Automated Feedback Generation for Automatic Programming Assessment: Its Conceptual and Initial Analysis of Mapping Studies

Ahmed Abdulsalam Abdulmajeed Al-Athwari[1] and Rohaida Romli[2]
School of Computing, College of Arts and Sciences, Universiti Utara Malaysia, Malaysia, {[1]ah.alathwari20112@hotmail.com, [2]aida@uum.edu.my}

ABSTRACT
A systematic way of handling programming assignments' assessment via an automated approach is highly demanding. Thus, a method that is called Automated Programming Assessment (or APA) has been widely utilized to support automated marking and grading on students' programming exercises or assignments. Generating useful and meaningful feedbacks automatically via APA is essentially reducing lecturers' efforts, and students could learn to identify their own programming mistakes so as towards the end of learning process they can themselves achieve certain extend of good quality in programming. In this paper, we reveal an initial analysis of a mapping study related to these two contexts of areas so as to identify the criteria and matrices used to support automated feedback generation for more comprehensive features of APA. A technique known as Systematic Mapping Study (SMS) was utilized to comprehensively review the focused studies considering both the fully and semi-automated APA.

Keywords: Automatic Programming Assessment, summative feedback, formative feedback, Systematic Mapping Study, software testing.

I INTRODUCTION

Learning programming languages is extremely important for university students who pursue their studies in the fields of Information Technology, Software Engineering, and Computer Science disciplines which are known as practical subjects to improve students' or learners' understanding of programming principles (Lajis et al., 2018). Renumol, Jayaprakash and Janakiram (2009) quoted that "programming is the process of writing, testing and debugging of computer programs using different programming languages". As to achieve learning programming efficiently, assessing the quality of learners' programming solutions (Insa & Silva, 2018) and providing useful and meaningful feedbacks are vital. Feedback quality is an important factor in improving learners' programming skills (Buyrukoglu, 2018). Furthermore, feedback helps students to understand problems and find suitable way to address them (Buyrukoglu, 2018).

One of the important activities in learning programming is evaluating students' assignments (Insa & Silva, 2018) or is commonly known as programming assessment. Generally, assessment on students' work can be either formative or summative (Buyrukoglu, Batmaz & Lock, 2016a; Buyrukoglu, Batmaz & Lock, 2016b). Scriven (1967) stated that a summative assessment is with regard to the measurement of students' learning and their own achievements in learning. Taras (2005) stated that the formative assessment is in fact a summary of assessment that is added to the feedback used by the learners. Formative assessment is directly related to enhancement of student education by providing immediate and periodic feedback (Buyrukoglu et al., 2016b; Melmer, Burmaster & James, 2008).

Automatic Programming Assessment (or APA) emerged long time ago and has long history since the 1960s and is still active to research field and also researchers' focus (Buyrukoglu et al., 2016a). The main purpose of APA is to implement automated assessment and provides consistent and effective feedback to learners for improving their learning in programming as well as promotes workload reduction for lecturers (Buyrukoglu et al., 2016a). Manually marking programming exercises or assignments is well known as troublesome and tedious works (Blau, 2015; Huang & Morreale, 2015). In addition, the lecturers face challenges with assessing efficiently a huge number of students' assignments (Bey, Jermann & Dillenbourg, 2018; Blau, 2015; Romli, Sulaiman & Zamli, 2015) and frustrating to give their students individualized attention (Blau et al., 2016). Therefore, APA has become one of choices for assessing students' programming assignments automatically without the need of humans' involvement (Saikkonen et al., 2001).

Programming assessment is a part of software testing techniques, which can be categorized as dynamic testing or static analysis (Lajis et al., 2018; Romli, Sulaiman & Zamli, 2010; Saikkonen et al., 2001). Software testing is a process to measure, define, locate and detect the errors in a program (Latiu et al., 2012). Static analysis is mainly used to detect and inspect the errors that are committed has not to prove the validity of the program (Lajis et al., 2018). On the other hands, dynamic analysis is performing an assessment for the execution of a program to assess the style, software metrics and design of programs (Lajis et al., 2018).

