Visual programming and automatic evaluation of exercises: an experience with a STEM course

Leônidas de Oliveira Brandão
Institute of Mathematics and Statistics
University of São Paulo, USP
São Paulo – Brazil
leo@ime.usp.br

Yorah Bosse
Department of Computing
Federal Univ. of Mato Grosso do Sul, UFMS
Ponta Porã – Brazil
yorah.bosse@ufms.br

Marco Aurélio Gerosa
Institute of Mathematics and Statistics
University of São Paulo, USP
São Paulo – Brazil
gerosa@ime.usp.br

***Abstract*—Programming capabilities are important to the new professionals. Although several initiatives all over the world haves been proposed for teaching programming to people at all levels. Many undergraduate students still fail in the programming courses. Proposed strategies have included visual programming and automatic evaluation of exercises. Nevertheless, there is still a lack of knowledge about students' perceived difficulties in using these strategies in practice: that is, their challenges to learning how to program. In this paper, we report a study aimed at understanding these difficulties and strategies in a STEM course. We used an environment comprising a visual programming tool to introduce algorithms, iVProg with iAssign, and the virtual programming lab (VPL) to introduce programming in C, both with automatic assessment integrated to Moodle. We report quantitative and qualitative results and future directions. Teachers and tool designers can leverage these results to better support programming learning.**

Keywords—programming; learning; novice; automatic evaluation; difficulty

I. INTRODUCTION

The software industry achieved 40.1% of the total amount of 2015 investments [1], [2], as pointed out by market researchers, such as MoneyTree[1]. However, many business fail before reaching their potential in the market; one cause is software development failures [1]. Therefore, a major challenge for modern society is preparing new generations of software developers, which means people skilled in algorithms and computer programming. Governments and large companies are attentive to promote initiatives, as in Scotland, Israel, New Zealand and South Korea [3]. These projects aim to encourage students to learn logic and programming since elementary school.

However, the area still faces many problems. Most evident is the high failure rates of students in their first programming course, usually during the first year of STEM (Science, Technology, Engineering, and Mathematics) courses.

This first programming course comprised about 28% of failures and dropout [4], [5]. In the University of São Paulo (USP) - Brazil, the rate of failure and dropout for the last five years is about 29%. More than 25% of the students try more than once to pass [5]. Students demonstrate difficulty in learning a new and formal syntax and abstractions [6]. While they can understand the syntax and semantics of commands, they cannot combine them into a single program [7]. Beginners tend to have superficial knowledge, and fail when they need to apply it [8]. Another factor that may contribute to this index is the teacher's workload. At the University of São Paulo, in the last 5 years, the average number of students per class has been 59 for the introduction to programming courses [8], which causes delayed feedback.

To assist in teaching and learning how to program, several researchers propose the use of visual programming [9], [10] and automatic evaluation systems [11]. With visual programming, students use flowcharts, programming structures blocks, and other visual aids to assist in the algorithm construction process. Automatic evaluation systems give immediate feedback to students about whether their algorithms or programs are working properly. Without such automation, this feedback would be expensive and time-consuming with large classes and many exercises.

The goal of this paper is to understand the difficulties perceived by students from a STEM course in learning how to program using visual programming and automated evaluation. These techniques were applied in course for astronomy and geophysics majors. The first research question that guided our study is RQ1: How do students perceive the use of visual programming and automated evaluation?

This study was conducted in the second semester of 2015. The class had 46 enrolled students, with 35 effective students going to classes. Of these, 10 (28.6% of the effective) were approved and 9 (25.7%) took the recovery test. Thus, 44.7% of the students failed, even with the use of the visual programming and automatic evaluation techniques.

Given this result, we decided to deepen the study, seeking to better understand the difficulties faced by programming students from STEM courses, what strategies they adopted to overcome them and how they use the tools for their studies. Thus, a second research question emerged to guide the second phase of our study: RQ2: What are student's difficulties, and what strategies do they adopt to overcome them? This second phase was conducted specifically with the students that were not approved in the regular timing.

This article contributes to the literature the elicitation of positive and negative aspects of using visual programming and automatic evaluation tools in introductory programming courses for students in STEM courses. Moreover, it also shows the difficulties and strategies used to solve them.

1https://www.pwcmoneytree.com/

II. RELATED WORK AND TOOLS USED

A. *Difficulty in learning how to program*

"Programming is a complicated business" [12]. This is evidenced by the high percentage of fail presented in the Introduction to Programming courses [4], [5]. Beaubourg and Mason studied the reasons for high rates, checking, among other factors, reduced problem resolution skills, use of laboratories provided for homework, and students entering directly into programming, without going through the analysis and design steps [13]. Initiatives to bring programming to schools help to develop skills needed for better performance. In his study, Hagan argues that having program experience before starting a programming course helps in better performance [14].

There are several skills needed to learn how to program, the more obvious being the ability to solve problems and fundamental knowledge of math. Besides these, Jenkins says it is necessary to know to use the computer to create the program, compile, test, and correct bugs [12]. However, it is fundamental to deliver to the market professionals who have these skills because "computers are useless without programs and programmers to develop them" [12].

Learning style and motivation are factors that influence the process of learning how to program [12]. Understanding the process of learning one's first programming language can help in the task of creating more effective learning environments [15], thereby reducing the difficulties encountered by beginners. Several researchers invest time in finding information about these difficulties. A study shows that syntax error is one of the barriers to programming novices, delaying the feedback provided to the student about the logic of the code developed [16].

