How Do Students Solve Parsons Programming Problems? – Execution-based vs. line-based feedback

Juha Helminen, Petri Ihantola, Ville Karavirta and Satu Alaoutinen
Department of Computer Science and Engineering
Aalto University
Espoo, Finland
Email: firstname.lastname@aalto.fi

***Abstract*—In large introductory programming classes, there typically are no resources for adequate individual guidance. Automatic feedback for programming tasks can facilitate students' learning by allowing them to get immediate individual feedback regardless of time and place. This paper presents a study on how the type of automatic feedback in Parsons problems affects how students solve them. Students on their first programming class were divided into two groups and, in two assignments, each group in turn received different type of feedback. The type of feedback had an effect on how students constructed their programs and how quickly they were able to complete them. With feedback based on execution as opposed to the visible arrangement of code, the programs were more frequently executable when feedback was requested and, overall, feedback was requested less frequently. Based on the analysis, we discuss possible future improvements to automatic feedback in this type of an assignment.**

I. Introduction

Ultimately, learning to program requires practice. For this process of improvement, continuous feedback is an essential component. At our institution, we have very large introductory programming classes and providing enough individual guidance is thus a major challenge. Automated ways of providing feedback to students are therefore of special interest to us. This paper describes an experiment to analyze and compare two different types of automatic feedback in a type of a programming assignment called the Parsons problem.

Parsons problems are scaffolded, simplified program construction tasks where given a set of code fragments in random order, the task is to select and arrange some of them in order to compose a program that meets some set requirements [1]. Typically, a fragment is a single line of code but may also consist of multiple consecutive lines. Previously, we have implemented a drag-and-drop web environment for Parsons problems in Python called js-parsons [2]. In Python, code blocks are defined by indentation and an indented statement falls into the block of a surrounding (control) structure that has lower indentation. Indeed, in js-parsons, students are not only required to order code fragments but must also indent them appropriately. The system provides immediate unlimited feedback on request. Previously, feedback was always given in terms of the order and indentation of the expected correct solution by highlighting code fragments that need to be moved somehow in order to fix the program. We call this *line-based feedback* and its implementation is described in detail by Karavirta *et al.* [3]. The possibility to give such precise suggestions for correcting the program is based on the assumption that the assignment is designed in such a way that there only exists a single correct arrangement of the given code fragments that meets the requirements.

Js-parsons logs all learner interaction within the environment including any operations on the code fragments and all feedback requests. To better understand the process that students follow in solving these assignments, we have previously analyzed and visualized these detailed interaction traces of students' solving sessions [4]. In that study, we observed some undesirable patterns of behavior such as trial-and-error strategies and cycles in the solution paths. Indeed, many students requested feedback excessively and some ended up going in circles in terms of the partial program they had constructed. Since then, we have designed and implemented some improvements to the original feedback mechanism of js-parsons to account for these behaviors [3]. However, these improvements were not deployed in the experiment described in this paper.

This paper presents two primary contributions to the existing knowledge of automatic assessment and feedback on Parsons programming problems:

- We have extended js-parsons with what we call *execution-based feedback* where tests are run on learner's code and then checked for expected results. In our implementation, the Python code is executed directly within the browser environment.
- We have compared the new execution-based feedback and the previously existing line-based feedback in terms of how students use feedback and how it affects their behavior while solving the problems.

II. Background

A. Automatic Assessment of Programming

Automatic assessment of programming assignments has been studied since the 60's. Douce *et al.* [5] describe the history of automatic assessment of programming assignments by dividing the tools into three generations: monolithic systems used by teachers only (1960s and 1970s), scripts and other command line utilities that were also able to provide feedback to the students (1980s' and 1990s), and web-based automatic assessment tools (since late 1990s). Since then, it has also been argued that web-based assessment platforms, where work is submitted to a remote server for grading, would be moving

978-0-7695-4960-6/13 $26.00 © 2013 IEEE
DOI 10.1109/LaTiCE.2013.26

towards assessment on the client side [6]. The work presented here, falls into this category.

