coppeto, Francesc Santanach Delisau, Cole Shaw, and Xavier Aracil Díaz. *Next generation learning architecture.* eLearn Center. Universitat Oberta de Catalunya, 2016.

12 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.

13 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, and Jakub Swacha. Yet Another Programming Exercises Interoperability Language (Short Paper). In Alberto Simões, Pedro Rangel Henriques, and Ricardo Queirós, editors, *9th Symposium on Languages, Applications and Technologies (SLATE 2020)*, volume 83 of *OpenAccess Series in Informatics (OASIcs)*, pages 14:1–14:8, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.SLATE.2020.14`.

14 Charles Severance, Ted Hanss, and Josepth Hardin. IMS learning tools interoperability: Enabling a mash-up approach to teaching and learning tools. *Technology, Instruction, Cognition and Learning*, 7(3-4):245–262, 2010.

15 Anuj Ramesh Shah. *Web-cat: A web-based center for automated testing.* PhD thesis, Virginia Tech, 2003.

16 Antonio J. Sierra, Álvaro Martín-Rodríguez, Teresa Ariza, Javier Muñoz-Calle, and Francisco J. Fernández-Jiménez. LTI for interoperating e-assessment tools with LMS. In Mauro Caporuscio, Fernando De la Prieta, Tania Di Mascio, Rosella Gennari, Javier Gutiérrez Rodríguez, and Pierpaolo Vittorini, editors, *Methodologies and Intelligent Systems for Technology Enhanced Learning*, pages 173–181, Cham, 2016. Springer International Publishing.

17 Zahid Ullah, Adidah Lajis, Mona Jamjoom, Abdulrahman Altalhi, Abdullah Al-Ghamdi, and Farrukh Saleem. The effect of automatic assessment on novice programming: Strengths and limitations of existing systems. *Computer Applications in Engineering Education*, 26(6):2328–2341, 2018. `doi:10.1002/cae.21974`.

WebPuppet – A Tiny Automated Web UI Testing Tool

Ricardo Queirós
CRACS – INESC-Porto LA & uniMAD, ESMAD/P. Porto, Portugal

Abstract

One of the most important phases in the Web development cycle is testing. There are several types of tests, different approaches to their use and a wide range of tools. However, most of them are not open source, require coding and do not have a pedagogical nature. This article introduces WebPuppet as an automated Web UI testing tool. The tool is distributed as a small Node package and can be easily integrated into any learning environment in the web development domain. In addition, it does not require coding in any language, just use a very simple domain-specific language that will generate a test script to run in client applications. In order to exemplify its use, a simple test scenario based on a login page is presented.

2012 ACM Subject Classification Applied computing → Computer-managed instruction; Applied computing → Interactive learning environments; Applied computing → E-learning

Keywords and phrases User Interface Testing, Web development, DOM

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.10

1 Introduction

Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do. There are several types of tests that can be performed in a Web product ranging from unit, functional to end-to-end and integration. Despite the usefulness of all these types, User interface (UI) testing is one of the most important in the Web development cycle. In order to validate whether applications have the desired aesthetics and functionalities, Quality Assurance (QA) professionals should test all interface components. This action will not only improve the software quality but also ensures a rich experience for end users while using the Web application.

UI testing plays a significant role before an application is released to production. UI testing is centered around two main things. First, checking how the application handles user actions carried out using the keyboard, mouse, and other input devices. Second, checking whether visual elements (e.g. text boxes, checkboxes, radio buttons, menus, toolbars, colors, fonts, and others) are displayed and working correctly.

The test can be performed manually or with an automated testing tool. There are several tools that support UI testing (e.g. Parasoft Selenic[1], Katalon[2], Selenium IDE[3], Mabl[4], Perfecto[5]). Most of these tools have great features such as recording abilities, locators recommendations, BDD support, CI/CD integration, self-healing capabilities and many supported languages. Although these features, some of them are proprietary, too complex and not pedagogical-driven. Regardless of the method and tool used, the goal is to ensure all UI elements meet the requested specifications.

[1] `https://www.parasoft.com/products/parasoft-selenic`
[2] `https://katalon.com/`
[3] `https://www.leapwork.com/discover/selenium-automation`
[4] `https://www.mabl.com`
[5] `https://www.perfecto.io/`

This paper presents WebPuppet as an automated Web UI Testing tool. The real motivation for its creation was to test the UI of small web applications created by students in web development courses. In this way, the teacher does not need to manually create tests that are always time consuming and error-prone and can thus concentrate on making more differentiating and impactful exercises. In its genesis, the tool receives as input an instance with all test cases based on a domain-specific language that will generate a test script to be executed later in all student applications. As an output, a report is generated for the student with the identification of all errors and possible solutions.

The remainder is organized as follows. Section 2 provides a background on UI testing approaches. Section 3 presents the WebPuppet tool and its domain-specific language. Section 4 presents a simple test scenario where the tool can be used. Finally, Section 5 summarizes the contributions of this work and points to future directions.

2 UI testing approaches

A test is a code which can run automatically in a client application to validate a user interface in order to meet design and logic requirements. Often we group a set of related tests in a test case and organize a set of test cases with different priorities and dependencies in a test scenario. A test case can have tests of two types:

- **Simulation tests** – those responsible for simulating the user's behavior;
- **Validation tests** – those responsible for checking, after we simulated some user behavior, if the system worked properly.

In the literature several solutions can be found to perform tests on graphical interfaces which resort on computer vision [1], web interface matching algorithms [2], structural comparison (comparing the trees resulting from the two HTML documents) [4], or with very different goals such as detecting phishing sites [3]. The goal of this section is not to compare the different solutions found, but rather compare the different approaches that can be used to automate Web UI testing. There are three main UI testing approaches, namely: manual testing, record/playback and keyword/data-driven scripting. The following subsections detail each one.

2.1 Manual testing

A common way of testing the UI of a Web application is to directly write code in a programming language like JavaScript, Java, PHP or C++. Often this will be the same programming language used to write the application that is being tested. Using this approach, a human tester (or a team) performs a set of operations to check whether the application is functioning correctly and that the graphical elements are conformed to the documented requirements.

Several frameworks allow the creation of tests for various platforms. The most notable examples are Selenium (for Web applications), Appium (for mobile), and Microsoft Coded UI (for Windows applications).

Despite being an flexible approach easy to implement, manual-based testing has some drawbacks such as it can be time-consuming, and the test coverage is extremely low. Additionally, the quality of testing in this approach depends excessively on the knowledge and capabilities of the human tester or testing team.

2.2 Record-and-Playback Testing

This approach resorts to automation tools which records all tasks, actions, and interactions with the application. The recorded steps are then reproduced, executed, and compared with the expected behavior. For further testing, the replay phase can be repeated with various data sets.

Using record-and-playback testing the simulation part of the script is relatively easy to capture just performing the relevant user actions and the system creates a script. However, the validation part is more difficult to achieve. After simulating the user actions, for each element to check on the screen, it is necessary to explicitly add steps to the script in order to identify these elements in the interface, and compare their values to expected values. As obvious, it is impossible to record these validations because they are not user actions. Thus, it is mandatory to define them one by one using the automation system's GUI.

This approach typically generate test scripts behind the scenes in simple scripting languages like VBScript. Often, advanced users manipulate the code directly to make small adjustments.

2.3 Keyword/Data-Driven Testing

Other test frameworks support the definition of a set of "keywords" which specify user actions. This approach calls **keyword testing**. A human tester can specify these keywords and a script will be generated that performs the desired actions on the system under test. Listing 1 shows a small example:

Listing 1 Keyword Scripting Test example.

```
Open Browser To Login Page
Input Username david
Input password 16485
Submit Credentials
```

In this case, the first line tell that the script should navigate to the login screen and enter certain user credentials. A script will be generated that knows how to navigate to the login screen, type in the data provided and submit the form.

Using this approach, the simulation part of the script will be handled by one keyword that defines the user action (e.g. "login page" in the example above). The validation part of your script will require multiple keywords and multiple values of expected data for each part of the UI to validate. From the tester's perspective it is just using the keywords, and don't need to deal with the code. However, this means that typically few validation options are available, and adding more will require help from the responsible for the generation script.

A variant of this approach, called **data-driven testing** (Figure 1), is when the same test needs to be repeated several times with different data values or different user operations.

Data-driven testing (DDT) is data that is external to your functional tests, and is loaded and used to extend the automated test cases. The same test case cab be taken and run it with as many different inputs, thus getting better coverage from a single test.

The advantage of this testing approach is that the code that simulates user operations is not repeated in every test script, rather it is defined in one place, and a tester can use it as a building block in multiple test cases. This reduces the code writing and also the maintenance required as the application under test changes.

One of the most famous testing frameworks which uses this approach is the open source Robot Framework. There are others test automation frameworks which provides a mix between keyword-driven or data-driven testing functionality.

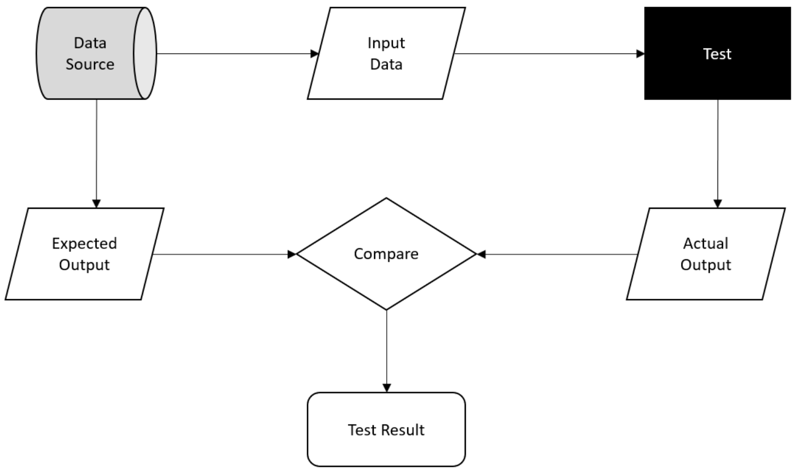


Figure 1 Data-driven Testing.

3 WebPuppet

WebPuppet is a small tool for automating web application UI tests. Its creation arises from the need to support the evaluation of students' performance in the creation of Web applications in learning environments. The architecture of WebPuppet is straightforward and is composed of the following components:

- The editor – Web-based component with a GUI for the definition of test scenarios.
- The engine – client component responsible for the generation of the test script that will run on the client application.

3.1 The Editor

The editor is a Web-based component for the creation of a test scenario. A test scenario is a document that explains how the application under test will be used in real life. A simple test scenario could be: "users will successfully sign in with a valid username and password". In this scenario, we can have tests for multiple GUI events (e.g. provide a valid username and password combination, enter an invalid username, hit the login button). These tests are grouped in test cases for logic organization and dependency.

A test scenario is defined as a domain-specific language (DSL) formalized with a JSON Schema. As said before, a test scenario is composed by one or more test cases. Listing 2 presents how a test case is described.

Listing 2 Test case schema.

```
{
 "$schema": "http://json-schema.org/draft-04/schema#",
 "description": "A schema to formalize a Test Case",
 "type": "object",
 "properties": {
   "id": {"type": "integer"},
   "description": {"type": "string"},
   "depends": {"type": "array", "items": {"type": "integer"}},
   "tests": {"type":"array", "items": {"$ref": "#/definitions/Test"}}
 },
 "required": ["id", "description", "depends", "tests"]
}
```

The `id` property is the test case identifier. The `description` property describes in natural language the test case. The `depends` property refers to the test case(s) that this test case depends on. Finally, the `tests` property is an array with all the tests that compose a test case.

A test (Listing 3) is the smallest unit in the DSL and will actually contain the test to be performed in the client code.

Listing 3 Test schema.

```
{
 "$schema": "http://json-schema.org/draft-04/schema#",
 "description": "A schema to formalize a single test",
 "type": "object",
 "properties": {
   "id": { "type": "integer" },
   "description": { "type": "string" },
   "selector": { "type": "string" },
   "action": { "enum": ["VALIDATE", "GET", "FILL", "CLICK"] },
   "operator": { "enum": ["=", "!=", ">", "<","..."] },
   "value": { "type":"string" },
   "error": {"type": "string" }
 },
 "required": ["id", "selector", "action"]
}
```

The `id` property is the test identifier. The `description` property describes in natural language the test. The `selector` property is a CSS selector responsible to find the element to test in the DOM tree. The `action` property is a enumeration of all the actions that can be made in the element selected:

- `VALIDATE` – verify if certain DOM element or attribute exist;
- `GET` – get a value from a DOM element or attribute and compare it with a specific value. In this case two more actions should be defined:
  - `OPERATOR` – the operator to use in the comparison;
  - `VALUE` – the value to compare.
- `FILL` – inject text in a selected text box or select an item in a selected combo box;
- `CLICK` – click in a selected button, radio button or checkbox.

The `error` property is a string with a feedback to be delivered to the student when the test is not passed.

3.2 The Engine

The engine is a JavaScript file that will receive as input an instance of a test scenario formalized in the previous DSL and will generate a test script to be executed in the client application code created by the student. All tests will run according to predefined dependencies and the generated feedback will be presented to the student.

A test script is code that can be run automatically to perform a test on a user interface. The code will typically do the following one or more times: (1) Identify input elements in the UI, (2) Simulate user input, (3) Identify output elements, (4) Assert that output value is equal to expected value, and (5) Write the result of the test to a log.

The engine uses Puppeteer to simulate user actions. Puppeteer is a Node library that provides a high-level API to control headless Chrome or Chromium browsers over the DevTools Protocol. It can also be configured to use full (non-headless) Chrome or Chromium.

In Table 1, we present some of the functions of the WebPuppet engine.

Table 1 WebPuppet engine functions.

Function	Description
TestScenario loadTestScenario(JSON scenario)	Creates a new test scenario based on a WebPuppet DSL scenario
TestCase TestScenario.getTestCase(int id)	Get a test cased based on a given id
List<Test> TestCase.getTests()	Obtains the tests of a specific test case
Object TestScenario.run()	Execute all test cases automatically and returns a JSON object with the feedback

In this moment, only the first and the last functions are implemented. This means that currently the test scenario must be created manually.

4 UI testing scenario

This section presents a typical test scenario of the login function on a website. In this scenario, we can do the following:

1. Load the website homepage
2. Locate the “username” and “password” text-boxes and the “submit” button in the home screen;
3. Type the username “ricardo” and password “12345”
4. identify the “submit” button and click it
5. Wait and locate the title of the Welcome screen that appears after login
6. Read the title of the Welcome screen.
7. Assert that the title text is “Welcome ricardo”.
8. If title text is as expected, record that the test passed. Otherwise, record that the test failed.

It is important to distinguish two important parts of the test scenario:

- The **simulation** part of the script (steps 1, 3 and 4) are responsible for simulating the user’s behavior;
- The **validation** part of the script (steps 2, 5–8) are responsible for checking, after we simulated some user behavior, if the system worked properly.

