A new marking technique in semi-automated assessment

A New Marking Technique in Semi-Automated Assessment

Selim Buyrukoglu, Firat Batmaz, Russell Lock
Computer Science
Loughborough University
Loughborough, UK
{s.buyrukoglu,f.batmaz,r.lock}@lboro.ac.uk

***Abstract*—The number of students learning programming languages in higher education and secondary schools has substantially increased, especially in the last decade. The increasing number of (novice) programmers makes code script assessment more important. Thus, this study proposes a new marking technique based on a semi-automated assessment approach. It advocates providing detailed and consistent feedback for novice programmers based on formative assessment. An experiment was carried out to check the feasibility of the proposed marking technique. The initial results and findings show that this is a potentially valuable approach.**

***Index Terms*— Marking Technique, Feedback, Programming Language, Semi-Automated, Novice Programmer**

I. INTRODUCTION

Feedback should be useful for students; otherwise, it cannot help them develop their own programming skills [1]. If feedback is comprehensive and is of sufficient quality, students can understand and fix their own mistakes more efficiently [2]. In this sense, the assessment purpose and marking techniques are just as important as the marker's experience in marking. There are three purposes of assessment, namely diagnostic, summative and formative. Diagnostic assessment measures students' weaknesses and strengths before they start their studies in order to give lecturers better information about their capabilities [3]. In summative assessment, students are generally provided with limited feedback and suggestions [4]. In contrast to summative assessment, formative assessment generally provides targeted feedback to both student and marker, rather than the results of their study [5]. Formative assessment also aims to improve learning with useful feedback. Many researchers have stated that the best way to teach a programming language to students is to use formative assessment [6][7][8][9]. In this case, students may focus deeply on learning programming. However, the assessment approach is as important as the purpose of the assessment. There are three ways to assess exercises in computer programming, manual assessment, fully automated and semi-automated assessment approaches. Many markers prefer to assess programming exercises manually. Seventy-four percent of examiners preferred to practice lab assessment manually [10]. However, the manual assessment of computer programming is inefficient and creates a heavy workload for the marker when the number of students in the class is high [11][12]. In fully automated assessment approaches, students submit their assignments and then these submissions are assessed purely by the computer [13]. In contrast, semi-automatic assessment approach enables humans to collaborate with the computer [13]. The fully automated and semi-automated approaches have some differences. The main differences are listed below [14]:

- The marking process is completed by a marker, using a semi-automated approach.
- In the semi-automated approach, novice programmer's code script does not need to produce an output – evaluation is based on a static assessment. In the static assessment, the code script is evaluated without executing the code [15]. The marker will mark the code script at less than 100% when it is provided in an incorrect state by the novice programmer. However, the fully-automated assessment approach focuses on dynamic assessment. In dynamic assessment, the code script is evaluated by executing the code in order to check its correctness [15].
- A human marker can provide more feedback about the code script's quality and efficiency, via the semi-automated approach, than a fully automated assessment system can.

Some advantages of a semi-automated assessment approach for novice programmers are highlighted [14]. This research focuses on a semi-automated assessment approach to give consistent and detailed feedback to novice programmers. The research also aims to reduce the marker's workload. The best assessment approach, in terms of providing consistent feedback, is the fully-automated system [16]. However, more detailed feedback can be provided using a human marker [17][18]. The reason for this is that the fully automated system may not be able to provide comments on critical aspects of code scripts. Thus, manual and fully-automated assessment approaches should be combined in order to provide consistent and detailed feedback. This constitutes the semi-automated assessment approach described in this research.

The structure of the paper is as follows: the next section introduces related marking techniques based on different semi-automated assessment approaches. Section III gives basic information of the proposed semi-automated assessment approach and Section IV describes the details of the

experiment. Section V provides information about the results and findings while Section VI provides the conclusion and future work.

II. Related Work

This section gives information about the marking techniques of current semi-automated assessment systems. This research also proposes a semi-automated assessment approach. Semi-automation in this research means the marking of students' programming solutions using both manual and automatic approaches.