Ribeiro et al. investigated the differences between using textual and visual programming environments in the introduction of computer programming [17]. After analyzing the data collected from NASA TLX, activity log, and survey, they concluded that visual programming is a good model for teaching algorithms and programming.

Lahtinen et al. conducted a survey at six universities in five countries, showing that students are very confident in independent studies; individualized study was found most effective when compared to the exercise sessions and practical classes. The result was the opposite when respondents were teachers [8].

Some support programming learning systems are used by many teachers to try to ease the difficulties faced by students. Scratch is an example of such systems. It enables the student to program with a visual model, presenting programming with blocks [18]. According Malan and Leitner, Scratch puts the focus on the logical problem and not in the syntax [18]. In a study with students from 5th grade primary school, who used Scratch, Kalelioglu and Gülbahar found that in the process of solving problems, half of the students had difficulties and the other not. Most tried to solve their problems in different ways [19]. Beyond Scratch, there are other systems such as Alice [20], Logo [21] , iVProg [22], among others.

B. *iVProg*

iVProg is an environment to support learning programming using the Visual Programming (VP) paradigm. This project started in 2009 [23]. Figure 1 presents the main interface of iVProg's current version (in Java - there is another in HTML5). In the visual programming paradigm, the students mainly use the mouse to construct the algorithm [24].

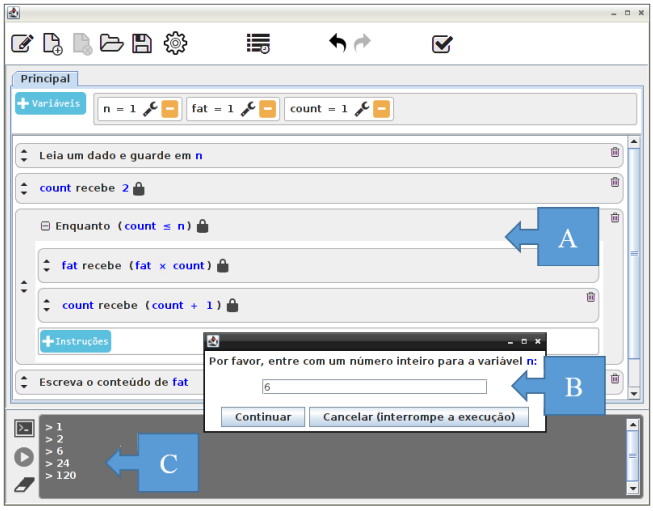

Fig. 1. The iVProg screenshot with the code for the calculation of Factorial (A). Request for a new input (B) and the results printing for previous inputs (C).

The version of iVProg adopted is an applet Java, working as an interactive Learning Module (iLM). An iLM is an educational system that can be integrated with any Learning Management System (LMS) that implements some special communication functions [25]. Moodle is an example of an environment prepared to receive iLM [26], [27]. Furthermore, iVProg can be used as an iAssign applet and it has an automatic evaluator system based on test cases. Basically, the outputs generated by the algorithm built with iVProg is compared to the outputs provided by the teacher [17]. The teacher-author prepares a set of inputs and their correspondent outputs. When the students submit their solution, the input set is used with the students algorithm and its outputs are compared against the correspondent test case [17].

In a prior experiment, the use of iVProg increased the presence in classes by more than 3.3%, and increased students' average grades by more than 0.53 points [27].

C. *Virtual Programming Lab – VPL*

Virtual Programming Lab (VPL) is also a Moodle plugin, developed at the University of Las Palmas Gran Canaria – ULPGC[2]. Its main features provide an integrated environment to teach and learn programming, similar to iAssing/iVprog. VPL offers an editor to the learners entering their program and use test cases to automatically validate their solutions [28]

VPL is independent of the programming language. It is only necessary that the server provides the correspondent compiler. The tool provides syntax highlighting for several languages, such as C, Java, and Python [28]. Figure 2 presents

2 Virtual Programming Lab – VPL. URL: http://vpl.dis.ulpgc.es/index.php/about

the VPL editor in a former version (this one as an *applet* Java). On the figure's left side, one can see the Java editor with C code, and on the right side the evaluation result, indicating the presence of 3 case tests. The immediate feedback usually stimulates the learners to keep trying, until they reach the perfect score.

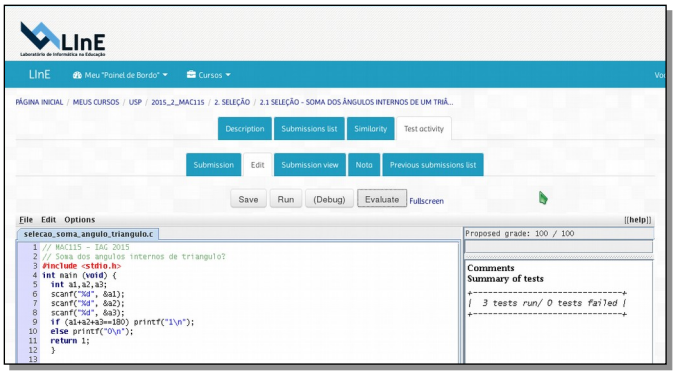

Fig. 2. VPL layout (with Java) - editor with syntax highlighting and the evaluated code.

The teacher-author can limit the time for submission of solutions, as well as the number of submissions for the same exercise. Furthermore, teachers have the option to prevent pasting of external texts into the editor area. Another interesting VPL feature allows viewing groups of similar codes.