This scenario should be defined through the DSL mentioned in the previous section. Listing 4 presents a snippet of a test instance, more precisely, the fulfilment of the login form with specific data and the verification, after submission, of the authenticated user name. If the verification is not successful, the student receives automatic feedback based on the value of the `error` property.

Listing 4 Test instance.

```
{
 "id": "3",
 "description": "Fill the login form elements
    and inspect the name of the logged user",
 "depends": "1",
 "tests": [{
   "id": "1",
   "selector": "#username",
   "action": "FILL",
   "value": "ricardo"
  }, {
   "id": "2",
   "selector": "#password",
   "action": "FILL",
   "value": "12345"
  }, {
   "id": "3",
   "selector": "#loginbtn",
   "action": "CLICK",
   "value": "new"
   }, {
    "id": "4",
    "selector": ".usertext",
    "action": "GET",
    "operator": "=",
    "value": "Ricardo Queiros",
    "error": "authentication failed"
    }
  ]
}
```

Using these basic elements of GUI automation, simulation and validation, allows testing even in very complex multi-step operations. In complex test scripts these elements will repeat themselves several times, each time for a different part of the user's workflow.

5 Conclusion

This article introduces an automated UI testing tool called WebPuppet. The purpose of the tool is not to compete with existing ones, but to simplify the testing process as much as possible and orient it towards a more pedagogical nature by defining a simple feedback system for each of the tests.

The tool can be easily integrated in learning environments where students are challenged to create Web applications and used to test their solutions. This way, the tool can relief the burden of manual evaluation of each Web application bu the teacher.

Right now the tool is in beta stage, but it can already be installed and used as an npm package (the package manager for the Node JavaScript platform). As future work it is intended:

- to allow the automatic creation of the test scenario through an API;
- to support the task recording feature in the browser.

References

1 Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller. Gui testing using computer vision. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1535–1544, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1753326.1753555.

2 Marco Primo and José Paulo Leal. Matching User Interfaces to Assess Simple Web Applications. In Pedro Rangel Henriques, Filipe Portela, Ricardo Queirós, and Alberto Simões, editors, *Second International Computer Programming Education Conference (ICPEC 2021)*, volume 91 of *Open Access Series in Informatics (OASIcs)*, pages 7:1–7:6, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2021.7.

3 Gaurav Varshney, Manoj Misra, and Pradeep K. Atrey. A survey and classification of web phishing detection schemes. *Security and Communication Networks*, 9(18):6266–6284, 2016. doi:10.1002/sec.1674.

4 Jiří Štěpánek and Monika Šimková. Comparing web pages in terms of inner structure. *Procedia - Social and Behavioral Sciences*, 83:458–462, 2013. 2nd World Conference on Educational Technology Research. doi:10.1016/j.sbspro.2013.06.090.

Learning Computer Programming: A Gamified Approach

Mário Pinto
uniMAD – ESMAD, Polytechnic of Porto, Portugal

Teresa Terroso
uniMAD – ESMAD, Polytechnic of Porto, Portugal

Abstract

Learning computer programming is a difficult task for most students who start learning in this field. In fact, many students refer that learning computer programming is an arduous and difficult task, presenting some fear in addressing these issues. However, the main challenge for beginners is not in the language or syntax, but in devising a solution to solve the proposed problem. On the other hand, the younger audience is used to clicking on an icon and seeing an application with an appealing interface! Thus, students are often discouraged when, in a classroom, they have to implement an algorithm to classify numbers or sequences of characters and print them, sometimes in unappealing development environments. The lack of immediate feedback on the solution proposed by the student is another aspect that promotes some demotivation, as students often have no real idea of where they went wrong and how they can improve the solution they present.
This paper describes the introduction of gamification elements in an introductory programming course, based on a conceptual framework proposed by Piteira. The main objective is to motivate and involve students in the learning process, through the introduction of strategies based on gamification (such as challenges, progression and levels), as well as strategies such as problem-based learning, seeking to make teaching programming more attractive and appealing to students. The paper describes a case study carried out in the course of algorithms and data structures in School of Media Arts and Design, at Polytechnic of Porto, in Portugal.

2012 ACM Subject Classification Computing methodologies; Applied computing → Computer-managed instruction

Keywords and phrases programming learning, teaching methodologies, teaching-learning, higher education, gamification

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.11

1 Introduction

Learning to program is often a difficult process to which traditional teaching approaches have not been able to respond effectively [15, 5, 7, 10]. Introduction to programming courses or disciplines have, with some frequency, failure rates above 50%, according to several studies. This reality also occurs in higher education institutions, in courses and areas of computer science and Web development. Numerous studies have been carried out on this reality, and point to some causes, such as [8, 12, 7]:

- Difficulties in interpreting and understanding the proposed problems, often leading students to start solving a problem without fully understanding it;
- Difficulties in solving logical and concrete problems, in the form of algorithms;
- Difficulties in capturing students' attention and interest in learning the fundamentals of programming, with learning often oriented towards solving more or less abstract problems and using unappealing interfaces.

Teaching programming can also be difficult, because learning programming requires much more than acquiring technical knowledge and skills. Students should learn about programming structures, logic, and syntax of the language used. But they must also quickly begin to build strategies to combine the knowledge acquired in solving problems through algorithms and their computer coding. The task is not yet complete because the code must be tested, debugged, and often reworked and optimized. In this context, some programming learning approaches and strategies tend to be adopted, in order to relieve the difficulties felt by students, such as [2, 6]:

- The introduction of dynamic elements in learning strategies, such as gamification, which uses game design elements in educational contexts. The introduction of techniques and strategies that include elements such as missions or challenges, embedded in a learning narrative, progression mechanisms such as points or levels, relationships that promote cooperation in the development of a solution and peer evaluation, among others, can help to increase student involvement and motivation [14];
- Automatic code evaluation systems, based on intelligent tutors. These systems allow students to progress in their learning outside the classroom environment, having automatic and immediate feedback on the solutions they submit. They also make it possible to increase students' degree of autonomy, often providing personalized learning paths depending on the profile and skills of each student [1];
- Systems that use visual representations, animations and simulation of algorithms, seeking to make learning more dynamic, visual and interactive. These are animation systems with the purpose of appealing to the potential of the human visual system, contributing to a better understanding of inherently dynamic concepts, when compared to the textual format.

These approaches do not, by themselves, solve the difficulties felt in the initial learning of programming. But when combined with appropriate pedagogical strategies, they can help to relieve these same difficulties, converting learning into a more accessible and motivating process for students.

Thus, this paper begins by contextualizing the concept of gamification elements, distinguishing them into components, mechanics and dynamics, followed by a brief summary about gamification design frameworks, which can guide us in the process of implementing a gamified strategy. Section 3 describes the methodology adopted, which is based on a simplified approach to the 6D framework [16], applied to the teaching context. The following section presents the results obtained in the curricular unit under study, followed by its discussion and brief conclusions.

2 Gamification in programming learning

Gamification is the use of typical game elements (such as experience points, badges, progress indicators, levels or achievements) in other different contexts than games, in order to involve and motivate users in achieving a specific goal. The gamification of teaching and learning environments aims to involve and motivate students through the inclusion of elements and mechanics present in game design. This inclusion makes it possible to reward students for successful daily tasks, provide instant and personalized feedback, present their progress in the course, and foster healthy cooperation and competition among students. In this context, gamification also seeks to involve students in learning activities, making them more fun and motivating. One of the main challenges in introductory programming education is the need to capture students' attention and motivation. Especially the younger audience, is accustomed to clicking on an icon and see an application with appealing interface popup! Thus, they become unmotivated when, in a classroom, they are asked to implement an algorithm to sort numbers or strings, and print them in the console, for instance. Gamification seeks precisely to address this problem through the implementation of new strategies for the development of knowledge, often also based on pedagogical approaches such as problem-based learning. According to some authors, developing a learning project based on gamification mechanisms, implies the implementation of three conceptual levels [16, 7]: i) components (such as achievements, badges, levels, points, etc.); ii) mechanisms (such as challenges, competition, cooperation, rewards, etc.); iii) dynamics (such as emoticons, relationships, progression status, etc.). Figure 1 shows an overview of game elements addressed for a gamification project implementation.

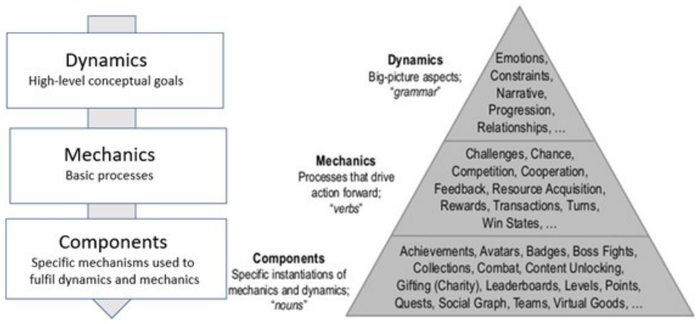

Dynamics
High-level conceptual goals
Mechanics
Basic processes
Components
Specific mechanisms used to fulfil dynamics and mechanics
Dynamics
Big-picture aspects; "grammar"
Emotions, Constraints, Narrative, Progression, Relationships, ...
Mechanics
Processes that drive action forward; "verbs"
Challenges, Chance, Competition, Cooperation, Feedback, Resource Acquisition, Rewards, Transactions, Turns, Win States, ...
Components
Specific instantiations of mechanics and dynamics; "nouns"
Achievements, Avatars, Badges, Boss Fights, Collections, Combat, Content Unlocking, Gifting (Charity), Leaderboards, Levels, Points, Quests, Social Graph, Teams, Virtual Goods, ...

Figure 1 Game elements pyramid for gamification [16].

Dynamics encompasses the big picture aspects of a gamified system. At the top of the pyramid, they are the most high-level conceptual elements in a game or gamified system. These are factors that absolutely must be considered, even if they don't enter directly into the game itself. Some examples of dynamics elements: constraints, emotions, narrative, progression, and relationships. The second group of elements is the mechanics. These are the basic processes that drive users to engage with the content and continue to drive the action forward, such as: challenges, chance, competition, cooperation, feedback, resource acquisition, rewards, transactions, turns, and win states. Components make up the largest group of game elements. In many ways the components are a more specific form of either dynamics and mechanics. These elements are less abstract than the first two categories and lead to actual tools that can be employed to begin to incorporate gamification in the environment of interest [10]: points, levels, badges, bonuses, notifications or peer evaluation of students are some of the elements that we can use at this level of abstraction [4, 9]. Based on the game elements pyramid for gamification (Figure 1), the gamified strategy adopted in this project sought to encourage students to progress in their teaching/learning process, in a more engaging and autonomous way. In this context, a mission was proposed to the students, based on 3 challenges. The challenges were associated with gamification elements such as points, content unlocking (according to the progress in challenges) and badges. Peer evaluation (by students) was also included, with the aim of rewarding those who were most active in cooperating with their colleagues. Challenges included various mechanisms for obtaining feedback from student work, such as:

- Points: based on user scenarios, points can contribute to other game mechanics, such as levels or progression. Points can also confer a bonus on the student's final evaluation. Points can be earned by completing the proposed challenges. We expected that students would get engaged to the acquisition of points and on re-taking their study.

- Levels: in an educational context, levels allow progression and sequence through contents and activities. Each module activity was designed to a level. Inside the module, students have sub-levels. Students needed to complete the activities to reach the next. You may also use levels to unlock content – new problems to solve!
- Badges: which are medals that reward users for specific behaviours, are some of the most visible elements of gamification because they confer status. Badges are most useful for students who rank high in external motivation.
- Peer evaluation of students: The goal is also to introduce some ethics in the process of student's assessment, rewarding those who are somehow recognized by their peers with better performance (be it more skills, greater availability to help colleagues, ability to cooperate, etc.). It often seeks to encourage cooperation, collaboration and recognition by your peers.

However, some authors [10, 11] emphasizes that deploying the appropriate mechanics and components actually comes at the very end of the design process. Several authors propose different frameworks that help to systematize the programming teaching/learning process, many of them using gamification elements. This paper does not seek to carry out an in-depth literature review on the processes of designing a gamified teaching-oriented strategy. However, in recent years there has been an effort to formalize the process of designing a gamification strategy, and several gamification design frameworks have emerged, such as MDA (Mechanics, Dynamics and Aesthetics), Octalysis, GAME (Gather, Act, Measure, Enrich) or the 6D framework [3, 13]. Piteira [8] in a recent research, proposes a conceptual framework to implement gamification on courses of computer programming learning. The proposed framework includes several progression steps, such as: i) knowing the target audience; ii) Learning goals and learning outcomes; iii) Structure of gamified learning; iv) Identify and organize the study contents; v) Apply gamification elements appropriate to the context, and considering the aspects mentioned in the above steps [16, 11].

3 Methodology Adopted

Recognizing the difficulties that students normally present in learning computer programming, we tried to define a new approach to improve the success rate of students in this field. Based on the concepts of the increasingly popular gamification, we tried to introduce some mechanisms of game design, in the curricular unit of algorithms and data structures.

This curricular unit takes place in 15 weeks, during the 1st semester of the 1st year, of the degree in technologies and information systems for the web, at Escola Superior de Media Artes e Design, Polytechnic of Porto. The course was attended by 56 students, and it was designed to cover the fundamental concepts of programming introduction using the Python programming language. The course was organized in 5 main modules: i) fundamentals of programming: data types, variables, operators, simple data structures, basic control structures (if, while, for, . . .); ii) more complex data structures such as arrays, lists, queues and stacks; iii) data persistence (files); iv) introduction to GUI programming: basic components (such as labels, text boxes, buttons, radio buttons, images, timers, etc.), objects properties and events; v) development of a final project, applying the previous concepts in the development of an application.

From the first week, the teaching/learning process was oriented towards the exposition of small theoretical-practical contents, followed by carrying out some practical exercises of application. However, from the beginning, there were many difficulties in understanding on the part of the students, as well as some lack of motivation and interest in carrying out the proposed practical activities. Once the first knowledge assessment test was carried out, scheduled for week 7 of the course, it was found that only 30 of the 56 students took the test (about 54% of the students). And among these, only 11 obtained a positive evaluation (representing about 37% of those who attend the evaluation moment, and only 20% of the enrolled students). So, considering:

- The results obtained, truly discouraging;
- The students' difficulty in assimilating basic programming concepts and applying them in practice;
- The lack of motivation of most students;
- The inability to understand and solve the proposed problems.

We tried to introduce new strategies, to respond to the difficulties felt by the students in this introductory course. The objective was to promote a new approach, based on the paradigm of gamification learning, as well as strategies such as problem-based learning, seeking to make teaching more attractive and appealing to students.