APA together with a function to generate feedback is to promote effective learning (Buyrukoglu et al., 2016a). Generating feedback automatically is essentially to reduce lecturers' efforts as a part of assessment as well as it can motivate learners and guide them to produce a better quality of programming solutions. According to Romli et al. (2013), providing immediate feedback for learners in learning programming lead them to mastery learners' levels on programming. Thus, feedback generation is particularly important in APA systems (or APAS) that can help students from different level to enhance their knowledge in programming as well as avoiding any unfair assessment (Insa & Silva, 2018; Lajis et al., 2018). Also, this feature enhances the understanding of learners, particularly for undergraduate studies who are in expansive classes where lecturers' time is constrained or limited.

Providing useful and meaningful feedback on students' programming exercises and assignments is necessarily important to develop and enhance students' programming skills. Furthermore, assessing students' programming manually has been proved as very cumbersome and may lead to timely feedback, resulting in significant failure (Lajis et al., 2018) and it becomes more problematic when the size of classes is huge. Thus, most of these issues can be resolved by utilizing APAS. However, there is a lack of personalized and comprehensive feedback in existing APAS due to the sheer number of student submissions precludes the manual assessment option (Koprinska, Stretton & Yacef, 2015).

Thus, this study intends to review comprehensively related works that focus on these two context of areas to reveal on the current state of criteria and matrices used to support automated feedback generation in programming assessments or APA. The criteria and matrices include software testing techniques covered: static analysis and/or dynamic testing (white-box and black-box testing), and their respective quality matrices/factors, types of assessment feedback (formative and or summative) and its details, and features included in the APAS. However, this paper merely reveals some concepts involved in APA and feedback generation to highlight among of the common criteria and matrices applied, some of the review studies done as well as an initial analysis of the conducted mapping studies.

The content of the remaining sections is organized as follows: Section 2 discusses related reviews of selected studies as primary studies. In Section 3, we describe the applied research methodology of conducting SMS and the process of collecting relevant research papers. Section 4 presents the initial analysis of the conducted SMS. Finally, Section 5 concludes the paper and provides a brief discussion of future works.

II RELATED WORK

This section covers the discussion on basic concepts of software testing and programming assessment, types of assessment, and their related review studies.

A. Software Testing and Programming Assessment

Software testing is defined as measuring the quality of the software products (Latiu et al., 2012) and involves the process of analyzing a program to identify errors, playing an essential role to guarantee and maintain the quality, correctness and reliability of the software products (Myers, Sendler & Bandgett, 2011).

Programming assessment is related to the theory of software testing (Jackson, 1996). Software testing is commonly the basic concepts applied for tools related to improving programing analysis and comprehension skills among students (Souza et al., 2016). According to Sharma, Banerjee, Vikas and Mandal (2014) , student programming code can be statically or dynamically analyzed. In dynamic testing, it involves an execution of the program code, and the result is then checked to ensure the correctness, accuracy and validity of the program (Buyrukoglu et al., 2016b; Zougari et al., 2016a). For dynamic testing, the assessment process can be done by looking at the structure of the code (white box) or simply based on the functional behaviour of a program (black box) (Romli et al., 2010).

Programming assessment may use static analysis for analyzing the program code structurally (Salman, 1999) based on the code's properties. Static analysis is a method used to assess students' programming solutions without an execution of the code (Rahman & Nordin, 2007). Among the quality factors applied in the static analysis include program properties, proof of its practicableness, and look for errors within the code (Novikov, Ivutin, Troshina & Vasiliev, 2017). Also, static analysis has its assessment criteria such as programming style analysis, error detection (syntax or semantic or logic), metric analysis, keyword analysis, structural analysis, plagiarism detection (Rahman & Nordin, 2007; Zougari et al., 2016a).

B. Types of Assessment

Formative, diagnostic and summative are types of assessment (Buyrukoglu, 2018). These types of assessment are very important so as to develop knowledge of students who learn programming (Buyrukoglu, 2018; Buyrukoglu et al., 2017).