During the first phase of this study, we used VPL with Java to be coherent with the Java version of iVProg. In the second phase, since the focus was C programming, we moved to the current version of VPL, using HTML5 related technologies. In fact, figures 2 and 3 present the same content, but with VPL Java and HTML5 interface, respectively.

D. The NASA-TLX Protocol

The NASA-TLX protocol was proposed to verify the perceived workload while someone performs a task [27]. It provides quantitative data for the overall workload assessment, based on the weighted average of six scales: mental demand, physical demand, temporal demand, level of achievement, level of effort, and level of frustration [29]. The NASA-TLX was used to evaluate systems interface and to measure the user's perception. It was applied in two steps after the users completed their work.

In the first step, the user answered, on a scale of 0 to 100, six questions related to the six aforementioned dimensions [14]. The six questions were:

1. Mental Demand (MD): How much mental and perceptual activities were required (e.g., thinking, deciding, calculating, remembering, observing, searching, etc.)? Was the task easy or difficult, simple or complex?
2. Physical Demand (PD): How much physical effort was required during the activity (for example, clicking, typing, pushing, pulling, controlling, activating, etc.)? Was the task fast or slow, light or heavy?
3. Temporal Demand (TD): How much time pressure was felt in the task execution pace? Was the pace slow and leisurely, or fast and frenetic?
4. Performance (P): How successful do you consider yourself when performing the task objectives? Were you pleased with your performance in task fulfillment?
5. Effort (E): How hard did you have to work (mentally and physically) to reach your level of performance?
6. Frustration (F): How insecure, discouraged, angry, stressed, or annoyed, versus secure, encouraged, satisfied, relaxed, or complacent, did you feel during the task?

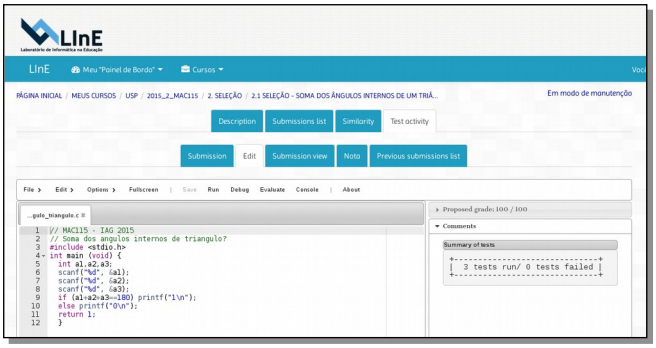

Fig. 3. VPL layout (with HTML5) - editor with syntax highlighting and the evaluated of the code.

In the second step, all 6 scales were presented to the user in pairwise basis, resulting in 15 contests. For each pair, the user decided between the pair, i.e., considering the two options, they decided which factor had the greater influence during the task under screening. After all the pairwise contests, the weights defined at the first step were used to compute the perceived workload, by the sum of the products between each level and their respective weights [14].

III. METHODOLOGY

In this study, we used a mixed-methods approach, aiming for a triangulation strategy. We combined qualitative and quantitative data, questionnaires, and interviews. The study focused on a mandatory introduction to programming course (PROG1), and was divided into two distinct phases. The first one occurred during the semester, and the second phase occurred with a small group of students from the same course.

A. Phase 1: quantitative analysis on automatic assessment

The adopted process for the first phase is shown in figure 4. It was conducted during the regular mandatory course at USP, regularly cited as the most prestigious university in South America.

The course had 46 students enrolled, and 35 students effectively participating in the classes. It was offered to freshmen in STEM courses to introduce them to programming concepts. It was a semester-long course, lasting 18 weeks, with 2 classes per week of 2 hours each. All classes took place in a computer laboratory, with one student per computer. During each class, typically a reduced set of problems were proposed, and the students were encouraged to find the algorithm that solved the problem, using VPL or iVProg. During the first 7 weeks, iVProg was adopted as a bridge for introducing C language with VPL. After week 8, when topics of function and strings were introduced, only VPL was used.

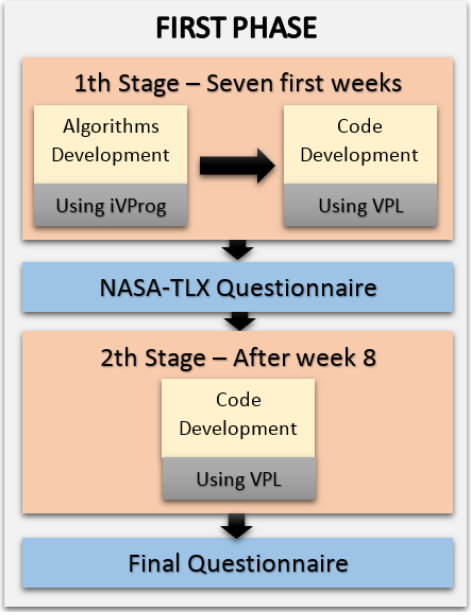

Fig 4. The first phase process.

Thirty different exercises were considered, 19 of them being presented twice to students: first with iVProg, then under C with the VPL. Since visual programming is considered less demanding to students, it was used as bridge to reduce the students' difficulties with the C programming language. These exercises were not mandatory, but aimed to stimulate them to try hard. Nevertheless, the exercises could add up to 1 point to the student final grade.

To reduce additional cognitive workload and teacher workload, Moodle was integrated with the plugins VPL and iAssign [28] [26]. iAssign was used with a particular iLM, the iVProg [17].

