Evaluating Feedback Tools in Introductory Programming Classes

Ruan Reis*, Gustavo Soares†, Melina Mongiovi*, Wilkerson L. Andrade*
* Federal University of Campina Grande, † Microsoft
ruanvictor@copin.ufcg.edu.br, gustavo.soares@microsoft.com,
melina@computacao.ufcg.edu.br, wilkerson@computacao.ufcg.edu.br

***Abstract*—This Research Full Paper presents a study on the evaluation of feedback tools in introductory programming classes. Recently, several tools have been proposed in order to provide guidance and help students overcome conceptual difficulties in programming education. Some tools leverage clustering algorithms and program repair techniques to automatically generate personalized hints for students' incorrect programs. In contrast, some teachers choose to present students with program visualization tools to help them understand the dynamic execution of a source code. These tools are used to help students get correct solutions for programming assignments. However, due to limitations in assessments, it is still unclear how effective the feedback provided by these tools is. In this study, we analyzed the effectiveness of a tool for generating personalized hints and a tool for visualizing programs. To do so, we conducted a user study in which students, assisted by these tools, implemented solutions for three programming problems. Our results show that personalized hints can significantly reduce student's effort to get correct solutions. In addition, personalized hints can provide students with an understanding of problem solving similar to when using test cases. However, students who used the program visualization tool got lower post-test performance than using other tools.**

***Index Terms*—programming education; feedback generation, program visualization, study, evaluation.**

I. Introduction

Learning to program is a challenge faced by students in most introductory programming courses [1]. In online and face-to-face classroom, students need to put the acquired knowledge into practice through practical programming assignments. To assist in these activities, teachers need to provide guidance and assistance, especially to novice learners who are getting their first programming experiences and need to overcome conceptual difficulties [2].

Feedback from teachers can help students get unstuck and correct their misconceptions [3], [4]. However, personalized attention does not scale easily, especially in massive programming classrooms [5], [6]. One of the most common practices used by teachers to provide feedback at scale is to present the student with a test suite. In this way, students can run their programs against test cases and receive reports from failing tests. Although it is a useful feedback, it may be difficult for a beginner to understand which misconceptions are made only through test cases results.

Recently, several tools have been proposed to support programming education. [7]–[15]. These tools use different approaches to generate, scale and personalize feedback to help teachers and students in the teaching environment. For example, Clara [7] can automatically repair incorrect programs, indicate the location of bugs (e.g., line number), and provide an exactly textual description of required changes. The approach used by Clara consists of cluster the existing solutions for a given assignment; select a target program from each cluster; and execute a trace-based repair procedure to repair new incorrect attempts. The Python Tutor [15] allows users to step forwards and backwards through execution to visualize the run-time state of a programs data structures. By using these features, students can debug their programs and, as a result, they can fix bugs and get correct solutions to programming assignments. Given a program as input, the Python Tutor performs an analysis under supervision of the standard Python debugger module (bdb), which stops execution after every executed line and records the program's run-time state.

These tools can be helpful in reducing teacher effort to propagate and personalize feedback. However, it is still unclear how effective these tools are, especially when the user is a novice programmer. The reason for the lack of clarity on this subject is due to the limitations in evaluations of these tools. The most common limitations found in papers are related to: (1) the lack of user studies, especially with beginners; (2) fail to get insight into learning improvement or skill acquisition; (3) the lack of comparative studies with other existing tools; and (4) focus only on evaluating the tool's performance in generating hints, but does not evaluate its usefulness. Therefore, it is necessary to investigate to what extent the feedback provided by these tools can be effective.