Formative assessment is a type that students can enhance their knowledge on learning programming on timely feedback (Buyrukoglu et al., 2016) as well as allowing students to enhance their thinking or behaviour in order to develop learning skills (Shute, 2008). Students can understand more deeply in their learning through formative assessment (Clark, 2011). Formative feedback is an important factor to help student learning and develop their works (Keuning et al., 2016).

The purpose of diagnostic assessment is similar to formative assessment (Buyrukoglu, 2018). However, diagnostic assessment is to measure the weaknesses and strengths of students while studying which can be useful for lecturers to know students' capabilities in certain knowledge (Conole & Warburton, 2005).

Meanwhile, summative assessment is a type of assessment that provides a report on the students understanding and achievement at the end period of study (Buyrukoglu, 2018). Most of the APAS and semi-APAS support both the formative and summative assessments.

Thus, it can be concluded that summative assessment is with regard to evaluating of students' works by providing results of assessment towards end of the assessment process, and on the other hand, formative assessment can be related to providing appropriate feedbacks based on the results of the evaluation such that they can develop and enhance their knowledge or skills on certain learning concepts.

C. Review Studies on Automatic Programming Assessment and Automated Feedback Generation

It has been found around ten review studies between the year of 2009 and 2018 related to APAS and Semi-APAS that focused on generating feedback. In terms of the review of APAS, thus far, several limited review studies had been conducted such as by Caiza and del Alamo Ramiro (2013), Edwards, Kandru and Rajagopal (2017), Ihantola et al. (2010); Lajis et al. (2018), Liang, Liu, Xu, and Wang (2009), Romli, Abdurahim, Mahmod and Omar (2016), Romli et al. (2010), Souza et al. (2016), and Striewe and Goedicke (2014). Ihantola et al. (2010) in their review covered developed APA tools in certain period from 2006 to 2010.

Another review study focuses on dynamic-structural testing (or white-box testing) conducted by Romli et al. (2016) reported that most of lecturers typically rely on the structural code coverage specified in programming assessment and even have a great learning to allow those criteria to be taken into consideration of implementing the APA. Similarly, Liang et al. (2009) reported that dynamic testing and static analysis as the major approaches of APA. Lajis et al. (2018) conducted a review that revealed most of APAS do not have a common grading model that refers to the learning taxonomy. Similarly, Caiza and del Alamo Ramiro (2013) also performed a review on the art of the APAS, which shows the lack of a common grading model as the major issue. Caiza and del Alamo Ramiro (2013) referred to those APAS that provide timely and consistent feedback on students' code scripts and among the related studies within the year of 2010, the main metric for grading is correctness. Edwards et al. (2017) in their study focused on investigating nearly 10 million static analysis errors found in over 500 thousand program submission made by students over five semesters. They used in their investigating two open-source static analysis tools (PMD and Checkstyle) to compare their features. They found that the most common static analyses errors are on formatting and documentation (Javadoc commenting) errors are the most common static analysis errors.

Ihantola et al. (2010) presented a Systematic Literature Review (SLR) to review the key features of related APA studies published between 2006 and 2010. They concluded that many proprietary APAS were developed and provide suggestions on APAS developers to make their systems open source such that it is easier for others to contribute enhancement on the tools because the lack of open source systems may be one of the reasons for the continuous development of newly refined APAS. Similarly, Souza et al. (2016), also performed a SLR to find out among APA tools that were developed for over last 10 years. They investigated 30 APA tools particularly to focus on their features in assisting lecturers to identify APA tools better for their needs. The selected tools have been found that they can provide immediate feedback which encourage students to improve their solutions continuously. Striewe and Goedicke (2014) reviewed APA tools that only focus on static analysis approaches in detail for diagnosing students' programs. They found that some of APA tools may be considered insufficient to use the full power of static analysis in terms of generating feedback in e-assessment systems. On top of that, Romli et al. (2010) reviewed the approaches implemented in several studies that focus on APA, test data generation and their integrations.

