Pedagogical Content for Professors of Introductory Programming Courses

Yorah Bosse*
University of Sao Paulo / Federal University of Mato Grosso do Sul
Sao Paulo / Ponta Pora, Brazil
yorah@ime.usp.br

David Redmiles
University of California
Irvine, USA
redmiles@ics.uci.edu

Marco A. Gerosa
Northern Arizona University / University of Sao Paulo
Flagstaff / Sao Paulo, USA / Brazil
marco.gerosa@nau.edu

ABSTRACT

Teaching introductory programming requires knowledge of both content and pedagogy. Pedagogy includes understanding the typical difficulties students face as they learn, as well as recognizing didactic strategies professors can use to help students to overcome these difficulties. Our research aims to improve the pedagogical knowledge instructors have to teach introductory programming courses, especially those new in this area. We conducted 16 semi-structured interviews with instructors who teach introductory programming courses and collected diaries filled by 110 students during their studies. Qualitative analysis of this data revealed a set of difficulties students faced when learning programming basics and a set of didactic strategies professors use to mitigate them. The results were reviewed by senior instructors in order to confirm them and by junior instructors to verify the importance of this material from their perspective. The main contribution of our paper is a set of difficulties faced by students learning programming, a classification of the most harmful challenges, and the didactic strategies usually used to teach and avoid them. Thus, we provide the basis for the pedagogical content necessary to junior and senior professors planning introductory programming courses.

KEYWORDS

pedagogical content; learning to program; novice learners; barriers to learning; introductory programming; strategies; computational thinking

ACM Reference Format:
Yorah Bosse, David Redmiles, and Marco A. Gerosa. 2019. Pedagogical Content for Professors of Introductory Programming Courses. In *Innovation and Technology in Computer Science Education (ITiCSE '19), July 15–17, 2019, Aberdeen, Scotland UK*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3304221.3319776

*The first author is financially supported by the UFMS - Federal University of Mato Grosso do Sul.

ITiCSE '19, July 15–17, 2019, Aberdeen, Scotland UK

ACM ISBN 978-1-4503-6301-3/19/07…$15.00
https://doi.org/10.1145/3304221.3319776

1 INTRODUCTION

Learners of programming around the world who are writing, running, and debugging code in introductory programming courses (CS1) inevitably run into frustrating hurdles [19], including understanding programming' basic concepts [2, 15, 16, 18, 23, 27, 31]. Though people learn to program in different ways, such as with graphical programming blocks, text, or both [17, 21, 31], and for different reasons [4], few consider it an easy task [1, 15, 18]. This can be perceived through failure and dropout rates, which reach about 28% among undergraduate students [3, 5].

Preparing new generations of professional and casual developers is a big challenge. If instructors are to overcome the difficulties students face, quell their frustrations, and help them achieve their goals, they need both knowledge of content and pedagogy [25, 29, 30]. Pedagogical knowledge includes three types of information: difficulties faced by students as they attempt to learn the content, typical paths students need to traverse to understand the content, and a set of strategies instructors can use in classes to avoid the difficulties [6].

Our research was developed to help instructors of CS1 prepare their classes and improve their teaching strategies so they can support students learning to program in C or Python. For that, we focus on two research questions:

RQ1 - What are the difficulties students face when learning how to program?

RQ2 - What didactic strategies do instructors use to overcome these difficulties?

Experienced teachers acquire knowledge of content and pedagogy and know the types of difficulties students have and how to circumvent them [6]. However, research has shown that expertise can lead to over-confidence in one's knowledge, meaning teachers can overlook details when teaching a topic they know well [13]. Even so, that knowledge can help other instructors in the task of how, for instance, to better organize the content for students [6]. Taking this into account, we collected qualitative data from two sources of data: semi-structured interviews with 16 professors from the University of Sao Paulo - USP - Brazil and diaries maintained by 110 students when taking CS1 at the same university. This data was analyzed using Grounded Theory techniques. The findings were reviewed by experienced instructors to confirm the results, in addition, it was collected the inexperienced instructors' perspectives about the importance of this material. The main contribution of this paper is a set of difficulties faced by students learning programming, a classification of the most harmful challenges, and the didactic strategies usually used to teach and avoid them. This material can

guide instructors, especially inexperienced ones, to prepare their classes, helping them improve their teaching and students' learning.

2 RELATED WORK

We organize our literature review according to the difficulties students face, and then the methods and strategies instructors use to teach the content.