Automatic assessment of programs can be based on various aspects and skills demonstrated. Ala-Mutka, for example, lists functionality, efficiency, tests, style, programming errors detectable in static analysis, various metrics such as cyclomatic complexity, design and special features [7]. Despite many possibilities, automatic assessment of functionality that is based on program execution, *i.e.* testing, is the most common way to give feedback [5], [8], [9]. According to Ihantola *et al.* [9], there are several ways to approach this task: using industrial testing tools, examining and comparing program output with test input, scripting, and other experimental approaches. Industrial unit testing tools are used, for example, in Web-CAT [10] and Marmoset [11]. Ala-Mutka reported on several systems using output comparison [7]. Scripting, again according to Ihantola *et al.*, can have different meanings, such as running a script and comparing the result saved by that script to an expected answer. In this paper, we introduce an implementation of execution-based automatic assessment and feedback for Parsons programming problems. Our solution falls somewhere between unit testing and scripting.

Although visualizations are widely utilized in computing education, visual feedback from the functionality of programming assignments is rarely used [6], [12]. A more common way to give feedback on the functionality is simply to tell the input and the expected vs. the observed output to the learner in a textual format. This is also what most unit testing libraries do. It can be argued that the type of feedback originally used in js-parsons, the line-based feedback, is visual, whereas the new execution-based feedback described in this paper takes after traditional unit testing.

B. Parsons Programming Problems

In addition to the code fragments given for selection and arrangement, Parsons programming problems may contain some static, immovable code around the blanks to be filled with the appropriate code fragments. Garner [13] has argued that this is important because it ties problems to a larger context and allows for more complex programs without making the task too difficult. Another variation is to have some extra fragments, known as *distractors*, that are not actually needed in the final program. Distractors generally make Parsons problems more challenging as they grow the set of possible program compositions. They can also be used to drill students' syntactical knowledge by providing alternative versions of a fragment with some minor syntactical errors. The js-parsons environment has support for distractors but not for static parts of code. Another approach to increase the difficulty and to move towards more free-form program construction is to allow creating blocks, for example, like in js-parsons with Python and its use of indentation. Different variations and studies on Parsons programming problems are discussed more in-depth by Helminen *et al.* [4].

Automatically assessing the correctness of small Parsons programming problems is straightforward. In that case, Parsons problems are essentially multiple-choice questions where the single correct arrangement is the answer. Indeed, originally, Parsons problems were implemented on top of a multiple-choice question engine called Hot-Potatoes [1]. A similar

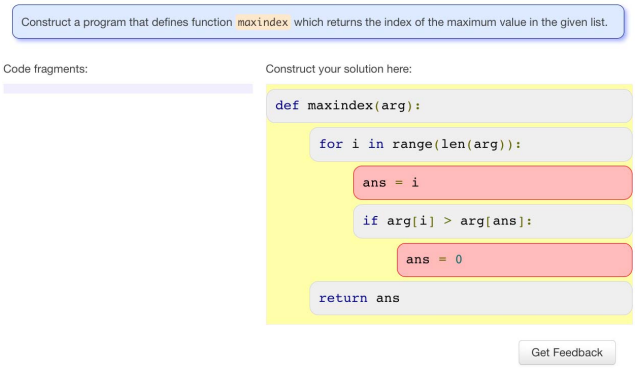

Fig. 1. Feedback based on code fragment positions and indentation in js-parsons, *i.e.*, line-based feedback.

assessment approach has also been utilized in js-parsons [2] and its mobile extension MobileParsons [3] – although in both of these tools code fragments are placed on a two dimensional grid and this requires some preprocessing of the data (*e.g.* normalizing the indentations). The challenge in this approach of assessing correctness purely based on the code arrangement is that, in many cases, especially when the number of code fragments grows, the correct order is no longer unique. There may exist multiple possible arrangements that meet the requirements. Avoiding such situations is one of the challenges in designing non-trivial Parsons problems when using this approach to assess correctness [2].

