Automatic Formative Assessment of Programming Tasks

Janet Liebenberg, North-West University, South Africa

The Paris Conference on Education 2023
Official Conference Proceedings

Abstract
The onset of Covid-19 has impacted educational processes, particularly assessment, in a way never seen before. Automatic Programming Assessment (APA) can be unfair and inaccurate when used for summative assessment. This paper aimed to investigate to what extent the students had to adapt to automatic assessment and to determine the value of APA as a formative assessment tool. During a practical session in the computer lab, seven tasks were assigned to the students. The tasks resembled a step-by-step guide for writing a complete program that takes a user-specified number of integers, determines the minimum and maximum of these values, and performs calculations involving the minimum and maximum. The code that the students had written was uploaded to the APA system, allowing students to resubmit their work and improve their solutions as they went along. The analysis included marks per task, final marks of students, number of uploads per task, and the total number of uploads per student. General trends of these metrics were also observed. It was established that the majority of the students could successfully complete small programming tasks when re-acting to about two feedback comments per task. APA systems can be instrumental in supporting learning and are useful as a formative assessment tool. As a result of this study, we can point the way to develop systems which are smarter and more flexible.

Keywords: Automatic Assessment, Introductory Programming, Formative Assessment

iafor
The International Academic Forum
www.iafor.org

Introduction

The onset of Covid-19 has impacted educational processes and particularly assessment in a way never seen before or imagined. Education systems across the globe have responded to the Covid-19 induced disruptions in the manners mediated by their contexts. In previous research, it was found that Automatic Programming Assessment (APA) can be unfair and inaccurate when used for summative assessment (Liebenberg & Pieterse, 2018; Pieterse & Liebenberg, 2017; Ullah et al., 2018). However, APA may show potential for formative assessment purposes. What makes APA very appealing for formative assessment is the fact that it allows instant feedback to support real-time learning. This research aimed to investigate to what extent the students had to adapt to automatic assessment and to determine the value of APA as a formative assessment tool.

Related Work

Automatic Assessment

Automatic program assessment systems have been used for more than 50 years (Douce, Livingstone, & Orwell, 2005). In a review of APA systems by Ihantola, Ahoniemi, Karavirta, and Seppälä (2010), developed in the period 2006 to 2010, it was observed that APAs are mainly used in programming contests and introductory programming courses. A tremendous number of tools and systems for APA have been developed (Ullah et al., 2018). Mekterović, Brkić, Milašinović, and Baranović (2020) and Cipriano, Fachada, and Alves (2022) remark that APA systems are rarely used outside the institutions in which they are developed and cite a number of systems which are not available or have not been updated in a long time.

Many benefits of applying automatic assessment of programming assignments have been reported. Automatic assessment is bound to be consistent and objective (Arifi, Abdellah, Zahi, & Benabbou, 2015; Staubitz, Klement, Teusner, Renz, & Meinel, 2016), enables rapid feedback (Arifi et al., 2015; Liu et al., 2016; Ullah et al., 2018), and allows students to submit multiple improved versions of the programs they have written (Del Fatto et al., 2017; Staubitz et al., 2016). It can play a motivational role to engage students in the educational process (Šťastná, Juhár, Biňas, & Tomášek, 2015; Staubitz, Klement, Renz, Teusner, & Meinel, 2015). The most appealing benefit seems to be the possibility of saving time (Ullah et al., 2018). This comes as no surprise as it has been reported that assessment is one of the most often mentioned tasks that lecturers find burdensome (Pieterse & Sonnekus, 2003). Del Fatto et al. (2017) report how they effectively saved time when using a system, which can automatically identify correct code, reducing manual assessment to involve only code, which contains errors.

Staubitz et al. (2015) describe a number of challenges associated with applying automatic assessment of programming tasks. Ullah et al. (2018) mention the problem that many APA systems suffer from inflexibility and unfair grading. An important challenge, which is often overlooked, is that considerable time and effort need to be devoted to the implementation of resources for automated assessment (Ala-Mutka, 2005; Pieterse, 2013; Pieterse & Liebenberg, 2017; Watanobe, Rahman, Rage, & Penugonda, 2021). Another problem is that the development of new exercises often requires considerable technical skills beyond the scope of the content being assessed (Korhonen & Malmi, 2000; Pieterse, 2013). To address this problem Solms and Pieterse (2016) and Ullah et al. (2018) call for standardization.

