Providing Better Feedback for Students Solving Programming Tasks Using Project Tomo

Gregor Jerše
Faculty of Computer and Information Science
University of Ljubljana
Ljubljana, Slovenia
Gregor.Jerse@fri.uni-lj.si

Matija Lokar
Faculty of Mathematics and Physics
University of Ljubljana
Ljubljana, Slovenia
Matija.Lokar@fmf.uni-lj.si

***Abstract*—Systems for automatic assessment of programming tasks have become a popular choice in programming courses as several advantages of using automatic assessment in teaching and learning process have been observed. One of the most important is the immediate feedback students get. However, the quality of the feedback is essential to achieve good learning results. At the University of Ljubljana we use our proprietary system called Project Tomo as a teaching tool. One of the most important aspects of our system is the possibility to return a detailed feedback and to analyze the student's solution since every submission is stored on the server. Until now we have collected more than 110,000 attempts along with their history. We are currently in the process of analyzing them. Currently we are concentrating on how to use this data to further improve the quality of the feedback given to the student. Some of the observations and the preliminary results are presented in the paper.**

I. Introduction

Teaching a beginner level programming course can be quite challenging. Since programming is a skill, it can be learnt best by solving as many programming tasks as possible. As Lee and Ko write in [1], for most beginners, the experience of writing computer programs is characterized by a distinct sense of failure. The first code beginners write, often leads to unexpected behaviors, such as syntax errors, run-time errors, or unintended program output. While all of these forms of feedback are essential helping a beginner understand what programs are and how computers interpret them, the experience can be quite discouraging [2], [3] and emotional [4]. As several researchers have pointed out, for example in [5], feedback is an important factor in the learning process.

Teachers should provide the students with the feedback beyond one that normal tools (compilers, interpreters and run time environments) provide. But the feedback must be immediate, otherwise the students can get stuck which slows down their progress considerably. Keuning, Jeuring and Heeren [6] write "*Formative feedback, aimed at helping students to improve their work, is an important factor in learning.*" Also Campos et al. [7] following [8] conclude that good feedback is essential for improving the students progress. They can learn more effectively if they receive quick and appropriate feedback about their actions in a short amount of time. In addition to its influence on the students' achievement, feedback is also a significant factor in providing motivating for learning. As Nicol states[1] *Assessment and feedback practices should be designed to enable students to become self-regulated learners, able to monitor and evaluate the quality and impact of their own work and that of others.*

But providing timely and instant feedback in overcrowded classrooms is a tough task. This calls for a tool that can automatically provide immediate feedback. In their paper Keuninig et all. [6] studied what kind of feedback is provided by systems for automated assessment of programming tasks (SAAP), which techniques are used to generate the feedback, how adaptable the feedback is, and how these tools are evaluated.

The paper is organized as follows. In Section II we describe our SAAP solution called Project Tomo and its properties. In Section III we present the results of the analysis of students' submissions and the conclusion in Section IV.

II. Project Tomo

After evaluating several SAAP (a systematic review can be found in [9]–[11]), we came to the similar conclusion as Keuninig et al. in [6] that "*Most SAAP tools only grade student solutions*" and "*tools do not often give feedback on fixing problems and taking the next step, and that teachers cannot easily adapt tools to their own needs.*". Also Rubio-Sanchez et al. mention in [13] "*despite acknowledging that using Mooshak (SAAP tool) was a good idea, students did not appreciate the experience as a whole, where the main reported drawback was related to its feedback.*" Most of the disappointment with the feedback is connected to the fact that the majority of SAAP tools work as explained in [13]: given a set of predefined instances of some computational problem consisting of input-output pairs, the tool compiles and runs source code in order to verify whether the program generates the desired outputs given the initial inputs.

So we developed a new web service for automatic assessment called *Project Tomo*[2] [14]. One of the main design goals was the flexibility a tool should provide in giving feedback to the students. Contrary to many SAAPs intended mostly to support assessment, our goal was to develop a small, flexible

[1] on webpage https://www.reap.ac.uk/
[2] https://tomo.fmf.uni-lj.si

solution, aiming at providing assistance for lab exercises where students are required to solve numerous programming tasks. We aimed towards the methods and tools that may help in providing the necessary feedback to improve the support for students during the learning of programming. So, our target was the formative feedback [5].

A. Basic Features

The main design objectives in developing our SAAP service were:

- Local execution,
- Possibility to use any of the existing programming environments,
- Being flexible enough to be functional with any programming language (currently we support Python, R and Octave) and
- Providing as much flexibility as possible in administering tests to achieve giving the appropriate feedback.