A separate but related aspect is how to provide feedback on Parsons problem solution attempts beyond the correct/incorrect label. The Hot-Potatoes implementation simply had slots for each line and it pointed out those that contained a correct one. In js-parsons, there are no slots but the learner can freely add, remove, and rearrange lines (code fragments), and the feedback points out one possible set of fragments that need to be moved to fix the program [3]. See Figure 1 for an example. On the other hand, Cort, for example, does not provide automatic feedback, but it encourages students to copy the code to an external interpreter where they can experiment with the program [13]. Finally, although the feature has not been reported in the literature, the ViLLE system [14] seems to support Parsons problems where the assessment and feedback is based on executing the learner's code on a remote server and comparing the output of the code to an expected output.

III. Implementation of Execution-Based Feedback Extension to js-parsons

The line-based feedback has two primary drawbacks. First, as mentioned, in a previous study we found that the feedback may allow or perhaps even encourage students to follow a trial-and-error strategy without truly thinking about the solution [4]. Second, for teachers creating assignments, the requirement of a unique correct solution significantly limits the types of assignments that can be designed. For example, assignments with classes are problematic, since the order in which methods and fields are specified makes no difference to the functionality of the class. Additionally, on the one hand, we like that the line-based feedback gives a direct suggestion to the student

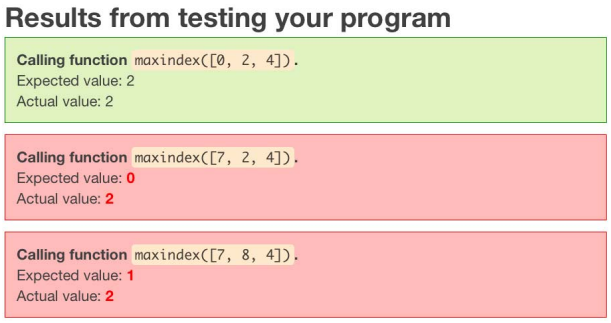

Fig. 2. Example of execution-based feedback for the solution shown in Figure 1.

on how to fix the program but, on the other hand, it does not clearly relate to the execution and flow of the program, *i.e.*, how it works incorrectly.

As an attempt to tackle this, we added support for execution-based feedback as an extension to the js-parsons tool. Our premise was to add this feature with as little effect to the way the assignments for the tool were specified. Essentially, this meant that the same assignment should work with the existing line-based feedback if no execution tests were specified (and there was a unique solution), and that every existing assignment could be changed to give execution-based feedback with only adding specifications for the tests.

From a student's point of view, the assignments look exactly the same as before. Students interact by drag-and-dropping lines to their solution just like before but when a student requests feedback, instead of showing line-based feedback, an assignment with execution-based feedback executes the student's current code with teacher-specified input data. The results of the execution are presented in a similar manner to what is typical for unit tests. See Figure 2 for an example. For each test, the student sees a description of the test which typically includes the function call or the name of the variable that the test examines. The result of each test can be:

- **A passed test** In this case, the test is colored green to indicate correctness.
- **Test fails an assertion** In this case, the test is colored red to indicate incorrectness. The student is also shown the expected value and the actual value produced by the student's program.
- **Test fails to parse/execute the program** Again, the test is colored red. In addition, the student is shown a Python-like exception message. This message can be, for example, "NameError: name 'maxindex' is not defined" in an assignment where the task is to specify a function named maxindex.

From a technical point of view, the js-parsons tool works entirely on the client-side, *i.e.*, in the browser. We wanted to keep it that way to avoid requiring instructors to install some backend server in order to use the assignments. This requirement meant executing the Python code in the browser. For this, we decided to use Skulpt[1], a JavaScript implementation of Python. Skulpt has previously been succesfully used in other larger projects such as the Runestone Interactive[2].

The programs in Parsons problems assignments are typically quite simple. Thus, we did not see a reason to adopt or implement a full-fledged unit testing framework. Instead, we decided to implement lightweight testing capabilities. In our implementation, all tests assert a value of a single variable after executing the student's code and the test code. The test definition is composed of a message to be shown to the student, code to be executed (in addition to the student's code), variable to inspect, and the expected value of the variable. An assignment can specify as many tests as necessary.