In this study, we evaluated the effectiveness of Clara and Python Tutor in assisting novice programmers in problem solving. Our goal is to analyze whether using these tools students can solve programming assignments better than when using only test cases. We selected Clara for our evaluation because it is a state-of-the-art tool for automatic hints generation. On the other hand, we selected Python Tutor because it is widely used in programming classes to visualize program execution. In our evaluation, we recruited 42 undergraduate students and asked them to implement Python solutions for three classic problems. For each problem, students were able to use Clara or Python Tutor, and a test-case suite to help them solve the problems. Subsequently, in order to evaluate the impact of the tools on the student's understanding of how to solve the problems, we proposed a post-test in which students should review four

978-1-7281-1746-1/19/$31.00 ©2019 IEEE

solutions for a specific problem and indicate whether each solution is correct or not.

Specifically, we analyzed the effectiveness of these tools with respect to three aspects: (i) how fast a student can get a correct solution; (ii) what is the impact of the tool on the student's understanding of how to solve the problem; (iii) how useful is the tool to help students correct bugs in their programs. Our results show that, when considering getting correct solutions faster, Clara can significantly reduce the student's effort (i.e. number of attempts) compared to other tools. In the post-test results, we did not find a significant difference comparing the performance of students who used Clara to students performance using only test cases. However, students who used Python Tutor got lower post-test performance than using other tools. Finally, students scored Clara as more useful than test cases to fix bugs in their programs. They mentioned that finding bugs using only test cases is difficult, but using Clara, they can figure out where the bugs are and immediately reflect on the hints provided.

II. Related Work

A. Automated Feedback Generation Tools

There are several feedback generation tools to help students in introductory programming assignments [7], [8], [11]–[14], [16], [17]. For example, AutoGrader [17] takes as input an incorrect student program, along with a reference solution and an error model consisting of potential corrections to student errors, and searches for the minimum number of corrections using a SAT-based program synthesis technique. As feedback, AutoGrader describes exactly what changes are needed to repair the program.

Clara [7] can automatically repair incorrect programs, indicate the location of bugs (e.g., line number), and provide an exact textual description of required changes. Its approach is to cluster the correct programs for a given assignment and select a canonical program from each cluster to form the reference solution set. Then, Clara runs a trace-based repair procedure against each program in the solution set and selects a minimal repair from the repair candidates.

Sarfgen [8] leverages the large number of available student solutions to generate instant, minimal, and semantic fixes to incorrect student submissions without any instructor effort. Its approach is to search for reference solutions similar to a given incorrect program. Then, Sarfgen aligns each statement in the incorrect program with a corresponding statement in the reference solutions to identify discrepancies for suggesting changes. Finally, it points out minimal fixes to patch the incorrect program.

These tools were subjected to a similar evaluation procedure. Basically, the researchers analyzed the effectiveness of their tools in generating repairs for incorrect programs. To do so, they ran their tools in a benchmark set of incorrect submissions and then computed the percentage of submissions that were successfully fixed. In addition, Clara and Sarfgen's assessments included a user study to evaluate the usefulness of the feedback provided.

Unlike previous evaluations, our study is focused only on analyzing the effectiveness of tools from the beginner student's perspective. We also developed a post-test to assess the impact of the tools on students' understanding of problem solving.

B. Program Visualization Tools

In programming education, most educational tools are focused on visualizing and animating program aspects based on its run-time execution [18]. These tools have been proposed to help students understand the dynamic execution of a program through a descriptive visualization. Examples of program visualization tools are Jeliot 3 [19], JIVE [20], VILLE [21], Whyline [22], Theseus [23], and Python Tutor [15]. These tools typically execute the program, store a snapshot of internal states at each execution step, and show a visual representation of run-time states such as stack frames, heap objects, and data structures. Recent studies have found that using program visualization tools can be pedagogically effective if students actively engage with the tool [24], [25].

Python Tutor [15] allows users to step forwards and backwards through execution to visualize the run-time state of a programs data structures. Karnalim et al. [26] evaluated Python Tutor based on a questionnaire survey applied in Basic Data Structure classes. The goal was to evaluate the impact of the tool to complete assignments, to understand programming aspects, and collect information about the students' experience.

Whyline [22] and Theseus [23] provide an overview of execution behavior and let a user find the cause of a bug through interactive question-answering or retroactive logging. In the evaluation of both tools, the authors asked the participants to identify the causes of bugs in given programs, using the respective tool to assist in the task.