In this case study, the framework presented by Piteira at al [8, 9] was adopted, since it is oriented towards programming learning courses. Furthermore, the framework was applied with remarkable success by its authors, thus providing an opportunity to confirm the results obtained in previous studies. According to the process of implementing a gamified strategy, described in the previous section, a new plan was drawn, organized in the following steps (Table 1):

At the end of the gamified activity, a questionnaire survey was carried out, in order to understand the students' degree satisfaction with this initiative. The main objectives of the questionnaire focused on understanding the students' feedback on the gamified activity carried out, as well as if it was useful to extend this initiative to other modules of the course. Before the questionnaire was applied, we conducted a pre-test applied to a group of ten people, with similar characteristics to the final sample, to identify omissions and the level of understanding of the questions addressed in the questionnaire. The data collection took place at the end of the activity, between January 3 and 10, 2022. We received 21 responses, which corresponds to more than 90% of the students who joined this gamified initiative.

4 Results and Discussion

The questionnaire had four questions, with answers based on a likert scale, between 1 and 5, where 1 meant very little and 5 very much. Table 2 summarizes the answers obtained in these four questions.

Analyzing the results obtained, and considering the responses classified as levels 4 and 5 on the Likert scale (as much or very much), we can see that most students were satisfied or very satisfied with the strategy adopted. Around 100% of the respondents considered that gamification helped them to improve their levels of motivation and involvement in the teaching/learning process, as well as allowing them to develop skills in programming more easily.

It should be noted that only 11 students had obtained a positive evaluation in the first moment of evaluation (representing about 37% of those who attend the evaluation moment, and only 20% of the enrolled students), which demonstrates the enormous difficulties felt.

It should also be noted that most students found it useful to replicate this teaching and learning strategy, which incorporates some gamification elements, in other modules of the discipline.

Table 1 Process of implementing a gamified strategy.

Process (step) of implementing a gamified strategy	Description
Target Audience	Students enrolled in the discipline of Algorithms and Data Structures. The joining to this gamified strategy was optional, consisting of a complementary way to the teaching and learning process previously defined.
Learning objectives	i) Consolidate knowledge about simple data structures, lists and arrays, as well as data persistence (the first three modules of the course, described above); ii) Develop skills related to problem-solving capability; iii) Develop student autonomy and self-expression.
Structure of gamified learning	The gamified strategy involved conceiving a mission, proposed to students, which is composed by 3 challenges. These challenges were organized into levels (unlockable, once students successfully complete the previous level), with successively higher degrees of difficulty. Once students have completed a level, they earned points, which are later converted into supplementary points in the final assessment of the course (cumulative points with all other assessment elements initially planned). They also earned a badge for completing each level. In addition, students are also encouraged to collaborate and cooperate with each other, through participation in internal discussion forums. At the end of the mission, the students would evaluate their peers, choosing the 3 students who stood out the most in helping and tutoring their colleagues.
Identify and organize the study	As mentioned, the mission was organized around 3 challenges: i) the first consisted of implementing a small rooster game (tic tac toe), a traditional game. It should consist on 2 players, who playing alternately; ii) After completing this first level, the second challenge consisted of adapting the same game, where one of the players is the computer. In this case, the algorithm should simulate the computer's movements, based on principle of playing against the computer; iii) once completed and submitted this challenge, the third level was unlocked. This level consisted on adapting the same game, but now the validation criteria for the assessment of the submitted solution was now focused on the efficiency with which the game simulated the computer movements. At this third level, efficiency and code quality were decisive key factors in awarding solutions. It is important to note that this third level awarded only the 3 best solutions submitted by students.
Apply gamification elements	This initiative is available on moodle between December 20, 2021 and January 3, 2022. The proposed mission should be completed in this period, for all those who joined this initiative (remember that it was not mandatory). One of the goals was to keep students motivated and involved in the discipline, even during the Christmas break.

5 Conclusion and Future Work

The experience presented in this paper describes the introduction of some gamification elements in the programming teaching/learning process, namely in an introductory course, based on a conceptual framework presented by Piteira. The results obtained are clearly in line with those presented in the study carried out by Piteira [9], confirming that this approach helped to achieve the desired results, with an increase in student motivation and involvement. Greater satisfaction, motivation and interest are clearly conclusions that we can draw from the questionnaires carried out.

Table 2 summary of responses obtained to the questionnaire.

Questionnaire questions	1	2	3	4	5
How satisfied are you with the gamified strategy adopted?			1	4	16
Does gamification help you to improve your levels of motivation and engagement with the learning process?				3	18
Did the challenges proposed helped you to develop your skills in programming?			1	5	15
Do you think gamification should be applied to other course modules?				4	17

However, further studies would be needed to allow us to compare the results obtained with those that would have been obtained if we continued the more expository teaching/learning process, followed by exercises, which was adopted in the first weeks of the course (and until the first moment of evaluation).

On the other hand, more substantial conclusions also imply a comparison with the results obtained in previous years, in the same course, as well as an analysis of results over a longer period of time. That is, it will be necessary to replicate this model in the following academic years, in order to obtain a data set that allows to perform a trend analysis of results and thus more informed conclusions.

References

1 Kirsti Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15:83–102, June 2005. doi:10.1080/08993400500150747.

2 Yorah Bosse and Marco Aurelio Gerosa. Why is programming so difficult to learn?: Patterns of difficulties related to programming learning mid-stage. *ACM SIGSOFT Software Engineering Notes*, 41:1–6, January 2017. doi:10.1145/3011286.3011301.

3 Patrick Buckley, Seamus Noonan, Conor Geary, Thomas Mackessy, and Eoghan Nagle. An empirical study of gamification frameworks. *Journal of Organizational and End User Computing*, 31:22–38, January 2019. doi:10.4018/JOEUC.2019010102.

4 Darina Dicheva, Christo Dichev, Gennady Agre, and Galia Angelova. Gamification in education: A systematic mapping study. *Educational Technology & Society*, 18:75–88, July 2015.

5 Anabela Gomes, Cristiana Areias, Joana Henriques, and Antonio Mendes. Aprendizagem de programação de computadores: dificuldades e ferramentas de suporte. *Revista Portuguesa de Pedagogia*, 42:161–179, July 2008. doi:10.14195/1647-8614_42-2_9.

6 José Paiva, José Leal, and Ricardo Queiros. *Authoring Game-Based Programming Challenges to Improve Students' Motivation*, pages 602–613. Advances in Intelligent Systems and Computing, Springer, January 2020. doi:10.1007/978-3-030-11932-4_57.

7 M. Pinto. Learning computer programming: The role of gamification. In *EDULEARN18 Proceedings*, 10th International Conference on Education and New Learning Technologies, pages 9492–9497. IATED, 2-4 july, 2018 2018. doi:10.21125/edulearn.2018.2263.

8 Martinha Piteira and Carlos Costa. Computer programming and novice programmers. In *Proceedings of the Workshop on Information Systems and Design of Communication*, pages 51–53, June 2012. doi:10.1145/2311917.2311927.

9 Martinha Piteira, Carlos Costa, and Manuela Aparicio. A conceptual framework to implement gamification on online courses of computer programming learning: Implementation. In *Proceedings CISTI 2017*, pages 7022–7031, November 2017. doi:10.21125/iceri.2017.1865.

10 Ricardo Queirós, Mário Pinto, and Teresa Terroso. Computer Programming Education in Portuguese Universities. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASIcs)*, pages 21:1–21:11, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2020.21.

11 Ricardo Queiros, Mário Pinto, Alberto Simões, and Filipe Portela. *A Primer on Gamification Standardization*, pages 1–13. IGI Global, January 2022. doi:10.4018/978-1-7998-8089-9.ch001.

12 Jože Rugelj and Maria Lapina. Game design based learning of programming. In *proceeding of International Scientific Conference Innovative Approaches to the Application of Digital Technologies in Education and Research (SLET-2019)*, May 2019.

13 Alberto Simões. *Using Game Frameworks to Teach Computer Programming*, chapter 10, pages 221–236. IGI Global, January 2017. doi:10.4018/978-1-5225-1034-5.ch010.

14 Jakub Swacha and Pawel Baszuro. Gamification-based e-learning platform for computer programming education. In *X World Conference on Computers in Education*, June 2013.

15 Christopher Watson and Fred Li. Failure rates in introductory programming revisited. *ITICSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference*, June 2014. doi:10.1145/2591708.2591749.

16 Kevin Werbach and Dan Hunter. *For the Win: How Game Thinking can Revolutionize your Business.* Wharton Digital Press, January 2012.

A Roadmap to Convert Educational Web Applications into LTI Tools

José Paulo Leal
CRACS & INESC Tec LA / Faculty of Sciences, University of Porto, Portugal

Ricardo Queirós
CRACS – INESC-Porto LA & uniMAD, ESMAD/P. Porto, Portugal

Pedro Ferreirinha
Faculty of Sciences, University of Porto, Portugal

Jakub Swacha
University of Szczecin, Poland

Abstract

This paper proposes a roadmap to integrate existing educational web applications into the ecosystem based on a learning management system. To achieve this integration, applications must support the Learning Tools Interoperability specification in the role of tool provider. The paper starts with an overview of the evolution of this specification, emphasizing the main features of the current stable version. Then, it proposes a set of design goals and milestones to guide the adaptation process. The proposed roadmap was validated with existing applications. This paper reports on the challenges faced to apply it in these concrete cases.

2012 ACM Subject Classification Applied computing → Computer-managed instruction; Applied computing → Interactive learning environments; Applied computing → E-learning

Keywords and phrases programming, interoperability, automatic assessment, programming exercises

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.12

Funding This paper is based on the work done within the FGPE Plus: Learning tools interoperability for gamified programming education project supported by the European Union's Erasmus Plus programme (agreement no. 2020-1-PL01-KA226-HE-095786).

1 Introduction

Learning management systems (LMS) play a pivotal role in the way instruction is delivered in many schools. These systems manage most of the learning activities in which students must participate. However, some of these activities may be better handled by other educational web applications.

For instance, a typical LMS provides automated assessment features but not on specialized domains such as programming languages. On the other hand, web applications such as Sololearn, CoderByte, CodeWorkout, CodeWars, CodinGame, HackerRank, and LearnJS provide this kind of assessment and would benefit from a tighter integration into the LMS ecosystem.

The Learning Tools Interoperability (LTI) specification [3] provides a standard approach for this kind of integration. It evolved from a simple mechanism to launch web applications from the LMS, to support multiple services to securely interchange data with these applications. LTI is a standard supported by all LMS vendors of reference, and thus maximizes the payoff of adapting an educational web application.

A practical example can be given using CodeWorkout as an external third-party tool provider and Canvas as an LTI-compliant LMS acting as a Tool Consumer. Using LTI, it is easy to seamlessly integrate CodeWorkout as a Canvas Assignment embedded in a Canvas page. This enables students to code the exercises from within Canvas in the same way that they might do with any Canvas-native assignment, and receive a score in the Canvas gradebook for doing that assignment.

Despite these advantages, adapting an existing educational web application to support LTI is far from a painless process. Designing a system from scratch to use this specification is complex enough. Redesigning an existing system in such as way as to retain all its previous features and simultaneously take the most from LTI integration is even more challenging. Moreover, there are several hurdles to consider, from the configuration of the LMS itself to make it support a given LTI version, to selecting libraries to support the specification on the web application side.

The remainder of the paper is organized as follows. Section 2 provides a background on the LTI specification and surveys the best frameworks and libraries to implement it. Section 3 identifies a set of design goals and milestones to guide the adaptation process. Section 4 reports on the use of a selected LTI library to integrate a code playground into an LMS. Finally, Section 5 summarizes the contributions of this work and points to future directions.

2 State of the Art

Learning platforms have evolved in several dimensions. The interoperability dimension was one of the dimensions that progressed the most [2]. The first platforms were monolithic and closed, with all the features provided by internal and proprietary components. In the last decades, platforms become decentralized as they are built by assembling external components exposed as services. These services are decoupled from their consumers, and thus can easily be reused by several learning processes.

These services led to the formalization of initiatives [8] to adapt Service-Oriented Architectures (SOA) to e-learning called e-learning frameworks. The ultimate goal of these initiatives was to simplify the integration of educational systems, allowing the creation of dynamic educational ecosystems easily adaptable to any learning domain. They were composed of open interfaces to numerous reusable services organized into genres or layers and combined in service usage models. Despite the good intentions, the complexity in formalizing the specifications of the frameworks undermined the adhesion of the major educational players [6].

In parallel, other interoperability initiatives focused on simplifying the educational systems' integration emerged, such as NSDL, POOL, OKI, EduSource, IMS DRI, IMS LTI. Based on these initiatives, several authors presented works on connecting LMS to external applications [9, 2, 1].

2.1 IMS Learning Tools Interoperability (LTI) specification

An initiative that stood out was the IMS Learning Tools Interoperability (LTI) specification [3] defined as a standardized way to integrate third-party content into courses within an LMS or platform. Without LTI, LMS and content providers would have to rely on custom integrations with prejudice to interoperability and scalability.

Nowadays, the IMS consortium recommends the adoption of version 1.3. In short, LTI 1.3 uses OAuth2 and signed messages using JSON Web Tokens (JWTs) to securely authenticate students and pass data between the LMS and the external learning tool.

This version includes LTI Advantage, defined as a package of services that add new features to enhance the integration of any tool with any LMS in the core of LTI 1.3. The three LTI Advantage feature services are Names and Role Provisioning Services, Deep Linking, and Assignment and Grade Services:

- Names and Role Provisioning Services (NRPS): grants the tool access to the list of users and their roles for a specific context (e.g. a course or group). NRPS allows the external tool to request a list of members from the context that launched it. Once the tool receives the membership list, it can read all students and teachers enrolled in the context, even if they have not yet launched the tool.
- Assignment and Grade Services (AGS): allows teachers to sync grades between third-party tools and their LMS. This service returns numeric scores and teacher comments to an LMS gradebook. For instance, a teacher can view when a student has started an assignment, or if it has been completed. The teacher can also receive the status of a grade, even if it requires manual input from the teacher.
- Deep Linking (DL): offers a more streamlined approach to adding LTI links to an LMS. Deep Linking messages allow external learning tools to appear within an LMS as internal tools would. With Deep Linking, external learning tools can now allow teachers to configure specific content or activities within the tool. In LTI 1.1, the whole tool content had to be exposed, even if the teacher only wanted the class to use a specific resource. Deep Linking allows teachers to select whichever content they need and share the link to launch that content with their course. For example, a tool may let a teacher configure a specific chapter from an e-book when their students select a link, rather than force them to launch the tool and then navigate to that chapter.

Adopting LTI 1.3 should be a careful decision. For most of the cases, LTI 1.1 is enough in use cases where a student launches an activity from the LMS, solves it in the external tool and then a grade is reported back to the LMS grade book. However, the new LTI 1.3 features could be applied in relevant use-cases. For instance, considering the use of leaderboards: names and roles obtained from provisioning services could be used to populate leaderboards with the names of students before they submitted for the first time; deep links could be used to embed in LMS content leaderboards generated on-the-fly by the tool.