Difficulties: The content covered in programming courses can be difficult to understand, such as pointers and abstract data types [22, 24]. According to Mhashi and Alakeel [22], the most difficult programming concepts are repetition, recursion, lists, pointers, passing parameters, abstract data types, and the use of libraries. Some of these difficulties also appear in the research from Sevella and Lee [28]. They further show that students confuse 'For' and 'While' concepts in C and find it difficult to use functions, sometimes developing programs without ever using this feature. In addition, they point out difficulties with variables: students forget to declare them or assign wrong types. They also note difficulties with selection structures, such as writing conditional statements and nested selections. They found no difficulty with understanding lists. However, for the instructors responding to the questionnaire by Piteira and Costa [24], lists are among the most difficult topic, aside from pointers, structured data types, error handling, and parameters. Pointers and passing parameter by reference are also considered two of the most difficult topics by Lahtinen and colleagues [1]. Piteira and Costa [24] add that other difficulties for students include the use of language libraries and abstract data types. Regarding syntax, Hristova and colleagues [27] report the wrong use of '=' vs '==', '&&' vs '&', '||' vs '|', different amounts of opening and closing parentheses, brackets and quotation marks, wrong separators in 'For' loops, and so on.

Methods and strategies used to teach basic programming content: Traditional teaching methods can be inadequate for many students: teaching is not personalized, and dynamic concepts are presented with static material [14]. In contrast, Deek and colleagues [20] have developed an approach to programming courses that helps students learn concepts and their uses. In this approach, language characteristics were gradually introduced to students only in the context of specific problems. This has been shown to have positive effects on student achievement and confidence in programming. There are also programming tools that allow students to overcome the barriers to learning to program themselves [7, 17, 21], and to give automatic feedback to students about their codes [11, 12], not only feedback on syntax questions, but also on program semantic issues [26]. Another approach for increasing students' knowledge and developing their abilities is to increase the number of exercises, carefully selecting and solving them afterwards [22]. While visualization techniques to show complex concepts can help, visualizations are most often used for algorithm animation, with less emphasis on illustrating the basic structure of programs and their execution [22].

While the literature tends to focus on the content students consider difficult, we seek to identify a set of difficulties students face in CS1 in order to organize a set of strategies that instructors may use to avoid them and improve learning.

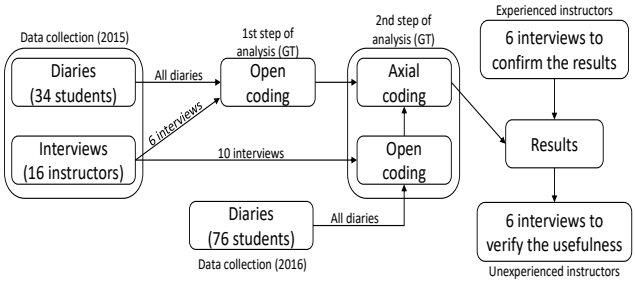

Figure 1: Scheme of the methodology.

Table 1: Instructors' data.

Professor ID	Years of Experience	Gender
P1 - P3	More than 40 years	1 F and 2 M
P4 - P5	31 to 40 years	2 M
P6 - P7	21 to 30 years	2 M
P8 - P12	11 to 20 years	2 F and 3 M
P13 - P16	1 to 10 years	4 M

3 METHODOLOGY

We conducted semi-structured interviews with 16 instructors and asked students to maintain diaries. Figure 1 shows an overview of our methodology, which is detailed below.

3.1 Data Collection

This study is motivated by a previous study conducted at the University of Sao Paulo (Brazil) from 2010 to 2014, based on data from approximately 18,500 students from various majors who enrolled in 29 CS1 courses [5]. The results showed that approximately 30% of these registrations resulted in failures and dropouts, which corroborates results obtained by Bennedsen and Caspersen [3].

In the present study, we chose to further investigate these courses and topic-related issues. We conducted interviews with instructors to delve into particulars, focusing on the topics to gain a perspective of the students' learning experiences and instructors' teaching experiences [10]. Sixteen instructors from the Computer Science Department at the University of Sao Paulo were selected for interviews. About 30 instructors from the department had taught this course at least once. The first 6 instructors interviewed were those who were teaching introductory programming for the six classes whose students were keeping diaries about the difficulties in learning how to program. The other 10 were randomly selected. Each instructor was identified with an ID comprising a P, followed by a number from 1 to 16. Table 1 shows the level of experience and gender of the subjects.

The interviews began with a very general question: 'In your view, what are the difficulties faced by students in CS1?' The instructor was then guided to talk about the following introductory topics: variables; assignment command; input and output functions; arithmetic, relational (<, >, =, ...) and logical (and, or not) expressions; selection (if...else) and repetition structures (while, for, ...); string manipulation; uni- and multi-dimensional arrays; structured data;

Table 2: Students' data.