In this paper, besides evaluating the usefulness of each tool from the student perspective, we also compared the effectiveness of using a program visualization tool (Python Tutor) with the use of an automatic feedback tool (Clara).

III. Study Design

In this study, we evaluated the effectiveness of Clara and Python Tutor in assisting novice programmers in problem solving. To do so, we conducted a controlled experiment to analyze whether using these tools students can solve programming problems better than when using only test cases.

A. Research Questions

- **RQ1:** *Do students using Clara or Python Tutor can solve problems faster than using only test case suites?* We are interested in analyzing whether Clara or Python Tutor can reduce the number of submissions needed to get a correct solution. This can be an indicator that the student is actually being helped by the tool.

- **RQ2:** *Do students using Clara or Python Tutor understand problem solving in the same level as when using test case suites?* We are interested in analyzing whether Clara or Python Tutor can impact the student's understanding

of how to solve the problem. It is important to investigate whether the tool is harming or benefiting student learning.

- **RQ3:** *Do students find Clara or Python Tutor more useful to fix bugs than test case suites?* Given the purposes of each tool, we are interested in finding out which approach students find most useful in bug fixing.

B. Tools

Test Cases: It is one of the most common practices used by teachers to provide feedback in programming classes. Generally, the teacher provides a set of test cases that describe the expected behavior of student programs for a given assignment. Thus, students can run their programs against the test suite and verify that their submissions are returning the expected result.

The feedback generated by test cases consists of indicating to which inputs a given program does not return the expected result. For example, consider a student program that should indicate whether a number is prime or not, however, the program does not return the expected result when the input value equals seven. The feedback provided to the student indicates which is the failed test, the expected result, and the result obtained.

GENERATED FEEDBACK

Testing: Prime Numbers of (7)
Expected: True
But got: False

Clara [7]: It is a fully automated program repair tool for introductory programming assignments. This tool can automatically repair incorrect programs, indicate the location of bugs (e.g., line number), and provide an exact textual description of required changes. The key idea of Clara is to use the existing correct student solutions to repair new incorrect student attempts. Their approach consists of clusters the correct programs for a given assignment and selects a canonical program from each cluster to form the reference solution set. Given an incorrect student attempt, it runs a trace-based repair procedure against each program in the solution set, and then selects a minimal repair from the repair candidates.

For example, consider the following incorrect student attempt for the *Prime Numbers* problem:

```
1  def is_prime_number(n):
2    n_div = 0
3    for i in range(1, n):
4      if n % i == 0:
5        n_div += 1
6    return n_div
```

In this example, Clara was able to identify two bugs in the given program as input. The first bug is found in the parameters of the *range* iterative expression *(line 3)*. The second bug is in the *return* statement at the end of the function *(line 6)*. As a result, the following corrections are suggested for the student program:

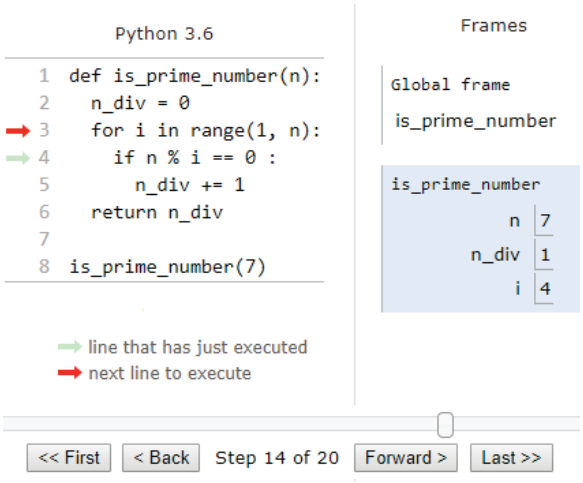

Fig. 1. The Python Tutor shows the state of variables and data structure in each line of the source code. Thus, students can visualize the execution of a program step-by-step.

