Computer Programming Education in Portuguese Universities

Ricardo Queirós
CRACS – INESC-Porto LA, Portugal
uniMAD, ESMAD, Polytechnic of Porto, Portugal
http://www.ricardoqueiros.com
ricardoqueiros@esmad.ipp.pt

Mário Pinto
uniMAD, ESMAD, Polytechnic of Porto, Portugal
mariopinto@esmad.ipp.pt

Teresa Terroso
uniMAD, ESMAD, Polytechnic of Porto, Portugal
teresaterroso@esmad.ipp.pt

Abstract

Computer programming plays a relevant role in the digital age as a key competency for project leverage and a driver of innovation for today's modern societies. Despite its importance, this domain is also well known for their higher learning failure rates. In this context, the study of how computer programming is taught is fundamental to clarify the teaching-learning process and to ensure the sharing of the best practices. This paper presents a survey on computer programming teaching in the first-year courses of Portuguese Universities, more precisely, what is taught and how it is taught. The study focuses essentially on the following facets: the class characterization, the methodologies used and the languages/technologies taught. Based on these criteria, a survey was done which gathers information of 59 courses included in a wide range of Universities spread across Portugal. The results were collected and analyzed. Based on this analysis a set of conclusions were taken revealing some interesting results on the teaching methods and languages used which can be useful to support a discussion on this subject and, consequently, to find new paths to shape the future of programming teaching.

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

Keywords and phrases computer programming, teaching-learning, universities

Digital Object Identifier 10.4230/OASIcs.ICPEC.2020.21

Funding This work is financed by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.

1 Introduction

Computer programming is considered one of the most important and emerging domains in today's society. As a domain with large market demand, educational institutions have been including in their curricula, a set of related disciplines, ranging from the introductory level to a more advanced one.

At the same time, this domain has high levels of failure, especially in introductory programming disciplines. There are several reasons for this fact [2, 5, 14, 11], ranging from traditional teaching methods, the difficulty of students in enhancing the problem-solving facet, the small and limited number of programming exercises, to the lack of automated tools to assist teachers in authoring and evaluating exercises and students in monitoring their resolutions [4, 6].

With the advent of automatic tools for evaluating programming exercises, which came in part to relief the teachers from the manual burden of manual assessing of student's code, and the Web learning environments that came to provide more sustained guidance to students while solving exercises, we continue to assist to a great lack of motivation in learning programming [1, 8].

In the last few years, in order to engage students and foster a collaborative and competitive spirit, elements associated with the mechanics that we typically see in games, the so-called gamification, began to emerge [12]. Gamification is now a crucial component in learning environments and has been used in order to motivate students to remain focused on overcoming their difficulties. Despite the huge buzz with gamification in education, the lack of systems that can be easily integrated into learning environments and their unbalanced use, reveals that this technique has not yet been fully exploited [7].

This paper presents the current state of the teaching-learning process of computer programming in Portugal. For this study, a survey was defined and distributed to several Portuguese educational institutions, more specifically, to teachers who teach introductory programming subjects. This survey raises several questions related to the characterization of the classes, the teaching methods, the languages and tools used, the reasons for the current difficulties and the desired improvements. Based on the results of the survey, an analysis is made that essentially aims:

- to characterize the teaching of computer programming in Portugal;
- to know which methodologies, languages and tools are adopted;
- to identify good practices implemented (with satisfactory results);
- to outline lines of action for the future of programming teaching.

The remainder of the paper is structured as follows: Section 2 explores the reasons for the failure of teaching programming. The next section describes the experience made to obtain data on teaching programming in Portugal, namely, the methodology used for data collection. The results analysis section presents the results of the survey and analyzes them. Finally, the conclusions, the main contributions of the article and possible paths for future work are presented.

2 Programming education issues

In this section, we begin by identifying the main reasons for the difficulties that teachers and students have in the teaching-learning process of computer programming.

In order to learn to program it is not enough to know the syntax of a language. There is a set of inherent concepts that requires a level of abstraction and structured reasoning from the student, which is difficult to achieve, especially in an introductory phase. Several scientific studies points out several reasons for failure in this area [2, 5, 14, 11]:

- Complex domain of complex programming;
- Traditional teaching and study methods;
- Psychological aspects;
- Difficulties in using/integrating automated tools.