The details for those decision are explained in [14] and [15]. The service is designed to require little or no additional work from students and teachers, enabling them to focus on the content.

The service works as follows: the students first download the files containing problem descriptions to their own computers. The files are opened in their preferred programming environment for the chosen programming language and the students start coding the solutions. Executing the files checks their solutions locally. If the server is available, the solutions are automatically stored on the server. The server also optionally checks for the validity of the solutions by comparing hashed output from students' solution to the hashed output of the official solution.

This approach has several benefits: the service provides instant insight into the obtained knowledge to both student and teacher, all without disturbing the teaching process. There is also no need for powerful servers since all executions are done on the students' computers.

B. Testing Possibilities

As much flexibility as possible in administering tests was another vital feature. For instance, one of the requirements was that there should be a possibility to administer tests that can check whether a specific method was (or was not) used in a student's submission. For example, if the student's ability to write recursive programs is to be tested, non-recursive methods should not be accepted, even if they give expected results.

After providing the instructions for the task the teachers enter the expected solution followed by the tests that it has to pass (see Fig. 1). The solution is separated from the tests with the `Check.part()` command. It should be noted that the officially provided solution needs to pass the same tests as the students' solutions. At first this approach seems slightly more demanding for the teachers compared to the traditional approach where the teachers only provide instructions in text form. However, this forces the teacher to test the quality of the instructions. Namely, it often happens that poorly formulated

```
# =====================================================
# Computing distances
# =======================================@000003=======
#
# Write a function dist(x1, y1, x2, y2) that returns the
# distance between points (x1, y1) and (x2, y2).
#
# >>> dist(1, 2, 3, 4)
# 2.82842712475
# =====================================================

def dist(x1, y1, x2, y2):
  '''Returns distance between two points'''
  return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5

Check.part()
Check.equal('dist(1, 2, 3, 4)', 2 ** 1.5)
Check.equal('dist(0, 0, 3, 4)', 5)
```

Fig. 1. Instructions, solution, and validation from the teachers file.

problems only prove as being such during the attempt to solve them. This approach also ensures that the official solutions exist and work properly.

There are two commands that are most often used in testing. The most simple one tests the equality of the expected result with the result acquired by evaluating the given expression (see Fig. 1).

However, Tomo's main strength is in the possibility that the teacher composes a test that goes beyond a direct comparison of the outputs. The tests have access to the source code of the submitted solution under `Check.current_part['solution']`. So it is simple to make tests that ensure that a solution did not use for or while loops (e.g. if the students are to write solutions in recursive style) - see Fig. 2.

```
s = Check.current_part['solution']
if 'while' not in s:
    Check.error('Use of while is obligatory')
if 'for' in s:
    Check.error('Use of for is forbidden')
```

Fig. 2. While is required, but for is forbidden

Commands `Check.out_file`, `Check.equal`, and `Check.run` return `True` or `False`. Therefore, they can be used to determine if additional tests should be run or not. For instance, if the first test fails, the submission is clearly not valid and additional tests are not necessary. However, if the goal is to provide the students with detailed information on which test data their programs fail, as many tests as desired can be run.

There are other commands available to test the programs, for example commands that work with files. Since the validation is essentially a program written in a chosen programming language (Python for example) using the capabilities of the `Check` class, it can be made more advanced using all programming constructs that language offers. Thus, there are numerous possibilities available to prepare the appropriate feedback.

C. Feedback

The basic feedback is the report on the success of the student's attempt. This feedback is issued as soon as the students runs their solutions. Figure 3 shows that the solution of the first part of the task is accepted, the second one failed at (at least) one test and there has been no attempt to solve the third part of the task yet.

```
1. part has a valid solution.
2. part does not have a valid solution.
  - Expression vsote([3,5,1], 4) returns [1] instead of [].
3. part has no solution.
```

Fig. 3. Basic feedback

We paid special attention to suitable wording of this basic feedback. One of the first versions of the systems declared "*solution is correct*". But this is not in accordance with the premise that passing all the tests is not yet the proof of correctness. Therefore, we changed that into "*solution is accepted*". Several teachers reported this change had positive influence on student's awareness on what is the 'right solution'.

