Design science research applied to difficulties of teaching and learning initial programming

José Figueiredo1 · Francisco José García-Peñalvo2

Accepted: 26 October 2022
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

Abstract
Learning and teaching to program is an arduous task. It requires a lot of commitment, dedication, and passion from everyone involved. Programming courses have high dropout and failure rates. Throughout time, several educational research works have been carried out to study the different learning processes and characteristics of students. With this work, we present and describe our vision and model of teaching and learning of initial programming to minimize the problems. We present a technological tool, called HTProgramming (Help To Programming), which complements the teaching and learning process. This allows students to practice a wide variety of activities with immediate feedback, directly related to content and themes for learning programming. It allows the teacher to follow the whole process and students’ results. Using a machine-learning (neural network) predictive model of student failure, it will allow the teacher to anticipate possible student failure and act quickly. In this paper, we apply the Design Scientific Research Methodology to tackle teaching and learning difficulties to initial programming. We also include the results and evaluation of the application. Students consider the application an important tool for their learning process. The student failure prediction model presents very realistic values.

Keywords CS0 · CS1 · Programming · Teaching programming · Learning programming · Design science research methodology

1 Introduction
The challenges in the teaching and learning of initial programming are a topic that has been addressed since the appearance of the first programming languages [1–5]. The theme has increasingly worrying contours since programming is a subject in the most diverse areas of knowledge. The problems in teaching and learning programming reach alarming proportions when programming is the main subject, as is the case in computer science courses. The difficulties in learning programming lead to high failure and dropout rates, lack of motivation, interest, and dedication of students [6–9].

The pandemic due to COVID-19 has added more problems. The necessary adaptations to the normal system of functioning of the classes, such as the transformation of the classes from a face-to-face system to an online format, and all the urgent adaptations concerning contents, subjects, the online assessments, and also the difficulty in monitoring the students, caused major problems to the teaching system [10–12].

This paper describes the work developed at Instituto Politécnico da Guarda (IPG), in the Computer Science course, in the curricular unit of Introduction to Programming. Over the three decades dedicated to teaching programming, there have been many experiences and constant challenges. Every year we carry out new experiments and adaptations to the characteristics of new students, to transmit the required skills. However, the result is invariably the same, high failure and dropout rates.
The main motivation for the development of this work is to understand what the real difficulties of students are, to motivate students for the hard work necessary for success in learning programming. With this goal in mind, we seek to apply a teaching and learning methodology, aided by a technological tool, that allows to assist students and teachers in the complex process of teaching and learning of initial programming.

Research papers usually follow a development pattern consisting of problem definition, literature review, hypothesis development, data collection, results analysis, and discussion. In recent years a new approach has emerged [13] called Design Science Research Methodology (DSRM). In this paper, we describe and develop the application of this methodology to the difficulties of teaching and learning of initial programming in a somewhat particular context and conditions.

This paper is structured as follows: section two describes the context and background of the problem of teaching and learning programming and describes our study group. Section three section an overview of Design Science Research Methodology. The fourth section describes the Design Scientific Research Methodology applied to initial programming problems. In the five section the conclusions are addressed.

2 Context and background

The main purpose of the teaching of initial programming is to provide students with the skills necessary for the manipulation of instructions, structures, and rules of a programming language, to create computer programs to solve problems [14]. This process requires the acquisition of a large set of specific knowledge in a short period of time. Learning initial programming is one of the major obstacles students face when they begin their studies in a STEM (science, technology, engineering, mathematics) related course [15].

In J. Bennedsen’s thesis [16], work is presented that proves the difficulties introductory programming courses have. Problems in the learning of initial programming are one of the main causes of failure and dropout of students. Besides the difficulties inherent to the beginning of a new study cycle, several factors have been pointed out as the origin of the problem [17], such as the lack of abstraction capacity, the lack of preparation for mental construction, in other words, lack of practice and training in computational thinking, the ability that allows us to create solutions to problems using computing techniques. The use of inappropriate teaching methods [18] is also a factors pointed out as a cause of students’ learning difficulties.

The approach to personalized education for learning programming is one example of how to solve such problems [19]. A model strongly focused on the student, recognizing his/her individuality, characteristics and learning pace, with good results in minimizing dropout rate and increasing grade average.

Teaching and learning programming is notoriously difficult. This is not a problem particular to one school, region, or country, it is a universal problem.