GENERATED FEEDBACK

- In iterator expression at line 3, change *range(1, n)* to *range(1, n+1)*.
- In return statement at line 6, change *return n_div* to *return n_div == 2*.

Python Tutor [15]: It is a web-based program visualization tool for Python. This tool allows students to step forwards and backwards through execution to view the run-time state of a program's data structures. Students can use these features to debug their programs. As a consequence, they can fix bugs, and get correct solutions to programming assignments.

Given a program as input, the Python Tutor performs an analysis under supervision of the standard Python debugger module (bdb), which stops execution after every executed line and records the program's run-time state. As a result, an ordered list of execution points is produced, where each point contains the state right before a line of code is about to execute.

The Python Tutor GUI (shown in Figure 1) presents a source code panel that shows the program that is being visualized, the currently-executing line highlighted, a visual representation of run-time state (e.g., stack frame contents, heap objects), and control widgets to allow the user to step forwards and backwards over executed lines.

C. Participants

We recruited 42 undergraduate students from introductory programming classes. All participants are from Computer

Science or Engineering courses and are getting their first programming experiences with the Python language. They have knowledge in assigning variables, mathematical and boolean expressions, conditional structures (if / elif / else) and iteration using loops (for / while). In order to enable interaction with the tools, we developed an integration platform where participants can write their programs and get feedback for incorrect attempts. Our platform includes a panel to describe the programming assignment, a text box to write the program code, a section to show test case results, and a section to present the feedback provided by the tools.

D. Method

At the start of each study session, we provided an 8-minute tutorial on Test Cases, Clara and Python Tutor for all participants to familiarize them with each tool. We then asked them to implement solutions for three programming problems: *Sum of Squares*, *Prime Numbers* and *Fibonacci*. Participants were able to choose the order in which problems would be solved, however, we recommend that they begin with the problem that they find easiest. For each problem, participants should use one of the following conditions to get a correct solution:

- ***Condition 1*** - They could use only the Test Cases as assistant.
- ***Condition 2*** - They could use Clara and Test Cases as assistants.
- ***Condition 3*** - They could use Python Tutor and Test Cases as assistants.

All conditions were randomly assigned to the problems. In addition, the same condition can not be attributed to different problems of the same participant.

We presented the participants with a description of each problem and asked them to solve everything within the class time. Whenever the participant submits an incorrect attempt to the problem, the assigned tool will provide some feedback to help with the solution. Participants were able to submit as many times as needed until a correct solution was reached. We also do not allow participants to run their programs on other platforms beyond our tool integration platform. To verify that a solution is correct, our platform runs the code against a test suite related to the problem.

Once the participants get a correct solution, they should do a post-test related to the problem solved. The post-test consists of four solutions to the same problem. The participant needs to review the solutions and indicate whether each solution is correct or not. Finally, we conducted a post-survey where participants could rate which tools they find most useful for fixing bugs.

E. Problems

The following programming problems were proposed to the participants in this study:

- **Sum of Squares:** write a program that receives a positive integer *n* as input and returns the sum of the squares of the first *n* terms in a sequence: $1^2 + 2^2 + 3^2 + ... + n^2$.
- **Prime Numbers:** write a program that receives a positive integer *n* as input and returns *True* if *n* is a prime number, or *False* otherwise.
- **Fibonacci Sequence:** write a program that receives a positive integer *n* as input and returns the *n*th element of the Fibonacci sequence.

The programming problems used in this study were selected from exercise repositories of Computer Science courses. We chose to select problems that are common in introductory programming classes.

F. Post-test

We proposed a post-test to evaluate the impact of the tools on the student's understanding of how to solve the problem. The post-test is presented immediately after the student has reached a correct solution to a problem. For each problem, we collected a set of correct and incorrect programs, and then we created a data source from them. The programs were obtained from a pilot study session and also from storage bases of programming assignments.