2.2 LTI implementation tools

Given the popularity of LTI and its consequent adoption by the community, it is worth analyzing the existing options to develop an external application with LTI support and, thus, benefit from secure integration with an LMS while using the services provided by this standard.

Currently, there are several frameworks and libraries (from now on, LTI development tools) to implement a tool provider with preconfigured routes and methods to manage the LTI 1.3 protocol. By using these frameworks, developers may profit from LTI without needing to implement all the security and validation requirements and, this way, concentrate on the tool logic.

The LTI development tools survey was based in the following methodology: firstly, a set of tools were collected using a “lti” keyword to search on GitHub Lab [1] – a site filter for GitHub repositories. Then, tools that weren’t update in the last year were discarded. Finally, the five with more stars were selected. The set of tools is the following: ltijs[2], TSUGI[3], LtiLibrary[4], ims-lti[5] and lti-1-3-php-library[6].

[1] `https://githublab.com/repositories?q=lti`
[2] `https://github.com/Cvmcosta/ltijs`
[3] `https://github.com/tsugiproject/tsugi`
[4] `https://github.com/LtiLibrary/LtiLibrary`
[5] `https://github.com/instructure/ims-lti`
[6] `https://github.com/IMSGlobal/lti-1-3-php-library`

The following subsections describe and compare these five LTI development tools. The analysis, summarized in Table 1, is based on three facets: maturity, specification coverage, and language binding.

Table 1 Analysis of the major LTI development tools.

Criteria	Properties	ltijs	TSUGI	LtiLibrary	ims-lti	lti-1-3
Maturity	First release date	May 19	Jun 17	Nov 16	May 20	Aug 18
	Last release date	May 21	Mar 22	Mar 22	May 20	Aug 20
	# Open issues	21	17	7	5	21
	# Pull requests	1	1	-	2	5
	# Commits	438	2879	604	246	110
	# Releases	5	18	14	1	-
	# Contributors	7	24	13	20	8
	# Stars	173	287	72	167	84
	# Forks	29	243	55	105	60
Coverage	LTI 1.0					
	LTI 1.1		X	X	X	
	LTI 1.3					X
	LTI 1.3 (DR included)	X				
Language Bindings	.NET			X		
	JavaScript	X				
	PHP		X			X
	Ruby				X	

Regarding maturity (based on the tools' GitHub repositories), although relatively recent, these tools have been growing with the evolution of the LTI specification. TSUGI and LtiLibrary are the oldest and more frequently updated, taking into consideration commits and releases. However, ims-lti, despite being the most recent, is the second most popular, with a reasonable number of stars. The number of forks of a repository is also relevant. Forking a repository allows to freely experiment with changing it without affecting the original project. Thus, this means more people are using the TSUGI code base to start their projects than any other. The last two tools are those with a lower number of commits and the oldest last release dates. Nevertheless, the lti-1-3-php-library, created by IMSGlobal, has the biggest number of open issues, which is revealing of the great community that the IMS presents.

LTI 1.1 is still the most popular and widespread option. In short, this version defines a standard way to launch a tool provider (typically an external learning tool) within a tool consumer (typically an LMS) and attach some user data, authentication data, and service references. Afterward, the learner uses the external learning tool. Finally, the tool provider calls a Learning Information Service (LIS) to submit the learner grade back to the LMS. Nevertheless, the IMSGlobal library has already implemented version 1.3, supporting all services from LTI Advantage. The ltijs library has the highest coverage level, being the first library to implement the new LTI Advantage Dynamic Registration Service, which turns the LTI Tool registration flow into a fast, completely automatic process. It exposes a registration endpoint through which platforms can initiate the registration flow.

Finally, most LTI development tools are coded in one of two languages, either PHP or JavaScript (NodeJS runtime). It was expectable as these are the most popular server-side languages. The initial unfiltered set included tools coded in other languages, such as Java and Python, but most of them were in beta state.

3 LTI roadmap – design goals and milestones

This section presents a roadmap to integrate an existing educational web application into a Learning Management Systems (LMS) using the Learning Tools Interoperability (LTI).

We envisioned the object of this integration process as an existing educational web application. This application must be able to provide activities to a group of students similar to a class. It must have structured content to be presented to students as activities. This content can be either expository (PDFs) or evaluative (automated assessment). Finally, the outcome of these activities should be reported back to the LMS for integration in a grade book.

To guide us in the creation of this LTI integration roadmap we considered the following design goals:

- The web application will become an LTI tool provider (TP) that can be launched and interact with an LTI tool consumer (TC) such as an LMS.
- The web application will continue to be usable outside the LTI context, supporting simultaneously LTI and non-LTI users.
- Modifications to the web application user interface will be minimal, desirably none.
- Global configurations of TC (ex: skinning, natural language) should be exposed to the LMS.
- When possible, changes made by LTI users in the web tool will be automatically reported to the TC.
- The adaptation process will start with simple and independent tasks and then proceed to more complex ones that depend on the former.

The adaptation process should profit as much as possible from all LTI features, ranging from the core features from the early versions of the protocol to those proposed by LTI Advantage and integrated into the latest version. The core features include launching the TP in an authenticated session and configuring it from the LMS. The latest features include the Names and Role Provisioning Services (NRPS), Deep Linking (DL), and Assignment and Grade Services (AGS).

The proposed roadmap is organized into three consecutive main milestones. Each milestone is a fully functional and improved version of the web application, providing a deeper integration into the LMS and building on the previous ones. The first aims at the core integration between the TP and LMS, with user ids shared by both systems. The second aims at exposing specific TP content to different LMS activities. Finally, the third milestone aims at reporting student activity on the TP back to LMS. The following subsections detail each of these milestones.

3.1 Entrance

The main objective of the first milestone is to enable a smooth transition between the LMS (e.g. Moodle) and the TP (e.g. Agni). Providing a single-sign-on is a major driving force behind LTI adoption; this feature is available since version 1.0 of this specification. However, entering the TP when coming from the LMS involves more than just authentication and authorization. This milestone is divided into the following sub-milestones.

1. Launch the TP.
2. Expose TP configuration to the LMS.
3. Create an authenticated session.
4. Create users for the activity.

The first sub-milestone is to launch the tool using LTI. This sub-milestone requires configuring the LMS to support LTI with the tool (more on that in the following section) and configuring an activity on the LMS. This sub-milestones is concluded when the activity link is presented on the LMS and, when followed, the tool screen is presented to the user. At this stage, the TP might simply present a welcome or login screen.

The second sub-milestone is to enable the configuration of the TP using LTI custom parameters. The TP may have general configurations to expose to the LMS. For instance, the TP might support different natural languages or some form of user interface customization, such as setting a background color or a logotype. If these configurations are available they may be exposed to LMS using LTI custom parameters. This way, the teacher may configure them when defining the LTI activity on the LMS.

The third sub-milestone is the creation of an authenticated session according to the profile received via LTI. The relevant profiles are teacher and student and these should be mapped to profiles available on the tool. At this stage, the student profile can be mapped to a guest user or generic student. The teacher may be mapped to an administration profile.

Educational systems usually require students to be registered in advance. This registration is necessary since most of these systems record student progress. Many of these systems support groups of students like classes. Ideally, a class or similar should be created and associated with each LTI activity. This class can be automatically created on the first launch of an LTI activity.

Using NRPS the TP can obtain from the LMS the list of users and their profiles. With this approach, all users can be created in batch at the first launch, typically when the teacher tests the activity. In subsequent LTI launches the authorization is reduced to checking if the student was already created.

Students may enroll later in the course after the activity is configured and the class populated on the TP. Hence, if the student is missing during authorization, the process described in the previous paragraph will have to be repeated.

3.2 Content

The main objective of the second milestone is to launch the TP with a specific content. For instance, in a TP with many exercises, the teacher may need to specify a few for a particular LTI activity. This kind of launch opens the TP in any part of it, which gives the possibility of skipping a login screen or showing a restrictive single small part. This can be achieved by configuring the TP activity on the LMS using two different approaches:

1. LTI custom parameter.
2. Deep Linking (DL).

An LTI custom parameter can be used on the activity configuration to specify the content to present to the students. The drawback of this approach is that it requires the teacher to know the parameter name and the content identifier, usually a pathname. If the TP hosts more than a few individual contents, this approach may be too cumbersome.

The alternative is to use Deep Linking to provide a content selector to the teacher when she configures the LTI activity on the LMS. Although integrated on the LMS, the content selector is created by the TP, reflecting its current content. Thus, if the content has many items, it can be organized using a tree widget, for instance. Alternatively or complementary, an incremental search field can be added to facilitate the selection process.

In general, DL is preferable to custom parameters to select content items. However, custom parameters may have other uses in content selection. For instance, a custom parameter may be used to disable navigation on the TP. Using such a configuration, after an LTI launch on specific content, students will be unable to navigate different to other content on that instance.

3.3 Feedback

The main objective of the third and final milestone is to send feedback from the TP back to the LMS. It is an important facet of LTI integration since it allows the instructor to consolidate grades in the LMS, but can be exploited also for other purposes. This milestone is divided into the following sub-milestones:

1. Reporting activity grades.
2. Reporting other information.

One of the tools in an LMS is a *gradebook* – a sort of table where each row refers to a student and each column refers to an activity. The teacher may use this gradebook to manage student grades in a course and grades from LTI activities may be automatically filled in by the TP.

When an LTI activity is created in the LMS, a column for that activity is automatically added to the gradebook. This column has two components: a grade and a description. The grade is a non negative number within a certain range (e.g. 0 to 100) – and the description is a string of characters. Both these values may be null. Using AGS, other similar columns may be added to the gradebook for the same activity.

The objective of the first sub-milestone is to use the automatic column and fill it with the grade of each student that completes the activity. Using AGS, the TP should automatically report grades as soon as the students complete the activity, without requiring any action from the teacher.

The second sub-milestone requires the creation of other columns in the gradebook. These columns may be used to report data besides grades. For instance, the TP may manage HTTP sessions associated with each student and use them to compute the time they spent on the activity. This data may be reported in a new column using the numeric value for the accumulated time and the descriptive text to inform if the student is currently solving the activity.

Although the AGS was designed to reporting student related information and in particular grades, it be also exploited as a general mechanism to return data to the tool consumer. This use may be relevant if the tool consumer is not an LMS but another TP, using the LTI link to integrate this third system with a common single-sign-on domain. In this case the tool consumer will need to implement an AGS to receive the data reported by the TP.

4 LTI Adaptation Example

To validate the proposed roadmap the authors applied it in two different contexts. One is a JavaScript web playground named Agni (previously called LearnJS [5]). The other is a programming exercise authoring tool named FGPE AuthorKit [4]. Subsection 4.1 describes the required LMS and external tools configurations, similar in both cases. Then, Subsection 4.2 describes implementation details specific to Agni, and Subsection 4.3 those related to FGPE AuthorKit.

4.1 Configuration

This subsection details the configurations needed to support the integration of a tool consumer (LMS) with a tool provider.

4.1.1 Tool Consumer (LMS)

The first milestone of the roadmap requires launching the TP from the LMS. To fulfill this objective is necessary to configure an LTI activity on the LMS.

Moodle was the LMS selected for testing the integration of both applications. These applications have in common the implementation language, and both rely on the same integration library – ltijs – introduced in Subsection 2.2. Hence, the initial configuration of both applications as TP in Moodle is very similar and requires the following steps:

1. Create a tool configuration.
2. Set the tool's public key in the configuration.
3. Define the redirection URL(s).

The initial step is to set the tool type `LTI 1.3` and obtain a client ID to register the platform in ltijs. After the platform registration, the tool generates a public key for the handshaking process, to be entered in the `Public Key` of the external tool configuration. Finally, all the URL(s) that are processed inside the context must be entered in the `Redirection URL(s)`. URLs must be registered for security reasons, so that the LMS recognizes all the possible redirections inside its context.

4.1.2 Tool Provider (external tool)

To implement LTI in the tool provider, the ltijs library was the obvious choice due to its full support for the v1.3 specification, including the dynamic registration service. Also both external tools have Node.js – a JavaScript runtime built on Chrome's V8 JavaScript engine – as back-end, which perfectly suits the use of ltijs.

The first step is to install the library npm package and, since this package natively uses mongoDB by default to store and manage the server data, it is also necessary to install it.

After basic installation, it is necessary to set up ltijs into the external tool in multiple steps. Most of them can be mapped to specific methods of the ltijs library, namely:

- **setup** – to configure the database and additional parameters (routes for the reserved endpoints used by ltijs). After the `lti.setup()` method is called, the `lti` object gives you access to various functionalities to help you create your LTI Provider. The `lti.app` object is an instance of the underlying Express server, through this object you can create routes just like you would when using regular Express.
- **deploy** – opens a connection to the configured database and starts the Express server.

- **register** – registers the platform on the tool. Platform manipulation methods require a connection to the database, so they can only be used after the deploy method. A LTI tool works in conjunction with an LTI ready platform (Moodle), so in order for Moodle to display the external tool first page, it needs to first be registered in the tool itself. This way, the tool is able to generate a public key used by Moodle for further connections.

Ltijs behavior is configured through callbacks. For instance, the `onConnect` callback is called whenever a successful launch request arrives at the main app URL, or the `onDeepLinking` callback is called when a successful deep linking request is received.

Every launch generates an `IdToken` that the tool uses to retrieve information about the user and the general context of the launch. The valid `idToken` is then separated into two parts:

- **idtoken** – contains the platform and user information that is context independent (e.g., platform name, user name and email);
- **contexttoken** – contain the context specific information (e.g. activity name, platform endpoints)

Both are stored in the database and passed along to the next route handler inside of a `response.locals` object.

Ltijs needs the correct `idtoken` and `contexttoken` when a tool makes a request. The authentication protocol relies on two items, a session cookie, and a `ltik` token. At the end of a successful launch, ltijs: 1) redirects the request to the desired endpoint; 2) sets a signed session cookie containing the `platformCode` and `userId` information; 3) sends a `ltik` JWT token containing the same platform and user information, with additional context information as a query parameter to the endpoint. When the tool receives a request, it attempts to validate it by matching the information received through the session cookie with the information contained in the `ltik` token. It is important to notice that the `ltik` token must be passed to the tool through either query parameters, body parameters, or as an Authorization header (bearer or LTIK-AUTH-V1).