Age	Total	%	TACB	Total	%
Not informed	1	1%	0	81	74%
15 - 24	84	76%	1	17	15%
25 - 34	17	15%	2	6	5%
35 - 44	6	5%	3	4	4%
45 - 54	1	1%	4	1	1%
55 - 64	1	1%	5	1	1%

functions; and pointers. Most of the interviews took about half an hour, but some lasted for up to an hour.

We arranged for students to fill out diaries during their studies, keeping us informed about their difficulties as they emerged. We collected data from 6 different classes in 2015 (34 students) and another 6 in 2016 (76 students), yielding a total of 110 student diaries. The diaries were arranged in shared documents (Google Docs). We asked the students to include: code snippets, especially the wrong ones; how they fixed the errors; and the doubts they had about learning the topics. To clarify possible misunderstandings about the text written by the students, the researchers sometimes posed questions in the diaries. Each student was also identified with an ID, composed by an S, followed by a number from 1 to 110. Table 2 shows the total and percentage of students by age range and the number of times they attended the course before (TACB).

3.2 Data Analysis

For the analysis of the data, we used Grounded Theory (GT) techniques, as described by Strauss and Corbin [9]. During the analysis, concepts, categories, and subcategories emerged. According to Corbin and Strauss [8], "the procedures of grounded theory are designed to develop a well-integrated set of concepts that provide a thorough theoretical explanation of social phenomena under study." The groupings of these concepts into a higher degree of abstraction are called "categories" [9].

The data was read and analyzed, and relevant information was marked with a tag, thus characterizing a concept. Grouping these concepts, categories emerged. We primarily focused on two of them: 'Difficulties with the Topics' and 'Didactic Strategies.' After that, all text highlighted with codes was read again to identify difficulties, strategies, and connections among both. To confirm the results obtained with the data analysis, the results were presented to six experienced instructors to validate the findings. Then, six other instructors were interviewed, all inexperienced with the teaching of programming, in order to verify their opinions about the importance of the results for preparing their classes.

4 FINDINGS AND DISCUSSION

In this section, we present the results obtained in two categories defined in the data analysis, Difficulties and Strategies, answering the two research questions that guided our study.

RQ1 - What are the difficulties students face when learning how to program?

The difficulties presented below are divided by topics addressed in the courses, which are listed in the methodology section. Figure 2 presents the set of difficulties reported by instructors and students, which we classified in three levels. The first level comprises difficulties considered the most impactful to learning programming, which are marked with two stars. The intermediate level is marked with a star, and the last level has no marking. After each difficulty appears the number X of times they are cited by instructors 'p' and the number Y of times cited by students 's': XpYs. The 16 interviewed instructors commented mainly on C and Python; most of the difficulties listed in the diagram are related to the C language.

We found difficulties related to 10 topics taught in CS1, namely: Variables, Expressions, Input/Output Functions, Selection and Repetition Structures, Uni- and Multidimensional Array, Function, Pointer, and String. One of the difficulties of the first level of classification is **initializing the variables**, as shown in the topic Variable of the diagram. Many students do not know if it is necessary to initialize and with what value: "*When compiling everything was fine, however, the final result (max) has a gigantic number ... problem tidy, I had not put the initial value of the variable max...*" - S18. In addition, the **difference in the treatment of variables in mathematics and programming** can generate doubt, as explained by P8: "*Then there are 'x = x + 1' and these students experiencing difficulty do not understand if you read as in math, 'x is equal to x + 1,' (student replies) that 'it is impossible for x to be equal to x + 1'.*"

Working with Expressions, the result shows difficulties related to **detecting the wrong result due to data type problems**: "*... when it is a numerical operation, it is difficult for them to perceive the error. So, for example, an algorithm to calculate, make an average, ... yields zero (as a result) and sometimes the student realizes that they used integer rather than float*" - P7. Another source of difficulty is **writing logical expressions**. According to P11, "*...something that exists in natural language and that normally does not exist in programming language is something like 'x > 3 and < 5' and in computing I cannot write like this, you have to write 'x > 3 and x < 5', then they write it wrong, they write 'x > 3 and < 5'.*" The students find it difficult to write logical expressions in the syntax of the programming language, which is usually different from that used in mathematics and natural language. And, as already cited by Hristova and colleagues [27], there is **confusion between '=' and '=='**. According to S34, this is a constant error: "*I wrote the program and noticed some difficulties I constantly face such as ... whether to use = or == within the Selections or Repetition Structures.*"