There are in a total of two review studies have been found for automatic feedback generation for programming exercises by using APAS conducted by Keuning et al. (2016) and Keuning et al. (2018). Keuning et al. (2016) reviewed 69 tools while Keuning et al. (2018) reviewed 101 tools that classified the kinds of feedback generation into five categories: Knowledge about Task Constraints (KTC), Knowledge about Concepts (KC), Knowledge about Mistakes (KM), Knowledge about How to proceed (KH), Knowledge about Meta-Cognition (KMC). These review studies mainly focused on formative feedback that is defined as “information communicated to the students with the intention to modify their thinking or behavior for the purpose of improving learning” (Shute, 2008). Keuning et al. (2018), Keuning et al. (2016) in their studies analyzed the selected APA tools for their review to find what kind of feedback that each APA tool support based on the five kinds of feedback that mentioned above. Keuning et al. (2018) referred that every APA tool of all selected APA tool in their study can provide more than one kind of feedback but they reported that, in general, the feedback that APAS generate is not very varied and focuses mainly on identifying errors. Keuning et al. (2016) concluded that most of the APA tools rely on test cases and the provided feedbacks were merely on how to correct errors rather than comprehensively providing meaningful and rich feedback that could help them in identifying on their mistakes for further improvement on the quality of their programming solutions. Keuning et al. (2018) found that solution errors based on the KM, are in the most of selected APA tools with 59.4% while Keuning et al.(2016) in their previous review found test failures in many of APA tools.

III METHODOLOGY

A Systematic Mapping Studies (SMS) technique (Petersen, Feldt, Mujtaba, & Mattsson, 2008) has been adopted to conduct the proposed review study. The following summarizes the process of SMS that consists of five steps:

i) Definition of Research Question

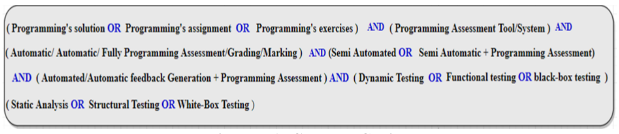
Specifying the research question (s) is the most important part of any systematic review including SMS. The following are the research questions that have been identified:

RQ1: To what extend does the criteria and matrices used to support automated feedback generation are implemented in APA?

RQ2: What are among the promising criteria and matrices that can be utilized in realizing automated feedback generation to support a better feature of APA?

ii) Conduct Search

Conduct search is a step to select the related studies (Petersen et al., 2008). It includes two steps: (1) using search string to retrieve information from electronic resources, and (2) databases selections. Figure 1 shows the keywords and search strings that were formed in searching the related studies from electronic databases.



(Programming's solution OR Programming's assignment OR Programming's exercises) AND (Programming Assessment Tool/System) AND (Automatic/ Automatic/ Fully Programming Assessment/Grading/Marking) AND (Semi Automated OR Semi Automatic + Programming Assessment) AND (Automated/Automatic feedback Generation + Programming Assessment) AND (Dynamic Testing OR Functional testing OR black-box testing) (Static Analysis OR Structural Testing OR White-Box Testing)

Figure 1. Search String

After designing the search string, the relevant databases have been chosen. Seven electronic databases were selected include ACM Digital Library, Google Scholar, IEEE Xplore, Scopus, Search Gate, EThOS e-thesis online services and ScienceDirect which are known as the most relevant to scientific sources which primary studies likely to be contained (Souza, Papadakis, Durelli, & Delamaro, 2014) and some of the selected databases are among the main databases in state of Computer Science.

iii) Screening of the papers

This stage is about screening the primary studies selected to the topic of this study such that the studies that are not related to answer the RQs can be excluded (Petersen et al., 2008). The inclusion and exclusion criteria applied in this study are depicted in Table 1.