In the next subsections, we explore all these facts.

2.1 Complex domain

Programming learning requires a range of skills ranging from problem-solving to abstraction. These skills associated with reasoning structured in order to find the best solutions for a given problem are decisive for successful progress in this domain. Several studies show [9, 11, 5], that, at an early stage, students have difficulties in assimilating all of these skills.

In fact, problem-solving is nowadays seen as one of the main soft-skills that anyone must have in order to be successful in their work. This skill is essentially characterized by five steps: In a **first phase**, you start by analyzing a problem (for instance, a programming assignment) and identifying what needs to be addressed to obtain a solution. In this step, the necessary skills focus on good reading, interpretation, and analysis of the problem and adequate identification of the requirements.

In a **second phase**, and after realizing the problem and identifying needs, it is time to discuss possible solutions. It is rare that a single strategy is an obvious answer to solving a complex problem. The creation of a set of alternatives helps to cover all needs and reduces the risk of exposure if the first strategy that implements fails. At this stage, the necessary skills focus on good planning for solving a problem (e.g., developing algorithms).

In a **third phase**, the best solutions are evaluated. Depending on the nature of the problem, the evaluation of the best solutions can be carried out taking into account several criteria (e.g. getting from point A to point B more quickly or spending less money). Here the necessary skills are discussion and teamwork, prioritization and test development.

In a **fourth phase**, the decision reached in the previous phase is implemented. Here, a programming language is typically used to implement the best solution in order to solve the problem. As necessary skills, we highlight the ability to codify and collaborate (in this case, group work).

Finally, the effectiveness of the solution execution is evaluated.

Many students have also deficits in mathematical and logical knowledge. Several experiments [9] were carried out to find correlations between mathematical knowledge and the lack of programming skills. In these experiments, the authors concluded that the students involved had profound difficulties in several areas, such as basic calculus and theory of numbers or simple geometric and trigonometric concepts. The authors also report difficulties related to the transformation of a textual problem into a mathematical formula that solves it. Limitations in terms of abstraction and logical reasoning have also been identified.

At the same time, and still related to the nature of programming, another problem persists that is closely linked to the syntax of languages. In fact, the syntax of languages is complex (in fact, they were designed to be used at a professional level and not to support your learning) and, in some cases, has evolved in a meteoric way [13], making students have difficulties in its adaptation, memorization and consequent application. Obviously, these problems can be alleviated by teaching and study techniques that are discussed in the next subsection.

2.2 Traditional teaching-learning methods

One of the main problems in teaching programming has to do with the fact that teachers are typically more focused on teaching a programming language and its syntax, rather than promoting problem-solving using a programming language. This enormous emphasis on syntax, at the expense of a more practical component, is an obstacle to student's progress. Also, in the programming area, where there is a great need for teaching dynamic concepts, this is usually done using materials of a static nature (e.g. drawings on the board, slides that are too long and confusing and verbal communication, sometimes deficient and of difficult understanding).

At the same time, the study methods are also not the most suitable. Programming requires a very practical and intensive study. Obviously there are many disciplines that require study methodologies based on reading and memorizing formulas or procedures. However, programming, like mathematics, requires a different method of study that involves intense training. The only way to learn to program is to program. Just watching classes, watching videos and reading specialized books is not enough. Moreover, students tend to give up problems whose solutions they cannot find in a simple and quick way, so monitoring 24x7 tools, outside the classroom, would be desirable.

It is unanimous that the most effective method for learning any domain is practice [11]. In computer programming learning, the practice comes down to solving programming exercises. In order to have exercises it is necessary to create them or reuse existing ones, which is complicated as the best exercises are often inaccessible or in proprietary formats. Even with these exercises, it is necessary to make them available to students in an attractive and practical way, organized into well-defined thematic modules and ordered by difficulty levels. This organization and sequencing benefits the student's progress and consequent motivation [16].