Combéfis and Schils (2016) point to the complexity of being able to provide sensible feedback as it is nearly impossible to anticipate all errors that can occur in novice programs and to have test cases to identify each of the anticipated errors. They propose similarity clustering to improve the accuracy of feedback. Lepp et al. (2016) report that the design of automatically assessed exercise tests was one of the most difficult challenges they faced when applying a Moodle plug-in VPL for automatic assessment of programming assignments.

When using an APA system, educators should keep the reasons for assessment in mind and also guide their students to appreciate these goals in order to gain the most value from the assessment.

Assessment Goals

Tew and Guzdial (2010) suggest that there is no agreement on what constitutes valid measures of student learning in computing. Researchers speculate that students' poor performance may be indicative of inaccurate measures of their ability and knowledge (Lister, 2010; Tew & Guzdial, 2010). Often Bloom's cognitive taxonomy (Battestilli & Korkes, 2020; Bloom & Committee of College and University Examiners, 1964; Thompson, Luxton-Reilly, Whalley, Hu, & Robbins, 2008; Ullah et al., 2019) or The Structure of the Observed Learning Outcome (SOLO) taxonomy (Biggs & Collis, 1982; Petersen, Craig, & Zingaro, 2011) are used to determine the assessment goals of questions asked to evaluate the programming competence of students.

The assessment of programming tasks is classified into three categories according to the assessment goals of the measure of student skills and understanding of programming tasks namely structural, functional and conceptual.

Structural: A structural evaluation may include scrutiny of syntactical constructs and compliance with coding standards. These aspects are usually achieved through manual inspection. However, some authors have endeavoured to automate aspects of the structural assessment of programs (Ala−Mutka, Uimonen, & Jarvinen, 2004; Ali, Shukur, & Idris, 2007). Parsons and Haden (2006) developed a drill and practice computer game for mastering syntax constructs. The game itself serves as a formative assessment of mastering these constructs and the scores of students when playing the game, can be used for summative assessment of the skills and knowledge of students regarding structural aspects of programs.

Functional: The assessment of the functional correctness of a program written by a student can be achieved through the execution of the program using well-designed test cases (V Pieterse, 2013). Functional correctness may include the evaluation of aspects such as efficiency and proper memory management such as avoiding memory leaks (Ala-Mutka, 2005). These may be measured using popular profiling tools such as Valgrind1, Pin2 and Dr. Memory3. The automation of the functional correctness of programs is commonplace (Arifi et al., 2015; Ihantola et al., 2010), and according to Tirronen and Tirronen (2016), modern techniques can practically ensure the functional correctness of student solutions.

Conceptual: Evaluating the programming accomplishments of students on a conceptual level is probably the most difficult of the assessment goals to achieve. It is common to evaluate this using code reading questions or questions asking definitions or explanations in written exams (Petersen et al., 2011). Visual programming environments such as Scratch (Resnick et

al., 2009) and Alice (Dann, Cooper, & Pausch, 2008) can be used to promote conceptual understanding. The evaluation of conceptual aspects when assessing programs written by students is, however, not easy to automate (Posavac, 2015).

Formative Assessment

Formative assessment or Assessment for learning aims at monitoring student learning to provide ongoing feedback that can aid in identifying students' strengths and weaknesses in order to improve student attainment (Black & Wiliam, 2009). Ideally, in programming assessment, one should aim to take the whole spectrum of student achievement namely the students' structural, functional and conceptual understanding of the work into account when assessing their programs. Although an automated assessment system is less suitable for summative assessment, it shows promise for formative assessment of programs written by students during their practical lab sessions – especially monitoring the performance of students and improving their programming ability over time (Ho, Chean, Chai, & Tan, 2019). Automated assessment can reduce educators' effort, besides benefiting students with immediate feedback. Good and immediate feedback is one of the key components of learning programming (Lokar, 2019).

Battestilli and Korkes (2020) found that it takes students more submission attempts in the APA when they are given questions that contain some starter code, than when they have to write their solution from scratch. However, when writing code from scratch, the students' code quality can be impaired because the students are not required to actually understand the concept being tested and might be able to find a way to bypass or hack the tests of the APA.