Many semi-automated assessment systems aim to improve on manual approaches by providing more detailed and richer feedback than either manual or fully-automated approaches. In some semi-automated marking systems, the marker is enabled to provide comments on a static assessment [19]. If the marker considers a comment necessary, they can provide comments. On the other hand, other systems initially focuses on dynamic assessment [20][16][21][22][23]. These systems check the correctness of the students' solutions. If the solutions are correct, these systems automatically give feedback. Then, the feedback can be extended by the marker if they consider that additional comments are necessary based on the static assessment. However, the marker checks each student's solution and feedback to decide whether the automatically given feedback is sufficient. This creates an extra workload for the marker. If the solutions are incorrect, the marker fixes the errors to run the code. Then, the system gives feedback.

These systems use different marking techniques. Initially, the student's and a model answer's output are character-by-character matched [22] [23]. If they match, the student's answer is accepted by the system as correct. In other cases, the student answer is normalised before being matched with the model answer so that redundant parts of the student's answer are removed [14]. The third marking technique is where the student must print the output based on the marker's definition in the question. For example, the student must use the variable names specified in the question. Additionally, the student's code script must pass all tests to be considered successful. If the student answer and model answer do not match, the marker does not provide feedback and the student fails [20][16]. Then, these systems give feedback on the student's solution which either passes or fails.

Some semi-automated assessment systems also focus on providing feedback with visual annotations [24][17][18]. Annotation is used to add explanatory notes to a text [24]. For example, the system enables students to run their own programming solutions and to get feedback [17]. If they need more feedback, they can ask online questions of the marker. In this case, the marker provides annotation to give more detailed feedback. In a different marking system, the student solution is commented on by the marker using annotation in the form of tags [18]. The marker, in a different marking system, compiles and views the student's solution [24]. Then, the marker provides feedback. In the feedback process, specific icons are used to highlight comments such as arrows or lines etc.

Semi-automated assessment systems can provide useful feedback for students; however, there are some problems:

- The systems cannot reduce the markers' workload as markers must check each student's solution and the feedback to extend the automatically provided system feedback.
- These systems can provide general comments rather than on specific parts of the student's solution. The provision of more specific comments using existing systems would require significant preparation on the part of the examiner, who would have to pre-empt potential scenarios and prepare responses for them.
- The marker may provide inconsistent feedback for similar parts of different programming codes.

This study purposes to sort out the problems described above without significantly increasing the marker's workload. Thus, the following assessment approach has been developed, advocating a novel marking technique based on the static assessment.

III. Semi-Automated Assessment Approach

Programming code is referred to as code scripts in this research. Fig. 1 illustrates an example of a code script. Each code script consists of code segments which are also illustrated in Fig. 1. A code segment refers to the control structures of a code script in this research. For example, Fig. 1 shows two code segments which are code segments A and B. This research deals with the marking of three types of code segment, which are sequential, if and loop type code segments. If a code segment does not contain any control statement, it is considered to be a sequential type code segment. For example, code segment A matches the criteria for a sequential type code segment in Fig. 1. If a code segment refers to a simple-if, else or else-if type code segment, it is considered as an if type code segment. For example, code segment B matches the criteria for an else-if type code segment in Fig. 1. If a code segment refers to a for-loop, while-loop or do-while-loop type code segment, it is considered as a loop type code segment.

Code Segment A	`import math` `r=int(raw_input("Enter value"))`

Code Segment B	`if r<0:`
	`  print "r is smaller than 0"`
	`elif r==0:`
	`  print "r equals to 0"`
	`else:`
	`  print "r is bigger than 0"`

Fig. 1. An example of code script and code segment

This research's approach advocates marking each code segment individually and so providing detailed feedback, rather than just giving general feedback for the complete code script. Furthermore, it is intended to instigate the re-using of the marker's comments for similar code segments – in order to reduce the marker's workload. In this sense, the feedback can be richer and more helpful for students. The following

assessment cycle has been developed to provide consistent feedback and reduce marking time.

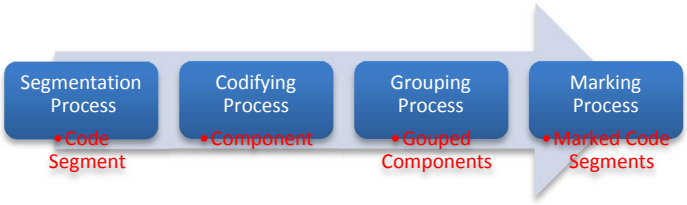

Fig. 2. Assessment cycle

The segmentation process parses code scripts to obtain code segments. For example, the illustrated code script in Fig. 1 is parsed as a two-code segment. Code segment A starts with the first line of the code script and finishes with the previous line from the if statement. Code segment B starts with the if statement and finishes with the last line in code segment B.