IV. Data Collection

To evaluate the effect of the two different types of feedback – line-based and execution-based – we divided students in a large introductory programming class into two groups, had both groups in turn use them both, and then analyzed the interaction traces of students' solving sessions as recorded by js-parsons. The js-parsons tool records each operation – changing the position or indentation of a fragment, adding and removing fragments, and feedback requests – and logs the current state of the student's code with a timestamp. Additionally, when feedback is requested, the feedback given to the student is also logged and the trace recorded so far is sent to a server for storage.

The study was conducted on the first programming course that students take when they first start at the university. Its focus is on object-oriented programming using Python. Most of the students are freshmen coming from the Computer Science, Electrical Engineering, Communications Engineering, Automation and Systems Technology, Engineering Physics and Mathematics, and Geomatics departments. The course has over 400 students and automatic assessment of programming assignments has been used on the course for years. Parsons programming problems had not been used previously on the course.

The course has 10 rounds of automatically assessed programming assignments and Parsons problems were included into the beginning of the first five. The weight of Parsons problems assignments in the grading scheme was quite insignificant – solving all of them gave 90 points whereas a total of 3190 points were available. Despite this, submission counts indicate that students were still well motivated to solve the them.

The js-parsons assignments used on the course are briefly described in Table I. Some of the assignments provided line-based feedback whereas others used the new execution-based feedback. Students were randomly divided into two groups (A and B) based on their student identification number. The third round included the research setup to compare the two types of feedback. In assignment 3.1, group A received execution-based feedback and group B line-based feedback. In assignment 3.2, the type of feedback was switched between the groups, so group B received the execution-based feedback. To ensure both groups were familiar with the execution based-feedback before the research setup, it was first introduced in assignment 2.2.

[1]http://www.skulpt.org/

[2]http://interactivepython.org

TABLE I. PARSONS PROBLEMS ASSIGNED ON THE COURSE. THE ASSIGNMENT ID INDICATES THE ROUND AND THE ASSIGNMENT'S POSITION ON THAT ROUND. FEEDBACK COLUMNS INFORM WHICH TYPE OF FEEDBACK EACH OF THE GROUPS RECEIVED IN THAT ASSIGNMENT.

Assignment	Topic	Solution Code Fragments / Distractors	Group A Feedback	Group B Feedback
1.1	Arithmetics and assignments	4 / 1	line	line
1.2	Arithmetics and assignments	4 / 1	line	line
2.1	Functions	6 / 2	line	line
2.2	If statement	10 / 0	execution	execution
3.1	For statement (find the max element from an array)	6 / 0	**execution**	**line**
3.2	While statement (merge sorted lists)	11 / 0	**line**	**execution**
3.3	Functions	14 / 0	execution	execution
4.1	Objects	13 / 0	execution	execution
5.1	Objects and classes	6 / 0	line	line

The results of analyzing the traces from this setup are reported in Section V.

Finally, after the round five was closed, all students (N=445) received a link to an anonymous survey, where we asked for their opinions about the js-parsons tool generally and about the two different types of feedback. The results of this survey are reported in Section V-C.

V. ANALYSIS AND RESULTS

In the following description and analysis of students' interaction traces of solving assignments in js-parsons, we use a specific terminology to refer to the components of the trace. The full trace of a student correctly solving, or *completing*, the assignment is called a *solving sequence*. A solving sequence may consist of several *solving sessions* each of which corresponds to a single session in the js-parsons environment where the assignment's web page was loaded and it was first initialized with an empty program construction area. Indeed, the student may at any time interrupt their current solving session and start a new one from scratch immediately or later on by reloading the page.

Furthermore, we have represented solving sequences as sequences of states of the student-constructed code in the program construction area. Each operation the student performs, such as moving or indenting a code fragment, changes this state thus leading to a state transition. We call these transitions *steps*. A solving sequence can thus also be thought of as a directed graph where the nodes correspond to states of the student's program code and edges correspond to operations invoking a transition from one state to another. The events in the recorded interaction traces translate to these concepts with little effort. However, in addition, in order to minimize the length of the sequences some additional filtering is done, such as combining consecutive actions that move the same code fragment into one move action. Finally, in comparing and analyzing these solving sequences, we only consider those that were completed. Overall, there were relatively few students that started a solving session but ultimately never finished the assignment. Additionally, we dropped sequences where the student had been solving the assignment in multiple parallel sessions, probably in multiple tabs. There were only three occurrences of this in the whole data set across all the students and assignments.