2.1 Characteristics of the study group

With constant hard work on both sides, students and teachers, with a lot of commitment, dedication and motivation, all students achieve the basic goals in learning programming. Teachers are required to be able to maintain the students’ levels of commitment, dedication, and motivation to achieve the goals. However, our study group has very particular characteristics that need to be described.

This work has involved students from an introductory programming course (Introduction to Programming), in the first semester and first year of the Computer Science course at the IPG, Portugal. In this course, the C language is used to teach basic programming concepts. The number of students per year is approximately 105 students on average. However, the number of students in the last academic year 2021–2022, has reduced considerably, to around 50 students. If we consider that only 50% to 60% of these students regularly attend classes, this leaves a rather small number of students that we can involve in our work.

The computer science course from IPG is generally not the first choice of students. This may reveal some lack of motivation for the course. However, most students have already had contact with some programming language. In the last few years, we have received students from Portugal-speaking African countries (Portuguese: Países Africanos de Língua Oficial Portuguesa; PALOP), with several problems in their general education.

In this section, we wanted to evaluate our students with respect to their knowledge of programming. For this, we used the study developed by the ITiCSE 2001 workgroup, namely the "McCracken group". The exercises are the same as the one presented in [20], we only reduced the number of exercises to 10 (from the 12 in the original) and translated it to Portuguese. Table 1 shows a breakdown of student numbers, by the quartile boundaries established, for the IPG students. Table 2 shows a breakdown of student numbers, by the quartile boundaries established, but excluding students from the institution that contributed most of the performance data, in the “McCracken group” report [20].

Analyzing of results between the two studies, we found that, in general, the results obtained by us (Table 1) are lower than the results obtained by the "McCracken group" (Table 2). We also found that the greatest differences are
located at the extremes of our scale, in the top quartile (from 13% to 20%) and the bottom quartile (37% to 29%), i.e., a difference of approximately 7% lower in the study carried out with the students from the IPG. Note the respective differences in sample size (N) and number of Multiple-Choice Questions (MCQ) (N=30, Q=10 for IPG and N=228, Q=12 for "MacCracken group"). It is also worth noting that in the intermediate results there are no significant differences, 27% and 23%, for the second and third quartile of the IPG and, the values of 25% and 26% for the "MacCracken group" results. This was also the case when analyzing the results in the study presented in [20], between the results of all institutions that participated in the study and the results excluding the institution with the largest contribution to the results.

| Quartile | Score range | No. of students | Percent of students (%) |
|----------|-------------|-----------------|------------------------|
| Top      | 9–10        | 4               | 13                     |
| Second   | 6–8         | 8               | 27                     |
| Third    | 3–5         | 7               | 23                     |
| Bottom   | 0–2         | 11              | 37                     |

| Quartile | Score range | No. of students | Percent of students (%) |
|----------|-------------|-----------------|------------------------|
| Top      | 10–12       | 46              | 20                     |
| Second   | 8–9         | 56              | 25                     |
| Third    | 5–7         | 60              | 26                     |
| Bottom   | 0–4         | 66              | 29                     |

Table 1 Shows the quartile boundaries for IPG students

Table 2 Shows the quartile boundaries for “McCracken group” students

3 Overview of design science research methodology

There has been growing interest in the engineering fields for Design Science Research Methodology (DSRM), as this methodology focuses on creating artifacts for a practical purpose [21, 22]. Some of the work developed proposes that an artifact should solve an important problem, suggesting four types of artifacts: representation constructs, models, methods, and instantiations [23]. Other work also suggests that research in DSRM should address a problem in a unique and innovative way or in a more effective way [24].

The work developed by Guido L. Geerts is an excellent description of the evolution of DSRM among the scientific community [22]. It describes the importance that some of the works have achieved, awakening the interest and need for the creation of a series of journals in the area of information systems (IS) to launch special issues on design science research. Some examples are the Journal of Information Technology Theory and Application (JITTA), the Journal of the Association of Information Systems (JAIS) and MIS Quarterly (MISQ). Also mentioned is the creation of an annual conference, the International Conference on Design Science Research in Information Systems and Technology (DESRIST) that focuses on design science research.

The lack of a model to use in design science research was a constant concern. Pefers et al. address the problem and present a methodology for design science research [13]. The model presented consists of a sequence of six activities: problem identification and motivation, defining the goals of a solution, design and development, demonstration, evaluation, and communication.