4.2 Agni Web Playground

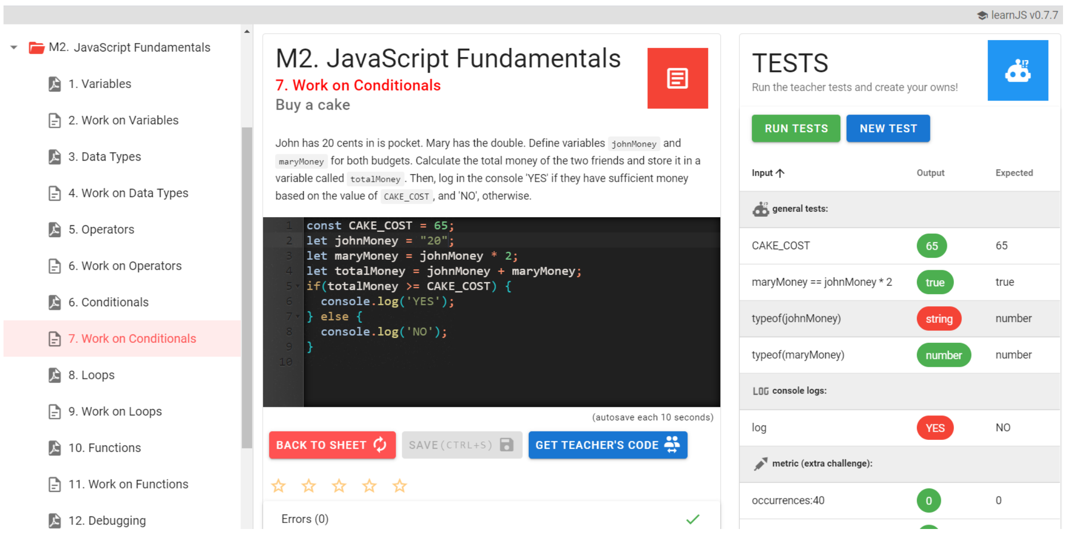
Agni (formerly LearnJS) is a Web playground that enables anyone to practice the JavaScript language [7]. In the playground, students access PDFs and videos on different topics, and solve exercises related to those topics with automatic feedback on their resolutions. The playground has two main components: 1) an editor where students code their solutions in an interactive environment and 2) an evaluator to assess students' code based on static and dynamic analyzers. Figure 1 shows the front-end GUI of the playground.

The following subsections discusses three Agni features aligned with the milestones presented at Section 3. The features are exercise sheet launching, leaderboard generation, and score submission.

4.2.1 Entrance – Leaderboard generation

In Agni, every successful launch from the platform to the tool should present two sections: the exercise sheet, so that the student can solve the exercises and a leaderboard with the current state of participant progress.

After a first successfully launch, NRPS is used to automatically create all the platform's users (referred to as members) and their roles within the context of the course where the LTI activity belongs. All members of a platform within the context can be retrieved simply by calling the **getMembers** method as shown in Listing 1.



M2. JavaScript Fundamentals

7. Work on Conditionals

Buy a cake

John has 20 cents in is pocket. Mary has the double. Define variables `johnMoney` and `maryMoney` for both budgets. Calculate the total money of the two friends and store it in a variable called `totalMoney`. Then, log in the console 'YES' if they have sufficient money based on the value of `CAKE_COST`, and 'NO', otherwise.

```
const CAKE_COST = 65;
let johnMoney = "20";
let maryMoney = johnMoney * 2;
let totalMoney = johnMoney + maryMoney;
if(totalMoney >= CAKE_COST) {
  console.log('YES');
} else {
  console.log('NO');
}
```

(autosave each 10 seconds)

Errors (0)

TESTS

Run the teacher tests and create your owns!

Input	Output	Expected
general tests:		
CAKE_COST	65	65
maryMoney == johnMoney * 2	true	true
typeof(johnMoney)	string	number
typeof(maryMoney)	number	number
console logs:		
log	YES	NO
metric (extra challenge):		
occurrences:40	0	0

Figure 1 Agni playground User Interface.

Listing 1 Members route.

```
lti.app.get('/members', async (req, res) => {
 // Gets context members
 const members = await lti.NamesAndRoles.getMembers(res.locals.token)
 res.send(members)
})
```

With this approach Agni, could present a leaderboard with the names of students (and respective scores) even before they submitted for the first time.

Despite its popularity, presenting leaderboards also has disadvantages, in particular, it can frustrate all those who recurrently remain in the lower places. Therefore, its use should be restricted, either through the creation of partial rankings or through the use of a boolean custom parameter called **show_leaderboard** which, when set to true, displays the leaderboard, otherwise it inhibits its visualization.

4.2.2 Content – Exercise sheet launching

Agni is a code playground targeted at JavaScript and organized into modules for different topics such as variables, data types, and arrays. Each module has two kinds of resources: expositive, such as videos and PDFs; and evaluative, such as exercise sheets. An exercise sheet contains a set of exercises of different types (e.g. blank sheet, buggy code, quiz).

To configure an exercise sheet in Moodle, the teacher should add an external LTI tool activity (Agni). On the form, press the "select content" button, which will call the `onDeepLinking` callback so that the tool can display a resource selection view. This view with the list of existing exercise sheets did not exist, so it was one of the first challenges to overcome. After selecting the desired sheet, the tool creates a deep linking request message and sends it to the platform, through the `createDeepLinkingMessage` method. Subsequently, when the activity is started in Moodle, the respective exercise sheet is displayed.

Another challenge was controlling the Agni UI, because when launching a specific exercise sheet, nothing prevented the student from selecting the other sheets. For this, a boolean custom parameter called **deactivate_ui** was added. If set to true, the student will only be able to access the associated sheet. Otherwise, they will have access to all the exercise sheets.

4.2.3 Feedback – Scores submission

During the problem solving process, the student's performance score is automatically sent to the Moodle gradebook. In this way, teachers are able to monitor the progress of their students in real time.

Using the Grading Service it is possible to submit grades from Agni to Moodle (with the **submitScore** method), get grades from Moodle (**getScores** method) and even manage Moodle gradebook columns (**getLineItems**, **createLineItem** and **deleteLineItems** methods).

The hardest challenge was to decide at which moment the score should be sent to the platform, since the Agni UI does not have a button to send grades. To overcome this difficulty, it was decided that whenever a student, after submitting his solution for evaluation, leaves the exercise back to the list of exercises, his score is automatically submitted into Moodle gradebook.

4.3 Authorkit

AuthorKit is a programming exercise authoring tool [4]. It is a web application where authors can create programming challenges that can later be retrieved as learning objects by other components of an e-learning environment, such as automated assessment systems. In a strict sense, AuthroKit is not the kind of web application envisioned by the proposed roadmap, since it is not targeted at students. However, by using LTI it can benefit from single-sign-on and use AGS to report data back to the calling system.

As the tool itself does not provide a numeric feedback nor is this feedback relevant in the context of having students and calculating their grade. This was done out of interest for how such a tool could be used in the LTI context, this implementation uses the fact that every grade has a description, and as such, provides the key generated by the tool as a description for a null grade. This key can then be seen through the LMS interface.

AuthorKit is initially set up by a login screen, the objective is to send through an LTI parameter informing that this launch is indeed an LTI one, skipping the login screen and entering the corresponding account from the TC. Afterwards, it is possible to create a programming exercise through the TP content in the TC, resulting in the exercise key as feedback.

5 Conclusion

This paper proposes a roadmap to integrate existing educational web applications into the ecosystem created by a learning management system. To achieve this integration applications must support the Learning Tools Interoperability specification in the role of tool provider. The proposed roadmap was validated with existing applications. This paper reports on the challenges faced to apply it in these concrete cases. As future work it is intended to implement all the services of LTI Advantage in a gamified programming learning environment that is an integral part of an ecosystem to facilitate the teaching of programming and that has been worked on in the context of a European project called FGPE Plus: Learning tools interoperability for gamified programming education (FGPE+)[7].

[7] `https://fgpeplus.usz.edu.pl/`

References

1 Marc Alier, María José Casany, Ariadna Llorens, Jesús Alcober, and Joana d'Arc Prat. Atenea exams, an ims lti application to solve scalability problems: A study case. *Applied Sciences*, 11(1):80, 2020.

2 Claudia-Melania Chituc and Marc Rittberger. Understanding the importance of interoperability standards in the classroom of the future. In *IECON 2019 – 45th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 6801–6806, 2019. doi:10.1109/IECON.2019.8927631.

3 IMS Global Learning Consortium. Learning tools interoperability core specification, 2019. accessed on 14 Apr 2022. URL: http://www.imsglobal.org/spec/lti/v1p3/.

4 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.

5 Ricardo Queirós. Learnjs – a javascript learning playground (short paper). In *7th Symposium on Languages, Applications and Technologies (SLATE 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

6 Ricardo Queirós and José Paulo Leal. Ensemble – an e-learning framework. *J. Univers. Comput. Sci.*, 19(14):2127–2149, 2013. doi:10.3217/jucs-019-14-2127.

7 Ricardo Queirós and José Paulo Leal. Fostering students-driven learning of computer programming with an ensemble of e-learning tools. In Álvaro Rocha, Hojjat Adeli, Luís Paulo Reis, and Sandra Costanzo, editors, *Trends and Advances in Information Systems and Technologies – Volume 2 [WorldCIST'18, Naples, Italy, March 27-29, 2018]*, volume 746 of *Advances in Intelligent Systems and Computing*, pages 289–298. Springer, 2018. doi:10.1007/978-3-319-77712-2_28.

8 Ricardo Queirós and José Paulo Leal. Elearning frameworks: A survey. In *INTED2010 Proceedings*, 4th International Technology, Education and Development Conference, pages 1345–1354. IATED, 8-10 march, 2010 2010.

9 G. Tuparov and D. Tuparova. Approaches for integration of educational computer games in e-learning environments. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 0772–0776, 2018. doi:10.23919/MIPRO.2018.8400143.

Programming for Non-Programmers: An Approach Using Creative Coding in Higher Education

Teresa Terroso
uniMAD – ESMAD, Polytechnic of Porto, Portugal

Mário Pinto
uniMAD – ESMAD, Polytechnic of Porto, Portugal

Abstract

Learning how to program can be a cumbersome task even for students who enroll in courses in the Computer Science field. It is well documented that computer programming courses have high failure rates and high drop out. Even at the initial stage of computer introduction courses, novice students often reveal difficulties and strong reactions to this subject. However, computer programming has been recognized as an essential skill and a necessary element in education in many different areas. This work reflects on the experience provided by teaching a Creative Programming course, being held as part of a Master's degree curriculum in School of Media Arts and Design (ESMAD), at Polytechnic of Porto (P.PORTO), in Portugal. The students' background is not uniform, therefore pedagogical learning strategies had to be adapted to these multidisciplinary backgrounds to foster student attention and interest, as well as being able to achieve the goals of teaching the fundamentals of computer programming. This article reflects on the strategies to teach programming for non-informatics: drifting from the traditional functional way, like developing a program or product to solve a problem, to the usage of creative coding and generate interactive animations, while simultaneously achieving the ambitious goals of learning programming concepts and paradigms.

2012 ACM Subject Classification Computing methodologies; Applied computing → Computer-managed instruction

Keywords and phrases creative coding, programming learning, teaching methodologies, higher education, visual applications, interactive graphics, open-source tools

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.13

1 Introduction

It is well known that many students have difficulties in learning computer programming since it is a complex activity, requires a lot of practice, and traditional teaching approaches have not been able to respond effectively [12, 4, 1, 10].

Different reasons are enumerated and can be arranged into five major groups: teaching methodologies (non-personalized methods, static materials for dynamic programming concepts, more focus on teaching a programming language rather than promoting problem-solving), study methods (rather than practice intensively, students focus on reading books or watching tutorials and memorize formulas or procedures), student's abilities (generic lack of problem understanding, relating knowledge and infer a solution), the nature of programming (abstract concepts and complex syntactic details of many programming languages) and psychological effects (usually taught at the beginning of a higher education course and negative connotation associated with programming).

The high dropout and failure rates in introductory programming courses are a universal problem that motivated many researchers to propose methodologies and tools to help students: game design and development, programmable physical/tangible tools, project-based learning, gamification, automatic code evaluation systems, and so on [7, 10].

Although these approaches may motivate students to learn to program, most are oriented to computer science programs and thus special attention must be taken to students with different academic profiles and/or professional backgrounds. Creative coding as been already discussed in some works as an additional approach concerning computer science education, therefore this work contributes to the topic by providing further data supporting that it can be an additional path in teaching the basics of computer programming.

2 Creative coding

Creative and problem-solving competencies are part of the so-called 21st-century skills and are considered crucial to succeed nowadays [14]. Everyone, not just students who major in computer science, can benefit from thinking like a computer scientist.

As stated in [3], creativity is "the tendency to generate or recognize ideas, alternatives, or possibilities that may be useful in solving problems, communicating with others, and entertaining ourselves and others". By extension, creative coding can be understood as a type of programming where students can use the computer to produce visual animations to self-express themselves rather than focus on functionality. As learning computer programming often emphasizes technical detail over creative potential, creative coding supports increasing fluency with computational thinking, build upon creativity, imagination, and students' personal interests. According to [8], computer technology "is not a tool; it is a new material for expression". This intersection between arts and technology has received attention in both professional and educational fields [9, 6]. New art forms are being developed through the medium of code daily, and the creative coding community is growing rapidly.

Creative coding, as a programming practice geared towards artistic content production, has already been found effective in learning computer programming [5, 15, 2]. This work can be placed together within diverse efforts for introducing programming in an artistic manner to students, beyond the engineering perspective.

This work reflects on the findings from teaching a one semester-long Creative Programming course, taught on the first year of a Master degree in Interactive Media and Systems at ESMAD, P.PORTO. The main goal of this course is to teach the elementary concepts of computer programming applied to the different contexts of multimedia digital arts, by presenting a creative programming language as a tool for artistic expression. At the end students should be able to design and produce programs for the generation of audiovisual content for performances, multimedia installations, or web applications. Most of its students do not have an inherent affinity towards programming. As a course in a master degree, the classes are composed by students with very different educational or professional backgrounds: from artists to designers, from physiotherapists to multimedia students. Traditional ways of teaching programming (lectures combined with exercises, assessed by only one project and a final exam) had been tried, but students found that it was hard for them to build the link between theory and practice. Students are often eager to put the just learned knowledge into practice, if not immediately, at least as quickly as possible. Most of them have passion in visual designs so, they would like to see immediately their creations "move". The followed approach focused less on manipulating and interacting with numerical or text data but more on digital animations (from building graphics, to audio or video manipulation), that have been found to be at their interests as young adults and also making them more amenable to a heterogeneous environment audience.

To attend the course no programming experience is required. However, students should be comfortable using a computer, to perform simple tasks like installing applications and working with files. Within the efforts of creating and expressing themselves through digital media are the building blocks for introducing the essentials of programming. Over the semester, with a total of 30 contact hours, students develop a portfolio of digital animations using P5.js (an interpretation of Processing written in JavaScript) and employing basic computing structures typically taught in traditional Computer Science courses.

3 p5.js

Different creative tools (like Processing or p5.js) have been integrated into the curriculum of several education institutes, since they foster experimentation and enable the use of computation to express ideas with visually engaging sketches or interactive installations. An extensive list of curated tools and other resources for creative coding can be found in [13] and table 1 summarizes some of the most found in literature regarding creative coding.

Table 1 Tools for Creative Coding.