A. Statistical Analysis

The following statistical analyses focus on the assignment 3.1 where a group of students received execution-based feedback while the other group received line-based feedback. In assignment 3.2, the groups were reversed. However, due to a bug in recording traces in js-parsons, we cannot infer anything solid from assignment 3.2. In this assignment, the student could construct a program that creates an infinite loop and with execution-based assessment this could cause a small part of the trace to be lost. Based on the frequency of states where students with line-based feedback requested feedback the data loss is likely quite insignificant but we cannot be sure. We can say that with the data we have on 3.2, we get the exact same results as presented below for assignment 3.1 and thus it seems to confirm our findings. However, since the traces may be incomplete to an unknown degree, we do not discuss this assignment any further.

Statistics on students' solving sequences in assignment 3.1 are shown in Table II. For each metric, the table shows the median, the first (25%) quartile, and the third quartile (75%) of the distribution of values. The first thing to notice from the table is the generally high amount of long solution sequences, especially the much longer sequences in the group with execution-based feedback. The amount of feedback requests in the third quartile is quite perplexing as well. After all, the assignment had only 6 code fragments, none of which were distractors. Still, the highest amount of feedback requests was as high as 409.

In addition to the descriptive statistics, the last column in the table shows the p-value for a statistical test performed to compare whether the observed values in the two groups are likely to be independent or sampled from the same population. Mann-Whitney-U test was used in all these tests for the difference of the location of the distributions. There was no statistically significant difference in how many steps there were in the solving sequences before the first feedback request. Unless the groups were different, such as composed of students of sufficiently different levels, this is to be expected because at this point the students have not interacted with the feedback. Furthermore, we can observe that there is no statistically significant difference in how many times students have requested feedback. It appears the nature of the feedback has not affected this. However, in all the other metrics listed in Table II, there was a statistically significant difference. Moreover, there was no such difference between the groups

in any other assignment in terms of any of the metrics listed in the table (except, of course, in assignment 3.2 as discussed above). This suggests that the observed differences are indeed due to the different types of feedback. If the assignment was not completed on the first feedback request, students who received execution-based feedback on average needed more steps to solve the assignment after the first feedback, had more solving sessions, and spent a longer time span trying to solve the assignment, that is, more time elapsed between the time they first tried to solve the assignment and when they eventually finished. Finally, students with execution-based feedback requested feedback less frequently.

B. Program Quality

Since in these assignments students are not actually running their code themselves, we wanted to investigate how students' programs evolve and are built in terms of their syntactical correctness and executability. For this, we analyzed the feedback states for both groups in all the assignments with one or both groups receiving execution-based feedback. Results of this analysis for assignment 3.1 are summarized in Table III. The metrics in the table are for the states that failed to execute, states that had a syntax error (a subset of the previous), and states that had an indentation error (a subset of the previous). To calculate the metrics, each feedback state in the solution sequence was tested. For the sequence, the percentage of feedback states where the solution failed to execute is reported.

The results show that the group who received execution-based feedback had significantly less feedback states where their code failed to execute. Furthermore, they had fewer feedback states with syntax and indentation errors. The same statistically significant differences were also found in assignment 3.2. In the assignments where both groups received execution-based feedback, there were no differences.

TABLE III. STATISTICS OF SYNTACTIC CORRECTNESS AND EXECUTION ERRORS OF STUDENTS' PROGRAM AT THE TIME OF FEEDBACK REQUESTS. GROUP A (N=216) RECEIVED EXECUTION-BASED FEEDBACK AND GROUP B (N=165) RECEIVED LINE-BASED FEEDBACK.