As explained before, the basic test is done with `Check.equal` method. This method directly compares the output from the official solution and the one (in [11], [12] see the discussion on drawbacks when only this kind of test is provided). Tomo offers much more flexibility. For example, especially the tasks that require to output text, students often complain "*but just one space is missing. Why is Tomo so picky!*" Here all teacher's pedagogical knowledge is to be exploited (as discussed in [5], [8]). The Project Tomo is just a tool in the hands of a teacher, who should decide on the purpose of a certain task. In this example the teacher has (at least) four different possibilities:

1) Leave the task as it is. The purpose of such a task is to get students to read the instructions, claims, and requirements carefully, and to keep to them consistently. SAAP here helps the teacher, because it is not necessary to explain to each student that his solution is not good because of a single capital letter.
2) Change the test so that if the student writes "enter" instead of "Enter", he receives the feedback that instructs them to look at the capitalization of the commands.
3) Change the test so that it does not matter which case is used. This makes sense in the case where the teacher's focus lies elsewhere and the output is of secondary significance.
4) Change the test so that it does not matter what wording the students use in their solutions.

And of course—most importantly—the teacher's task is to react to events that may occur during exercises. And that is precisely the basic reason why we are developing the Project Tomo: to relieve the teachers of simple tasks and give them additional time to interact with the student during lab exercises.

In Project Tomo it is possible to give a student feedback not only when a given test fails but anytime during the test program. This can be used to notify the students that the solution has passed some test cases and that they are on a good track. This is achieved using the `Check.feedback` construct which prints out the string given. See example in Fig. 4.

```
if Check.equal("""start('miha', 'meta')""", 1):
  Check.feedback("Bravo! Strings 'miha' and" +
           " 'meta' match in the first character")
```

Fig. 4. Positive feedback

A good way to learn is also to observe the official solution (see Fig. 5). In Project Tomo the teacher can decide when the students see it for every task. Currently the possibilities for official solution visibility are "always", "never" and "after they have submitted a valid submission".

The first option is rarely used since it gives the students an easy possibility to 'cheat': if their solution is not accepted, they look at the official one and use obtained information to solver the task. The second one is used during exams, where the official solutions are made visible after the end of the exam and the third one is the default setting.

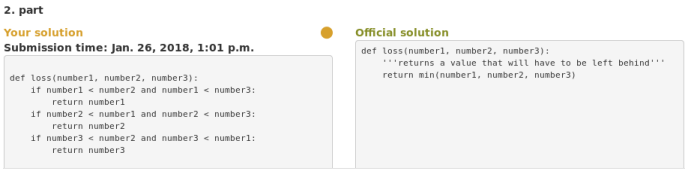

Fig. 5. Comparing with official solution

III. Analyzing the Data

Till now, we have developed quite an extensive library of programming tasks with high-quality feedback. Despite that we are constantly adding new tests and feedbacks to the tasks. Ideas for additional tasks arise from observing the mistakes that students make while programming. Many of the mistakes are missed by the teacher since it is impossible to observe each and every student all of the time. Since Project Tomo stores the history of every submission, we can do that retroactively: checking the history of the students' submissions we can analyze them, extract typical mistakes and use that knowledge to improve the quality of the feedback even further. So our workflow is as follows.

First the task is created by the teacher. The teacher tries to predict typical mistakes the students will make and include them into the test cases. The task is used in a teaching course and a lot of submissions are acquired from the students. Then these submissions are analyzed and if need for additional test cases is seen, they are added. Then the updated task is used in a teaching course and the entire process is repeated. So the quality of the feedback (and test cases) is checked and improved continuously.

Currently we are just starting a thorough analysis of the submissions. The goal of the first step of the analysis is to detect problematic tasks. Our assumption is that the exercises

where the average number of unsuccessful attempts students made before the accepted one, is high, are prime candidates for being labeled as problematic. If we manage to improve the quality of the feedback for these tasks, the students would benefit the most.

We concentrated on the last year programming course, where all data is available. For each successful attempt we checked how many unsuccessful attempts had been made before the valid one. Using the above mentioned criteria we detected several tasks where the average number of unsuccessful submissions was higher than 10. We analyzed the source code of those unsuccessful attempts.

Since the number of attempts is quite large, we have only managed to analyze some of the tasks so far. One task that was particularly interesting was a simple one, where students had to print the amount of money on an bank account in a grammatically correct way. The average number of unsuccessful submissions for this task was higher than 15. When we analyzed the history of the 495 attempts made, we found out that the students had made two typical mistakes. The first one was that they were unaware of the grammar rules of their own mother tongue and the second one was that they were very careless with their output, which usually only slightly deviated from the official one. Both of them combined caused the students to submit a lot of attempts that only slightly deviated in the output from the official solution, but were rejected by Project Tomo. It looks like students did not manage to see the difference in the outputs since they submitted lots of solutions with seemingly random changes to the source code that did not really fix the problem.