Regarding Selection Structures, the most complicated task is working with **nested selections** with a set of pairs 'IF..ELSE': "*When I had nesting, one selection within another, that caused a lot of confusion*" - P15. Similarly, the most complicated Repetition Structure to learn is the creation and use of **nested repetition** structures, that is, two or more structures of repetition, with one inside the other. A classic example of using nested repetitions is the manipulation of multidimensional arrays. Other difficulties include the creation of the **stopping condition** and manipulating the **control variable**. According to P15, students begin to use the control variable without even initializing it, "*... they use a variable in the stop condition before assigning an initial value to this variable,*" and then they misuse the increment or decrement operators so that it satisfies the desired stop condition: "*... their challenge to generate a repetition structure was to understand the concept of a stop condition,*

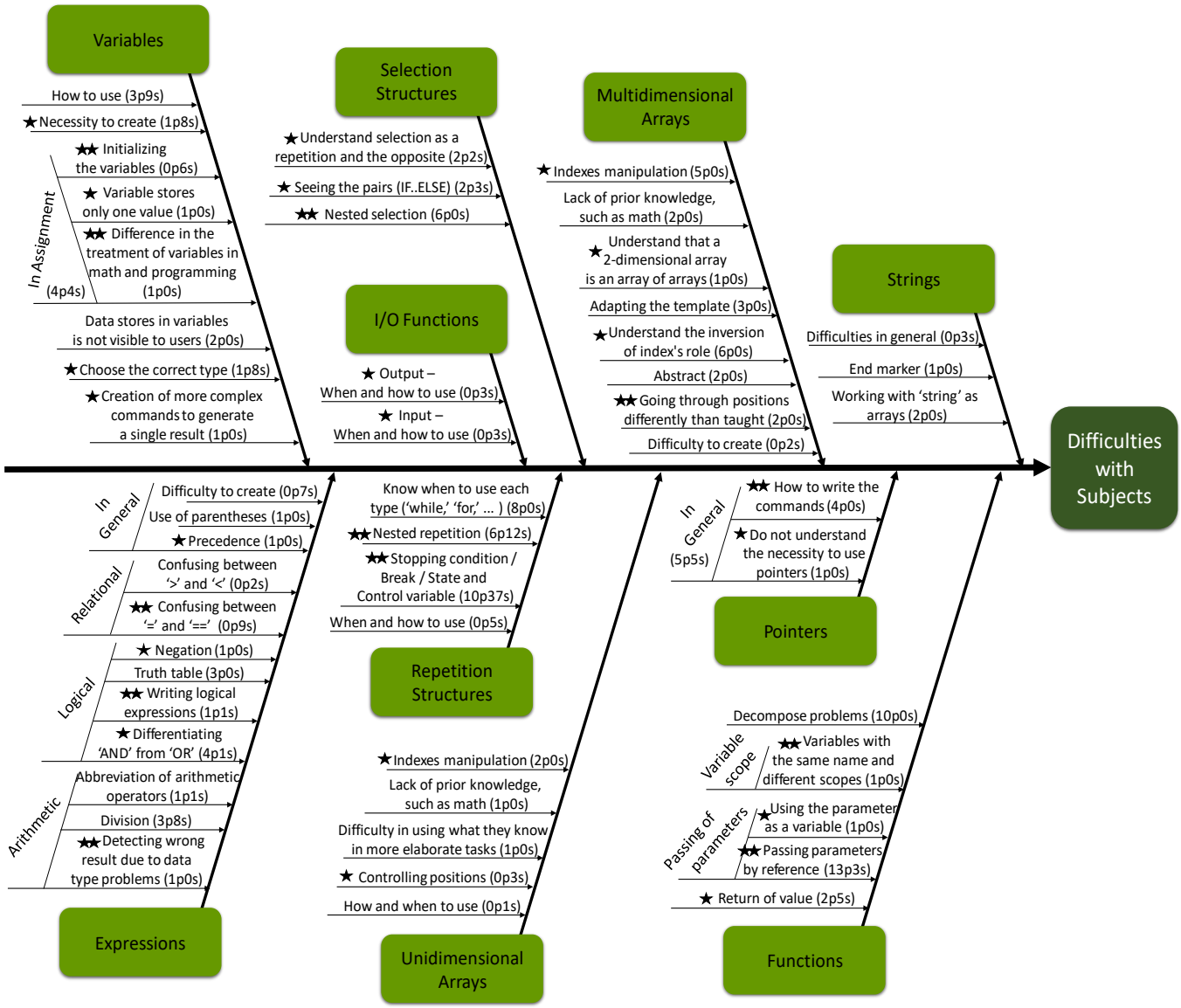

Figure 2: Diagram showing the list of difficulties per topics taught in introductory programming courses.