The codifying process aims to normalise code segments through generic rules such as the removing of blank lines from the code segments. The process normalises code segments in terms of code structure. Although some code segments are similar, their similarity can be increased through generic rules. Each code segment is considered as a component after the codifying process which is illustrated in Fig. 2. The generic rules are described with examples in [25].

The grouping process initially measures the similarity of components. String matching is applied to measure the components' similarity. Then, components are put into the same groups if they are matched in this process. The grouping process is the most important process in this approach. Feedback consistency mostly depends on the grouping process. In the marking process, the marker provides comments for segments of code scripts rather than for components. The reason for this is that components are transformed into code which is then used for the grouping process. Thus, the marker must provide comments for real code (code segments).

In the marking process, each student's code script is commented by the marker. When the marker provides a comment for a code segment in a code script, the comment is re-used to mark code segments, in different code scripts, which are in the same group as the marked code segment. Thus, when the marker provides comments for one student's code script, some or every code segment in the same group may be automatically marked via this marking technique. Hence, the marker need make comments for only the unmarked code segments – if, indeed, the code script has any unmarked code segments. Thus, this research can provide consistent feedback and reduce the marker's workload. The marker can check the generated feedbacks after the marking process. In this case, the marker can edit any comment to provide more helpful comments. Thus, the marker may ensure the correctness of the feedback. More detailed information about the proposed assessment approach can be obtained from [26].

IV. The Experiment: Feasibility of the Marking Technique

An experiment was carried out about the proposed semi-automated approach which is based on a new marking technique. This section gives details about the experiment, including the experiment aim, participant details, marking page layout design and data collection.

A. Experiment Aim and Participants

The aim of the experiment is to check the feasibility of the proposed marking technique. A marking page layout was designed for the experiment. The marking page layout consists of two marking areas: one for annotation and one for code segment marking; these are illustrated in Fig. 3. The proposed assessment approach supports the re-use of markers' comments. This only applies to comments made in the code segment marking areas; this is called segmented marking in this research. This constitutes a new marking technique. If segmented marking is found to be preferred by the participants of this research, then it can be inferred that the proposed assessment approach works well. A marking tool is required which will re-use the markers' comments. However, before implementing this marking tool, the feasibility of the proposed marking technique, based on the proposed assessment approach, will be measured by this experiment. That is, this experiment focuses on measuring the preferred marking areas of the marking page layout.

Twelve participants provide feedback in the experiment. They are PhD students with experience in Python programming and they know how to make comments. Three of them are teaching assistants in a Python programming module and two are teaching assistants in a C++ programming module. They were required to provide feedback for 16 paper-based code scripts (manually) using the designed marking page layout which is illustrated in Fig. 3. Normally, 55 code scripts are available to mark. However, the manual marking process is quite tedious for participants. Thus, the number of code scripts was reduced from 55 to 16 based on the following criteria: code segment types, programming errors and code layout. In this sense, dissimilar code scripts were chosen for the experiment.

B. Marking Page Layout Design

The marking page layout, illustrated in Fig. 3., is constructed from the two marking areas which were described in the previous section.

Annotation Marking Area	Code Segment Marking Area
Code Segment – I	Comment Box – I
Code Segment – II	Comment Box – II

Fig. 3. Proposed marking page layout

Each marking areas' size in the code script is fixed by the biggest code segment's size in the marking page layout. Thus, participant should have a sufficient marking area in which to make comments. In this sense, if participants mostly prefer segmented marking, the feasibility of the proposed marking

technique can therefore be considered high. Furthermore, annotation can still be important although participants mainly prefer segmented marking. The reason is that annotation with segmented marking can make the feedback more detailed and helpful for novice programmers, based on formative assessment [24].

C. Data Collection

Data were collected in two ways in this experiment. Initially, the participants marked students' code scripts using the designed marking page layout which was illustrated in Fig. 3. Second, a questionnaire was made with each participant after marking the code scripts. However, participants were not informed of the importance of segmented marking to this research. The questionnaire aims to get participants' opinion about the segmented marking based on the approach, even if they do not prefer segmented marking.

V. Results and Findings

This section presents two main results which are the preferred marking areas in the marking page layout by the participants and their responses to the questionnaire, respectively.