A post-test for a specific problem contains four different solutions for it, which may be correct or incorrect solutions. The proportion of correct/incorrect and selection of solutions are randomly defined by our platform. The participant needs to review the solutions and indicate whether each solution is correct or not. We did not allow participants to run the solutions presented in the post-test. For each correct statement in the post-test, one point is accumulated in the participant's score on the problem addressed. As a result of the post-test, a range score of zero to four is generated to represent the student's understanding of how to solve a given problem. This score is associated with the tool the participant used to help solve the problem.

IV. Results

Overall, 42 undergraduate student from introductory programming classes participated in this study. In total, all participants produced 876 submissions. Our integration platform was able to provide feedback for 387 incorrect submissions. However, among other submissions, 9 failed in generation, 114 were correct and 366 had syntax errors. The tools evaluated in this study are not able to produce feedback when a submission contains syntax errors.

For purposes of analysis, we considered only correct student solutions that received some feedback in at least one of the submissions. Therefore, submission data from 42 correct student solutions were discarded. In total, we analyzed 72 correct student solutions, of which 23 were obtained using only Test Cases (TC), 21 were obtained working with Clara (CL) and 28 were obtained by interacting with Python Tutor (PT). In order to better understand the effect of the tools on student performance, we segmented our data into the following datasets:

- ***general***: includes student solutions to all programming problems addressed in this study.

- ***simplest***: includes only student solutions to *Sum of Squares* problem. Participants found this problem the easiest to solve.
- ***complex***: includes student solutions to *Prime Numbers* and *Fibonacci* problems. Participants found these two problems the most difficult to solve.

RQ1: *Do students using Clara or Python Tutor can solve problems faster than using only test case suites?* In general dataset, students who used only Test Cases required an average of 8.3 attempts to get a correct solution. This number decreased when students used Clara (4.05) or Python Tutor (6.68) as shown in Table I. These differences are statistically significant when comparing Clara with Test Cases (Z = -2.5, $p < 0.02$) and also when compared Clara with Python Tutor (Z = -2.65, $p < 0.01$) by Wilcoxon-Mann-Whitney test. However, we did not find a statistically significant difference comparing Test Cases with Python Tutor (Z = -0.66, $p > 0.5$).

We also consider getting results for the simplest and complex datasets separately. In the simplest dataset, we did not observe any significant difference in number of attempts. Possibly because students rely less on feedback tools to solve simple problems. The results for complex dataset confirm that Clara is able to reduce the number of attempts to get a correct solution when compared to other tools (Clara *vs* Test Cases: Z = -3.19, $p < 0.002$; Clara *vs* Python Tutor: Z = -2.14, $p < 0.03$). Finally, we did not observe a significant difference when comparing Python Tutor and Test Cases even in complex dataset.

TABLE I
SUMMARY OF RESULTS FOR THE NUMBER OF ATTEMPTS REQUIRED TO GET CORRECT SOLUTIONS.

NUMBER OF ATTEMPTS			
	TC	**CL**	**PT**
general	8.3 (7.2)	4.0 (2.5)	6.7 (5.3)
simplest	3.4 (1.9)	3.0 (1.4)	5.3 (2.5)
complex	10.4 (7.7)	4.3 (2.7)	7.3 (6.1)
	on average (SD)		

RQ2: *Do students using Clara or Python Tutor understand problem solving in the same level as when using test case suites?* We analyzed the results of the post-test as an indicator of the student's understanding of how to solve the problems addressed. Table II shows an overview of the scores obtained in our post-test. Our general analysis shows that there is no significant difference when comparing the post-test scores of students who used Test Cases (2.39) with students who used Clara (2.43). However, we found significant differences in scores obtained using Python Tutor (1.79), when compared to Test Cases (Z = -1.88, $p < 0.05$) and when compared to Clara (Z = 2.01, $p < 0.05$) by Wilcoxon-Mann-Whitney test.

In analyzing the simplest and complex datasets separately, we did not observe any significant differences. However, in trying to understand where the difference found in the general analysis came from, we found that the combination of data from *Sum of Squares* and *Prime Numbers* problems are major

TABLE II
SUMMARY OF RESULTS FOR THE POST-TEST SCORE.