Frameworks for desktop development		
	Language	Mainly used for
Processing	Java	General purpose (IDE for visual arts)
OpenFrameworks	C++	General purpose (no IDE, toolkit for creative coding)
Cinder	C++	Professional-quality creative coding (uses OpenGL)
Libraries for web development		
	Language	Mainly used for
p5.js	JavaScript	General purpose (alternative for Processing)
three.js	JavaScript	3D graphics (simplifies working with WebGL)
Paper.js	JavaScript	Vector graphics scripting

In the course presented in this work, the visual programming tool is p5.js, a JavaScript library, as a web-based alternative to Processing, but with the same goal of making coding accessible for artists, designers, and overall beginners.

Processing was created in 2001 as a language for teaching art students how to program and to give an easier way to work with graphics to a more technical audience. P5.js emerged as a more accessible version, but still adhering to the syntax and conventions of Processing. Open-source and with a strong community of contributors, p5.js has capitalized on the web browsers' features and ubiquity. Compared to plain JavaScript, p5.js provides a more readable and clean code, making it easier to work with the Canvas HTML element and implement frame-by-frame animations, Table 2.

Table 2 Code snippets to illustrate some differences between plain JavaScript and p5.js.

Drawing a circle		**Frame animation**	
Plain JavaScript	p5.js	Plain JavaScript	p5.js

c.beginPath();
c.arc(x,y,r,0,2*Math.PI);
c.closePath();
	circle(x,y,r);	function draw() {
		/*some code*/
		requestAnimationFrame(draw);
		}
		draw();
			function draw(){
			/*some code*/
			}

For students who are learning how to program, p5.js provides encouragement and motivation with the immediate visual feedback. P5.js adds custom features related to graphics and interaction and provides easier access to native HTML5 features already supported by the browser. Several libraries extend p5.js even further, like DOM manipulation, webcam capture, image/sound processing and so on. Experimenting is made easier as there is an online editor[1] available on their website, allowing for a quick start to the library.

Choosing between Processing and p5.js was not difficult, even with the shortlist of bibliography or online examples. Having to choose between Java (Processing) and JavaScript (p5.js) as the base programming language, the choice fell towards the latter as its dynamic language features, and ability to merge functional and object-oriented programming techniques make it particularly well suited to exploratory creative practice. It has also been identified as an excellent language for introducing computer science students to programming [11].

P5.js follows the concept of sketching as its programming method, where it basically consists of quickly throwing 2D and 3D graphics for rapid prototyping of a dynamic visual work, similar to how drawn sketches are created by artists. This contrasts with the standard and disciplinary approach of software engineering for software development where first specifications are created, then implemented and evaluated. Despite its easiness to start to work with, it is a full-featured library capable of rendering stunning graphics and animations, using a simplified syntax and graphics programming model.

4 Course design

The course presented in this work consists of the following topics, for the total duration of one semester:

Course Introduction Introduction to Computing: structure, logic, syntax and algorithms. What is creative computing? Historical context, artistic references and examples.

Drawing primitives 2D and 3D coordinate systems. Simple shapes: points, lines, shapes, curves, text, images and 3D objects. Drawing and styling.

Control structures Variables and data types, operators, expressions, conditionals and loops.

User interaction Mouse and keyboard events, basic animation.

Functions Procedural abstraction, declaration and invocation of functions, parameters and return values.

Arrays and Objects Introduction to arrays, lists and indexing. Object-Oriented Programming (OOP): objects and classes.

Mathematics and Physics Basic trigonometry, direction, oscillation, acceleration, inertia and friction.

Creative coding examples Transformations, iteration and randomization.

Each topic was structured around 1-2 weeks of lectures and exercises, complemented with creative coding extra-class assignments. Although the textbook contents and topics ordering follows more or less the same as traditional introductory computer programming courses, the key concepts are most of the times driven by the assignments themselves or appear as tools to respond to the students' wanderings 'what if we would like to...?'.

During classes, the explanation of programming concepts is supported using visual graphics, Figure 1. In experience gained from teaching this course, it was found that using program visualization techniques helps grab the students' attention and support learning key programming concepts. Examples taught in classes serve as entry point to analyze the logic, decompose the algorithm within, or spotting for patterns or similarities. Exercises usually

[1] `https://editor.p5js.org/`

have an inspiration as starting point, were students are challenged to experiment and create something new, or debug to find and fix errors or even to collaborate by working together to find a solution. Some researches have demonstrated the impact of visual programming tools in upgrading pass rates and overall improvement when compared with other teaching interventions to facilitate student learning like taking an intensive preliminary training course or workshop, [7]. Like many, and from early on, the professors of this course were faced with problems aroused from the fact that students enter the Master degree with widely different experience levels with key course topics. If the material is covered too slowly, those with greater experience get bored and lose interest. If the material is covered too quickly, those with less experience get lost and feel incompetent. That led them to promote participation in short introductory programming courses, available for free in various e-learning platforms, targeting students with little or no background in programming. This intervention is still in practice today but can be improved since it is not to mandatory and the professors cannot control the course contents, acting only as curators in the selection of courses.

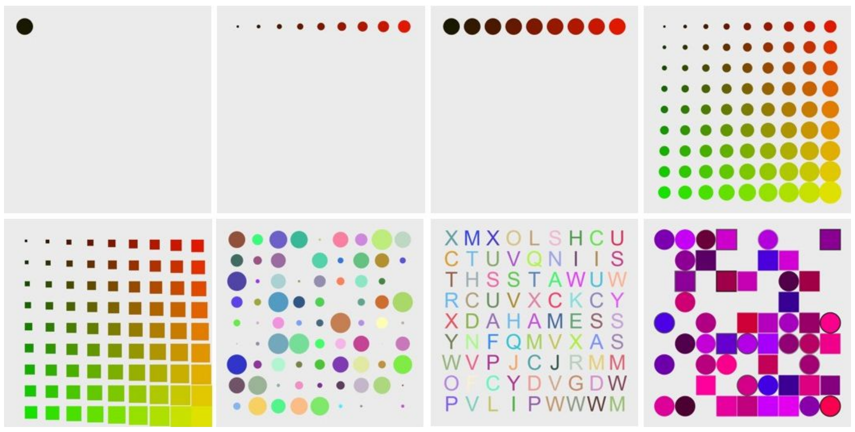


Figure 1 Examples presented during classes to visually demonstrate key programming concepts like loops, conditionals and operators. Top to bottom, left to right, from the simplest static sketch (no loop iterations and no conditionals) to showcasing different variations, by gradually introducing loops, nested-loops, conditionals, user inputs and animations.

Based on this work experience, the usage of media computation (mostly computer graphics, but also including image or sound processing), the foundation of creative programming, gave the students a context in which they already find the computers useful, which combined with open-ended exercises and assessments provided creative activities and restrained the stereotypes of computing as boring and anti-social, Figure 2.

Over the course of a semester, students are encouraged to develop further and share their sketches (in our school's learning management system or on public websites like openprocessing.org), so they can learn from each other's work.

4.1 Assessment

Students are asked to develop two projects during the course. The first project, developed individually, focus on simple and short 2D animations, using functional programming, where the theme is set by the professor, Figure 3. The second is a collaborative project with another course of the master's degree, Interfaces and Interaction Design, and it is developed in small groups (2 to 3 students). In this last project, students develop an interactive installation or application on a topic chosen by them, with a mandatory requirement of using OOP. In

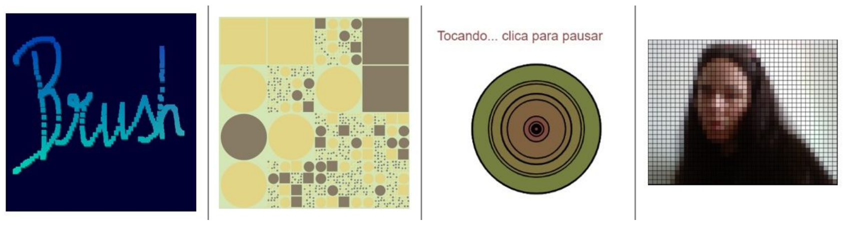


Figure 2 Examples of computer art techniques: from left to right, virtual brushes, recursive structures, sound visualization, pixel art.

this sense, students must develop an intuitive interface (with the input of user information through a camera or computer user interfaces, like mouse or keyboard) and/or tangible (through the Arduino micro-controller) for the public to be able to generate and manipulate audiovisual content. The format of the projects may vary from web applications, to 2D or 3D games or artistic installations, Figure 4.

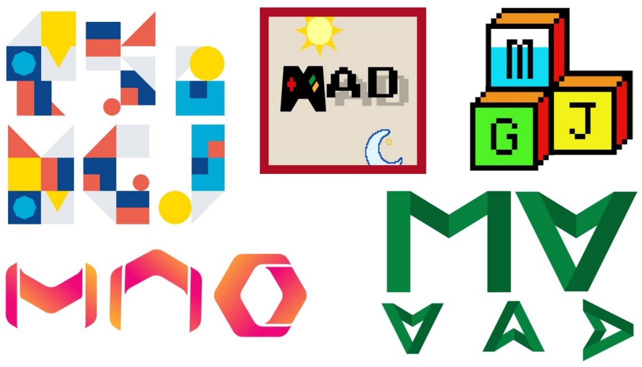


Figure 3 Project 1 example: creation of an animated logotype for one of our school's annual event, the Mad Game Jam.

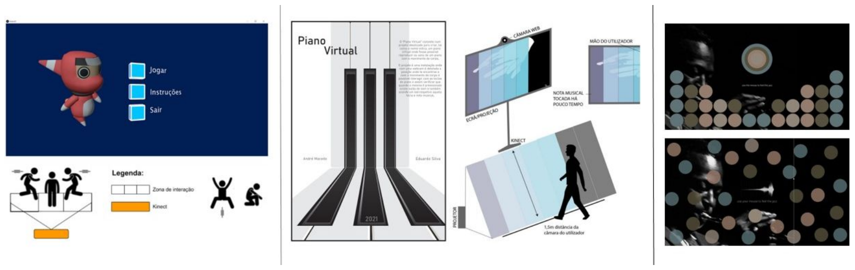


Figure 4 Work examples for Project 2: from left to right, 3D game (player controlled by body movement), virtual piano (audio and visuals controlled by hand movement), web music player (visuals controlled by sound samples).

Assessment of the students includes traditional instruments such as critiques and presentations. The critique process comes out of the classroom, where student work is assessed primarily through discussions including both technical aspects of the projects, such as a review of the source code, and also basic aesthetic issues. During final presentations, students are asked to perform auto- and hetero-evaluations.

5 Preliminary results

In the past three editions of this course, from a total of 39 enrolled students, only 6 dropped-out. From the remaining 33 students, only 4 (little more than 12%) had some previous knowledge about computer programming. All students were asked to answer a small survey about their findings, but only the students that completed the course answered, gathering a total of 16 responses (about 48%). No extensive statistical data analysis is presented in time for this work, since the surveys have suffered some alterations from year to year and the sample size is relatively small. However, some observations can be pointed out from the students' answers:

Course rating Students' were asked to grade the course on a 1 to 5 scale, with 1 as "it did not meet my expectations at all", 3 equals "it met my expectations" and 5 as "it exceeded all my expectations". The average rating was 3.75, having increased when the course assessment has shifted from individual tests (with theoretical questions and small practical exercises) and one group project, to two projects (one individual and the other developed in small groups).

Main strengths Some of the students' answers included "Learn to make generative art, and give yourself freedom with code", "Learning creative programming allowed me to start exploring areas that I had no knowledge of until now" and "I found myself entertained with some of the homework exercises because it was fun". 78% of students responded very favorably to the context of art when learning how to program and 43% of those claimed to spend extra time on assignments because they enjoyed it. Not so significantly, but a few students stated that the course instilled in them the desire to further explore the area of programming.

Main weaknesses A significant percentage of students (around 87%) pointed out the reduced in-class time to consolidate concepts and a few would have liked to explore some topics more in-depth. 24% of the respondents, even with the online course taken at the beginning, still struggled with the programming basics and "ended up confusing everything".

6 Conclusion and Future Work

The work presented in this paper reflects the usage of creative coding in Higher Education, as a way to overcome the difficulties of learning how to program, especially in classes that have different learning backgrounds, therefore fostering their interest and demystifying some of the negative connotations associated with programming. Creative coding can act as an approach to minimize the issues of code illiteracy and provide the tools for empowerment in this digital era. As a way to create art through code, creative coding can also be understood as a path for learning programming through creative projects. Therefore, it can be encouraged as a teaching tool, as a new way for exploration, computational reasoning development, and critical reflection about programming. The authors will continue to explore the opportunity to refine the course design and develop course materials (e.g. using the power of visual environments, that rather than just showing code structure and results, can offer dynamic data behavior) and accumulate student feedback, so that this work findings can be better substantiated and compared with other teaching interventions to facilitate student's learning in introductory programming courses.

References

1 Yorah Bosse and Marco Aurelio Gerosa. Why is programming so difficult to learn?: Patterns of difficulties related to programming learning mid-stage. *ACM SIGSOFT Software Engineering Notes*, 41:1–6, January 2017. doi:10.1145/3011286.3011301.

2 Vincenzo Fragapane and Bernhard Standl. Work in progress: Creative coding and computer science education – from approach to concept. In *2021 IEEE Global Engineering Education Conference (EDUCON)*, pages 1233–1236, 2021. doi:10.1109/EDUCON46332.2021.9453951.

3 R.E. Franken. *Human Motivation*. Brooks/Cole Publishing Company, 1994. URL: https://books.google.pt/books?id=hfW6AAAACAAJ.

4 Anabela Gomes, Cristiana Areias, Joana Henriques, and Antonio Mendes. Aprendizagem de programação de computadores: dificuldades e ferramentas de suporte. *Revista Portuguesa de Pedagogia*, 42:161–179, July 2008. doi:10.14195/1647-8614_42-2_9.

5 Ira Greenberg, Deepak Kumar, and Dianna Xu. Creative coding and visual portfolios for cs1. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, pages 247–252, 2012. doi:10.1145/2157136.2157214.

6 Edmund Harcourt. Exploring the intersection between art and technology, May 2021. URL: https://hackernoon.com/exploring-the-intersection-between-art-and-technology-en1s34fd.

7 Arto Hellas, Jonne Airaksinen, and Christopher Watson. A systematic review of approaches for teaching introductory programming and their influence on success. *ICER 2014 - Proceedings of the 10th Annual International Conference on International Computing Education Research*, July 2014. doi:10.1145/2632320.2632349.

8 J. Maeda, R. Burns, Thames, Hudson, and Massachusetts Institute of Technology. Media Laboratory. *Creative Code*. Thames & Hudson, 2004. URL: https://books.google.pt/books?id=VeO6GwAACAAJ.

9 Kylie Peppler and Yasmin Kafai. Creative coding: Programming for personal expression. In *8th International Conference on Computer Supported Collaborative Learning (CSCL)*, pages 76–78, June 2009.