In order to better understand the students' perception while using iVProg and VPL, the NASA-TLX questionnaires were applied to assess the workload when using both iVProg and VPL. This instrument was not to be used as a comparison between both technologies, since iVProg was used at first by the students to conceive of the algorithm that solves the problem. With this solution, they used VPL to master the C syntax. In addition, a final questionnaire containing objective and open questions was used to verify the students' perception about the course and the technologies used.

At the end of the semester, the students that failed, could take a final exam. If their grade on the exam compensated for (outweighed) the semester's failing grade, they passed. This recuperation option is a common procedure in Brazilian universities, usually offered to any student that fails a course.

B. Phase 2: qualitative analysis on a small subset

The intention of the second phase was the better understand the students' difficulties in programming. It was designed as a short course of PROG1 (8 weeks), and before this, we conducted individual interviews with a small set of students. Conveniently, we invited all the students that failed the course PROG1 (Fig. 5).

There was no reward for students' participation in phase 2. However, the student could be motivated to get more assistance to eventually learn "how to program" and, perhaps, succeed on recuperation. As a result, we achieved the consent of all 9 invited students.

The individual interviews occurred before the exam. We used the Think Aloud Protocol. According to Villanueva, the technique consists of observing users performing specific tasks within a controlled environment [30]. During the interview, the students described aloud, in real time, what they were thinking. To capture these spoken aloud thoughts, there are two possible techniques: (1) recording on video or (2) transcription by a moderator. Although video or audio recording has the advantage of documenting everything that is done and said, the disadvantage is that the user may become intimidated and not verbalize everything they are experiencing. In the case of real time annotation, the moderator writes down everything that happens and what the interviewed says. The advantage is that the environment becomes more relaxed, leading the users to feel more comfortable exposing everything they think and know. The disadvantage is that the speed required for such note-taking can lead to loss of important research information [30].

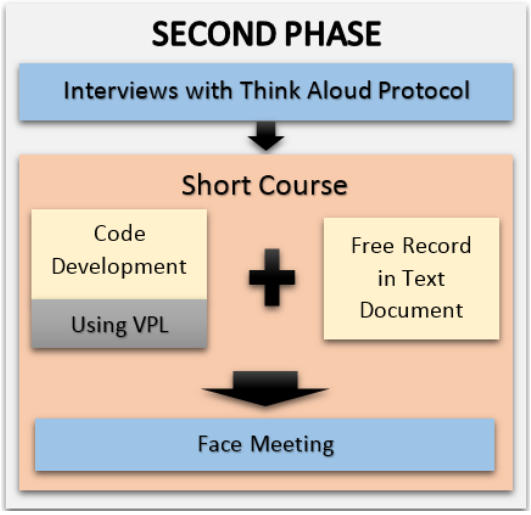

Fig. 5. The second phase process.

Genise (apud Renzi et al.) describes the protocol application procedure in five steps: (1) Organize a small group of users, about 4; (2) the researcher meets these users; (3) the researcher decides which tasks should be carried out and in which environment; (4) the researcher explains the method to users by directing them to verbalize their thoughts; (5) the researcher notifies the necessary changes in the tool [30]. As our goal is not the tool but the difficulties of programming, step 5 does not apply to our study.

We performed the Think Aloud interview session with the recording process, and asked the students to solve four exercises with different degrees of difficulty in VPL with C language. Each interview lasted about one hour, and the students were encouraged to talk constantly about what they were doing and thinking. During the process, the moderator asked a few questions, such as "What do you think about using the VPL in class?", in order to seek information beyond what the students spontaneously exposed.

The next step in the second phase was the short course, which used the same environment from the semester classes, but with an updated VPL version. In the version used during the semester, students faced a lot of problems setting up Java on their computers. With HTML5, it is not necessary to install anything in the students' computers.

Every two weeks in this short course, students received a set of exercises to solve. Each set of exercises increased in degree of difficulty. In addition to solving the exercises using VPL, the students made free record of this process in a text document. After solving each set of exercises, there was a face-to-face meeting where doubts were clarified.

IV. Main Results

We conducted a two-phase study, the first involving all students from the class, and the second involving only the students that had failed during the semester. In the first phase, we used quantitative instruments, whereas the second focused on qualitative analysis.

The research questions for this paper were "RQ1 – How do students perceive the use of visual programming and automated evaluation?" and "RQ2 - What are student's difficulties, and what strategies do they adopt to overcome them?". To show the results, we created sub-research questions, as follows.

A. *RQ1a – What benefits and difficulties do students perceive regarding the use of iVProg and VPL?*

At the end of PROG1 in the second half of 2015, the teacher asked students to answer a questionnaire with seven questions, designated herein as the Final Questionnaire. Questions can be divided into objective (the first four) and discursive (the remaining).

The objective questions obtained 100% of responses. Some results from the answers are that 65% of respondents had taken this course before, even partially, and had not been approved. When asked about their preferences concerning classes in the laboratory, 47% said they most liked when classes take place in a laboratory setting.

Another question checked if the students liked the individual class activities of solving exercises in the laboratory. Furthermore, it was important to have feedback from students about the number of questions provided for them. The answers showed that 53% enjoyed solving exercises in class, (Fig 6.), and the same amount considered the amount of exercises appropriate (Fig. 7). As previously stated, the course lasts for one semester, with 18 weeks and it was used 49 exercises, 19 of them under two different technologies, C with VPL and visual programming with iVProg.

In the two following questions the students were asked to identify positive and negative aspects about the use of automatic evaluation systems such as iVProg and VPL. The answers are compiled in TABLE I. Three students from 16 declared "I do not see negatives aspects"$_{s1, s2, s3}$.