Zingaro, Petersen, and Craig (2012) state that traditional code-writing exam questions seem to require in addition to a mastery of several concepts, the ability to design with or synthesize those concepts. As a result, students obtain marks for peripheral code in addition to code that satisfies the aim of the question and the mark awarded may therefore not alert students to core misconceptions. Zingaro et al. (2012) propose that single-concept questions - questions targeting one concept, or adding one concept over a previous concept question - are more effective formative feedback tools.

Tirronen and Tirronen (2016) saw how insufficient feedback, such as complicated compiler error messages or mere pass/fail feedback can hinder student progress. However, it is possible to induce positive behaviour with formative feedback.

Methodology

Scenario

The study was carried out at a South African university. A convenience sample of 187 first-year students enrolled in a Java programming course in the second semester made up the participants.

Prior to the study's session, the students completed an assignment during the seven weekly practical lab sessions while they could seek assistance and feedback from the tutors and lecturer. Following completion of the task, a student submitted the program for manual assessment by a tutor.

Data Collection

The assignment that served as the source of the data for this study was automatically evaluated by the APA system (APAS) Fitchfork, which uses dynamic testing-oriented assessment. The system runs the student's solution against a number of test inputs, checking the results against a regular expression that specifies the expected results for each test input. This method has two significant drawbacks: it requires a working program, and the test cases that are selected for evaluation may reduce the effectiveness of the evaluation.

Seven tasks were assigned to the students. During the session, the code that the students had written was posted to the APAS. It was assumed that they would complete the tasks in the prescribed order, which resembles a step-by-step guide for writing a complete program that takes a user-specified number of integers, determines the minimum and maximum of these values, and performs calculations involving the minimum and maximum. Table 1 summarizes the seven tasks.

Table 1: The seven tasks

Task Nr	Task Name	Max marks
1	Input	4
2	Loop	9.5
3	MinMax	12
4	getSum	2
5	getDiff	3
6	getQuo	4
7	Main	6

Task 1 as Example

To complete the first task, the students had to create a simple program that would ask the user for an integer number and then display that value in an output statement. A single test case was used to assess the task. Table 2 displays the feedback for projected output lines created by the student programs.

```
Enter an integer: 9
You entered 9
That's all folks!
```

Figure 1: Expected output for Task 1

Table 2: Assessment of Task 1 with test value = 5

Line	Possible output	Message	Mark (max)
1	Enter an integer:	PASS prompt line	1 (1)
	Other	FAIL The prompt should end with a colon and a space	0 (1)
2&3	You entered 5 That's all folks!	PASS output line	3 (3)
	You entered: 5 That's all folks!	FAIL Program output line 2 should NOT contain a colon	2 (3)
	5	FAIL The input should be on the same line as the prompt	1 (3)
		FAIL The acknowledgement line is missing	1 (3)
	Other	FAIL User input	0 (3)

Analysis

The uploaded files were downloaded after the students completed the assignment. The uploaded files as well as the marks and feedback that was given to students were analysed. The quantitative analysis included marks per task, final marks of students, number of uploads per task, and the total number of uploads per student. General trends of these metrics were also observed.

Results and Discussion

First Encounter

The way students participated in the first task was investigated in order to determine to what extent the students had to adapt to automatic assessment, as this was their first encounter with an APAS.

Figure 2 shows the number of students in three categories namely, file naming errors, other errors and no errors. Of the 184 students who attempted this task, only 128 (69.57%) completed the task successfully on their first attempt. Since this task is very easy, it is alarming that 56 (30.43%) of the students encountered problems when attempting this task.

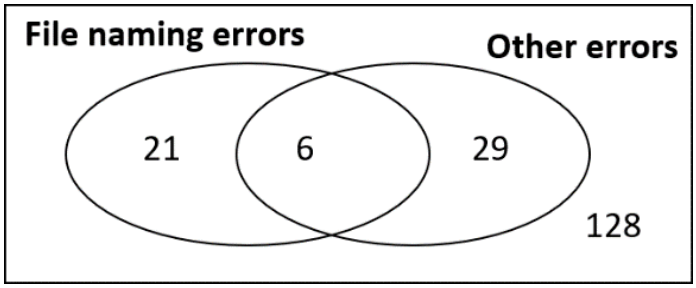

Figure 2: Types of errors in the first task (n=184)

Analysis of the types of errors made by the students who were unsuccessful on their first attempt to get full marks for the task reveals that 27 (14.67%) saved their solutions using a file name other than the prescribed name at some stage. Under manual assessment, this kind of error would have no ill effect. Yet under automatic assessment, this error causes Fitchfork

to fail, as the name of the file that is automatically compiled, executed and evaluated is key to the assessment of the code. If the file name is wrong the code is not assessed.