10 Ricardo Queirós, Mário Pinto, and Teresa Terroso. Computer Programming Education in Portuguese Universities. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASIcs)*, pages 21:1–21:11, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2020.21.

11 John Resig. Javascript as a first language, December 2011. URL: https://johnresig.com/blog/javascript-as-a-first-language/.

12 Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003. doi:10.1076/csed.13.2.137.14200.

13 Terkelg. Awesome creative coding. URL: https://github.com/terkelg/awesome-creative-coding.

14 Joke Voogt and Natalie Pareja Roblin. A comparative analysis of international frameworks for 21st century competences: Implications for national curriculum policies. *Journal of Curriculum Studies*, 44(3):299–321, 2012. doi:10.1080/00220272.2012.668938.

15 Zoe J. Wood, Paul Muhl, and Katelyn Hicks. Computational art: Introducing high school students to computing via art. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 261–266, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2839509.2844614.

Program Comprehension and Quality Experiments in Programming Education

Maria Medvidova
Department of Computers and Informatics, Technical University of Kosice, Slovakia
Jaroslav Porubän
Department of Computers and Informatics, Technical University of Kosice, Slovakia

Abstract

The paper deals with the design of a new experimental method designed to measure the understanding of the code of subjects who do not know any programming language in connection with the implementation of empirical and analytical study. The aim of this work is the analysis of students' knowledge before and after the course Basics of Algorithmization and Programming at Technical University in Kosice, Slovakia, and the subsequent static analysis of their codes from one of the assignments. The theoretical part provides a look at the various models and ways to measure program comprehension, code quality metrics, examines the most common analysis tools, suggests recommendations for improving comprehensibility, and provides a closer look at program comprehension issues in the teaching context.

2012 ACM Subject Classification General and reference → Surveys and overviews

Keywords and phrases Program comprehension, static code analysis, empirical software engineering, code as a story, students

Digital Object Identifier 10.4230/OASIcs.ICPEC.2022.14

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

1 Summary of basic information about the program comprehension

During software development and maintenance, developers spend a significant amount of time in the process of understanding the program [12]. Studies suggest that understanding the code is a major maintenance activity because it absorbs approximately 50 percent of the cost. Problems in understanding the code are known from the time of the first developed software. The partial goal of this work is to observe these challenges from the student's point of view. As software engineering teaching expands, computer science teachers are encouraged to help students develop their understanding. Today we know that even if a programmer puts together valuable code, he may not understand it. Precisely because of this, our work is devoted to evaluating the level of understanding of code as an important part in the life of a developer [2]. We gradually analyze their ability to understand from the beginning to the end of the semester using a new empirical story method that we have developed for this purpose only, as most current studies unfortunately focus only on professionals due to the need to maintain existing software and lack student studies. In our research, we focus on various metrics that we can use to quantify code quality. We are thus preparing a unique opportunity to observe the development of participants in the subject Basics of Algorithmization and Programming, which we can later use in the evaluation of students and their codes representing potential security risks in the event of poor program implementation. Thanks to this measurement, we monitor the improvement or deterioration of potential students who may have difficulty mastering this subject in the future, and we can thus set preventive steps for the successful completion of the course [9]. In the first part, we cannot generalize their knowledge in a specific programming language, as everyone underwent a different teaching in high school. Therefore, there will be general questions in order to determine the level of the student's understanding of the code using the proposed "code as a story" method. To measure understanding in the second stage, we decided to use the "understand code" protocol. Subsequently, we evaluate the level in understanding the program before and after completing the course. In both sections, the participant was measured from the time the question was displayed until they were answered using a programmed form using Google Apps Script.

1.1 Models of understanding of program

Models are created using theories based on empirical studies by programmers who perform tasks that require them to read and understand the program.The differences between human understanding models lie in the terminology that describes the content of the knowledge base and each approach to the process of assimilation.

- Top-down model: The process begins with a hypothesis about the general nature of the program. This initial hypothesis is then refined hierarchically by creating sub-hypotheses. Sub-hypotheses are refined and evaluated in depth first [1].
- Bottom-up model: Bottom-up program understanding theories assume that programmers first read code statements and then mentally divide or group these statements into higher-level abstractions [10].
- Cognitive model: The basic structure of cognitive models consists of four components. Target system, which is to be understood. The second component is the knowledge base, which represents the previous experience of the person. The third component is a mental model that shows the current state of understanding. The last component is an assimilation process that interacts with the other three components and updates the state of understanding [4].

1.2 Complications in understanding the program

In this section, we examine the scientific understanding of code odors. A clear example from the studies we are discussing is that long methods are particularly problematic. They make it harder for developers to understand what's going on and make changes, are more prone to error, and require more effort to change. The code smells like "blob class" and "spaghetti code", which are clearly related to it.

1.2.1 Code smells

The term "odor code" was popularized by Martin Fowler. Any symptom in the code that indicates a deeper problem can be considered "odors" in the code. This problem often cannot be seen right away, but can be uncovered if the code is thoroughly analyzed. Code smells are not program errors. They only show software design issues that make it difficult to understand the code and then make it difficult to develop or extend it. Code smells also increase the likelihood of future errors in this software. The presence of odors in the code is a sign of poor code quality. There are many potential odors. The most common code smells and thus the causes of poor code level in terms of understanding include:

- Duplicate code: If you see the same code structure in multiple places, you can be sure that your program will be better if you find a way to unify them.
- Shotgun Surgery: Occurs when one change requires adjustments in many classes.

- Feature Envy: This is code smell that describes when an object accesses another object's fields to perform an operation, instead of just telling the object what to do.
- Middle Man: Occurs when a function has too much delegation to other functions and methods and does almost nothing else because the work gradually moves to other methods.
- Extremely short identifiers: The name of a variable should reflect its function, unless the function is obvious.
- Too complex conditions: Extensive nested conditions tend to grow more and more complicated over time as developers constantly add conditions and more levels. The deeper the nesting, the more time it will eventually take to remodel.
- Spaghetti code: The nested if, for, do, while, switch, and try statements are a key component in creating what is known as "spaghetti code." Such code is difficult to read, refactor and therefore maintained.
- Unnecessary comments: Unused code should be deleted and, if necessary, retrieved from the source check history.
- Dead code: These are methods and classes that are no longer used.

1.2.2 Technical debt

The technical code debt is an insufficient and misspelled code that violates best practices and coding rules.Technical debt is like a collection of code odors along with bad architectural decisions. Resolving a technical debt usually takes longer. It is most often increased by adding another method to an already too long class, copying and pasting an existing class or method, but with a different name. But whatever the cause, technical debt will inevitably cause more work in the future by not introducing better solutions now. That's why Ward Cunningham coined the term "debt." The more you pay, the more interest you pay [3]. And in this analogy, refactoring is the process of paying off this debt.

1.3 Code comprehension measurement concepts

In most software engineering processes, a complete understanding of the entire program is unnecessary and often impossible. Change requests are often formulated in terms of domain concepts, such as "Add the ability to save a trusted device to the authorization system." An important task is then to understand where and how the relevant terms are implemented in the code. We know many protocols designed to measure it, the most well-known are think out loud protocol, remember, understand the code.

1.4 Approaches to improve the comprehensibility of existing code

There are a number of factors that can be helpful in improving the accuracy, correctness, completeness, and simplicity that a program can understand. In this subsection, we list the most common ones.

- Syntax highlighting: The function displays text, especially source code, in different colors and fonts depending on the category of expressions. It is commonly used to highlight different code constructs using different colors and fonts for different identifiers. Although this is a very basic approach, it undoubtedly contributes greatly to improving comprehensibility.
- Expertise in the domain: Many empirical studies have shown that expertise is important in building fast and accurate comprehensibility. Well-founded knowledge of the problem domain, troubleshooting, and programming languages make it easier to understand the program [6].

- Cross-references: Support is used to provide a link between an identifier definition and its use in different places in the source code.
- Annotations and comments: Annotation is another way to help a programmer easily understand code. These are virtually added comments that did not originally exist in the code, but are often more useful.

1.5 Tools helping to understand programs

In the early days of programming, source code was often written using a standard editor such as Emacs or Vi. However, as the complexity of the programs grew, the standard text editor was no longer enough, so the need for integrated development environments grew rapidly. The area of program understanding research has resulted in many different tools that help in understanding the program, and in this subsection we discuss the features of the tools that support program understanding. The tools for understanding the program can be roughly categorized according to three categories:

- extraction,
- analysis and
- presentation.

Extraction tools include analyzers and data collection tools. Analytical tools perform static and dynamic analyzes to support activities such as clustering, concept assignment, feature identification, transformations, domain analysis, division, and metric calculations. Presentation tools include code editors, browsers, hypertext, and visualizations.

It is best to use the integrated development environment (IDE) and its built-in application add-ins, which combine common development tools into a single graphical user interface and provide these features. However, if that's not enough, we provide a brief overview of the tools that help to understand. Rigi supports multiple views, cross-references, and bottom-up comprehension queries. Reflection tool supports a top-down approach by generating and validating hypotheses and helping to bridge the gap between source code and high-level models.The Bauhaus tool has features to support clustering and concept analysis for software architecture, software maintenance and program understanding. And Codecrawler is a search tool specifically designed for use to search for source code. We can classify the mentioned tools mainly in the category of analysis but also partially in the extraction.

2 Student and understanding of code

As the teaching of computer science expands, computer science teachers are encouraged to help students develop an accurate understanding. Introductory programming courses are challenging for students, and novices often show misconceptions that hamper their ability to learn and make progress. Students should test and analyze their code to identify and correct problems [8]. Here are some of the methods used to improve teaching: work by learning and learning by example, live coding, gameplay and gamification, mentor support, peer mentoring and programming, information and demonstration of bad practices leading to technical debt [7].

We think that the key knowledge for a beginning student to learn to program is the ability to solve programs, math skills, abstraction and creativity. A good knowledge of English also forms a solid foundation, due to the great support of the community in the English language, available material also in the form of videos, courses, forums but also English keywords in programming languages. Beginning programmers have trouble understanding abstract concepts as real-life equivalents, the syntactic side of the language, and the actual execution and debugging of the program. On the other hand, the main challenge for teachers is to maintain motivation, relationship and choice of the right tools and language [11]. We will need more new professionals in the field than will be produced next year. The need for individuals in the field of informatics is expected to increase faster than the average of all professions.

3 Source code analysis

Code quality is important because it affects the overall quality of the software. And quality affects the extent to which your code base is secure, seamless, and reliable. When the code is of low quality, it can pose security risks. If the software fails due to a security breach or security flaw, the consequences can be catastrophic or even fatal, so high quality should be the goal of the entire development process.

3.1 Most used code metrics

There is no specific way to measure code quality, but there are several metrics that can be used to quantify code quality. Different teams may use different definitions and metrics based on the context of their area [6]. Code that is considered good can mean one thing to a car developer. And for a web application developer, that might mean something else. For this reason, we explain what code quality is, how to improve code quality, what important code quality metrics are, and how code quality tools can help, but we don't list just one specific procedure or metric. The chapter introduces the more well-known metrics used and their value.

The most widely used and discussed metric for estimating project scope is the number of lines of source code, referred to as LOC. It is currently used as a basis for other metrics and is used to evaluate the size of a software system or the effort in writing code, but it depends significantly on the programming language used. The basic premise is that the larger the software system, the more difficult it will be to understand and maintain it. LOC is a size metric, another simple alternative may be the number of functions. Halstead's metrics link metrics calculated from the line count and syntax elements of the program's source code.

In addition to indicators for evaluating the volume of work on a project, indicators for assessing its complexity are also very important for obtaining objective estimates for a project. As a rule, these indicators cannot be calculated at the earliest stages of project work, as they require at least a detailed design.

Cyclomatic complexity measures the number of linearly independent paths through the program source code and is calculated using a program flow control graph. The graph nodes correspond to the indivisible groups of program commands, and the leading edge connects the two nodes if the second command could be executed immediately after the first command. It can also be applied to individual functions, modules, methods or classes within a program.

Control flow metrics are a separate group of software metrics. These are based on the assumption that the more complicated the management flow, the more complex the program. The best known is the Chepin metric. The calculation is performed by analyzing the nature of the use of variables from the input-output list and serves to determine the degree of difficulty in understanding programs based on input and output data. The calculated weightings are used to express the different effects on the complexity of each functional group's program.

3.2 Static code analysis

Static analysis is best described as a debugging method by automatically examining the source code before running the program. The popularity of software quality analysis tools has increased over the years, with special attention paid to tools that calculate technical debt based on a set of rules, detect code odors, and usually also provide an estimate of the time required to correct technical debt. Of course, this can also be achieved by manual code checking. However, using automated tools is much more efficient.

There are many static analyzers on the market. Examples of some tools are CodeScene, Clang Static Analyzer, SonarQube, DeepSource, OCLint, Embold, Klocwork, Coverity Scan, Fortify Static Code Analyzer. Based on the above list of synthetic analyzers, SonarQube was selected for research. It is suitable for evaluating programs taking into account the desired metrics to be evaluated in this work at the same time is efficient and easy to use.It has a paid and unpaid version, which we used for our needs.

4 Study design

In designing the empirical study, we identified the key variables to be measured during the study, the appropriate subject population for the study, and the reference tasks [5]. These steps will be described in more detail in this section. We used the most suitable tool selected in 3.2 and then evaluated all the collected data.

4.1 Division of the study

The study is divided into empirical and analytical parts using different methods. In the analytics, student codes will be examined and measured using quality metrics for one and the same task. The course of research will be divided into 2 main parts, namely the data collection before and after the introductory course. With regard to the above division, research questions will also differ. In the first part, we cannot generalize students' knowledge of a specific programming language, as everyone underwent a different teaching in high school. Therefore, there will be general questions in order to determine the level of the student's understanding of the code using the proposed code as a story method. In both stages, the participant will be measured the time from the display of the question to his answer using the programmed form in Google Apps Script. The second part of the study will be carried out after the completion of Basics of Algorithmization and Programming course, so it is assumed that students' knowledge should be unified. In this introductory course, the aim is to acquire basic knowledge and skills of programs in procedural language C. Questions will therefore use programming language C.

4.2 Studied population

One of the goals of our study is to find out how students perform activities related to understanding the program. The research was focused on full-time students of the introductory course Fundamentals of Algorithmization and Programming at the Technical University in Kosice in the academic year 2021/2022, who were given the opportunity to answer questions electronically at the beginning and end of the course. The majority of participants are 18-20 years of Slovak nationality with a predominant male gender. The course has more than 400 participants every year. Although everyone was contacted, in the first stage, the form was completed by only 27 students. In the second stage, only the participants of the first were contacted, but we received feedback from the survey from 7 students. The researched codes are available from 19 students participating in the first series of studies. In the study, the response rate was 7 percent in the first and 26 percent in the second phase of the questionnaire, so the number of people who responded to the survey was divided by the number of people in the sample. We attribute the low level of response to the lack of interest of students in the researched topic and no obligation to fill in the questionnaire.