Based on the work presented by Pefers et al., we present a brief description of each of the activities [13].

Step 1: Problem identification and motivation. Define the problem and justify the value of a solution. Frame the importance of the problem and identify weaknesses in current solutions.

Step 2: Define the objectives of a solution. How the problem should be solved. Define the general objectives, such as feasibility and performance, and specific objectives that a solution to the problem should include. In the definition we must consider what is possible and feasible. What methods, theories and technologies can help in defining the objectives.

Step 3: Create an artifact. Build a model, the application of a theory, methods, or technology that solves the problem.

Step 4: Demonstration. Prove that the artifact works by solving one or more instances of the problem.

Step 5: Evaluation. Observe and measure the results of the artifact in solving the problem. Compare objectives and observed results. Evaluate results and if necessary, repeat step 2 and/or step 3.

Step 6: Communication. Communicate and present the problem, its solution and usefulness. Communicate and publicize the novelty and effectiveness of the solution to the community. Evaluate solution and its usefulness and if necessary, repeat step2 and/or step3.

This seems to be a good path to adopt in the development of our work. In our study there are two groups of students and a small number. It does not seem fair to us to take two different approaches, that is, to have a test group and a control group, as in other methodological research approaches. We do not want to take this kind of approach, where one group is applying concepts, methods, and technological means to...
help the development and acquisition of knowledge, and another group with a more traditional approach of exposition and evaluation of the contents, strongly penalizing for the students’ goals.

4 The design scientific research methodology applied to initial programming problems

In this section, we address the DSRM methodology, applied to our case of teaching and learning strategies for initial programming, with the goal of improving the process of teaching and learning of initial programming and, consequently, making the vast majority of students able to acquire the skills required in programming.

4.1 Step 1: Problem identification and motivation

Our teaching and learning model consists of a face-to-face system where the fundamental contents of the learning of a programming language are covered, in our particular case the C programming language is used.

In our educational institution, a system of theoretical-practical classes, practices, and tutorial guidance is defined, throughout the first semester of the first year of the computer science course, in a total of 90 h, divided into three blocks of two hours per week.

The theoretical-practical lessons aim to give students the necessary knowledge to be able to identify and apply the rules of constructing computer programs to solve problems. In theoretical-practical classes we use simple, frontal teaching lessons with the aid of written materials and/or audiovisual aids, lasting approximately for one hour, usually consisting of an introduction to the subject, a presentation and examples of use, and a conclusion. We complement the information with a set of academic exercises, using learning by doing [25], a very successful method among beginners in programming. In these classes exercises are also proposed for individual resolution.

After studying the syntax and semantics of the programming language, as well as concepts for problem solving, students are ready to check it all out, in practice. In the practical classes an active methodology is used where students are encouraged to experiment, analyze different implementations and solutions, interact with peers, and investigate.

The tutorial classes aim to increase students’ involvement in educational activities, namely by planning and monitoring their learning process. However, most of these classes do not achieve the purpose for which they were created, such as identifying and analyzing individual problems and pointing out solutions. The tutorial classes have more of a function of creating a closeness and trust with the students. We find that students have difficulty verbalizing their difficulties. It is in these classes that we try to use other less conventional means, in a more informal way, approaching and applying a set of activities, such as: follow and give instruction, map design, paper folding, origami, memory transfer language and parson problems [26–29]. These activities aim to increase motivation, passion, beauty, joy, and awe for programming.

In our teaching model, we find that it is difficult to follow the evolution and work of the students. There are many reasons for this, such as students’ difficulty verbalizing their difficulties, or because they are shy or embarrassed to expose their doubts in front of other colleagues, or because they think they are insignificant problems and will expose themselves to ridicule, or even because of the high number of students to whom the teacher cannot give the necessary attention.

We found that some of the problems in implementation and problem solving are due to lack of knowledge of programming language syntax. They are due to the students’ difficulty in identifying and applying the most basic concepts of constructing a program, such as identifying and applying reserved words, identifiers, data types, instructions, and the basic structure of a program. Students are required to work continuously through the different phases of learning programming.