However, it is not enough to put a battery of exercises for the student to solve in order to master programming. For practice to be efficient, feedback is required. If the feedback is null or inaccurate, then practice can be detrimental to the student's sustained progress. To have feedback, the teacher must have time to be able to answer all requests from the class, typically with a large number of students

Another major problem in teaching programming is that it is not personalized. Typically, teachers' strategies do not usually address all student learning styles. It is a fact that we all learn in different ways and consequently have different preferences in learning in order to assimilate content and good practices. However, by adopting traditional methods, the teacher is forcing all students to have uniform learning, at the same pace and according to their pedagogical strategies. However, the high number of students in the classroom combined with time constraints makes more personalized approaches impossible. In an optimal world, the teacher should be able to contemplate the enormous diversity of learning styles present in the classroom and adapt teaching to each of the profiles found. One solution to this problem is the use of automated tools that support this personalized teaching [10].

2.3 Psychological aspects

The cognitive and motivational aspects are fundamental to the success of learning computer programming. The lack of motivation is perhaps one of the biggest reasons for school's failure. Many students are not motivated enough to study programming, due to its reputation for being difficult and the extremely negative connotation associated with it. There are same studies that indicate that there is a public image of a "programmer" as a "social inadequate" [5,6].

In addition, students have introductory programming disciplines during one of the most difficult periods of their student life, that is, at the beginning of a college degree in computer science, coinciding with a period of transition and instability in their life. There are even authors who consider that the programming disciplines are poorly located in the curriculum [5,6].

Gamification strategies can be used in the educational process of programming learning. These strategies foster engagement through collaboration (e.g. students interact each other in order to solve a challenge) and competition (e.g. students compete to be the first to solve a challenge). In fact, new methodologies and techniques are appearing aiming to improve retention and foster the motivation and competitiveness of computer programming learning [12]. While the concept of "winners and losers" can hinder the motivation of students [15], competitive learning is becoming a trendy learning paradigm that relies on the competitiveness of students to increase their programming skills [3] with promising results.

2.4 Difficulties in the use/integration of automated tools

Another major problem with this process is the fact that classes are typically very long which severely undermines the work of teachers. In addition to manually correct students' resolutions and give feedback, teachers have to give classes more quickly in order to teach all the course subjects and to foster the delivery of assignments to students.

These issues can be mitigated with the use of specialized online tools to support and guide the entire teaching-learning process of computer programming. Currently, there is a vast set of tools ranging from repositories of programming exercises to dynamic code evaluators [13]. However, despite the existence of several tools, their continued use is still scarce. There are several reasons for that ranging from the lack of time for its adoption and the interoperability issues in the most diverse infrastructures scenarios.

In this realm, we can organize existing tools into the following categories:

- Teaching-learning environments - environments that allow the teacher to create and manage their exercises and make them available to students and students to solve and submit them and access the resolution feedback;
- Exercise repositories - systems that allow the storage, cataloging and subsequent discovery of exercises by teaching-learning environments;
- Assessment tools - tools or services that receive the resolution for a given programming exercise and that return an evaluation of it;
- Gamification services - services that provide gamification components to be included in the teaching-learning environment with the specific aim of fostering student's engagement.

It is also important to state that computer programming learning tools are not limited to this list. Other systems can be used to assist in the process such as anti-plagiarism tools, recommendation systems, feedback animators, bots, and others.

3 Survey on Computer Programming Teaching

In order to understand how Higher Education Institutions (HEI) approach the teaching and learning processes of computer programming, particularly in the introductory units, a questionnaire survey was conducted. This questionnaire aimed to characterize what is taught and how it is taught, namely the topics covered in the unit courses, the methodologies adopted, tools, languages, good practices and the main difficulties encountered in the programming teaching process. The questionnaire ends with a request for suggestions on what might improve the teaching and learning process of computer programming. Considering these objectives, the questionnaire was organized into four sections:

- Respondent characterization (institution, course degree and course unit, number of contact hours and type of classes);
- Programming teaching (covered topics, languages and learning and pedagogical resources);
- Editors and teaching support tools (code evaluators, testing tools, plagiarism detection or gamification);
- Final considerations (average pass rate, main difficulties identified, good practices and tools that it intends to incorporate in the teaching process).