how to create a stop condition, and how to make an increment of a variable that satisfies this condition." Manipulating variables is also a task necessary for working with Multidimensional Arrays, where **going through positions differently than taught** is a source of difficulty, as exemplified by P14: "*... if we use a 2-dimensional array, for example, to represent a game board and ask to move diagonally, or backward, ... then things start to get confusing. Especially if the 'i' and 'j' (variables used to indicate the position in the row and column) that are there, begin to appear in a switched order.*"

Finally, for P4, when teaching Pointers, "*the trouble starts to get bigger.*" Pointers are cited by many researchers as one of the most difficult topics [22, 24, 28], and the results of this study confirmed this, as can be observed in the comment made by P14: "*But of the content of the course, I think the most difficult thing was for them to understand pointers.*" P13 reports that the students "*end up learning by heart,*" and adds "*I think they can understand, closer to the end of the second semester,*" that is, only in the subsequent course. Many even avoid teaching pointers: "*We try to avoid talking about pointers too*" - P10. Others end up showing them only because they are necessary to pass parameters by reference, and the main difficulty is **how to write the commands** using pointers. Working with Functions, besides the difficulty with **passing parameters by reference**, students confuse **variables with the same name and different scopes**.

RQ2 - What didactic strategies do instructors use to overcome these difficulties?

During the interviews, instructors mentioned many strategies they used to teach the topics and to mitigate difficulties students

Strat ID	Times cited	Strategy Name	Main difficulties that it helps
S1	16	Using visual resources	Variables. Nested repetition. Function. Uni and multidimensional array. General
S2	10	Using 'pseudocode and programming language' as a cycle for each new concept	General
S3	9	Solving together with students in class	General
S4	8	Explain using problems and similar exercises	General
S5	8	Omit some concepts and language details	Pointer - how to write the commands. Function - passing parameters by reference. Programming language - syntax issues. General
S6	8	Using "recipes"	Programming language - syntax issues. Multidimensional array - indexes manipulation and going through positions differently than taught. Pointer - do not understand the necessity to use pointers. Function – return of value. General
S7	7	Step-by-step execution	Function – variable scope. Repetition structure - stopping condition and manipulation of the control variable. General
S8	6	Making an analogy with known concepts / objects	Variables. Expressions. Function. Selection structure. General
S9	6	Intensive practice	General
S10	5	Programming on the projector screen	General

Table 3: Strategy used to avoid or minimize the difficulties faced by students to learn how to program.

face. In this paper, we present the strategies mentioned 5 times or more in the interviews. The third column of Table 3 reports the topics or difficulties linked to the strategies. If the term General is used in this column, this means that this strategy was quoted at least once in each of the topics covered.

Using visual resources (S1). Abstraction is one of the major problems of the course. It is difficult for the novice student to visualize where the data is, how it is processed, or how the commands work: "*...there is a difficulty in working with abstract concepts...*" - P9 and P15 remembered "*...classes that at first did not work, you had to have something visual too*" and cited an example "*Variable declaration needs to be visual ... explaining that declaring a variable will occupy space in memory, this abstraction is a bit difficult.*" The 'visuals'can be drawings: "*...I draw the memory...*" - P9 or made using features like arrows that represent the input and output of the data in a function, for example: "*I always do in a graphic way the inputs and outputs, with arrows*" - P8. Another form of visual presentation is "*...animations, if I needed to show that visually...*" - P15 and adds that "*...there are many visual resources on the internet....*"

Using 'pseudocode and programming language' as a cycle for each new concept (S2). Few instructors begin the course with pseudocode before progressing to a programming language, "*With the algorithm in pseudo-language, I did not see much success ... having two simultaneous languages was not going to be as effective*" - P16. Most of the interviewed instructors use pseudocode only to teach the concept and then immediately show the same concept in the programming language: "*... I give the algorithmic concept and after this, I go to the program ... I do it like this, coming and going*" - P9. According to P11, "*The programming language is a tool to teach algorithms and data structures*" and the pseudocode or pseudo language is quickly used, without many rules, only to explain the concepts.

Solving together with students in class (S3). This is another highly used strategy: to "*solve and write problems together with the students ... the idea is to let them participate and they try to develop the program together. Often times, I show the program even knowing that it is wrong and we try to find (the mistake)*" - P3. Many instructors reported using this strategy and claim it works because students participate and can see step-by-step how the program is developed.

Explain using problems and similar exercises (S4). To introduce and show the usefulness of the concepts that need to be taught in the course, one strategy is to use problems: "*I usually start explaining a content by describing a problem*" - P16. For P10, it is a way to motivate students, "*...we try to motivate by using some problems,*" because the student sees the concept's usefulness in practice and learns how it should be used. Between presenting the first problem to the student and more complex exercises, P16 advises "*...also bring to class exercises that are similar but do not solve exactly that problem.*" In this way, the student can practice what was taught, and then use it in other situations.