It is important for students to have continuous and immediate feedback on activities performed. Feedback is important for students to gauge their knowledge and to keep their motivation levels high. A student without feedback, or with a large time lag in getting feedback, is more easily demotivated. On the other hand, it is important for the teacher to also have objective feedback on student knowledge at every moment of the learning process. In our teaching and learning model this objectivity of continuous and immediate feedback is not clear.

Given the years of experience teaching initial programming, it is almost empirical to perceive the final result of each student as we see the results of the activities, doubts and difficulties of each student. However, this knowledge is only possible if there is a strong interaction between teacher and student, which in most cases does not exist, or is very difficult. The existence of tools and means can greatly help with this task.

In summary, we clearly identified the problems of our model of teaching and initial learning of programming, such as: difficulty in following the evolution and work of the students, small quantity of activities, and difficulty in responding quickly to the activities developed by the students.
4.2  Step 2: Define the objectives of a solution

In this model there needs to be continuous and immediate feedback. The student should receive feedback on the various activities he/she carries out and his/her progress in the learning model. On the other hand, the teacher must receive feedback on his or her students’ activities and progress. Only in this way will it be possible for the teacher to help the students, motivate and encourage for to more work or specific work.

From the above, we find it necessary to use technological means to help students and teachers in monitoring the learning process. To this end, we developed a technological prototype, an application called HTProgramming (Help to programming), to help address our problem.

The artifact to be developed should contain a set of activities guided by the teacher, covering the different phases of the initial learning process of C programming. This set of activities should allow the student to develop the activities at his/her own pace, without fear of making mistakes; it should have continuous and immediate feedback; and it must have a global assessment that allows the student and the teacher to assess their situation. To evaluate the student’s situation, we use the machine-learning (neural network) predictive model of student failure [30], with the objective of helping the teacher in this evaluation.

4.3  Step 3: Create an artifact

In this section, we describe the content of the HTProgramming application, the set of activities and methods that allow solving the problem. Figure 1 shows the general model of the HTProgramming application. The application is divided into two modules, the administrator module, and the student module. The application was developed in Java, with the Netbeans IDE. The desktop application interacts with a remote MySQL database, in a Hosting Smart Linux service. To use the application students must install it and also install a C language compiler, such as GCC (GNU Compiler Collection).

4.3.1  Artifact characterization—administrator module

The administrator module is used by the teacher to manage the entire teaching and learning process. It is in this module that the teacher creates and configures the activities to present to students. The teacher has at his/her disposal the feedback of the activities performed by the students. He/she also has access to the versions and attempts of the activities performed. The results of the student activities are used in the predictive neural network model of student failure, with the goal of predicting the student’s final result. Figure 2 shows the general appearance of the administrator module, where we can see a list with the identification of the students and their score.
4.3.2 Artifact characterization—student module

The student module is used by the student in his learning process. In Fig. 3, we can see the student module with the identification of the different areas and activities. The student performs a set of activities that will complement his learning process. By carrying out the activities, the student can practice, test, and evaluate his knowledge in the various areas of knowledge that make up his/her learning model. The student can carry out the activities at their own pace and without fear. For each activity the student automatically receives the grades obtained, suggestions for reading or reviewing, or even suggestions for new activities.

Our learning support artifact includes activities that best represent the needs and difficulties of our students. We start the construction of the student profile with the student’s characterization, collecting data such as age, area of previous studies, the final average of the high school, and the order of preference in the higher education admission forms, identified in Fig. 3 as area 6. We added to this characterization the test of spatial visualization skills, a feature strongly related to programming [31], area 7.

The set of activities directly related to the initial learning of C programming are the areas identified as numbers 4-Basic concepts, 5-Coding and 8-Parson problems.

The basic concepts are introductory activities such as data types, names and identifiers, identifying the structures of a program, and identifying errors. These types of activities are MCQ (Multiple Choice Question).

Parson Problems are programming instructions in which the student must select and order code instructions. According to [31–35] and proven in our work, learning and practicing using Parent Problems is a good exercise. Students find it interesting and are motivated to solve these types of problems. These types of activities are great for the early stage of learning programming because students do not make syntax mistakes, plus the possibility to do other types of activity like interpreting instructions and analyzing their set to form a logical solution.