A pilot test of the questionnaire was carried out with four users. These users were invited to answer a test version of the questionnaire, providing us with suggestions for improvement. Several suggestions were received, which were incorporated in its final version.

The questionnaire was addressed to university and polytechnic higher education teachers, who teach introductory programming subjects. This target audience includes professors from higher professional technical courses (TeSP), bachelor's and master's degrees. Considering the Bologna process, some universities have created courses that combine a bachelor's with a master's, called integrated master's (five years), which are also included in the target audience.

The questionnaire was distributed through an online survey, being disseminated by eighty contacts, which fit in the target audience described above, requesting collaboration in the response. Some contacts were obtained from research carried out on the institutions' web pages, when available. Others were obtained through personal networks. The respondents were informed that the questionnaire was anonymous, ensuring the confidentiality of responses. After forty-eight hours a reminder was sent to all contacts, reinforcing the invitation to participate in this survey, which was available to respond for 10 days.

A total of 59 responses were gathered, which represents a response rate of 74%. The responses were from teachers of 4 Polytechnic Institutes (Bragança, Cávado e Ave, Oporto and Viseu) and 9 Universities (Azores, Algarve, Aveiro, Beira Interior, Coimbra, Évora, Minho, Lisbon, and Oporto). One can see that the responses obtained come from a wide-ranging geographical scope, with responses from north to south of the country, including islands (Azores).

4 Results analysis

4.1 Respondent characterization

The questionnaire was answered mostly from professors of curricular units belonging to bachelor degrees, either from Polytechnics or Universities (Figure 1). Programming classes can have four different typologies: theoretical (40.7%), theoretical-practical (57.6%), practical (18.6%) and practical-laboratory class (52.5%). More than 50% of the inquired provide 4 weekly contact hours and 20 to 25 students per class.

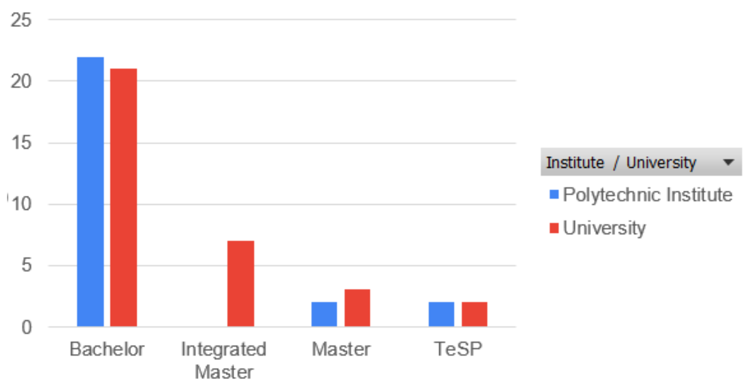

Figure 1 Degree.

4.2 Programming teaching

Regarding programming teaching (covered topics, programming languages and learning resources), the questionnaire survey offered options in a multiple-choice format. As expected, overall introductory programming curricular units covered the basics of programming, like variables (89.8%), operators (83.1%), structures and data types (89.8%), control structures (89.8%) and functions (91.5%). As for the languages used in initial programming classes, 16 of the inquired use C, followed by Python and Java (12 and 11 answers, respectively). Only 6 teachers answered C#, and the remaining languages had under 5 responses, Figure 2.

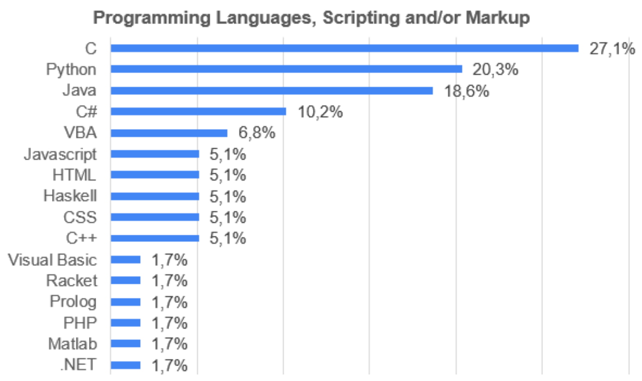

Figure 2 Programming languages.