Further analysis of the submissions of the 35 (19.02%) students who made mistakes other than file naming errors revealed only three types of errors:

1. Uploading in the wrong slot: The solution to each task had to be uploaded to the designated upload slot which is specifically configured to assess the given task. Since there were seven tasks, there were seven different upload slots. Students can easily by accident just pick the wrong slot.
2. Compiler errors: If the student code does not compile, it cannot be executed and evaluated, leaving the student with zero marks for the specific task.
3. Layout errors: In order to simplify the memo specifications, a coding convention was adopted that required command line prompts to end with a colon and the input value to be typed on the same line as the prompt. Also, program output, other than prompts, should not contain colons. The students who violated this convention got feedback pinpointing this specific violation and corrected their transgression in their subsequent uploads.
4. All errors students encountered when uploading their solutions to the first task had to do with the limitations and quirks of Fitchfork.

Marks Distribution

The distribution of the total marks that were automatically awarded to the students while completing the assignment is shown in Figure 3. It can be seen that the bulk of the students was awarded more than 60% for the assignment and that the final mark for the vast majority of the students (78 ÷ 187) = 41.7% were in the category $92\% < x \leq 100\%$.

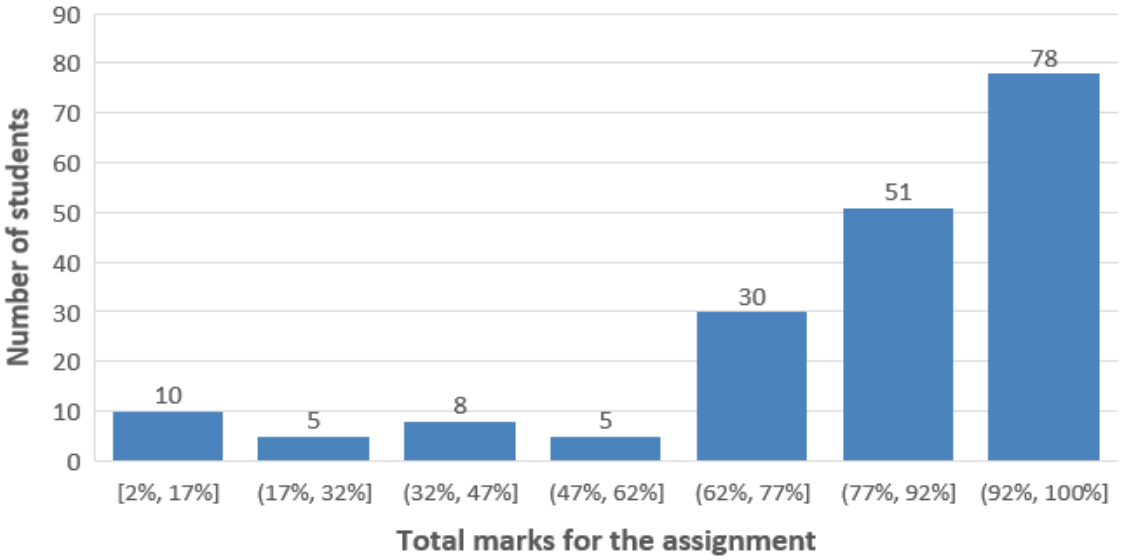

Figure 3: Distribution of marks (n=187)

When comparing this distribution with the distribution of automatically assigned marks by Matthíasdóttir and Arnalds (2015), the marks in our example show an obvious difference in the absence of high volumes of students being awarded very low marks. The same can be observed when comparing the distribution of marks reported by Liebenberg and Pieterse (2018) with the distribution of marks of our sample in Figure 3. This difference can be

attributed to the use of formative assessment and the option granted to students to resubmit their solutions to rectify their mistakes.