Omit some concepts and language details (S5). Details of the coding language can hinder the development of the code. If the instructor omits some details, they may improve initial learning: "*The idea was to abstract the language syntax somewhat*" - P15. P5 also said "*...if I have to teach in C, I do not explain an input command. I even joke: 'Look, God told you to do it like this.'*," because if you explain every detail of the syntax, the content does progress. Concepts are also omitted at times when they are not indispensable. An example of this was offered by P1, with the following statement: "*Passing parameter is also all by pointer, I do not need to speak of pointers, they do the exercise without problems ... they are hidden from them, these concepts, but they are in fact using (them).*"

Using "recipes" (S6). In content that is considered more complicated, as is the case of passing parameters by reference using pointers,: "*When we teach with C we make the passage by reference, but we provide a 'recipe', so it is (I use) '&' when I call the function and within the function '*'.* " - P12. In addition, to avoid language syntax explanations often unnecessary for the moment, such as 'scanf' and 'printf' in the C programming language, instructors use a kind of 'recipe;' that is, a step-by-step instruction of how something should be done.

Step-by-step execution (S7). Another frequently used technique is step-by-step execution, so that the student sees what is happening in the program, which also helps them find existing errors in the code. This execution is done on the board or even using the projector. Everything that happens, from change of values in the variables to the output of a repetition or change of variable scope, is line-by-line shown and analyzed during this exercise: "*I used slides that were showing repetition actually happening as a debugging of the program and going through each instruction line*" - P15. However, for more advanced students, this exercise can be a bit tedious.

Making an analogy with known concepts / objects (S8). This involves using an everyday example from the real world, with situations familiar to the student, as P3 suggested: "*make an association with a real-world problem where you have the same type*

of separation (situation). I think this makes sense, so it helps a lot." In reference to arithmetic, logical, and relational expressions, P1 said that "*they (the students) confuse a lot... I started with day to day examples*" and P11 reinforces that "*I think this is a fundamental thing, exercises that have to do with the universe of the student.*"

Intensive practice (S9). Five instructors cited intensive practice during the interviews, corroborating results presented by Mhashi and Alakeel [22]. According to P13, "*The student has to arrive, sit, study, work, exercise and solve as many exercises as possible. It is very Kumon style. It is training, training, training.*" Feedback is also important, as P11 said "*I give a lot of exercises for them to do in the classroom, from there they do on paper..., if we have three solutions (made by students in the classroom), I ask the 3 students to go (to write on the board) and I show the three different solutions. Then we discuss the 3 solutions.*"

Programming on the projector screen (S10). Lastly, this strategy includes developing the code during class, projecting it so that the student can see and participate, and collaboratively making changes and showing the results: "*I use blackboard and projector ... writing a program in the act and running it*" - P15 and P14 said that this "*...makes a significant difference.*"

In addition, future confirmatory research could yield data beyond instructors' experiences, exploring the degree of student development in the CS1 courses that use those strategies. Linking the findings to existing pedagogical theories could be of great value to add knowledge and increase the contribution of these results.

5 VALIDATION OF THE RESULTS

After compiling the results, they were validated by 6 experienced instructors. Each of them was asked about the difficulties listed by topic and didactic strategies mentioned. During these interviews, one new idea arose: a suggestion to classify the difficulties as explained in the Findings section. Having confirmed our findings, six inexperienced instructors (Pu) were interviewed. These instructors had taught less than three CS1 courses and work in universities of different states in Brazil. We asked them to comment on the importance of this information about difficulties and strategies. The difficulties were showed to them without the classification presented in this paper. Even unaware of this suggestion given by experienced instructors, two of them commented on the importance of classification, as indicated in a comment made by Pu4: "*...the list of difficulties could help me to organize my classes, but I think that if the difficulties had a classification, showing which of them are the most harmful for students learning programming, it could be easier to use in practice...*"

All of the six inexperienced interviewed instructors said that the list of difficulties and strategies could greatly help to plan the material used with students and in explaining the topics, as commented by Pu6: "*...knowing the difficulties, I would work out the topic in the classroom so that the difficulty arise and so I would help them solve the doubts...*" Three of them commented that most of the difficulties could be minimized or eliminated depending on the programming language used, citing Python as an example. They added that the difficulties exist not only for C language, but also for many others, like Java. One important point they addressed regarded the topics and/or difficulties linked to the strategies. Even though most strategies are known , this connection could help to determine the best strategy at certain times in class.

5.1 Threats to Validity

The data was collected from experienced instructors from the same department who teach in different colleges at a single university. This factor may raise some doubts as to the accuracy of the results in other contexts. A survey could be created to confirm the results obtained and to complete the set of difficulties and didactic strategies.