Regarding the learning resources, mostly use classic non-interactive approaches such as presentation slides (88.1%), notebooks (25.4%) and books (72.4%). Online tools already have relevance as a resource in programming teaching, as 22.4% use online tutorials, 6.9% adopt learning platforms in their classes, like Udemy or code.org, and 5.2% make use of YouTube or other online videos. Exercise solving is a feature on which the process of teaching computer programming learning depends on. More than 85% of the professors state that the exercises are created from scratch to the curricular unit and more than 60% claim their exercises are revised each year. Almost 80% of the exercises are solved in a code editor; the remaining use some sort of platform (online or adopted to the programming curricular unit).

4.3 Editors and teaching tools

The questionnaire results revealed that a multiplicity of code editors is used in Portuguese higher education programming classes, Figure 3.

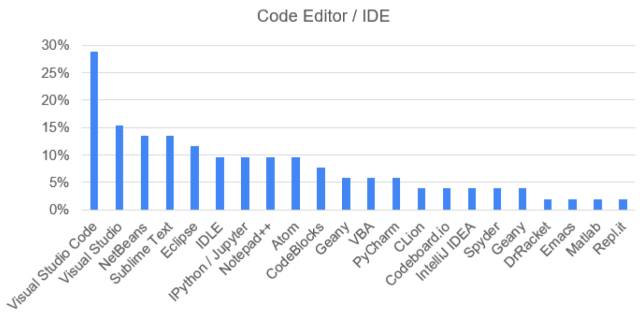

Figure 3 Code editors.

Regarding code evaluation tools, the responses obtained demonstrate that most respondents do not use code evaluators. Among those who reported using it, Mooshak stands out as the preferred option, Figure 4.

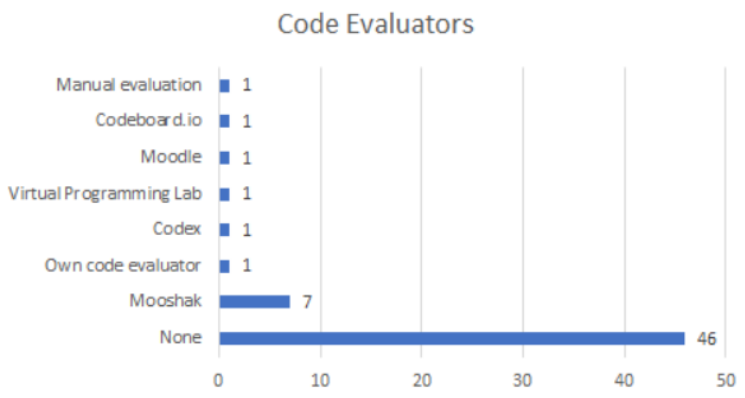

Figure 4 Assessment tools.

The same is shown when asked about testing, plagiarism detection or gamification tools. Only 4 respondents make use of some sort test framework, like Jasmine, Mocha, Enzyme, Jest, PandionJ, JUnit or QuickCheck, but none stands out as the most used. As for gamification, 6 professors employ gamification in the computer programming learning process: 2 use some tool integrated with the Learning Management System (LMS), 2 developed their own gamification and 2 others take advantage of web-based platforms like code.org and Kahoot. Plagiarism detection proved to be a major concern when compared with the latter two topics: testing and gamification. 11 answered positively when asked if they used an anti-plagiarism tool. From those, MOSS stands out as the most used (5 responses), followed, ex aequo, by Codequiry, JPlag, Urkund, Virtual Programming Lab, Blackboard SafeAssign and a proprietary application developed by the teaching staff, all of the above with 1 response each.

4.4 Final considerations

The fourth and last section of the questionnaire was composed of three open questions regarding main difficulties identified in teaching programming, best practices, and tools that could improve the computer programming teaching/learning process. More than 20% pointed out that the students' lack of strong know-how foundations and the complexity of the programming domain as the most prominent difficulties. Around 18.6% make reference to the classes with a high number of students and few contact hours (Figure 5).

Even though the average approval rate is considerably high (88% responded that the approval rate is higher than 50%), there is still potential to improve (Figure 6).