4.3 Research questions

The purpose of this study is to examine the relationship of understanding the code of early programmers before and after the introductory course and the correlation of study performance in programming among first-year students of the Technical University who attended the Basics of Algorithmization and Programming. In view of the above objectives, we have identified the following research questions:

1. What are the students' abilities in understanding the programs?
2. Is there a difference in students' understanding of the code before and after the introductory course?
3. Are there any relationships between the ability to understand the code and the learning performance in the introductory programming course?

4.4 Proposed experimental method "code as a story" and demonstration of its use in research

When designing the study, we solved one fundamental problem, and that was how to examine the understanding of code in a group with different knowledge. First-year students come with different academic training. Some have not even encountered programming yet. An experimental method has been proposed in which the code is presented as a story. It focuses on comprehension of the text, memorization, attention and detail. The text was categorized on the basis of programming properties and relationships to individual groups and represented as a text game called "Peter's Friday". The identified categories were condition (problem with nested if-else-if, switch logic problem), cycle (for cycle, while - do cycle), bits operations (AND, OR), fields and pointers. The "code as a story" method used is based on using real-life situations or their simulations in connection with programming turns and focusing the content of the story on the researched population.

The whole story begins with the student's introduction to the story and the first question whether the main character Peter will take an umbrella when the logical condition is formulated: " in the weather at the FM station they said that it would be rainless in the afternoon or my mother warned me that it would rain all day from nine, I will take the umbrella anyway ". This part is focused on bit operations AND and OR, the student must be aware of this and correctly evaluate the statement. When evaluating the whole expression, it is found that the statement takes the value 1 and therefore the correct answer should be YES.

The problem with the nested condition "if-else-if" where the goal is for the student to correctly identify two situations that are "game changer" and evaluate what outputs can arise from situation. It can help students to compile a flowchart to find out that there can be only 2 situations. The third question falls into the category of pointers that we simulated with a finger pointing situation. The array category is represented by the fourth part of the story. Specifically, it focuses on going through them from beginning to end and vice versa and thus the sequence of elements in the field. In our story, the field represents a dryer and things hung on it. The student must be aware of the sequence of things on the dryer (sequence of elements in the field) and based on that determine whether the chosen things are correct.

The fifth task is focused on understanding the main condition of the cycle. The logic of this snippet can also be understood as a while-do loop. There is a catch there, the condition states “yellow building on the right”, while the text ends that Peter saw “yellow building on the left”. The next passage contains a reference to the SWITCH construct. So the condition can be understood, depending on what beer I order, I pay so much for three pieces. The story states that Peter ordered a dark beer, which means that the correct answer is 7.5 euros. The last question and its text bridge to the “for” cycle, where we know the exact number of repetitions in advance and also to remember the information about the price of the product mentioned in the previous question. Determines whether the student can identify the specified number of repetitions.

4.5 Data collection

We decided to conduct the research online using a form, taking into account culminating wave of the COVID pandemic at the time of the research. A questionnaire programmed with Google Apps Script was used. It was advisable to have the time of each student’s response recorded, and no free software met this condition. Completing the questionnaire online was quick, easy, participants just needed to write the answers in the appropriate field. The answers were automatically stored in a clear table in a file, which was then used to evaluate the research. In the report, we simply see the time needed to understand each task. Significant differences were excluded from the responses examined. Based on time, we ruled out answers to one question in less than 10 seconds and longer than 30 minutes.

5 Summary of results

An important step in the survey was to map the current state of knowledge of students’ code with the intention of subsequent analysis of their codes for the same assignment. In this part, the collected data from the empirical and analytical part are represented.

5.1 Evaluation of questionnaire research

The total number of participants was 27, the first part deals with the evaluation of their understanding of the program using the story structure of the code and then, the second paragraph, evaluates the data from the form with the same categories submitted in C.

5.1.1 The first series before completing the course

The questionnaire was filled in on 26.10. - 16.11.2021. After analyzing the first ten answers, we decided to ask question number 2 differently, because we found that the analysis focused on the knowledge of determining the probability and not the “if-else-if” condition. A modified version of this question finds out the number of different paths, the original wanted to know the percentage of probability of the described situation. Following the change in wording helped to increase the success by 6 times. Questions from the field categories, the “switch” and the “loop for” have the most correct answers, while the least from the “while-do” and pointers have the most correct answers. Students answered the longest question in part of the fields, averaging 305 seconds, which is about 5 minutes, and the shortest, only an average of 46 seconds, a question from the “for cycle” category. For a better idea of what story form of the questionnaire looked like, we offer a sample question. The logic of this passage can be understood as a while-do loop with a focus on understanding the main condition of the loop. There is also a catch, the condition states "yellow building on the right", while the text ends that Peter saw "yellow building on the left". Thus, the condition for ending the cycle was not met and main character of story, Peter, had to continue straight on, still moving 10 meters, and so the student has to identify whether the specified minimum number of cycle repetitions is correct or not. According to story example below, the correct answer is yes, Peter walked more than 65 meters. It is necessary to think that the questionnaire was designed to gradually introduce the student to the story and therefore, this sample is torn from the story.

> When he finally manages to find one of his T-shirts, he sets off. He and his friend Michael have long ago agreed that they will go for a beer, but since Michael did not remember the exact name of the company they were going to, he only gave Peter a description of the route and the name of the stop he is to get off: "When you leave the stop straight until you find the yellow building on the right and then immediately turn left." Each building is exactly ten meters away from the next, and the distance between the stop and the first building is exactly ten meters. Pete walked past the five buildings and began to doubt that Michael was really remembering the way when he suddenly noticed the first yellow building on the left. After a while, he finally got to the business Michael was telling him about, and he looked angrily at the distance traveled on his watch. Did Peter walk more than 65 meters from the stop until he finally found the yellow building? (Yes/No)

The research shows the expected results. Simpler concepts such as fields or cycles have a success rate of 80 percent or more with a smaller average response time. On the other hand, the concept of pointers or bit operations has a success rate of 60 percent or less with an average time of 100 seconds or more. A total of five students out of 27 answered all the questions correctly. The relatively simple concept of "if-else-if" remains an unexpected result, where the observed increase in success from 10 percent to more than 60 percent is still a relatively small ratio, and its value ranks among the less frequently answered categories of questions.

Data from the point of view of individual questions were evaluated above. Figure no. 1 shows the success rate of each student and the total response time. Significant differences in overall time are visible and it cannot be said unequivocally that completion time plays a role in success. The best results using the experimental method are shown by students s13 and s17, who completed the form in 10 minutes and at the same time achieved 100 percent success.

5.1.2 The second series after completing the course

The second series was attended by 7 students, namely students with numbers s2, s9, s10, s13, s18, s25 and s26. The form was filled in on 03.02. - 14.02.2022. The percentage of correct answers was 100 percent only for a question from the field category and, conversely, the success rate of 70 percent was achieved by questions from the categories of both cycles. Students answered the longest question in the section of cycles and the switch, namely an average of 2.5 minutes and the shortest, only an average of 48 seconds to a question from the category of the cycle "if-else-if".

The average time to complete the questionnaire is around 10 minutes, up to a significantly higher value of student s25 with a completion time of 28 minutes. Students s9, s13 and s17, who completed the form in 9 minutes and achieved 100 percent success, show the best results using the C code output method. Students with these numbers also achieved 100 percent

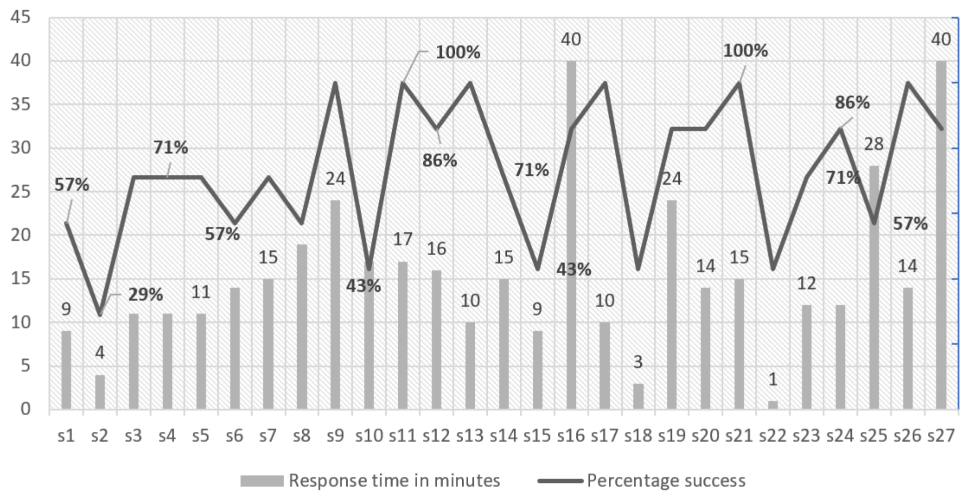


Figure 1 The success rate of each student and the total time to respond.

success in the first stage. The student with the number s2 with 57 percent achieved the lowest success rate from the monitored sample. The participant with serial numbers s25 has the worst ratio between the length of answers and the overall success rate. Below is a sample code in one of the questions and we want from students to enter the output of the program.

```
int main(){
   int a = 6;
   int b = -1;
   while (b < 2) {
     if (a <= 10) {
       a = a + 4;
     }
     else {
       a = a - 10;
     }
     printf("%d, ", a);
     b = b + 1;
   }
   return 0;
 }
```

5.2 Static analysis of student code quality

An analysis of the codes of the students involved in the research was performed on submitted codes from one of the assignments, so they all wrote program for the same problem. The results show that student s12 achieved the highest cognitive complexity 48 and the lowest student s25 only 18. Student s21 had the highest cyclomatic complexity and the lowest again s25. Except for students s16 and s24, they all have a cognitive complexity higher than cyclomatic. In the examined sample, the average cognitive complexity was 34 and cyclomatic 30.Student s11 has the highest number of comments 34.4 percent. Of course, there are also students like s24, s5 and s4 who have no comments. Although student s13 has only 129 lines, his number of logical lines is 108, which is the smallest difference between the values. The lowest number of LOCs is s25, which may be why it achieved the lowest cyclomatic and cognitive complexity.

Regarding the number of errors, we observe only four students who recorded 1 to 3 errors. These were errors performing arithmetic and logical operations with uninitialized variables. The number of odors in the code is interesting. Student s11 has the highest number, namely 29, and participant s3 has the lowest number of odors 3. This is a significant disparity between students. Student s3 performed best with 0 errors, odors 6 and the lowest technical debt 0.7 percent. Given the achieved LOCs numbers, we observe a very high incidence of odors on disproportionately small programs. The technical debt ranges from 30 to 150 minutes. All students achieve very well the results of technical debt repair within 1 or 2 hours.

Based on these data, we can evaluate that students in the first year of the introductory course of the Technical University best understand the logical turnover of fields, switches and bit operation evaluation, on the contrary, they understand the cycle and pointer the least. The students who did the worst in the form sections (s2, s10, s18, s15 and s22) did not fulfill the selected assignment from the course and therefore we could not examine their codes, which they never submitted. It is also possible to note that the nominees had one of the worst successes in the experimental method and the results were confirmed after the end of the course using a validated C output determination procedure. Comparing static code analysis with success in measuring code comprehension yields interesting data. We believe that if the students who did the worst in the empirical part submitted the assignments, they would be evaluated poorly in code quality and we can also say that the ability to program the assignment may be related to understanding results, as the worst participants by research did not submit the assignment. Furthermore, students with numbers s1, s11, s14 and s25, whose codes also have a high degree of complexity, errors and technical debt ratio, show the worst results of the submitted assignments in the odor number metric. These participants have an average ability to understand the code.

6 Conclusion

An experimental "code as a story" method was designed and used, and data from 27 students from two stages of research were processed. Finally, 19 student assignments underwent a static analysis. The biggest problem was the low turnout, but mapping this process provided some interesting insights. Research shows that more than half of students incorrectly answered one or more questions about the implementation of basic programs and the associated understanding of the code. There is a slight improvement for participants who have also completed the second stage of the forms after completing the course. Less than half of the students were able to answer trivial questions, and in interviews with them we found out that they knew the programming terms used to program the assignment, but had trouble determining the output of the program shown, so they did not understand the code. In static code analysis, we observe a correlation in bad results and non-submission. With average and above-average results in the empirical part, students have a lot of odors in the code. In 68 percent of students, 10 or more odors were identified, which is an average of LOC 137 for every 10 lines of code. The most common odors were spaghetti code, empty-body functions and methods, never used but defined variables, extremely short naming. The highest number of odors was 29 and the lowest 3. This is a significant disparity between students. We observe the appropriate use of the acquired knowledge in the possibility of improving the teaching of introductory programming courses. The path is in the integration of coding standards when passing mandatory fields. Using a coding standard is one of the best ways to ensure good code quality. The coding standard ensures that everyone uses the right style. It improves consistency and readability, and this is the key to reducing complexity and improving quality. We recommend that students complete a text story in the first days of study to identify weaker course participants and then draw more attention to their shortcomings in order to avoid failure in the exam or credit in this subject. We see an opportunity to continue our studies, but a larger sample of students will be needed.

References

1 Rodney A Brooks. Planning collision-free motions for pick-and-place operations. *The International Journal of Robotics Research*, 2(4):19–44, 1983.

2 Malcolm Corney, Donna Teague, Alireza Ahadi, and Raymond Lister. Some empirical results for neo-piagetian reasoning in novice programmers and the relationship to code explanation questions. In *Proceedings of the fourteenth australasian computing education conference*, volume 123, pages 77–86, 2012.

3 Rosemary T Cunningham. The effects of debt burden on economic growth in heavily indebted developing nations. *Journal of economic development*, 18(1):115–126, 1993.

4 Françoise Détienne. *Software design–cognitive aspect.* Springer Science & Business Media, 2001.

5 Massimiliano Di Penta, RE Kurt Stirewalt, and Eileen Kraemer. Designing your next empirical study on program comprehension. In *15th IEEE International Conference on Program Comprehension (ICPC'07)*, pages 281–285. IEEE, 2007.

6 Amy J Ko and Bob Uttl. Individual differences in program comprehension strategies in unfamiliar programming systems. In *11th IEEE International Workshop on Program Comprehension, 2003.*, pages 175–184. IEEE, 2003.

7 Raymond Lister, Elizabeth S Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, et al. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4):119–150, 2004.

8 Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 125–180. ACM, 2001.

9 Rodrigo Pessoa Medeiros, Geber Lisboa Ramalho, and Taciana Pontual Falcão. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, 62(2):77–90, 2018.

10 Nancy Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive psychology*, 19(3):295–341, 1987.

11 Yizhou Qian and James D Lehman. Correlates of success in introductory programming: A study with middle school students. *Journal of Education and Learning*, 5(2):73–83, 2016.

12 Elliot Soloway and Kate Ehrlich. Empirical studies of programming knowledge. *IEEE Transactions on software engineering*, 5:595–609, 1984.