At the end of the questionnaire, we asked the students for suggestions, criticism, and praise for the semester class, using an automatic evaluation system. Some students praised the class model, stating sentences like "I loved doing the course in this model"$_{a9}$, "I loved having a practical class, I believe that for a computer course, it is essential"$_{a6}$ and "I liked the way the course was given a lot, the automatic evaluator, the fact of being online, the iVProg, etc."$_{a7}$. A suggestion given by students was to switch between practical and theoretical classes, stating phrases such as "take turns between conventional classes and the computer"$_{a4}$ and "as all classes were practical, I missed introductory lectures"$_{a6}$. The students asked for "more exercises available to be done at home before the assessments."$_{a3}$.

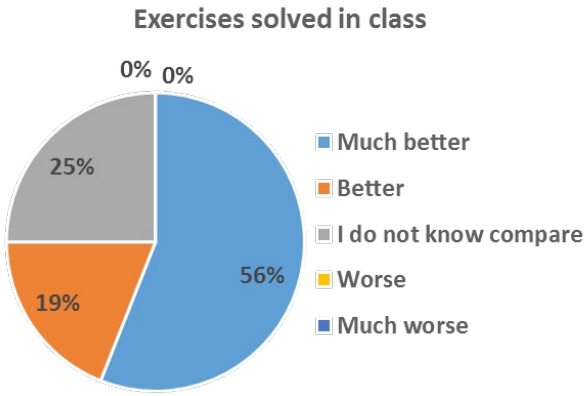

Fig. 6. Solve exercise during class.

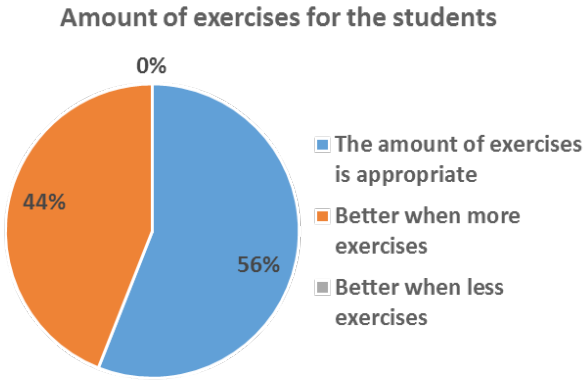

Fig. 7. Evaluation as the amount of exercises available to students.

Wanting to gather more information about the students and their behavior during the studies, using the Think Aloud method we conducted interviews with six students, lasting about one hour each. The students invited to participate in these interviews were in the same situation: they did not succeed during the semester, and needed to take the recovery test to be approved. During the interviews, they were challenged to solve four exercises with increasing degree of difficulty. Their interview session was documented, including the computer screen and audio recordings, for future analysis. In this report, the students are numbered from 1 to 6.

TABLE I. POSITIVES AND NEGATIVES ASPECTS IN USE AN AUTOMATIC EVALUATION

Positives Aspects	Negatives Aspects
Presentation of results through the test cases lets you know if your code is working correctly $_{s8, s11}$	Dependence on compilation $_{s4, s5, s6}$ and of the results of the test cases $_{s6}$, making the student does not look for errors alone $_{s5}$
Easy to view the syntax errors $_{s1, s9}$	Missed print messages / instructions to users $_{s7, s8}$
Immediate feedback to the student $_{s2, s6}$	Show that the automatic evaluation has flaws $_{s9, s10}$
Saving time for solving exercises $_{s3}$	Inhibits the practice of how to test the algorithm $_{s11}$
It gives an idea of your note through the percentage of correct answers of the exercises $_{s3}$	
Easy to correct $_{s1}$	

One of the observed attitudes, adopted by 2 of the students, was to take notes while they read the statements (student 1 and 3). These 2 had no better results than the others, but one of them, when asked by the interview moderator, stated "annotating helps to remember what needs to be done, because otherwise I cannot remember". Analyzing the behavior of respondents while running the session, we observed that this annotation process helped, for example, in the definition of which and how many variables were required to solve the task. One difference between these students and the others was that they had less mistakes in declaring the variables and setting their types; they practically did not need to return to the code to change what they had written.

The interview moderator observed in two students a reaction while reading the statement. Student 6 had not read the entire statement when he stopped reading to make the comment "I get nervous when I see the word matrix." Student 1, when starting to read the second question, spoke instantly "I do not like function" and "I have difficulty with function parameters." During question 6, student 1 said "At a first glance I dislike this exercise, I like exercises that have numbers." In these three situations, the students did not succeed in solving the exercise. This may be a sign that the students create a barrier to the content for which they face more difficulty.

We also noticed insecurity in students and some degree of absence of thinking. They were used to copying and pasting the code to read matrix elements, but when faced with compilation errors, they made comments like "We will see now. Must be something wrong. There is always something wrong." The moderator noted that the commands to which they referred were correctly written, but with undeclared names. Moreover, in some instances, the students faced problems with intention and practice. They verbalized something, but wrote something different. This situation was detected during interviews, and can be observed through the comments "I do not know if it's like this to read an array, but okay,"$_{a1}$ and "I think something is missing in this print"$_{a6}$.

Syntax errors were common in all interviews and resolutions, e.g. opening and closing structures with brackets, colons, correct spelling of the commands, among others. Some errors are noteworthy, such as: (A) attempting to read the data in the matrix; (B) creating an unnamed function, besides the incorrect declaration of the variables to receive the parameters, and (C) semi-colons ending a structure of repetition and selection that has not even started (Fig. 8).