Using the above data we added two additional feedbacks to the task. The first one informs the student of the necessary grammar rules in detail if the test detects that they are not respected. This will hopefully reduce the number of incorrect attempts since a student can fix all grammatical mistakes in one step. The second one deals with the sloppy outputs. On one hand it is good that students learn how to be accurate. On the other hand it can be very frustrating if one has been working on a task for hours without visible progress, even more so for beginners. So we decided to modify the comparison function between the expected and the given output so that it will show more clearly where the outputs differ. We hope this will reduce the number of unsuccessful attempts even further and teach the students how to be precise at the same time.

We plan to analyze more tasks in the similar manner and use the updated tasks during future programming courses. We hope to notice a reduction of unsuccessful attempts.

IV. Conclusions and future work

Our goal is to use our analysis results to improve the feedback for the most problematic tasks and use the improved exercises in the class next year.

Currently most of the analysis is done manually, which is a very slow process. In the future we plan to use machine learning algorithms to extract common patterns from unsuccessful submissions for the given task. This would save our time trying to analyze the history of all attempts and allow us to focus on the most common mistake patterns.

Also some aditional features in providing feedback are planed. We are currently looking into the possibility of adding an additional option when the official solution can be seen: make it visible after a specified number of unsuccessful attempts. This would allow the students to see the official solution after they had made some real effort towards solving the task but failed to provide a valid solution. However, we have to ensure way of verifying that those attempts are “real”, not merely faking some output in order to reach the required number of submissions. Here we will probably use some of the approaches suggested in literature, for example in [12].

References

[1] M. J. Lee and A. J. Ko. 2011. Personifying programming tool feedback improves novice programrs' learning. In Proceedings of the seventh international workshop on Computing education research (ICER '11). ACM, New York, NY, USA, 109-116. DOI: http://dx.doi.org/10.1145/2016911.2016934

[2] A. J. Ko, B. A. Myers, and H. Aung, 2004. Six Learning Barriers in End-User Programming Systems. IEEE VL/HCC, 199-206.

[3] A. J. Ko, and B. A. Myers, 2009. Attitudes and Self-Efficacy in Young Adults' Computing Autobiographies. IEEE VL/HCC, 67-74.

[4] P. Kinnunen, and B. Simon. 2010. Experiencing programming assignments in CS1: the emotional toll. ICER, 77-86.

[5] V. J. Shute. Focus on formative feedback. Review of Educational Research, 78(1):153–189, 2008.

[6] H. Keuning, J. Jeuring, and B. Heeren. 2016. Towards a Systematic Review of Automated Feedback Generation for Programming Exercises. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16). ACM, New York, NY, USA, 41-46. DOI: https://doi.org/10.1145/2899415.2899422

[7] D. S. Campos, A. J. Mendes, M. J. Marcelino, D. J. Ferreira and L. M. Alves, "A multinational case study on using diverse feedback types applied to introductory programming learning," 2012 Frontiers in Education Conference Proceedings, Seattle, WA, 2012, pp. 1-6. doi: 10.1109/FIE.2012.6462412

[8] J. Hattie and H. Timperley, The Power of Feedback. Review of Educational Research. Volume 77, No. 1, pp. 81-112, 2007.

[9] K. M. Ala-Mutka. A survey of automated assessment approaches for programming assignments. Computer Science Education, 15(2):83–102, 2005.

[10] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppala. Review of recent systems for automatic assessment of programming assignments. In Koli Calling, pages 86–93, 2010.

[11] D. M. Souza, K. R. Felizardo and E. F. Barbosa, "A Systematic Literature Review of Assessment Tools for Programming Assignments," 2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET), Dallas, TX, 2016, pp. 147–156.

[12] B. Cheang, A. Kurnia, A. Lim, W. Oon, On automated grading of programming assignments in an academic institution, In Computers and Education, Volume 41, Issue 2, 2003, Pages 121-131, ISSN 0360-1315, https://doi.org/10.1016/S0360-1315(03)00030-7. (http://www.sciencedirect.com/science/article/pii/S0360131503000307)

[13] M. Rubio-Sanchez, P. Kinnunen, C. Pareja-Flores, and Angel Velazquez-Iturbide. 2014. Student perception and usage of an automated programming assessment tool. Comput. Hum. Behav. 31 (February 2014), 453-460. DOI: http://dx.doi.org/10.1016/j.chb.2013.04.001

[14] M. Lokar, M. Pretnar. "A Low Overhead Automated Service for Teaching Programming", Proceedings of the 15th Koli Calling Conference on Computing Education Research, Koli, Finland 2015, http://doi.acm.org/10.1145/2828959.2828964

[15] G. Jerše, M. Lokar. Learning and teaching numerical methods with a system for automatic assessment. The international journal for technology in mathematics education, ISSN 1744-2710, 2017, vol. 24, no. 3, 121–127