In area 5-Coding, the student can perform coding activities. The coding activities are divided into four levels: basic, conditions, loops and advanced. In each level the student codes a solution to a proposed problem, in the IDE (Integrated Development Environment) that they prefer and submits his/her solution to the application. The application runs a set of verification tests and presents the result in the form of a numerical value on a scale of 0 to 20, the scale usually used for evaluation. For all activities whose result is lower than 12 points, or a value defined by the teacher, a message is automatically generated with a suggestion to review the activity, and suggestions to read or consult information. The messages can be consulted by the student in area 2-Messages.

In area 3-work done, depicted in Fig. 4, the student is presented with the activities done and their results (score). The overall score, student profile and results grouped by type of activity are also shown.

4.4 Step 4: Demonstration

The common problems, such as difficulty in following the evolution and work of the students, small quantity of activities, and difficulty in responding quickly to the activities developed by the students (immediate feedback) are identified in the course of step 1. We will demonstrate, in this step, that our solution solves the stated problems.
Concerning the problem of difficulty in following the evolution and work of the students, it is demonstrated through the characterization and explanation of the operation of the HTProgramming application, in step 3, that the teacher has at his/her disposal all the information of the activities performed by the student. The teacher can check and monitor the results of each student’s activities, as well as the number of attempts, and the topics covered. The teacher also has at his/her disposal a tool for predicting student success. This prediction model is based on the training of a neural network from the results of previous years. Therefore, over time we can improve the prediction model.

Regarding the problem of the small quantity of activities, so far, our focus has been on application development, testing and correction, so the number of activities has been underestimated. At any time, the teacher can easily include more activities.

All activities have immediate feedback, to respond to the problem identified as difficulty in responding quickly to the activities developed by the students.

4.5 Step 5: Evaluation

The evaluation of our artifact was done through student observation and comments on the use of the HTProgramming application, with the results of an opinion survey on the use of the HTProgramming application, and, with the analysis and observation of the prediction of student failure at a given moment.

From the students’ observation and comments, we found that Parson Problems activities should be given special attention. We found that students like this type of activity. However, the students made some comments about their assessment results. In fact, the evaluation was done through line-by-line comparison between the student’s program and the solution. Either we take extreme care in the program to be used, or there can be several solutions. In the example shown in Fig. 5, we present one such example where all solutions produce the same output. For the example shown other combinations could still be considered.

Having analyzed the problem, we choose to solve with a test case to the solution proposed by the student. This allows us to verify that the output is correct. Regardless of the order the solution will always have the expected output.

4.5.1 Opinion survey

To know the students’ opinion on the use of the HTProgramming application and to what extent it contributed to their learning process, we developed an opinion survey. In this survey, students were asked to evaluate, on a scale of 1 (unimportant) to 5 (very important), the importance of each

Fig. 4 Student module, area of work done example
of the activities in their learning process. The results of
the survey are analyzed in Table 3.

Twenty-four students responded to the survey. As we
can see in Table 3, the results are very good. Most students
rate the contribution of the basic concepts, parson problems
and coding activities to their learning process as important
or very important, 91.7%, 83.3% and 91.7%, respectively.
Regarding the general opinion of the importance of the
application in their learning process, 91.7% of the students
classified it as important or very important.

Some of the students also made comments such as: "Useful
and necessary for basic student learning", "Quite useful
for those starting to study C from zero", "Useful", "Fundamen
tal", "These are exercises that allow students to expand
their knowledge."

4.5.2 Analysis of the prediction of student failure

In Table 4, we present the result of the application of the
predictive neural network model of student failure.

We present the summary of Table 4, in Table 5, in num-
ber of students and their percentage. We considered 26 stu-
dents who actively used the HTProgramming application,
as a result of the predictive neural network model of student
failure we found that 11 of the students (42.3%) probably
fail. By observation and identified students, the results seem
adequate. Based on our experience and knowledge in class-
room observation.

We can compare these results with the values obtained in
previous academic years, see Table 6. The result prediction
for this academic year 2021–2022, presents a realistic value,
similar to the result obtained in the previous year 2020–2021
where the HTProgramming application was used.

It should be noted that the current academic year
2021–2022, at the time of writing this paper, has not yet
ended, so the values presented are based only on the use
of the HTProgramming application and do not take into
account the final results and the number of students who
have undergone assessment, parameters used in the calcu-
lation of the percentage of the other years.

4.6 Step 6: Communication

The last step in DSRM is to communicate and present the
problem, its solution and utility. The elaboration of this
paper is one of the ways to accomplish this goal. This work
has evolved over the past few years, and has only been pos-
sible with an organized data collection and controlled by a
technological tool, such as HTProgramming.