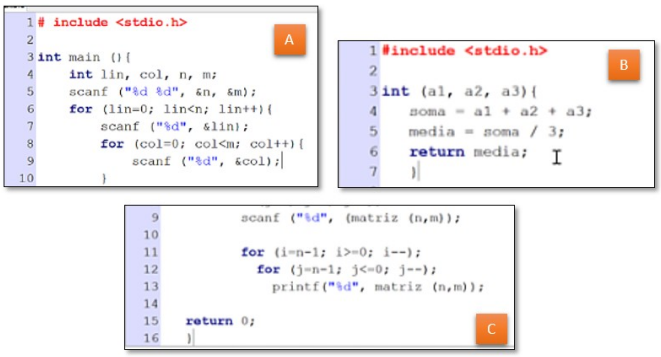

Fig. 8. Less common syntax errors.

When semantic errors occurred, students usually became more disappointed than with syntax errors. With syntax errors, they seem more accustomed. The semantic errors made students drop out of the exercise faster, because this is naturally more difficult, which they already realized (facing more time to fix semantic errors).

B. *RQ1b – What is the perceived effort by the student?*

The NASA-TLX were applied at the end of the first block of activities using iVProg and after VPL. The questionnaire evaluated the students' perceptions of workload concerning 4 elementary exercises (related to input-output, comparison, and operation with the rest of the integer division).

It is worth remembering that the activities were first released for iVProg, and then for VPL. Figure 9 presents the NASA-TLX for iVProg, and Fig. 10 shows the results to VPL. Note that, despite iVProg being used first, the mental demand (MD) for it was smaller than using C language with VPL.

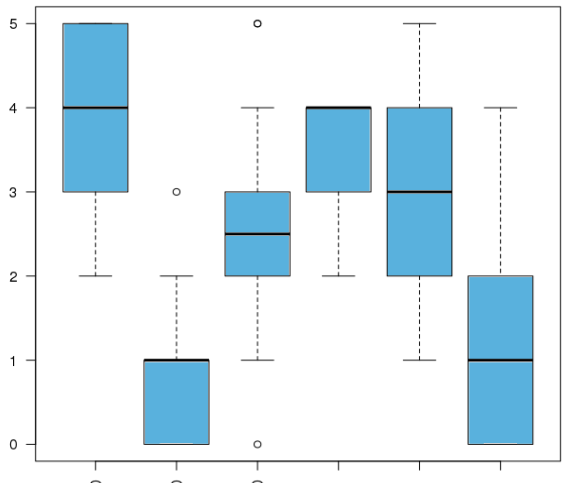

Fig. 9. NASA-TLX for the first block of activities with iVProg.

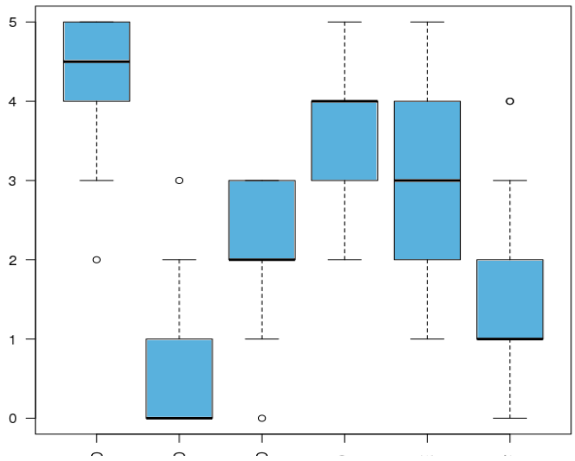

Fig. 10. NASA-TLX for the first block of activities using VPL with C language

Automatic evaluations

We divided the course into 6 sections, each with a number of exercises. During the first 9 weeks, the students were challenged with solving a set of activities (solving a problem with an algorithm) first using iVProg, then with C, for the same problem.

These programming activities had a small impact on the final grade, up to 10% of the student's final grade. These exercises were proposed to stimulate students to solve a large number of exercises. We used 17 iVProg exercises and 28 of VPL/C (the first 17 of VPL were the same 17 of iVProg).

The number of answers for VPL was 531, but some activities had more than 50 submissions for the same exercise, from several students. For iVProg, there were 197 submissions.

Since iVProg was used as a bridge to introduce C, we focused this analysis in VPL/C. We tested the correlation between several measures, considering student performance in VPL activities, compared to their final situation (approved, fail, final grade). The activities measures were (TABLE II.): the number of activities with maximum score (N10); the number of activities with minimum grade (N0); and the number of activities with minimum grade for compilation errors (NC). The student situation was: student approval (1) or fail (0) (AP); student final grade, where 5, or more, means approved (FG).

TABLE II. FINAL SITUATION TO EACH STUDENT IN THE COURSE

Student	*N10*	*N0*	*NC*	*AP*	*FG*
1	13	0	0	0	0
2	3	2	2	0	1.01
3	11	0	0	1	5.87
4	26	0	0	1	7.52
5	18	0	0	1	9.81
6	10	0	1	0	0.31
7	16	2	0	0	4.21
8	15	0	0	0	0
9	22	0	1	1	5.56
10	22	1	0	1	9.41
11	14	2	3	0	0.73
12	16	0	0	0	4.17
13	9	0	1	0	0
14	13	2	0	0	0.36
15	0	0	2	0	0
16	18	3	2	0	2.92
17	14	0	0	1	5.45
18	6	0	0	0	0
19	0	0	0	0	1.23
20	9	0	0	1	8.28
21	0	0	1	0	0
22	15	0	0	0	4.15
23	21	2	1	1	6.21
24	18	0	2	0	3.41
25	15	0	0	0	4.64
26	15	0	3	0	2.05
27	2	0	0	0	0
28	5	4	1	0	4.39
29	15	0	0	1	7.76
30	5	1	0	0	0
31	10	3	3	0	3.61
32	14	2	3	0	1.3
33	21	0	0	0	1.21
34	5	0	1	0	0
35	26	0	0	1	7.98

C. *RQ2a – What are the difficulties in programming learning using iVProg and VPL?*

The short course lasted 8 weeks. The average number of submissions per exercise was 7, and 84% of them had 100% success in the test cases analyzed by the system. The students submitted twice on average until their code was compiled, and three times until correct results began to appear.