	25%	med	75%	p
Failed to execute				
Group A	33.3%	55.4%	75.0%	0.033
Group B	33.3%	66.7%	83.3%	
Failed due syntax error				
Group A	0.0%	33.3%	51.7%	0.002
Group B	0.0%	50.0%	79.6%	
Failed due indentation error				
Group A	0.0%	29.1%	50.0%	0.003
Group B	0.0%	50.0%	76.8%	

C. Students' Perceptions

1) Background Data: Students' perceptions were surveyed with a multiple-choice questionnaire that also allowed free text feedback. The survey was sent to all 445 students that had at least logged in to the automatic assessment system. 163 of them submitted the survey (37%). Most of them were Computer Science students but there were many respondents from other majors too, as can be seen in Table IV. The majors that were represented by less than four respondents were grouped together as others.

TABLE IV. MAJORS OF THE RESPONDENTS.

Major	Number of respondents
Computer Science	50
Engineering Physics and Mathematics	27
Automation and Systems Technology	25
Geomatics	16
Electrical Engineering	14
Communications Engineering	8
Others	23

90% of the respondents were doing their bachelor's degree. The rest were either master's students, graduate students, or not students at the university. Most of the respondents were first year students (64%). 16% were second year students, 8% third year, and the remaining 14% had more years or were not students at the university.

The students were also asked about their assignment points at that moment. Table V shows the categories and how the points were distributed. Most of the respondents were in the 1500 - 2000 points category. This means that they had already earned a grade 1 out of 5 and were working for grade 2. The high number of students in the last category means that they had already done all the assignments so that they could participate in a more challenging version of the course.

TABLE V. DISTRIBUTION OF ASSIGNMENT POINTS.

Category	Number of respondents
0 - 499	2
500 - 999	5
1000 - 1499	30
1500 - 1999	83
2000 - 2499	18
2500 - 3190	25

2) Js-parsons Assignments: The survey also included a question on the assignments' level of difficulty. A large majority (102 students out 163) of the students found the difficulty of the js-parsons assignments suitable for them. This was the only question where the background data had any correlation with the other questions. Those students who had less points, felt the assignments were difficult and vice versa (Spearman $r_s = 0.35, p < 0.01$).

Students generally solved the assignments entirely in the browser environment. 18% had used an additional separate editor and 24% had executed code elsewhere in at least one assignment, and when they had done one, they had done both (Spearman $r_s = 0.55, p < 0.01$).

Students were also asked to express whether they agree or disagree with some statements. Table VI shows the statements and the results.

Reading from the Table VI, we see that the students' attitude towards js-parsons assignments was mostly positive. With regard to line-based and execution-based feedback and how they support learning, it is not an either-or situation. The answers correlate quite strongly (Spearman $r_s = 0.47, p < 0.01$) meaning that if a student felt that the line-based feedback supported learning, he or she felt that the execution-based feedback supported learning too. Only a minority of the students preferred one over the other (N=38, 23%) and

TABLE II. STATISTICS OF STUDENTS' SOLVING SEQUENCES IN ASSIGNMENT 3.1. GROUP A (N=216) RECEIVED EXECUTION-BASED FEEDBACK AND GROUP B (N=165) RECEIVED LINE-BASED FEEDBACK.

	25%	med	75%	max	p
Steps before 1st feedback					
Group A	5	6	8	17	0.145
Group B	6	6	8	20	
Steps after 1st feedback					
Group A	7	16.5	62	787	< 0.001
Group B	4	8	21.8	106	
Feedback count					
Group A	2	6	28.5	409	0.100
Group B	2	6	13	59	
Session count					
Group A	1	1	3	14	< 0.001
Group B	1	1	1	4	
Feedback frequency					
Group A	16.5%	37.6%	57.1%	100%	< 0.001
Group B	29.5%	54.5%	80%	100%	
Sequence duration					
Group A	2m 43s	36m 49s	1d 23h	23d	< 0.001
Group B	35s	1m 34s	9m 14s	14d	

TABLE VI. STUDENT OPINIONS ABOUT THE JS-PARSONS ASSIGNMENTS.