POST-TEST SCORE			
	TC	**CL**	**PT**
general	2.4 (1.2)	2.4 (1.1)	1.8 (0.8)
simplest	2.3 (1.1)	2.7 (0.9)	1.9 (0.8)
complex	2.4 (1.2)	2.3 (1.2)	1.7 (0.8)
	on average (SD)		

responsible for these significant differences. In this dataset, students who used Python Tutor got an average score of 1.68, while students who used Test Cases and Clara got an average of 2.47 and 2.54, respectively. These differences are statistically significant when Python Tutor is compared to Test Cases (Z = -2.23, $p < 0.03$) and also when compared to Clara (Z = 2.11, $p < 0.04$).

RQ3: *Do students find Clara or Python Tutor more useful to fix bugs than test case suites?* We asked participants how much each tool was useful for fixing bugs in their programs. The results for this research question are shown in Table III. In general dataset, students scored Clara as more useful than Test Cases for bug fixes (Z = 2.10, $p < 0.04$) by Wilcoxon-Mann-Whitney test. However, we did not observe any significant differences when comparing the Python Tutor with Test Cases (Z = 0.45, $p > 0.6$) and Python Tutor with Clara (Z = 1.36, $p > 0.1$).

In the complex dataset, we also observed that Clara's score on utility for bug fixes was significantly higher than the score of Test Cases (Z = 1.90, $p < 0.05$). When we compared the Python Tutor with the other tools, no significant differences were found. In addition, no significant difference was found when analyzing the simplest dataset.

TABLE III
7-POINT LIKERT SCALE QUESTION - USEFULNESS IN FIXING BUGS.

HELP TO FIX BUGS			
	TC	**CL**	**PT**
general	5.3 (1.4)	5.9 (1.8)	5.4 (1.7)
simplest	6.2 (0.7)	6.7 (0.5)	6.0 (1.1)
complex	4.8 (1.5)	5.7 (1.9)	5.0 (1.9)
	on average (SD)		

V. DISCUSSION

Through the results we found that Clara can significantly reduce student effort, in number of attempts, to get correct solutions (RQ1). Using Clara students required an average of 4.05 attempts to get correct solutions, while using Python Tutor and Test Cases were required an average of 6.68 and 8.30 attempts, respectively. This result was already expected, since Clara provides specific hints on how to get a correct solution to programming problems. However, we also expected that the Python Tutor could reduce the number of attempts to get correct solutions. This was not observed in any of our analyzes. We thought that by debugging the code with Python

Tutor, students would solve problems faster than using Test Cases. We believe that there are two possible explanations for this result: (i) although we have provided a tutorial on Python Tutor, students may need more practice with the tool for better results; and (ii) as our study was conducted in introductory programming classes, students may not have enough experience for debugging activities.

In the post-test results, we found that students who used Clara and those who used only Test Cases got approximate scores (RQ2). This may mean that although Clara provides specific hints on how to solve program bugs, the student's understanding is not impaired. We noticed that, most of the time, the students were trying to understand why they should apply the hints given by Clara. This behavior may have led students to better understand how to solve problems, resulting in better performance in our post-test. This result is supported by recent studies that have found that specific hints, such as those provided by Clara, may be good for learning [27], [28].

In contrast, students who used Python Tutor got lower post-test performance than other tools. This is an unexpected result. We thought that by debugging the code, students would have an better overview of how to solve the problem. This result may be due to the unexpectedness of students with debugging activities. Another possible reason for this result would be that, when debugging their code, students focus only on a particular way of solving a problem. However, our post-test consists of analyzing different solutions to the same problem. In addition, recent studies have found that for more effective pedagogical results using program visualization tools, such as Python Tutor, students need to be actively engaged with the tool [24], [25].

Finally, students scored Clara as more useful than Test Cases to fix bugs in their programs (RQ3). They mentioned that finding bugs in their programs using only Test Cases was difficult, but using Clara, they could easily find out where the bugs were.