A. Preferred Marking Areas

Participants made comments for each code script including a total of 30 code segments. Fig. 4. illustrates a bar chart to show the total percentages of used marking areas in the experiment.

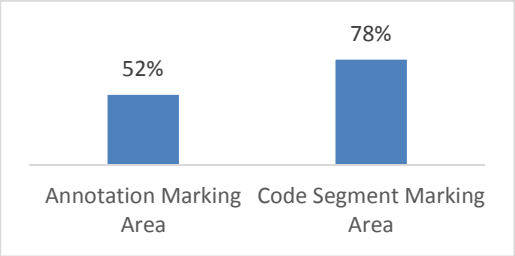

Fig. 4. Total percentages of used marking areas

Fig. 4. shows that 78% of the code segment marking areas were used by the participants. It is also the most preferred marking area which is an advantage since the proposed assessment approach supports re-using comments made in the code segment marking areas to mark repetitive code segments based on the formative assessment. On the other hand, 52% of the annotation marking areas were preferred by the participants. Annotations can be re-used for other students' code scripts, only where the code segments are identical across the code scripts. Of course, there are very few occasions in which code segments can be found which are identical. Code segments will usually have different syntactical and lexical structures even where they are very similar in terms of their control structures. Hence, the participants' workloads may actually increase due to the large number of non-identical code segments. Also, the proposed assessment approach may not provide consistent feedback due to their being these differences in the syntax of the lines. Thus, annotations should be optional for this research. Thus, participant can be make annotations if she/he considers them necessary, but participants do not have to provide annotations.

Although feedback quality is important for students (especially for novice programmers), a preference for segmented marking is more important than the feedback quality in this research. The feedback quality also completely depends on the participant's experience in marking [12].

B. Participants' Responses to the Questionnaire

Questions were asked of the participants based on the proposed assessment approach and proposed marking page layout in the experiment. The participants' responses in the questionnaire contribute to measuring the feasibility of the proposed marking technique and identifying the requirements of the marking tool.

Question: Which of the following statements is closest to you?

a) Providing feedback via a code segment marking area is more effective for me than using the annotation marking area on the code itself.
b) Providing feedback via the annotation marking area is more effective for me than using the code segment marking area on the code itself.
c) Despite the use of the code segment marking area, the annotation marking area is still important to provide detailed and helpful feedback.
d) Despite the using of the annotation marking area, the code segment marking area is still important to provide detailed and helpful feedback.
e) None of the above.

Rationale: The question was asked to find out whether the participants agree that the code segment marking area is more important than the annotation marking area although they are used together to provide feedback. Participants' responses are shown in a bar chart in Fig. 5.

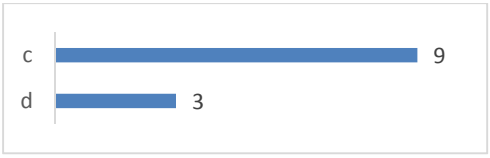

Fig. 5. Efficiency of the marking areas

The chart shows that the annotation marking area is still important even if the code segment marking area is the most effective marking area. However, use of annotations may decrease feedback consistency and increase marking time. The reason is that annotations cannot easily be re-used as much as comments made in the code segment marking area based on the proposed assessment approach, since code lines should be identical between different code segments.

Question: Which of the following statements is closest to you?

a) I believe that annotation made in the annotation marking area should only take a single form (for example, underlining) to ensure comprehension.
b) I believe that annotations made in the annotation marking area should take multiple forms to convey complex information (underlining, tick boxes, arrows, etc.) to ensure comprehension.
c) None of the above.

Rationale: The question was asked to find out whether the participants prefer to use a single form or multiple forms in the marking process. Participants' responses are shown in a bar chart in Fig. 6.

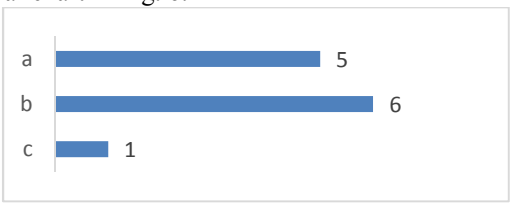

Fig. 6. Annotation forms

The chart indicates that half of the participants believe that the annotation marking area should take multiple forms. Five participants believe that the annotation marking area should take a single form while one participant has no preference. In this sense, annotation can be provided with a single or multiple forms.

Question: I believe that in order for a comment to be unambiguous the distance between the comment and the code itself should be minimised.