	Completely disagree	Mostly disagree	Mostly agree	Completely agree	Cannot say
The js-parsons assignments were nice.	13	25	78	44	4
I really thought about the js-parsons assignments when solving them.	9	27	80	47	1
The js-parsons assignments supported my learning well.	11	45	69	29	8
The line-based feedback supported my learning.	9	44	69	33	8
The execution-based feedback supported my learning.	12	41	69	29	11
The js-parsons assignments helped me to understand program structure and syntax.	7	38	81	34	4
The js-parsons assignments helped me to understand other programs.	9	30	73	35	5
The js-parsons assignments helped me to write programs.	6	51	70	29	5
The js-parsons assignments were not related to the rest of the course.	35	64	40	13	11
The js-parsons assignments are worth using in the future.	11	20	59	65	6

some could not tell either way (N=17, 10%). Of those 17 students who preferred execution-based feedback, 14 felt that the assignments were suitably difficult. On the other hand, of those 21 students who preferred line-based feedback, 8 felt the assignments were too difficult and 9 regarded them as suitable. However, this difference is not statistically significant (Mann-Whitney U test: $p = 0.25$). The two groups were also similar considering the assignment points they had earned.

There was also a possibility to give free text feedback about the js-parsons assignments. In total, 59 students wrote something. According to the comments:

- The assignments are useful, especially at the beginning of the course, when the basic concepts should be learned.
- If the assignments are not too difficult, they can act as an introduction to the actual coding assignments of a round.
- The assignments teach code structure and syntax.
- Some assignments were too difficult to understand (like the Pythonic way to compute prime numbers).
- If an assignment was too difficult to understand, you would end up just trying out different arrangements of lines.
- Execution-based feedback did not tell what went wrong.
- The students most experienced in programming felt js-parsons assignments were too easy and useless.

In addition, some technical improvements to the system were suggested. These included the possibility to save a solution and return to it later and showing a grid while dragging in order to make indentation easier and more clear.

VI. DISCUSSION

The overall attitude towards js-parsons assignments was mostly positive and students also found their difficulty to be appropriate. However, when compared to the significant amount of desperately long solution sequences, these results from our survey are in some sense surprising and raise some concerns about their validity. While 37% of students did take the survey, this is still a self-selected sample with a possible bias.

With regard to which feedback seems more preferable, as the results show, with feedback based on execution, students pay more attention to ensuring that their code is syntactically correct and executable. This, in itself, is a good thing as it may indicate that students are more inclined to think of their solution as a whole instead of just somewhat blindly focusing on the lines highlighted by the feedback like they may end up doing with the line-based feedback. However, this is to be expected since the execution-based feedback does not help the student at all if the program is not syntactically correct or fails to execute due to some other error.

When it comes to designing these types of assignments, line-based feedback as it is implemented now requires the solution to the assignment to be unique. With even slightly more complicated programs, finding sensible assignments with such a solution becomes difficult. With execution-based feedback, there is no such limitation, which makes the tool suitable for a larger set of programs. However, writing comprehensive tests is of course no trivial task either. In fact, we had to slightly modify some tests because incorrect answers passed the original tests. The assignments analyzed in more detail in this paper did not change during the course.

Overall, drawing upon this study, we are not convinced that either type of feedback is the better option. Especially if the programs in the assignments are more complex, students seem to struggle and generate extremely long solving sequences regardless of the feedback. In both groups, some students also relied on a trial-and-error solution strategy. This was somewhat surprising since the execution-based feedback does not really give hints on how to improve the solution unless you actually try to trace and understand the flow and execution of the program. We believe the usefulness of the execution-based feedback could be improved by additionally providing visualizations of the executions of failing tests. This kind of feedback is provided in Parsons problems -like code sorting assignments in the ViLLE system.

VII. Conclusions

In this paper, we have introduced an extension to the js-parsons Parsons problems web environment that gives students feedback based on the execution of their solution with test data. We have demonstrated how the execution-based feedback can be constructed on the client-side, *i.e.*, in the browser. The benefit of this is that js-parsons widgets, even with the execution-based feedback, can be embedded to any web page without the need of a separate assessment server. To evaluate this new type of feedback on an introductory programming course, we carried out an experiment where we randomly divided the students into two groups and gave the groups different types of feedback in two assignments. Our analysis shows that the type of feedback has a significant effect on how students solve the assignments. With execution-based feedback, feedback was requested more often in states where the student's solution was syntactically correct and less frequently resulted in runtime exceptions when executed. Also, students with execution-based feedback took more steps after the first feedback request to complete the assignment and requested feedback less frequently.