VI. Threats to Validity

In this section, we discuss possible threats to validity of our study. Our evaluation design sought to minimize the threats discussed whenever possible.

A. Construct validity

This study proposes a post-test to evaluate the student's understanding of the problem solution. However, the post-test score may not fully represent the student's understanding. There are many social aspects that can partially or totally affect the measurement of this construct. In addition, although we have observed a significant effect of the tools on the metrics, it is possible that the results found may not be entirely due to the tools used.

B. Internal validity

Since the study involves the active participation of humans, it is subject to internal threats. It is possible that the results were affected due to the moment and place where the experiments were conducted. Some of our study sessions happened in the classroom during class time. Participants were not previously advised that they would participate in this experiment. However, we let students know that their participation was not mandatory and that they could participate in the study in a private session. It is important to consider that students were solving programming problems, so it is possible that at some moment they are too tired or bored to perform their activities with involvement.

Participants in this study were recruited from Computer Science and Engineering courses. Although they are all enrolled in introductory programming classes, they may have different motivations and knowledge. Therefore, it is possible that some students were more experienced than others. To minimize this threat, we considered only in our analyzes the solutions that were correct, this ensures that the student was experienced enough to solve the problem. In addition, we discarded solutions from students who did not need any feedback to solve the problems, they were considered more experienced.

C. External validity

The participants of this study are representative only for the context of introductory programming subjects of local universities where the study was conducted. In addition, in our experiment only three programming problems and three tools were addressed. Therefore, we may not be able to generalize the results of this experiment to other contexts. For more general results, this study should be replicated in other introductory programming subjects.

VII. Conclusions and Future Work

In this article, we conducted user studies in introductory programming classes to evaluate the effectiveness of a tool for generating personalized hints (Clara) and a program visualization tool (Python Tutor). Specifically, we analyzed the effectiveness of these tools with respect to three aspects: (i) how fast a student can get a correct solution; (ii) what is the impact of the tool on the student's understanding of how to solve the problem; (iii) how useful is the tool to help students correct bugs in their programs. Participants solved three classic programming problems. For each problem, we provided a feedback tool and a test-case suite to assist in the resolution process. Once the participants get a correct solution, they should do a post-test related to the problem solved.

Our results show that, when considering getting correct solutions faster, Clara can significantly reduce the student's effort, in number of attempts, compared to other tools. We observed that students who used Clara and those who used only Test Cases got approximate scores. However, students using Python Tutor got lower post-test performance than using other tools.

As future work, we intend to: (i) evaluate other feedback tools used in programming education; (ii) establish guidelines for the use of feedback tools at each stage of programming learning; and (iii) develop a new post-test capable of assessing

other aspects of learning outcomes. Feedback has an important role in cognitive learning and is essential for improving knowledge and skills acquisition [29]. This article contributes to studies about the effect of feedback tools in programming learning. However, studies still need to be done to assess the quality of feedback provided by tools.

VIII. Acknowledgments

This work was partially supported by CNPq and CAPES grants.

References

[1] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year cs students," in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '01, 2001.

[2] M. Butler and M. Morgan, "Learning challenges faced by novice programming students studying high level and low feedback concepts," in *Proceedings of ASCILITE - Australian Society for Computers in Learning in Tertiary Education Annual Conference 2007*, 2007.

[3] A. T. Corbett and J. R. Anderson, "Locus of feedback control in computer-based tutoring: Impact on learning rate, achievement and attitudes," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '01, 2001.

[4] P. J. Guo, "Codeopticon: Real-time, one-to-many human tutoring for computer programming," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, ser. UIST '15, 2015.

[5] L. D'antoni, D. Kini, R. Alur, S. Gulwani, M. Viswanathan, and B. Hartmann, "How can automatic feedback help students construct automata?" *ACM Trans. Comput.-Hum. Interact.*, 2015.