During the analysis of the code, some syntax (TABLE III.) and semantics (TABLE IV.) errors were detected. In this paper, we show errors that appeared more than once.

We also noticed syntax errors in which we observe that the student was using mathematical commands in the code, such as the use of point instead of an asterisk for multiplication and writing condition as follows: "(600 <x <= 1200)" instead of "(x> 600 && x <= 1200)."

TABLE III. SYNTAX ERROR

Amount	Description
7	Not opened with "{" and closed with "}" any structure or function
3	Error in the formatting of the number of decimal places in a float
3	Use of undeclared variables
2	Forgot ";" on some lines
2	In the statement of the float type variables, separated the names with ";" and not ","
2	Do not separate the variables with comma within the parentheses of the scanf
2	Use comma rather than dot on real numbers.

TABLE IV. SEMANTIC ERRORS

Amount	Description
9	They did not pay attention in division by 0 (zero)
5	Control variable from "while" not initialized
4	Wrong formula
2	If..else structure mounted incorrectly
2	Used "return" to show -1 instead of printf: return "-1"

These are problems that make the program less effective, such as not using "else" in the selection structures "if." This problem was noticed 11 times, i.e. a high number compared to other problems. Other mistakes include loading unnecessary libraries and lack of indentation.

D. *RQ2b – What study strategies do students use?*

When students encounter difficulties during their studies and cannot proceed, they seek help from a classmate or friend who knows the subject, or seek content from the internet, or similar examples of exercises that they are solving.

Another students' strategy to solve problems was to divide the code into small portions and check each one, trying to identify where the error was located.

E. *RQ2c – How have they used the tools?*

Students who participated in the short course reported that the use of iVProg was very helpful to begin developing logical thinking, and that continued use of this tool would be interesting. VPL plays an IDE role. Providing information on syntax errors shows the student that there are also semantic errors.

Students said the results of the test cases pointed out by VPL helped to find semantic errors, which they unanimously considered the most difficult errors to detect and resolve. Test cases, according the students, were not completely analyzed, at times were analyzed only those by accused of error and others only by those who have succeeded. How to use the test cases to help resolve the errors would be a good exercise to perform with the students.

V. CONCLUSION AND FUTURE WORKS

In this study, we observed that students see value in automatic evaluation, even considering students that already had failed in the programming course. In addition, the system with automatic evaluation enabled the teacher to identify students with low participation levels. We found correlation between the number of correct exercises and overall course approval.

For the teacher, it became possible to present a greater number of exercises. However, this presents a challenge for using these tools with a large number of students. Perhaps it could be used with exercises, but not in formal evaluation that demands the presence of the student. This is now under investigation. About the preparation of the exercises for the students, one difficulty faced by us is how to create good test cases. The first condition is that it test all the important cases, like a sorting *problem* the presence of unitary sequence, increasing sequence, and decreasing sequence.

In our research, we found that the students approved to learn programing using automated evaluation with iVProg and VPL as well as visual programming with iVProg. However, a difficulty mentioned by students is the reduced number of theoretical classes and one suggestion is to take turns between conventional classes and in the laboratory. About using iVProg and VPL, various positives aspects was cited by the students and fewer negatives points. In the same time, that knows if the code working correctly with the presentation of the results of test cases, the use of this systems create a dependence making the student does not look for error alone.

About difficulties and strategies, we find that take note during the studies help learners to organize their ideas such as the amount and types of variables needed to solve the problem. Semantics errors are considered by students most difficult to solve than syntax errors. To help them in their studies, when they don't can proceed, they ask classmates, friends or demand for similar exercises on the internet.

It is worthy to note that automatic evaluation can promote an important aspect: allow to challenge students with new problems.

This study pointed some new questions that we are considering as future studies. One of them is to deeper the comparison study presented by Ribeiro et al [16]. This paper compared visual and textual programming, mainly considering the cognitive workload. It would be interesting to create a new method to compare the final skill of the students to solve problems by algorithms after a course of each model of programming.

ACKNOWLEDGMENTS

The authors would like to thank all students that participate in this study for their involvement. This work is partially supported by grant of São Paulo Research Foundation (FAPESP).

REFERENCES

[1] M. Crowne, "Why software product startups fail and what to do about it. Evolution of software product development in startup companies," in *IEEE International Engineering Management Conference*, 2002, vol. 1, pp. 338–343.

[2] T. Reuters, "MoneyTree Report - Q4 2015/Full Year 2015 Summary," 2016. Online at https://www.pwcmoneytree.com.

[3] S. P. Jones, "Computing at School International comparisons," *Retrieved May*, no. November, pp. 1–12, 2011.

[4] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," *ACM SIGCSE Bull.*, vol. 39, no. 2, p. 32, 2007.