6 CONCLUSION

Learning to program is not an easy task [1]. Students who study programming face many difficulties, as do their instructors. Some difficulties related to content are generally mild; they can be quickly clarified with examples of application and practical exercises, as is the case in understanding variables. Other difficulties are more complicated, requiring instructors to use several resources to improve learning and not always successfully. One such case is the topic Pointer. Many instructors shared their desire to avoid this topic in CS1, because they consider it too difficult. However, it is necessary for teaching other content, such as pass parameters by reference in C, in the topic Function. Pointer concepts are often omitted (S4) and/or used in the form of 'recipes' (S8), so there is no need to delve deeper into them in CS1. In addition, functions, uni- and multi-dimensional arrays, and repetition structures (while, for, ...) are considered the three most difficult topics in the instructors' view.

To teach CS1, inexperienced instructors consider it useful to have the list of difficulties students face and the strategies they can use to teach the content and avoid them. According to them, having knowledge of the difficulties could change the way they teach; for example, they can choose exercises that make the difficulties arise during the class, so that they can support students in solving them. About the strategies, we received good feedback from inexperienced instructors as well. Although many of them were already aware of many of the strategies on the list, they considered it important to have the list, in part because some of the strategies were linked to the difficulties that could be avoided, and also because the list tends to make them remember each one, making it easier to integrate them into a class. The contribution of our paper may improve pedagogical knowledge to better teach and consequently better learn programming.

ACKNOWLEDGMENTS

We thank the instructors and students who kindly provided us with the data that made this study possible. We also thanks to CAPES, this study was financed in part by the "Coordenacao de Aperfeicoamento de Pessoal de Nivel Superior " - Brazil (CAPES) - Finance Code 001.

REFERENCES

[1] Essi Lahtinen; Kirsti Ala-Mutka and Hannu-Matti Jarvinen. 2005. A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin* 37, 3 (Sept 2005), 14. https://doi.org/10.1145/1067445.1067453

[2] Kirsti M. Ala-Mutka. 2004. Problems in learning and teaching programming-a literature study for developing visualizations in the Codewitz-Minerva project. *Codewitz Needs Analysis* (2004), 1-13.

[3] Jens Bennedsen and Michael E. Caspersen. 2007. Failure rates in Introductory Programming. *ACM SIGCSE Bulletin* 39, 2 (Jun 2007), 32-36. https://doi.org/10.1145/1272848.1272879

[4] Ilias Bergstrom and Alan F Blackwell. 2016. The practices of programming. *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2016), 190-198. https://doi.org/10.1109/VLHCC.2016.7739684

[5] Yorah Bosse and Marco Aurelio Gerosa. 2016. Why is programming so difficult to learn? Patterns of Difficulties Related to Programming Learning. *ACM SIGSOFT Software Engineering Notes* 41, 6 (2016), 1-6. https://doi.org/10.1145/3011286.3011301

[6] John D. Bransford; Ann L. Brown and Rodney R. Cocking. 2000. How People Learn: Brain, Mind, Experience, and School. Expanded Edition. *National Academy Press* (2000), 384. https://doi.org/10.17226/9853

[7] Jill Cao; Irwin Kwan; Rachel White; Scott D. Fleming; Margaret M. Burnett and Christopher Scaffidi. 2012. From barriers to learning in the idea garden: An empirical study. *IEEE Symposium on Visual Languages and Human* (2012), 59-66. https://doi.org/10.1109/VLHCC.2012.6344483

[8] Juliet Corbin and Anselm Strauss. 1990. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology* 13, 1 (1990), 3-21. https://doi.org/10.1007/BF00988593

[9] Juliet Corbin and Anselm Strauss. 2015. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory* (4rd ed.). SAGE Publications, Inc. 456 pages. https://us.sagepub.com/en-us/nam/basics-of-qualitative-research/book235578

[10] John W. Creswell. 2014. *Research design: Qualitative, quantitative, and mixed methods approaches* (4rd ed.). SAGE Publications, Inc. 304 pages.

[11] Juan C. Rodriguez del Pino; Enrique Rubio-Royo and Zenon J. Hernandez-Figueroa. 2012. A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features. *Conf. e-Learning, e-Business, Entrep. Inf. Syst. e-Government* (2012).