Table 1. Inclusion and Exclusion Criteria

Inclusion Criteria	Exclusion Criteria
• Abstracts and keywords are written in English • APAs studies • Semi-APAs studies • Studies that explain software testing techniques	• Papers that directly related to programming assignment • Studies that did not focus on programming assessment • Papers where the main language is not English • Duplicated papers • Studies that do not include related tool for fully or semi-APA • Studies that do not indicate issues related to programming assessment. • Papers that are not in the fields of programming education • Secondary studies (e.g. review studies)

iv) Keywording using Abstracts

Keywording was performed in two steps firstly, the abstract was read with its keywords and then, concepts that reflected the contribution of the paper were identified.

v) Data Extraction

Data Extraction is the final stage of SMS process. The data extraction procedure was conducted in one stage. The stage was to collect the information about the paper to address the RQs of this mapping study by presenting them in tabulation. This involved grouping out the data based on the combinations of the dimensions and categories and displaying them using bubble plot. In this stage, nine categories or dimension were classified by using a classification scheme for the selected studies. The classification scheme is as shown in Table 2.

Table 2. Classification Scheme of Selected Study

Categories	Description
Study identifier	Study Id (e.g. S001)
Author (s) and year (ref)	Name of author(s) of the selected studies
Software Qualities Metrics (static analysis)	Static analysis metrices applied to access students' programming solutions
Software Qualities Metrics (dynamic testing)	Dynamic testing metrices applied to access students' programming solutions
Programming Languages used/supported	The programming languages used/supported by the developed APAS. For example: Java, Python, C, C++ and etc.
Technique (s) of testing applied	Type (s) of the testing applied in the assessment, could be the static analysis or dynamic testing or both of them.
Type of the tool	Type of the tool developed could be stand alone or web-based or mobile-based
Feedback details	The detailed descriptions of feedback provided via the performed assessment
Feedback types	Type of feedback provided either formative or summative feedback

IV INITIAL ANALYSIS OF MAPPING STUDY

This section presents the initial analysis obtained from the conducted mapping study.

A. Search and Selection Results

Initially 281 papers were retrieved when the designed search protocol was applied to the selected scientific databases. Inclusion and exclusion criteria were then applied based on the titles of the retrieved papers. By examining all of the papers based on relevancy of their titles, 158 papers were selected. The reason of excluding 123 papers was due to, they were not truly related to the programming assessment. For example, some of the excluded papers were discussed on learning programming but not focused on assessing students' codes. After the selection of 158 papers, the duplicated papers were removed as the second criteria of inclusion and exclusion. This round resulted in selection of 112 papers. After that, 87 papers were selected based on their abstracts. Some of the excluded papers were not written in English and some of them were not related to the desired topic. All of the 87 papers were then passed to the next selection round for in depth analysis, which involved reading completely their contents. Finally, 71 papers were selected as primary studies. From the 87 papers, one paper was dropped out because it focused on learning programming rather than programming assessment. The remaining papers were excluded because they were categorized as review papers.

B. Publication Year

After the process of searching and selecting the final primary studies by applying inclusion and exclusion criteria, it has been found that all of the selected articles were published from 1982 until 2019. Thus, although APA is a research area that has a long history, it still attracts the researchers' focus and attention until recently. Result on the trends of the publication year distribution of the selected primary studies includes: 1 paper (1.4%) was published in 1982, 1 paper (1.4%) was published in 1993, 1 paper (1.4%) was published in 1995 in (1.4%), 1 paper was published in 1997, 1 paper (1.4%) was published in 1999, 1 paper (1.4%) was published in 2000, 2 papers (2.8%) were published in 2003, 2 papers (2.8%) were published in 2004, 3 papers (4.22%) were published in 2005, 1 paper (1.4%) was published in 2006, 3 papers (4.22 %) were published in 2007, 5 papers (7%) were published in 2008, 1 paper (1.4 %) was published in 2010, 2 papers (2.8%) were published in 2011, 3 papers (4.22%) were published in 2012, 4 papers (5.63 %) were published in 2013, 4 papers (5.63 %) were published in 2014, 7 papers (9.85 %) were published in 2015, 11 papers (15.49 %) were published in 2016, 3 papers (4.22 %) were published in 2017, 3 papers (14 %) were published in 2018, 4 papers (5.63 %) were published in 2019. It can be concluded that the publications on fully APAS and Semi-APAS are keep on increasing through the years.