Number of Uploads

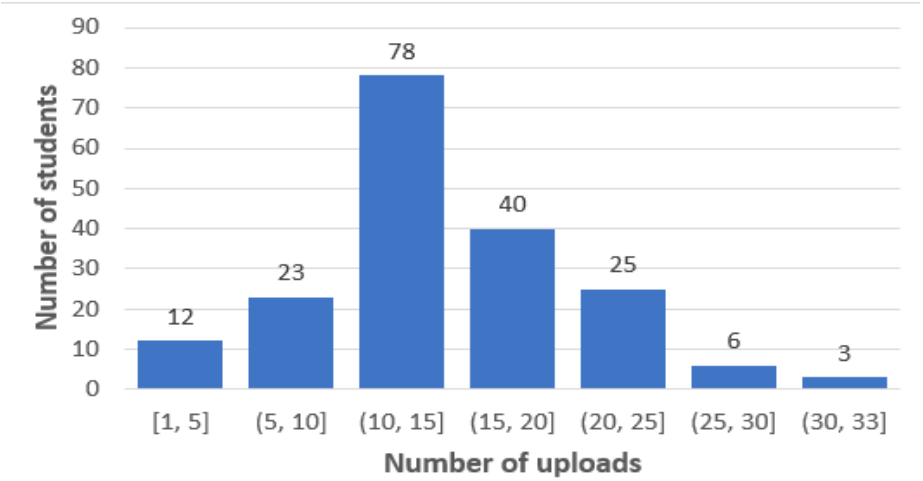

Figure 4: Distribution of the number of uploads (n = 187)

In Figure 4 the distribution of the total number of uploads for the assignment is shown. There were seven tasks and at least one upload per task was required to complete the assignment. The histogram is showing an almost normal distribution, slightly skewed to the left and having a peak in the number of students who uploaded between 10 and 15 times.

Bottom 18 Students

There were 18 students who had 10 or fewer uploads and achieved low marks. All but one did not complete all the tasks. They completed an average of 2.4 tasks and their average mark is 17.5%. The behaviour of these students represents that of non-committed students who often put in very little effort and just want to get out of the lab as soon as possible.

Careless or Effortless

Only two students uploaded exactly seven times with one upload per task. They obtained an average mark of 76.5%. Their behaviour might be similar to the abovementioned non-committed students, the only difference being they were able to complete most of the tasks correctly. They seemed to be careless about the tasks they did not complete correctly and probably did not even read the feedback.

Ninety-three students completed the assignment using between 9 and 15 uploads. Their average mark is 84.9% and 29% of them achieved 100%. The average number of uploads per task is 1.82. These students seem competent in terms of programming abilities, can follow instructions carefully and can successfully resolve problems based on the feedback.

Majority Group

There are 74 students in the category of students who used more than 16 uploads during the assignment. Three of them uploaded more than 30 times. The average mark of these 74 students is 85.1% and 24.3% of them achieved 100%. The average number of uploads per

task is 2.98. These students achieved on average slightly better than those discussed above, in terms of marks. They, however, needed more uploads to reach this achievement. These students may be less competent in programming than the previous category, yet they achieved good results. Since every upload is accompanied by feedback, pinpointing specific errors, they may have benefited from the feedback.

Conclusion

Despite this study being conducted before the pandemic, the rapid transition to remote emergency teaching and learning has accelerated the adoption of automatic assessment systems, particularly in the computing fields, leading to the widespread implementation of APASs.

Arguably, the most advantageous aspect of APASs is their ability to offer instant feedback. Unlike the situation where students had to wait for several minutes for a tutor to address their concerns, or even not receive feedback at all during the lockdown, all students utilizing the APAS in this lab session received feedback for each submission. This immediate feedback empowers students to make corrections to their programs and, in the process, facilitates their learning as they progress. The majority of the students uploaded about 3 times per task and managed to perform quite well. Their achievement might be explained by their learning from the feedback as intended with formative assessment.

The students who are used to manual assessment where they receive partial marks even if their programs do not compile are forced by Fitchfork to upload programs that do compile. This can be considered an educational benefit in terms of encouraging careless students to pay more attention to syntax and programming language features when writing their solutions. Based on our observations, students tend to be lax in following instructions, whereas the APAS demands meticulous attention to instructions. Consequently, the APAS presents valuable opportunities for students to hone their ability to pay attention to detail and follow instructions accurately. This skill holds particular significance in the field of IT.

The assessment done with Fitchfork is based only on the output of the student's solution. Sometimes a small deviation in the code can derail the whole assessment process. For example, a spelling error in a variable name can cause a compiler error resulting in Fitchfork being unable to assess the code. If the code was assessed manually, the assessor can give partial marks for the overall structure and algorithm used whereas automatic assessment awards zero in such cases. Furthermore, it is not possible to verify compliance with coding standards such as identifier names, indentation and comments, use of system constants, implicit and explicit type casting and types of loops used.