[12] Michelle Ichinco; Yoanna Dosouto and Caitlin Kelleher. 2014. A tool for authoring programs that automatically distribute feedback to novice programmers. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2014), 207-208. https://doi.org/10.1109/VLHCC.2014.6883058

[13] Matthew Fisher and Frank C. Keil. 2015. The Curse of Expertise: When More Knowledge Leads to Miscalibrated Explanatory Insight. *Cognitive Science: A Multidisciplinary Journal* 40, 5 (Sep 2015), 1251-1269. https://doi.org/10.1111/cogs.12280

[14] Anabela Gomes and Antonio Mendes. 2007. Learning to program - difficulties and solutions. *International Conference on Engineering Education - ICEE 2007* (2007). http://icee2007.dei.uc.pt/proceedings/papers/411.pdf

[15] Anabela Gomes and Antonio Mendes. 2015. A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations. *2014 IEEE Frontiers in Education Conference (FIE) Proceedings* (Feb 2015), 1-8. https://doi.org/10.1109/FIE.2014.7044086

[16] Sandy Garner; Patricia Haden and Anthony Robins. 2005. My program is correct but it doesn't run: A preliminary investigation of novice programmers' problems. *ACE '05 Proceedings of the 7th Australasian conference on Computing education* 42 (2005), 173-180.
vfill

[17] Michelle Ichinco and Caitlin Kelleher. 2017. Towards Block Code Examples that Help Young Novices Notice Critical Elements. *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2017), 335-336. https://doi.org/10.1109/VLHCC.2017.8103497

[18] Tony Jenkins. 2002. On the Difficulty of Learning to Program. *Language* 4 (2002), 53-58. https://doi.org/10.1109/ISIT.2013.6620675

[19] Ian Drosos; Philip J.Guo and Chris Parnin. 2017. HappyFace: Identifying and predicting frustrating obstacles for learning programming at scale. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC* (2017), 171–179. https://doi.org/10.1109/VLHCC.2017.8103465

[20] Fadi P. Deek; Howard Kimmel and James A. McHugh. 1998. Pedagogical Changes in the Delivery of the First-Course in Computer Science: Problem Solving, Then Programming. *Journal of Engineering Education* 87, 3 (July 1998), 313-320. https://doi.org/10.1002/j.2168-9830.1998.tb00359.x

[21] Michael J. Lee. 2014. Gidget: An online debugging game for learning and engagement in computing education. *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2014), 193-194. https://doi.org/10.1109/VLHCC.2014.6883051

[22] Mahmoud M. Mhashi and Ali M. Alakeel. 2013. Difficulties Facing Students in Learning Computer Programming Skills at Tabuk University. *Recent Advances in Modern Educational Technologies* (2013), 15-24. https://www.tib.eu/en/search/id/BLCP%3ACN084897952/Difficulties-Facing-Students-in-Learning-Computer/

[23] Iain Milne and Glenn Rowe. 2002. Difficulties in Learning and Teaching Programming - Views of Students and Tutors. *Education and Information Technologies* 7, 1 (Mar 2002), 55-66. https://doi.org/10.1023/A:1015362608943

[24] Martinha Piteira and Carlos Costa. 2013. Learning Computer Programming: Study of difficulties in learning programming. *Proceedings of the 2013 International Conference on Information Systems and Design of Communication - ISDOC '13* (2013), 75-80. https://doi.org/10.1145/2503859.2503871

[25] Edward F. Redish. 1996. Discipline-Specific Science Education and Educational Research: The Case of Physics. *Journal of Applied Developmental Psychology* 21, 1 (1996), 85-96.

[26] Alexander Repenning. 2011. Making programming more conversational. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2011), 191-194. https://doi.org/10.1109/VLHCC.2011.6070398

[27] Maria Hristova; Ananya Misra; Megan Rutter and Rebecca Mercuri. 2003. Identifying and correcting Java programming errors for introductory computer science students. *ACM SIGCSE Bulletin* 35, 1 (2003), 19-23. https://doi.org/10.1145/792548.611956

[28] Pranay Kumar Sevella and Young Lee. 2013. Determining the barriers faced by novice programmers. *International Journal of Software Engineering (IJSE)* 4, 1 (2013), 10-22. https://vpn.utm.my/docview/1416417332?accountid=41678

[29] Lee S. Shulman. 1986. Those Who Understand: Knowledge Growth in Teaching. *American Educational Researcher Association* 15, 2 (Feb 1986), 4-14. http://www.jstor.org/stable/1175860

[30] Lee S. Shulman. 1987. Knowledge and Teaching: Foundations of the New Reform. *Harvard Educational Review* 57, 1 (Apr 1987), 1-23. http://hepgjournals.org/doi/10.17763/haer.57.1.j463w79r56455411

[31] David Weintrop. 2015. Blocks, text, and the space between: The role of representations in novice programming environments. *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* 2015-Decem, C (2015), 301-302. https://doi.org/10.1109/VLHCC.2015.7357237