C. Venue Name and Type

A total of 71 venue names collected from searching the related primary studies, which come from a wide variety of journal, conference, symposium, peer-reviewed journal, colloquium and EThOS (Electronic Theses Online Service). Figure 2 shows the venue types included in the conducted SMS. Most of the studies were published in conferences representing 44 studies (62%) and journals representing 20 studies (28%). The rest studies were published in peer-reviewed journals as 3 studies (4%), 1 study in a symposium (1.4 %), 1 study in a colloquium (1.4%), and 1 study in EThOS as labeled 'Thesis' (1.4 %) and 1 other study (1.4 %). As a conclusion, majority of the articles are categorized as conference articles and journals.

D. Classification of Relevant Papers

Table 3 shows the distribution of the assessment tools by types (either APAS or Semi-APAS), their features (based on stand-alone, web-based, web-mobile-based), software testing techniques (static analysis or dynamic testing), dynamic testing techniques (black-box testing or white-box testing or a combination both of them), the feedback types (formative or summative feedback or an integration between both of them).

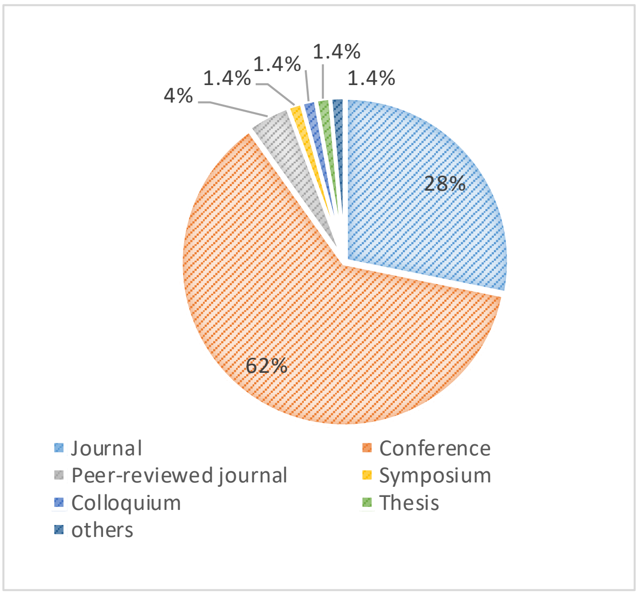


Figure 2. Venue Type

V CONCLUSION

Overall, results from conducted SMS revealed that applying the static analysis and dynamic testing techniques are the researchers' focus and interest. Nevertheless, it is also found that most of the selected primary studies integrate both dynamic testing and static analysis techniques in their proposed APA. Another finding is the black-box testing was used and focused in the most of selected studies rather than white box testing. Furthermore, it is observed that the studies which have supported formative feedback are close to those studies which supported the summative feedback. It has been observed that many of fully APAS and Semi-APAS studies supported both the summative and formative feedback. Semi-APAS studies focused on providing formative feedback more than fully APAS due to the need of the lecturers in providing useful feedback for students to get better understanding on their programming solutions. Additionally, Semi-APAS are close to fully APAS in providing summative feedback.

Currently, Semi-APAS in terms of generating feedback depend on the human instead of being fully automatic due to some reasons. However, some criteria used in current Semi-APAS can be used later for achieving fully APAS. In this regard, Insa and Silva (2015) referred to this issue which the fully APAS are lacking in terms of generating feedback, since most of the concern was more on fixing some errors to run the code. As to overcome the lacking in utilizing human involvement, it is more promising to integrate automated feedback generation in APAS as a mandatory rather than an optional for a better consistent feedback provided to students. This add-on feature in APAS is more significant in evading biasness and inconsistency feedbacks.