Several ideas were pointed out regarding what could improve the computer programming teaching-learning process, namely:

- More student work and responsibility
- Increase contact hours
- Motivate students (quizzes, inverted classrooms, other tools)
- Emphasis on algorithmic logic
- Introduce an automatic code evaluator
- Code execution and testing tools that provide intuitive feedback
- Smaller classes

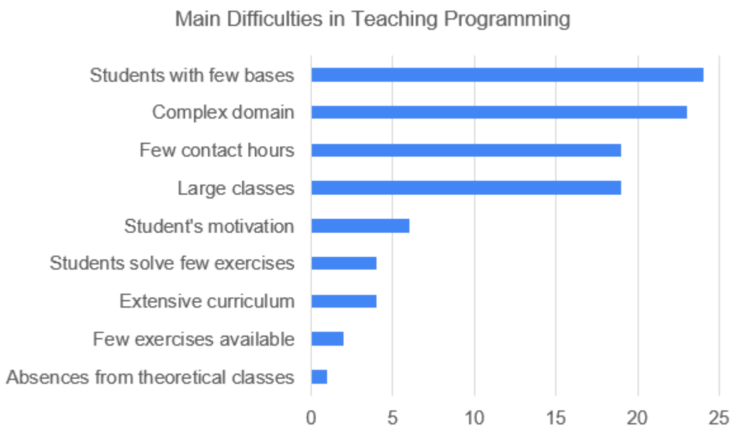

Figure 5 Main difficulties in teaching programming.

- Continuous training of teachers
- Introduce problem-based learning methodologies
- More projects
- More time for tutorial support for each student
- Introduce mob programming tool
- A pedagogical approach that motivates students
- Introductory classes with a view to standardizing students' knowledge
- Peer learning and active learning
- Guide learning to topics of interest to students
- Individual tutorial guidance
- Gamification mechanisms
- Use programming together with other curricular units

As for tools professors would like to introduce in their classes, 20 respondents did not answer or said they do not know which tool to point out. However, 29 promptly acknowledge that an automated code evaluator would improve their classes. 19 would like to include a code analyzer and 18 pointed out an open exercise repository would be a plus. Gamification, anti-plagiarism and recommendation systems were also chosen by 14, 11 and 7 of the inquired professors.

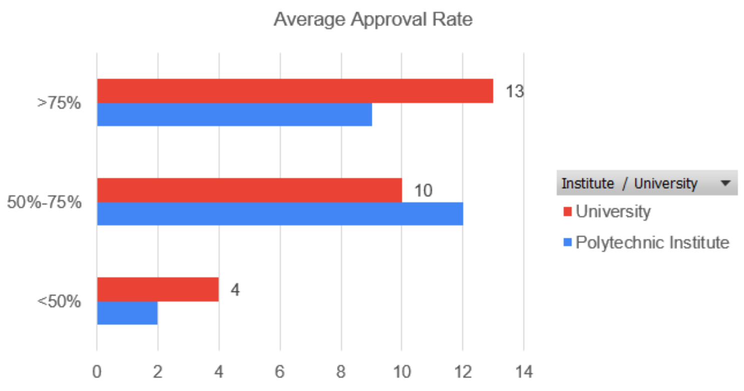

Figure 6 Average approval rate.

5 Conclusions

Learning to program is difficult. In this paper, we identify several factors that make students feel unmotivated from the methodologies used in the classroom to the psychological aspects inherent to the programming domain.

In order to try to understand how programming is taught in Portugal, a survey was carried out on more than fifty existing courses in Portugal covering a large part of the national territory and islands. The objective was to assess how programming classes are constituted, which methodologies, languages, and tools are used and what are the respondents' opinions regarding the main difficulties and which are the best approaches to solve them.

Regarding the characterization of the respondents and their classes, we had responses essentially from undergraduate courses where teachers give theoretical and practical classes for 4 hours per week.

The topics covered are initially linked to the basic concepts of languages (variables, operators, structures and data types). In terms of languages, most responses indicated C, Phyton, and Java as the programming languages taught in introductory courses. The teaching approaches are combined between slides exposing the theoretical part and the resolution of exercises in a code editor (preferably Visual Studio Code). Most exercises are created from scratch, with slight adaptations at the beginning of each year. The evaluation of the exercises is mostly done manually. In fact, the same methodology is used for testing, gamification and plagiarism detection.