Since this was the students' first encounter with an APAS, we suspect that some of the problems mentioned above will get better over time as the students get acquainted with the system.

In this study, we established that the majority of our students managed to successfully complete small programming tasks using an average of about three tries per task. This means that they reacted about two times to feedback comments and improved their solutions towards the requirements for the task. Regrettably, we observed that there were cases where the feedback the students received was confusing and misleading. This is particularly true for cases where students' programs did not compile - this is not conducive to learning.

In light of the above-mentioned advantages and drawbacks of APASs in the context of post-pandemic pedagogy, we conclude that APASs may effectively support learning. APA systems can be instrumental in supporting learning and are useful as a formative assessment tool. As a result of this study, we can point the way to develop systems which are smarter and more flexible.

References

Ala-Mutka, K. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education, 15*(2), 83-102.

Ala−Mutka, K., Uimonen, T., & Jarvinen, H.-M. (2004). Supporting students in C++ programming courses with automatic program style assessment. *Journal of Information Technology Education: Research, 3*, 245-262.

Ali, N. H., Shukur, Z., & Idris, S. (2007). Assessment system for UML class diagram using notations extraction. *International Journal on Computer Science Network Security, 7*, 181-187.

Arifi, S. M., Abdellah, I. N., Zahi, A., & Benabbou, R. (2015). Automatic program assessment using static and dynamic analysis. *Proceedings of the 2015 Third World Conference on Complex Systems (WCCS)*, 1-6.

Battestilli, L., & Korkes, S. (2020). *Writing effective autograded exercises using Bloom's Taxonomy*. Paper presented at the 2020 ASEE Virtual Conference, Virtual conference.

Biggs, J. B., & Collis, K. F. (1982). *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*: Academic Press.

Black, P., & Wiliam, D. (2009). Developing the theory of formative assessment. *Educational Assessment, Evaluation and Accountability (formerly: Journal of Personnel Evaluation in Education), 21*(1), 5.

Bloom, B. S., & Committee of College and University Examiners. (1964). *Taxonomy of educational objectives: The Classification of Educational Goals* (Vol. 2): Longmans, Green New York.

Cipriano, B. P., Fachada, N., & Alves, P. (2022). Drop Project: An automatic assessment tool for programming assignments. *SoftwareX, 18*, 101079.

Combéfis, S., & Schils, A. (2016). Automatic programming error class identification with code plagiarism-based clustering. *Proceedings of the 2nd International Code Hunt Workshop on Educational Software Engineering*, 1-6.

Dann, W. P., Cooper, S., & Pausch, R. (2008). *Learning to program with Alice*: Prentice Hall Press.

Del Fatto, V., Dodero, G., Gennari, R., Gruber, B., Helmer, S., & Raimato, G. (2017). Automating Assessment of Exercises as Means to Decrease MOOC Teachers' Efforts. *Proceedings of the Conference on Smart Learning Ecosystems and Regional Development*, 201-208.

Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC), 5*(3), 4.

Ho, S. B., Chean, S.-L., Chai, I., & Tan, C. H. (2019). *Engineering Meaningful Computing Education: Programming Learning Experience Model.* Paper presented at the 2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM).

Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. *Proceedings of the 10th Koli Calling international conference on computing education research*, 86-93.

Korhonen, A., & Malmi, L. (2000). Algorithm simulation with automatic assessment. *ACM SIGCSE bulletin, 32*(3), 160-163.

Lepp, M., Luik, P., Palts, T., Papli, K., Suviste, R., Säde, M., Tõnisson, E. (2016). Self-and Automated Assessment in Programming MOOCs. *Proceedings of the International Computer Assisted Assessment Conference*, 72-85.

Liebenberg, J., & Pieterse, V. (2018). Investigating the Feasibility of Automatic Assessment of Programming Tasks. *Journal of Information Technology Education: Innovations in Practice, 17*, 201-223.

Lister, R. (2010). Computing Education Research - Geek genes and bimodal grades. *ACM Inroads, 1*(3), 16-17.

Liu, L., Vernica, R., Hassan, T., Damera Venkata, N., Lei, Y., Fan, J., . . . Wu, S. (2016). Metis: A multi-faceted hybrid book learning platform. *Proceedings of the 2016 ACM Symposium on Document Engineering*, 31-34.