We started in 2016, with "ne-Course for Learning Pro-
gramming", [26], where we used and reflected on the use
of a set of activities, without the use of electronic means,
ne-course (no electronic course), for learning programming,
develop their skills in computational thinking.

The results of the ne-course for learning programming
activities indicated that following and giving instructions
are a good way to develop their skills in computational

| Activities       | 1 | 2 | 3 | [1, 3] | 4 | 5 | [4, 5] |
|------------------|---|---|---|-------|---|---|-------|
| Basic concepts   | 0 | 0 | 2 | 2     | 15| 7 | 22    | 91.7% |
| Parson problems  | 1 | 0 | 3 | 4     | 16.7% | 9 | 11 | 20 | 83.3% |
| Coding           | 0 | 0 | 2 | 2     | 8.3% | 9 | 13 | 22 | 91.7% |
| HTProgramming    | 0 | 0 | 2 | 2     | 8.3% | 12| 10 | 22 | 91.7% |
thinking, and consequently, to improve the students’ results. This study, in 2017, originated the paper "Improving Computational Thinking Using Follow and Give Instructions", in [27].

After the latter paper, in 2018 we felt the need for the development of a data collection and visualization tool. We started the construction of our technological artifact to organize and concentrate the collection of the data produced by the students in Making the analogy with the fantastic world of games, where the player can build and improve his/her characteristics through activities and challenges. In the same way we built the student’s profile, through the realization of a set of activities, related to initial programming learning. From this idea resulted the work developed in [28], entitled “Building Skills in Introductory Programming”.

Based on the identified profile (skill score) of each student, we consider it possible to calculate a threshold over which we can predict student failure. The work developed in “Predicting Student Failure in an Introductory Programming Course with Multiple Back-Propagation”, [30] in 2019, describes all the work done to test and verify the proposed machine-learning (neural network) predictive model of student failure based on the student profile, where the main goal is early identification of potential problems and immediate response.

In the following years we focused our work on the development and improvement of the tool to complement students and help teachers in the process of teaching and learning of initial programming, in [29, 36, 37].

This period also witnessed a phenomenon that nobody was expecting, the pandemic of COVID-19. This period affected the development of our work. The pandemic had a great impact on all of us, especially on how the whole teaching and learning system had to adapt [38]. There were major disruptions to the normal functioning of the education system, such as large numbers of students not attending classes, periods of online and face-to-face time, online assessments, not to mention all the family and social factors involved.

Regarding the solution and its utility, as demonstrated in step 5, all the results seem to indicate that we are on the right track. The students consider the tool to be very useful and complementary to their learning process of programming, and they like to use it. Naturally, the application needs revision and improvements in its functioning. At this point the DSRM methodology is working, and according to our evaluation it is necessary to review step 2 and/or step 3. We point out as some of these problems as the initial time to access the tool and improve the way we evaluate the coding activities.

The results obtained for our predictive neural network model of student failure indicate good results. However, with larger amounts of data, and with a more stable set of variables/activities, our machine-learning (neural network)
predictive model of student failure based on the student profile, should produce significant improvements.

5 Conclusion

In this work, we decided to use an emerging research methodology, namely Design Science Research Methodology addressing the problem of teaching and learning initial programming applied to our particular case of a higher education institution, with a study group with very special characteristics. We think that many researchers see themselves in the themes and problems described in this work. We are of the opinion that this methodology is the best suited for a research work.

The indicators on the approach taken to our model of teaching and learning of initial programming, complemented with the technological tool developed, are very good. Therefore, the problems identified will be minimized over time, and consequently, the failure and dropout rates caused by programming will be minimized.

Despite the difficulties caused by the pandemic, perhaps a new opportunity has arisen for the use of the tool presented. Although the tool was not designed and built with the purpose of working in distance learning, the characteristics of the HTProgramming application are naturally excellent to complement and use in a distance learning environment. In conclusion, but of special importance, it is crucial that the HTProgramming application is appealing, and uses the latest design and technology standards, so that students will use the tool.