Rationale: The question was asked to find out how the distance between the comment and code should be provided in feedback. Participants' responses are shown in the chart in Fig. 7.

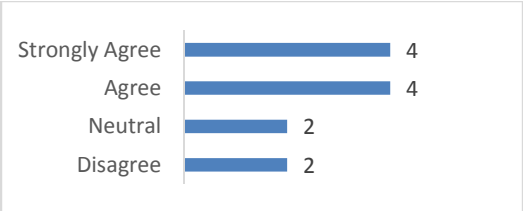

Fig. 7. Distance between comment and code

The bar chart shows that most (8/12) of the participants believe the distance should be minimised. Hence, the distance can be minimised to enhance comprehension of the comments.

Question: I believe that the code segment marking area need to be unrestricted in length as the comment length is independent of the code segment length.

Rationale: The question was asked to find out whether the participants agree about making the marking areas unrestricted to make comments freely. Participants' responses are shown in the chart in Fig. 8.

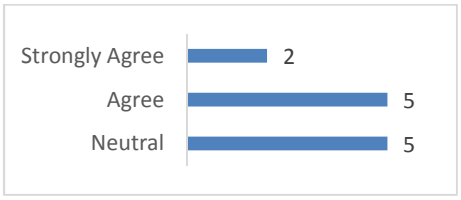

Fig. 8. Size of code segment marking areas

The bar chart shows that most (7/12) of the participants thought that the code segment marking area should be unrestricted. The rest (5/12) of the participants are neutral about the length. In this case, the length should be unrestricted. Due to the paper-based experiment, the code segment marking areas were restricted in this experiment. However, participants' thoughts about the length are captured through the question.

Question: I find that many of the comments I make are generic or common enough that an efficient mechanism to write them once and use them many times without rewriting them would be valuable.

Rationale: The question was asked to find out whether the participants agree about the automatic reuse of participants' comments for repetitive code segments based on the proposed assessment approach. Participants' responses are shown in the chart in Fig. 9.

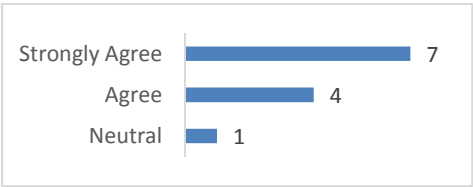

Fig. 9. Re-using comments for repetitive code segments

The bar chart shows that most (11/12) of the participants thought that the comments should be re-used for similar code segments. In this case, segmented marking can be considered quite important. The reason is that a participant's comments are re-used if they are provided through the segmented marking.

The result of the experiment shows that the proposed marking technique is feasible. However, this technique has some issues in relation to the proposed assessment approach. The first issue is that of the use of different variable names and different print messages across code segments. Two code segments may be identical in terms of structures; however, the variable names used in these code segments may be different. In this case, the two code segments cannot be matched and the number of groups unnecessarily increases, which causes an extra workload for the marker. A user interface could be designed and implemented to solve this issue. Students could drag and drop each part of their code segments through the tool. Thus, variable names and print messages could be made identical. The second issue is that of differing sequences of code segments within code scripts. The sequence of manually marked code segments within a code script may be different from the sequence of the same code segments in another script (which is to be automatically marked). Thus, the automatically marked code segment may be incorrectly marked. This issue

needs to be resolved for the future. The final issue is that comments made in a code segment marking area need to be generically reusable comments rather than specific comments. Generically reusable comments mean that the participant makes comments which can be used for more than one code segment. Specific comments mean that the participant provides detailed comments generally for code lines in code segments. In this case, annotation marking area can be preferred to provide specific comments. If a marker provides specific comments in their segmented marking for a code segment, this comment may be incorrect for the automatically marked code segments.

VI. Conclusion

This research paper has introduced a new marking technique based on a proposed semi-automated assessment approach. An experiment was manually carried out to check the feasibility of the marking technique. The initial results are encouraging. The technique can be considered feasible since approximately 75% of the participants are satisfied with segmented marking while 92% of the participants are satisfied with re-use of the participant's comments. To conclude, a participant's workload can be reduced and consistent and detailed feedback can be provided through the proposed marking technique.

A marking tool can be designed and implemented based on the semi-automated assessment approach described in Section III. Thus, efficiency of the marking technique can be proved through the tool. Finally, it can be used to assess novice programmers' code scripts in lab practices or assignments.