Lokar, M. (2019). *Project Tomo: automated feedback service in teaching programming in Slovenian high schools.* Paper presented at the Proceedings of the 8th Computer Science Education Research Conference.

Matthíasdóttir, Á., & Arnalds, H. (2015). Rethinking teaching and assessing in a programming course a case study. *Proceedings of the 16th International Conference on Computer Systems and Technologies*, 313-318.

Mekterović, I., Brkić, L., Milašinović, B., & Baranović, M. (2020). Building a comprehensive automated programming assessment system. *IEEE Access, 8*, 81154-81172.

Parsons, D., & Haden, P. (2006). Parson's programming puzzles: a fun and effective learning tool for first programming courses. *Proceedings of the 8th Australasian Conference on Computing Education, 52*, 157-163.

Petersen, A., Craig, M., & Zingaro, D. (2011). Reviewing CS1 exam question content. *Proceedings of the 42nd ACM technical symposium on Computer science education*, 631-636.

Pieterse, V. (2013). Automated assessment of programming assignments. *Proceedings of the 3rd Computer Science Education Research Conference (CSERC 2013)*, 45-56.

Pieterse, V., & Liebenberg, J. (2017). *Automatic vs manual assessment of programming tasks.* Paper presented at the Proceedings of the 17th Koli Calling International Conference on Computing Education Research.

Pieterse, V., & Sonnekus, I. P. (2003). Why are we doing IT to ourselves? *Proceedings of the 33rd annual conference of the Southern African Computer Lecturers' Association (SACLA)*, Paper 9.

Posavac, E. J. (2015). *Program evaluation: Methods and case studies* (8th ed.). New York: Routledge.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., . . . Silverman, B. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60-67.

Solms, F., & Pieterse, V. (2016). *Towards a generic DSL for automated marking systems.* Paper presented at the Annual Conference of the Southern African Computer Lecturers' Association.

Šťastná, J., Juhár, J., Biňas, M., & Tomášek, M. (2015). Security Measures in Automated Assessment System for Programming Courses. *Acta Informatica Pragensia, 4*(3), 226-241.

Staubitz, T., Klement, H., Renz, J., Teusner, R., & Meinel, C. (2015). Towards practical programming exercises and automated assessment in Massive Open Online Courses. *Proceedings of the 2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, 23-30.

Staubitz, T., Klement, H., Teusner, R., Renz, J., & Meinel, C. (2016). CodeOcean-A versatile platform for practical programming exercises in online environments. *Proceedings of the 2016 IEEE Global Engineering Education Conference (EDUCON)*, 314-323.

Tew, A. E., & Guzdial, M. (2010). Developing a validated assessment of fundamental CS1 concepts. *Proceedings of the 41st ACM technical symposium on Computer science education*, 97-101.

Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., & Robbins, P. (2008). Bloom's taxonomy for CS assessment. *Proceedings of the tenth conference on Australasian computing education, 78*, 155-161.

Tirronen, V., & Tirronen, M. (2016). *A framework for evaluating student interaction with automatically assessed exercises.* Paper presented at the Proceedings of the 16th Koli Calling International Conference on Computing Education Research.

Ullah, Z., Lajis, A., Jamjoom, M., Altalhi, A., Al‐Ghamdi, A., & Saleem, F. (2018). The effect of automatic assessment on novice programming: Strengths and limitations of existing systems. *Computer Applications in Engineering Education, 26*(6), 2328-2341.

Ullah, Z., Lajis, A., Jamjoom, M., Altalhi, A. H., Shah, J., & Saleem, F. (2019). A rule-based method for cognitive competency assessment in computer programming using Bloom's taxonomy. *IEEE Access, 7*, 64663-64675.

Watanobe, Y., Rahman, M. M., Rage, U. K., & Penugonda, R. (2021). *Online automatic assessment system for program code: Architecture and experiences.* Paper presented at the 34th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2021, Kuala Lumpur, Malaysia, July 26–29, 2021, Part II 34.

Zingaro, D., Petersen, A., & Craig, M. (2012). *Stepping up to integrative questions on CS1 exams.* Paper presented at the Proceedings of the 43rd ACM technical symposium on Computer Science Education.

Contact email: janet.liebenberg@nwu.ac.za