References

1. Zhao, H., Li, M., Lin, T., Wang, R., Wu, Z.: Prolog2vec: detecting novices’ difficulty in programming using deep learning. IEEE Access 9, 53243–53254 (2021). https://doi.org/10.1109/ACCESS.2021.3067505

2. McBroom, J., Paassen, B., Jeffries, B., Koprinska, I., Yacef, K.: Progress networks as a tool for analysing student programming difficulties. In: Australasian Computing Education Conference. ACE ’21, pp. 158–167. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3441636.3442366

3. López-Pernas, S., Saqr, M., Viberg, O.: Putting it all together: combining learning analytics methods and data sources to understand students’ approaches to learning programming. Sustainability (2021). https://doi.org/10.3390/su13094825

4. Ömer, U., Farooq, M.S., Abid, A.: Introductory programming course: review and future implications. PeerJ Comput. Sci. (2021). https://doi.org/10.7717/peerj-cs.647

5. Song, D., Hong, H., Oh, E.Y.: Applying computational analysis of novice learners’ computer programming patterns to reveal self-regulated learning, computational thinking, and learning performance. Comput. Human Behav. 120, 106746 (2021). https://doi.org/10.1016/j.chb.2021.106746

6. Bennedsen, J., Caspersen, M.E.: Failure rates in introductory programming. ACM SIGCSE Bull. 39(2), 32–36 (2007). https://doi.org/10.1145/1272848.1272879

7. Bennedsen, J., Caspersen, M.E.: Failure rates in introductory programming: 12 years later. ACM Inroads 10(2), 30–36 (2019). https://doi.org/10.1145/3324888

8. Watson, C., Li, F.W.B.: Failure Rates in Introductory Programming Revisited. In: Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education. ITiCSE ’14, pp. 39–44. Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2591708.2591749

9. Jenkins, T.: On the difficulty of learning to program. Language 4, 53–58 (2002). https://doi.org/10.1109/ISIT.2013.6620675

10. García-Peñalvo, F.J., Corell, A., Abella-García, V., Grande-De-Prado, M.: Online assessment in higher education in the time of covid-19. Education in the Knowledge Society (EKS) 21 (2020). https://doi.org/10.14201/eks.23013

11. García-Peñalvo, F.J., Corell, A., Abella-García, V., Grande-De-Prado, M.: Recommendations for mandatory online assessment in higher education during the covid-19 pandemic. Springer Nature, 85–98 (2021). https://doi.org/10.1007/978-981-15-7869-4_6

12. Knopik, T., Oszwa, U.: E-cooperative problem solving as a strategy for learning mathematics during the covid-19 pandemic. Education in the Knowledge Society (EKS) 22 (2021). https://doi.org/10.14201/eks.25176

13. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. J. Manag. Inf. Syst. 24(3), 45–77 (2007). https://doi.org/10.2753/MIS0742-122240302

14. Hoc, J.-M., Nguyen-Xuan, A.: Language Semantics, Mental Models and Analogy. In: Hoc, J.-M., Green, T.R.G., Samurçay, R., & Gilmore, D.J. (Eds.), Psychology of Programming, 139–156 (1990)

15. Ramírez-Montoya, M.-S. (ed.): Handbook of Research on Driving STEM Learning With Educational Technologies. Advances in Educational Technologies and Instructional Design. IGI Global, ?? (2017). https://doi.org/10.4018/978-1-5225-2026-9

16. Bennedsen, J.: Teaching and learning introductory programming: a model-based approach. Doctoral thesis, University of Oslo (2008)

17. Jenkins, T.: On the difficulty of learning to program. Language 4, 53–58 (2002). https://doi.org/10.1109/ISIT.2013.6620675

18. Bergin, S., Reilly, R.: Programming: factors that Influence Success. In: SIGCSE ’05: Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, St. Louis, Missouri, United States, pp. 411–415 (2005)

19. Rojas-López, A., García-Peñalvo, F.J.: Personalized education for programming in higher education. Innovative Trends in Flipped Teaching and Adaptive Learning, 203–227 (2019). https://doi.org/10.4018/978-1-5225-8142-0.ch0010

20. Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCarrtn, R., Moström, J.E., Sanders, K., Seppälä, O., Simon, B., Thomas, L.: A multi-national study of reading and tracing skills in novice programmers. SIGCSE Bull. 36(4), 119–150 (2004). https://doi.org/10.1145/1041624.1041673

21. Choi, J., Nazareth, D.L., Jain, H.K.: Implementing service-oriented architecture in organizations. J. Manag. Inf. Syst. 26(4), 253–286 (2010). https://doi.org/10.2753/MIS0742-1222260409