Both the line-based and the execution-based feedback have their strengths and weaknesses. In the future, we will study how to identify students struggling with the execution-based feedback and, only then, give line-based feedback in an attempt to more precisely guide them towards the correct solution. Furthermore, we will investigate ways to give more meaningful feedback based on the block structure of the program instead of merely the line positioning and indentation. Additionally, program visualizations of the execution of the tests might be beneficial to students. Finally, we need to study more closely how the feedback affects students' learning in addition to analyzing their behavior.

References

[1] D. Parsons and P. Haden, "Parson's programming puzzles: a fun and effective learning tool for first programming courses," in *ACE '06: Proceedings of the 8th Australian conference on Computing education*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 157–163.

[2] P. Ihantola and V. Karavirta, "Two-dimensional parson's puzzles: The concept, tools, and first observations," *Journal of Information Technology Education: Innovations in Practice*, vol. 10, pp. 1–14, 2011.

[3] V. Karavirta, J. Helminen, and P. Ihantola, "A Mobile Learning Application for Parsons Problems with Automatic Feedback," in *Proceedings of the 12th Koli Calling International Conference on Computing Education Research (Koli Calling '12)*. ACM, November 2012, pp. 11–18.

[4] J. Helminen, P. Ihantola, V. Karavirta, and L. Malmi, "How Do Students Solve Parsons Programming Problems? – An Analysis of Interaction Traces," in *Proceedings of the Eighth Annual International Computing Education Research Conference (ICER '12)*. ACM, September 2012, pp. 119–126.

[5] C. Douce, D. Livingstone, and J. Orwell, "Automatic test-based assessment of programming: A review," *Journal on Educational Resources in Computing*, vol. 5, no. 3, pp. 1–13, 2005.

[6] P. Ihantola, "Automated assessment of programming assignments: Visual feedback, assignment mobility, and assessment of students' testing skills," Doctoral Dissertation (Aalto University publication series, doctoral dissertations, 131/2011), Aalto University, 2011.

[7] K. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer Science Education*, vol. 15, no. 2, pp. 83–102, 2005.

[8] J. Carter, J. English, K. Ala-Mutka, M. Dick, W. Fone, U. Fuller, and J. Sheard, "ITICSE working group report: How shall we assess this?" *SIGCSE Bulletin*, vol. 35, no. 4, pp. 107–123, 2003.

[9] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '10. New York, NY, USA: ACM, 2010, pp. 86–93.

[10] S. H. Edwards, "Rethinking computer science education from a test-first perspective," in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, Anaheim, California, USA, 26–30 October*. ACM, New York, NY, USA, 2003, pp. 148–155.

[11] J. Spacco, D. Hovemeyer, W. Pugh, F. Emad, J. K. Hollingsworth, and N. Padua-Perez, "Experiences with marmoset: designing and using an advanced submission and testing system for programming courses," in *ITICSE '06: Proceedings of the 11th annual SIGCSE Conf. on Innovation and technology in computer science education*. New York, NY, USA: ACM, 2006, pp. 13–17.

[12] P. Ihantola, V. Karavirta, and O. Seppälä, "Automated visual feedback from programming assignments," in *Proceedings of the Sixth Program Visualization Workshop*, Darmstadt, Germany, 2011, pp. 87–95.

[13] S. Garner, "An exploration of how a technology-facilitated part-complete solution method supports the learning of computer programming," *Journal of Issues in Informing Science and Information Technology*, vol. 4, pp. 491–501, 2007.

[14] T. Rajala, M.-J. Laakso, E. Kaila, and T. Salakoski, "VILLE — a Language-Independent Program Visualization Tool," in *Seventh Baltic Sea Conference on Computing Education Research*, 2007, pp. 151–159.