Regarding the main obstacles to teaching programming, most teachers complain about the few student bases, the fact that programming is a complex field that combined with large classes and few hours of contact makes the process's time consuming and complex. Despite this, there has been a reasonable number of approvals.

As ideas for approaches to address programming learning failure, teachers point to the use of tools that automate various stages of the life cycle of the teaching process and the decrease in the number of students per class so that teaching can be more personalized.

As future work, the authors wish to improve the survey with new questions and extend the sample to international universities.

References

1 Kirsti M Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005. doi:10.1080/08993400500150747.

2 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, January 2017. doi:10.1145/3011286.3011301.

3 Juan C. Burguillo. Using game theory and competition-based learning to stimulate student motivation and performance. *Comput. Educ.*, 55(2):566–575, September 2010. doi:10.1016/j.compedu.2010.02.018.

4 Micaela Esteves, Benjamim Fonseca, Leonel Morgado, and Paulo Martins. Improving teaching and learning of computer programming through the use of the second life virtual world. *British Journal of Educational Technology*, 42(4):624–637, 2011. doi:10.1111/j.1467-8535.2010.01056.x.

5 Anabela Gomes, Cristiana Areias, Joana Henriques, and António José Nunes Mendes. Aprendizagem de programação de computadores: dificuldades e ferramentas de suporte. *Revista Portuguesa de Pedagogia*, 42:161–179, 2008.

6 Tony Jenkins. On the difficulty of learning to program. In *3rd Annual LTSN-ICS Conference*, pages 53–58, 2002.

7 Derviş Kayımbaşıoğlu, Bora Oktekin, and Hüseyin Hacı. Integration of gamification technology in education. *Procedia Computer Science*, 102:668–676, 2016. 12th International Conference on Application of Fuzzy Systems and Soft Computing, ICAFS 2016, 29-30 August 2016, Vienna, Austria. `doi:10.1016/j.procs.2016.09.460`.

8 Jackie O'Kelly and J. Paul Gibson. Robocode – problem-based learning: A non-prescriptive approach to teaching programming. *SIGCSE Bull.*, 38(3):217–221, June 2006. `doi:10.1145/1140123.1140182`.

9 Ana Pacheco, Anabela Gomes, Joana Henriques, Ana Maria de Almeida, and António José Mendes. Mathematics and programming: Some studies. In *Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, CompSysTech '08, New York, NY, USA, 2008. Association for Computing Machinery. `doi:10.1145/1500879.1500963`.

10 José Carlos Paiva, José Paulo Leal, and Ricardo Queirós. Authoring game-based programming challenges to improve students' motivation. In Michael E. Auer and Thrasyvoulos Tsiatsos, editors, *The Challenges of the Digital Transformation in Education*, page 602–613, Cham, 2019. Springer International Publishing, Springer International Publishing.

11 Ricardo Queirós. A framework for practice-based learning applied to computer programming. Master's thesis, FCUP, Porto, 2012.

12 Ricardo Queirós. *Gamification-Based E-Learning Strategies for Computer Programming Education.* IGI GLOBAL, 2016. `doi:10.4018/978-1-5225-1034-5`.

13 Ricardo Queirós. *A Survey on Computer Programming Learning Environments*, volume 1 of *1*, chapter 4, pages 90–105. IGI GLOBAL, 2019. `doi:10.4018/978-1-5225-7455-2.ch004`.

14 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`.

15 Maarten Vansteenkiste and E. L. Deci. Competitively contingent rewards and intrinsic motivation: Can losers remain motivated? *Motivation and Emotion*, 27(4):273–299, 2003.

16 Jacqueline L. Whalley, Raymond Lister, Errol Thompson, Tony Clear, Phil Robbins, P. K. Ajith Kumar, and Christine Prasad. An australasian study of reading and comprehension skills in novice programmers, using the bloom and solo taxonomies. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE '06, page 243–252, AUS, 2006. Australian Computer Society, Inc.