22. Geerts, G.L.: A design science research methodology and its application to accounting information systems research. Int. J. Account. Inf. Syst. 12(2), 142–151 (2011). https://doi.org/10.1016/j.ijaccinf.2011.02.004

23. March, S.T., Smith, G.F.: Design and natural science research on information technology. Decis. Support Syst. 15(4), 251–266 (1995). https://doi.org/10.1016/0167-9236(94)00041-2
24. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. Manag. Inf. Syst. Q. 28, 75 (2004)
25. Vihavainen, A., Paksula, M., Luukkainen, M.: Extreme apprenticeship method in teaching programming for beginners. Proceedings of the 42nd ACM technical symposium on Computer science education—SIGCSE ’11, 93 (2011). https://doi.org/10.1145/1953163.1953196
26. Figueiredo, J., Gomes, N., García-Peñalvo, F.J.: Ne-course for learning programming. In: Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality—TEEM’16, pp. 549–553. ACM Press, New York, New York, USA (2016). https://doi.org/10.1145/3012430.3012572
27. Figueiredo, J.A.Q.: How to improve computational thinking: a case study. Educ. Knowl. Soc. (EKS) 18(4), 35–51 (2017). https://doi.org/10.14201/eks20171843551
28. Figueiredo, J., García-Peñalvo, F.J.: Building Skills in Introductory Programming. In: Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality—TEEM’18, pp. 46–50. ACM Press, New York, New York, USA (2018). https://doi.org/10.1145/3284179.3284190
29. Figueiredo, J., García-peñalvo, F.J.: Increasing student motivation in computer programming with gamification. In: 2020 IEEE Global Engineering Education Conference (EDUCON), Porto, pp. 997–1000 (2020). 978-1-7281-0930-5/20
30. Figueiredo, J., Lopes, N., García-Peñalvo, F.J.: Predicting Student Failure in an Introductory Programming Course with Multiple Back-Propagation. In: Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality - TEEM’19, pp. 44–49. ACM Press, New York, New York, USA (2019). https://doi.org/10.1145/3362789.3362925
31. Jaeger, A.J.: What Does the Punched Holes Task Measure? Doctoral thesis, University of Illinois at Chicago (2015)
32. Denny, P., Luxton-Reilly, A., Simon, B.: Evaluating a new exam question: Parsons problems. Proceedings of the fourth international workshop on Computing education research, 113–124 (2008). https://doi.org/10.1000/3362789.3362925
33. Ericson, B.J.: Adaptive Parsons Problems with Discourse Rules. Icer ’14, 145–146 (2014). https://doi.org/10.1145/2632320.2632324
34. Morrison, B.B., Margulieux, L.E., Ericson, B., Guzdial, M.: Sub-goals help students solve parsons problems. Proceedings of the 47th ACM Technical Symposium on Computing Science Education, 42–47 (2016). https://doi.org/10.1145/2839509.2844617
35. Du, Y., Luxton-Reilly, A., Denny, P.: A review of research on Parsons problems. In: ACE 2020—Proceedings of the 22nd Australasian Computing Education Conference. Held in Conjunction with Australasian Computer Science Week, vol. 8, pp. 195–202. Association for Computing Machinery, Inc, ??? (2020). https://doi.org/10.1145/3373165.3373187
36. Figueiredo, J., García-Peñalvo, F.J.: Intelligent Tutoring Systems Approach to Introductory Programming Courses. In: Eighth International Conference on Technological Ecosystems for Enhancing Multiculturality, TEEM’20, pp. 34–39. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3434780.3436614
37. Figueiredo, J.A.Q., García-Peñalvo, F.J.: A Tool Help for Introductory Programming Courses. ACM, New York, NY, USA, October 26–29, 2021, Barcelona, Spain (2021). https://doi.org/10.1145/3486011.3486413
38. García-Peñalvo, F.J., Corell, A., Rivero-Ortega, R., Rodríguez-Conde, M.J., Rodríguez-García, N.: Impact of the covid-19 on higher education: An experience-based approach. In: In Information Technology Trends for a Global and Interdisciplinary Research Community, F.J. García-Peñalvo Ed. IGI Global, Hershey, PA, USA, pp. 1–18 (2021). https://doi.org/10.4018/978-1-7998-4156-2.ch001
Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.