[6] E. L. Glassman, J. Scott, R. Singh, P. J. Guo, and R. C. Miller, "Overcode: Visualizing variation in student solutions to programming problems at scale," *ACM Trans. Comput.-Hum. Interact.*, 2015.

[7] S. Gulwani, I. Radiček, and F. Zuleger, "Automated clustering and program repair for introductory programming assignments," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2018, 2018.

[8] K. Wang, R. Singh, and Z. Su, "Search, align, and repair: Data-driven feedback generation for introductory programming exercises," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2018, 2018.

[9] R. Suzuki, G. Soares, A. Head, E. Glassman, R. Reis, M. Mongiovi, L. D'Antoni, and B. Hartmann, "Tracediff: Debugging unexpected code behavior using trace divergences," *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2017.

[10] A. Head, E. Glassman, G. Soares, R. Suzuki, L. Figueredo, L. D'Antoni, and B. Hartmann, "Writing reusable code feedback at scale with mixed-initiative program synthesis," in *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, ser. L@S '17, 2017.

[11] R. Rolim, G. Soares, L. D'Antoni, O. Polozov, S. Gulwani, R. Gheyi, R. Suzuki, and B. Hartmann, "Learning syntactic program transformations from examples," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17, 2017.

[12] S. Kaleeswaran, A. Santhiar, A. Kanade, and S. Gulwani, "Semi-supervised verified feedback generation," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016, 2016.

[13] L. D'Antoni, R. Samanta, and R. Singh, "Qlose: Program repair with quantitative objectives," in *Computer Aided Verification*, S. Chaudhuri and A. Farzan, Eds., 2016.

[14] Y. Pu, K. Narasimhan, A. Solar-Lezama, and R. Barzilay, "Sk-p: A neural program corrector for moocs," in *Companion Proceedings of the 2016 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity*, ser. SPLASH Companion 2016, 2016.

[15] P. J. Guo, "Online python tutor: Embeddable web-based program visualization for cs education," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '13, 2013.

[16] K. Rivers and K. R. Koedinger, "Data-driven hint generation in vast solution spaces: a self-improving python programming tutor," *International Journal of Artificial Intelligence in Education*, 2017.

[17] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '13, 2013.

[18] S. BENTRAD and D. Meslati, "Visual programming and program visualization towards an ideal visual software engineering system ," *ACEEE International Journal on Information Technology*, 2011.

[19] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari, "Visualizing programs with jeliot 3," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, ser. AVI '04, 2004.

[20] P. Gestwicki and B. Jayaraman, "Interactive visualization of java programs," in *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, 2002.

[21] T. Rajala, M.-J. Laakso, E. Kaila, and T. Salakoski, "Ville: A language-independent program visualization tool," in *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88*, ser. Koli Calling '07, 2007.

[22] A. J. Ko and B. A. Myers, "Debugging reinvented: Asking and answering why and why not questions about program behavior," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08, 2008.

[23] T. Lieber, J. R. Brandt, and R. C. Miller, "Addressing misconceptions about code with always-on programming visualizations," in *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '14, 2014.

[24] C. D. HUNDHAUSEN, S. A. DOUGLAS, and J. T. STASKO, "A meta-study of algorithm visualization effectiveness," *Journal of Visual Languages & Computing*, 2002.

[25] J. Sorva, V. Karavirta, and L. Malmi, "A review of generic program visualization systems for introductory programming education," *Trans. Comput. Educ.*, 2013.

[26] O. Karnalim and M. Ayub, "The use of python tutor on programming laboratory session: Student perspectives," *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, 2017.

[27] B. Shih, K. Koedinger, and R. Scheines, "A response-time model for bottom-out hints as worked examples," 2008.

[28] M. Muir and C. Conati, "An analysis of attention to student – adaptive hints in an educational game," in *Intelligent Tutoring Systems*. Springer Berlin Heidelberg, 2012.

[29] V. J. Shute, "Focus on formative feedback," *Review of Educational Research*, 2008.