References

[1] J. Bull, and C. A. McKenna, "blueprint for computer-assisted assessment", Routledge, 2003.

[2] G. Wiggins, "Seven keys to effective feedback", On Formative Assessment: Readings from Educational Leadership (EL Essentials), 24, 2016.

[3] G. Conole and B. A. Warburton, "review of computer-assisted assessment". ALT-J, 2005, 13(1), 17-31.

[4] D. Chalmers and W. D. M. McAusland, "Computer-assisted assessment", The Handbook for Economics Lecturers: Assessment, Bristol: Economics LTSN, 2002, http://www. economicsnetwork. ac. uk/handbook.

[5] C. Gütl, "Moving towards a fully automatic knowledge assessment tool", International Journal of Emerging Technologies in Learning, 2008, 3(1), 1-11.

[6] I. Clark, "Formative assessment: Policy, perspectives and practice", Florida journal of educational Administration & Policy, 2011, 4(2), 158-180.

[7] S. Bedford and G. A. Price, "A study into the use of computer aided assessment to enhance formative assessment during the early stages of undergraduate chemistry courses", 2007.

[8] F. Scass, "Attributes of effective formative assessment": A work product coordinated by Sarah McManus, NC Department of Public Instruction, for the Formative Assessment for Students and Teachers (FAST) Collaborative. Washington, DC: Council of Chief State School Officers, 2008.

[9] L. A. Shepard, "Formative assessment: Caveat emptor", The future of assessment: Shaping teaching and learning, 2008, 279-303.

[10] J. Carter et al., "How shall we assess this?", In ACM SIGCSE Bulletin, 2003 (Vol. 35, No. 4, pp. 107-123). ACM.

[11] D. A. Kolb, "Experiential learning: Experience as the source of learning and development", FT press, 2014.

[12] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments", Computer science education, 2005, 15(2): 83-102.

[13] A. Weinberger, and H. Dreher, "Semi-Automatic Essay Assessment based on a flexible Rubric". 2011.

[14] H. Suleman, "Automatic marking with Sakai", In Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology, 2008, (pp. 229-236). ACM.

[15] K. K. Sharma, K. Banerjee, I. Vikas, and C. Mandal, "Automated checking of the violation of precedence of conditions in else-if constructs in students' programs." MOOC, Innovation and Technology in Education (MITE), 2014 IEEE International Conference on IEEE, 201.

[16] K. Ala-Mutka and H. M. Jarvinen, "Assessment process for programming assignments", In Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on (pp. 181-185). IEEE.

[17] N. Truong, P. Bancroft and P. Roe, "Learning to program through the web". ACM SIGCSE Bulletin, 2005, 37(3), 9-13.

[18] S. Cummins, L. Burd and A. Hatch, "Tag based feedback for programming courses", ACM SIGCSE Bulletin, 2010, 41(4), 62-65.

[19] D. Jackson, "A semi-automated approach to online assessment", *ACM SIGCSE Bulletin* ACM, 2000, 32(3): 164-167.

[20] D. Insa and J. Silva, "Semi-Automatic Assessment of Unrestrained Java Code", In Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, ACM, 2015: 39-44.

[21] M. Joy, N. Griffiths, R. Boyatt, "The BOSS online submission and assessment system", ACM Journal of Educational Resources in Computing, 2005, 5(3):2.

[22] B. Cheang, A. Kurnia, A. Lim and W.C. Oon, "On automated grading of programming assignments in an academic institution", Computers & Education, 2003,41(2): 121-131.

[23] K. A. Reek, "The TRY system-or-how to avoid testing student programs", In ACM SIGCSE Bulletin, 1989, (Vol. 21, No. 1, pp. 112-116). ACM.

[24] T. C. Ahren, "Using online annotation software to provide timely feedback in an introductory programming course", In Frontiers in Education, 2005. FIE'05. Proceedings 35th Annual Conference (pp. T2H-1). IEEE.

[25] S. Buyrukoglu, F. Batmaz, and R. Lock, "Increasing the similarity of programming code structures to accelerate the marking process in a new semi-automated assessment approach", In Computer Science & Education (ICCSE), 2016 11th International Conference on, 2016, (pp. 371-376). IEEE.

[26] S. Buyrukoglu, F. Batmaz, and R. Lock, "Semi-automatic assessment approach to programming code for novice students", 2016.