[5] Y. Bosse and M. A. Gerosa, "Reprovações e Trancamentos nas Disciplinas de Introdução à Programação da Universidade de São Paulo : Um Estudo Preliminar," *Work. sobre Educ. em Comput. (WEI)*, pp. 1–10, 2015.

[6] C. M. Lewis, "How Programming Environment Shapes Perception, Learning and Goals: Logo vs . Scratch," *Sigcse '10*, pp. 346–350, 2010.

[7] L. E. Winslow, "Programming Pedagogy - A Psychological Overview," *ACM SIGCSE Bull.*, vol. 28, no. 3, pp. 17–22, 1996.

[8] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," *ACM SIGCSE Bull.*, vol. 37,

no. 3, p. 14, 2005.

[9] M. C. Carlisle, T. A. Wilson, J. W. Humphries, and S. M. Hadfield, “RAPTOR: Introducing Programming to Non-Majors with Flowcharts,” *J. Comput. Sci. Coll.*, vol. 19, no. 4, pp. 52–60, 2004.

[10] M. Kölling, “Greenfoot - A Highly Graphical IDE for Learning Object-Oriented Programming,” *ACM SIGCSE Bull.*, vol. 13, no. 4, p. 60558, 2008.

[11] C. Reade, “A System for Automatic Evaluation of Programs for Correctness and Performance,” *Web Information Systems and Technologies*, 1997.

[12] T. Jenkins, “On the Difficulty of Learning to Program,” *ICS - Int. Conf. Supercomput.*, 2002.

[13] T. Beaubouef and J. Mason, “Why the high attrition rate for computer science students,” *ACM SIGCSE Bull.*, vol. 37, no. 2, p. 103, 2005.

[14] D. Hagan and S. Markham, “Does it help to have some programming experience before beginning a computing degree program?,” *ACM SIGCSE Bull.*, vol. 32, no. 3, pp. 25–28, 2000.

[15] S. Garner, P. Haden, and A. Robins, “My program is correct but it doesn’t run: A preliminary investigation of novice programmers' problems,” *Conf. Res. Pract. Inf. Technol. Ser.*, vol. 42, pp. 173–180, 2005.

[16] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx, “Understanding the syntax barrier for novices,” *Proc. 16th ACM Conf. Innov. Technol. Comput. Sci. Educ. - ITiCSE ’11*, p. 208, 2011.

[17] R. S. Ribeiro, L. O. Brandão, T. V. M. Faria, and A. A. F. Brandão, “Programming web-course analysis: how to introduce computer programming?,” *ASEE/IEEE Frontiers in Education Conference (FIE)*, 2014, pp. 1–8.

[18] D. J. Malan and H. H. Leitner, “Scratch for budding computer scientists,” *ACM SIGCSE Bull.*, vol. 39, p. 223, 2007.

[19] F. Kalelioğlu and Y. Gülbahar, “The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners’Perspective,” *Informatics Educ. Int. J.*, vol. 13, no. 1, pp. 33–50, 2014.

[20] S. Cooper, W. Dann, and R. Pausch, “Alice: a 3-D tool for introductory programming concepts,” *J. Comput. Sci. Coll.*, vol. 15 no. May 2000, pp. 107–116, 2000.

[21] S. Papert, *“Mindstorms: Children, Computers and Powerful Ideas”*. Basic Books, Inc. New York, NY, USA, 1980.

[22] L. O. Brandão, R. S. Ribeiro, and A. A. F. Brandão, “A system to help teaching and learning algorithms,” ASEE/IEEE Frontiers in *Education Conference (FIE), 2012*, pp. 1–6..

[23] R. R. Kamiya and L. O. Brandão, “iVProg - um sistema para introdução à Programação através de um modelo Visual na Internet,” *XX Simpósio Bras. Informática na Educ.*, 2009.

[24] E. P. Glinert and S. L. Tanimoto, “Pict: An Interactive Graphical Programming Environment,” *Computer (Long. Beach. Calif).*, vol. 17, no. 11, pp. 7–25, Nov. 1984.

[25] J. G. Moura, L. O. Brandão, and A. A. F. Brandão, “A web-based learning management system with automatic assessment resources,” ASEE/IEEE Frontiers in *Education Conference (FIE)*, 2007, pp. F2D–1–F2D–6.

[26] P. A. Rodrigues, L. O. Brandão, and A. A. F. Brandão, “Interactive Assignment: A moodle component to enrich the learning process,” ASEE/IEEE Frontiers in *Education Conference (FIE)*, pp. 1–6, 2010.

[27] R. S. Ribeiro, “Construção e uso de ambiente visual para o ensino de programação introdutória,” Doctoral dissertation, Universidade de São Paulo, 2015.

[28] J. C. Rodríguez-del-Pino, E. Rubio-Royo, and Z. Hernández-Figueroa, “A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features,” *Conf. e-Learning, e-Business, Entrep. Inf. Syst. e-Government*, 2012.

[29] B. M. Guimarães, L. B. Martins, L. S. Azevedo, and M. D. A. Andrade, “Análise da carga de trabalho de analistas de sistemas e dos distúrbios osteomusculares,” *Fisioter. em Mov.*, vol. 24, no. 1, pp. 115–124, 2011.

[30] A. B. Renzi, S. Freitas, I. Kuhn, and S. Köhler, “Use of Think-Aloud Protocol to Verify Usability Problems and Flow During Use of Entertainment and Personal Journal,” in *12o Congresso Internacional de Ergonomia e Usabilidade de Interfaces Humano-Computador*, 2012, p. 7.