Table 3. Summary of the Classification Scheme

Studies	APAS	SAPAS	Stand-alone		Web-based		Web-Mobile-based		Static Analysis	Dynamic Testing			FT		
										BB	WB	Both	FF	SF	B
			APA	SAPAS	APA	SAPAS	APA	SAPAS							
71	34	37	18	17	15	20	1	NA	7	27	8	27	28	27	16

Notes: SAPAS stands for Semi-APA; FT stands for Feedback Types; FF Stands for Formative feedback; SF Stands for Summative Feedback; B stands for Both (formative and summative).

REFERENCES

Bey, A., Jermann, P., & Dillenbourg, P. (2018). A comparison between two automatic assessment approaches for programming: An empirical study on MOOCs. Educational Technology and Society, 21(2), 259–272.

Blau, H. (2015). Automated style feedback for advanced beginner Java programmers (UNIVERSITY OF MASSACHUSETTS AMHERST Directed; Vol. 2016-Novem). https://doi.org/10.1109/FIE.2016.7757728

Blau, H., Kolovson, S., Adrion, W. R., & Moll, R. (2016). Automated style feedback for advanced beginner Java programmers. Proceedings - Frontiers in Education Conference, FIE, 2016-Novem. https://doi.org/10.1109/FIE.2016.7757728

Buyrukoglu, S. (2018). Semi-automated assessment of programming languages for novice programmers.

Buyrukoglu, S., Batmaz, F., & Lock, R. (2016a). Increasing the Similarity of Programming Code Structures to Accelerate the Marking Process in a New Semi-Automated Assessment Approach. In 2016 11th International Conference on Computer Science & Education (ICCSE), 371–376.

Buyrukoglu, S., Batmaz, F., & Lock, R. (2016b). Semi-automatic assessment approach to programming code for novice students. Proceedings of CSEDU 2016, the International Conference on Computer Supported Education, 1, 289–297.

Buyrukoglu, S., Batmaz, F., & Lock, R. (2017). A new marking technique in semi-Automated assessment. ICCSE 2017 - 12th International Conference on Computer Science and Education, (Iccse), 545–550. https://doi.org/10.1109/ICCSE.2017.8085551.

Caiza, J. C., & del Alamo Ramiro, J. M. (2013). Programming Assignments Automatic Grading: Review of Tools and Implementations. 7th International Technology, Education and

Development Conference, 5691–5700. Retrieved from http://library.iated.org/view/CAIZA2013PRO

Conole, G. and Warburton, B. (2005) “A review of computer-assisted assessment,” ALT-J, vol. 13, no. 1, pp. 17–31.

Edwards, S. H., Kandru, N., & Rajagopal, M. B. M. (2017). Investigating static analysis errors in student Java programs. In Proceedings of the 2017 ACM Conference on International Computing Education Research, 65–73. https://doi.org/10.1145/3105726.3106182

Huang, C., & Morreale, P. A. (2015). An Integrated Automatic Compiling System for Student Feedback on Java Programs. 201–204.

Ihantola, P., Ahoniemi, T., Ville, K., & Seppälä, O. (2010). Review of Recent Systems for Automatic Assessment of Programming Assignments. In Proceedings of the 10th Koli Calling International Conference on Computing Education Research, (January). https://doi.org/10.1145/1930464.1930480

Insa, D., & Silva, J. (2015). Semi-automatic assessment of unrestrained Java code: a library, a DSL, and a workbench to assess exams and exercises. In Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, 39–44.

Insa, D., & Silva, J. (2018). Automatic assessment of Java code. Computer Languages, Systems and Structures, 53, 59–72. https://doi.org/10.1016/j.cl.2018.01.004

Jackson, D. (1996). A software system for grading student computer programs. Computers and Education, 27(3–4), 171–180. https://doi.org/10.1016/s0360-1315(96)00025-5

Rahman, K. A., Ahmad, S., & Nordin, M. J. (2007). The Design of an Automated C Programming Assessment Using Pseudocode Comparison Technique. National Conference on Software Engineering and Computer Systems 2007.

Koprinska, I., Stretton, J., & Yacef, K. (2015). Students at Risk : Detection and Remediation. Proceeding of the 8th International Conference on Educational Data Mining, 512–515.

Keuning, H., Jeuring, J., & Heeren, B. (2016). Towards a Systematic Review of Automated Feedback Generation for Programming Exercises. *ACM Transactions on Computing Education*, *19*(1), 1–43. https://doi.org/10.1145/3231711

Keuning, H., Jeuring, J., & Heeren, B. (2018). A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education*, *19*(1). https://doi.org/10.1145/3231711

Lajis, A., Baharudin, S. A., Kadir, D. A., Ralim, N. M., Nasir, H. M., & Aziz, N. A. (2018). A review of techniques in automatic programming assessment for practical skill test. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 10(2–5), 109–113.

Latiu, G. I., Cret, O. A., & Vacariu, L. (2012). Automatic Test Data Generation for Software Path Testing Using Evolutionary Algorithms. 2012 Third International Conference on Emerging Intelligent Data and Web Technologies, 1–8. https://doi.org/10.1109/EIDWT.2012.25

Liang, Y., Liu, Q., Xu, J., & Wang, D. (2009). The recent development of automated programming assessment. In 2009 International Conference on Computational Intelligence and Software Engineering. IEEE, 1–5. https://doi.org/10.1109/CISE.2009.5365307

Myers, G. J., Sendler, C. and Bandgett, T. (2011) The Art of Software Testing, 3rd ed. John Wiley & Sons.

Novikov, A. S., Ivutin, A. N., Troshina, A. G. and Vasiliev, S. N. (2017). “The approach to finding errors in program code based on static analysis methodology,” 2017 6th Mediterr. Conf. Embed. Comput. MECO 2017 - Incl. ECYPS 2017, Proc., pp. 4–77.

Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). Systematic Mapping Studies in Software Engineering. In Ease, 8(1), 68–77. https://doi.org/10.1142/S0218194007003112

Renumol, V., Jayaprakash, S., & Janakiram, D. (2009). Classification of cognitive difficulties of students to learn computer programming. Indian Institute of Technology, India, 12.

Romli, R., Abdurahim, E. A., Mahmod, M., & Omar, M. (2016). Current practices of dynamic-structural testing in programming assessments. Journal of Telecommunication, Electronic and Computer Engineering, 8(2), 153–159.

Romli, R., Sulaiman, S., & Zamli, K. Z. (2010). Automatic programming assessment and test data generation a review on its approaches. ITSim ’10, 1186–1192. https://doi.org/10.1109/ITSIM.2010.5561488

Romli, R., Sulaiman, S., & Zamli, K. Z. (2013). Designing a test set for structural testing in automatic programming assessment. International Journal of Advances in Soft Computing and Its Applications, 5(SPECIALISSUE.3), 41–64.

Romli, R., Sulaiman, S., & Zamli, K. Z. (2015). Improving Automated Programming Assessments: User Experience Evaluation Using FaSt-generator. Procedia Computer Science, 72, 186–193. https://doi.org/10.1016/j.procs.2015.12.120

Saikkonen, R., Malmi, L., & Korhonen, A. (2001). Fully Automatic Assessment of Programming Exercises. ACM SIGCSE Bulletin, 33(3), 133–136. https://doi.org/10.1145/507758.377666

Salman, A. (1999). Automatic marking of Shell programs for students coursework assessment (Doctoral dissertation), Oxford Brookes University.

Scriven, M. S. (1967). The methodology of evaluation. Chicago: Rand McNally.

Sharma, K. K., Banerjee, K., Vikas, I. and Mandal, C. (2014). “Automated Checking of the Violation of Precedence of Conditions in else-if Constructs in Students’ Programs,” 2014 IEEE Int. Conf. MOOC, Innov. Technol. Educ., pp. 201–204.

Shute, V. J. (2008). Focus on formative feedback. Review of Educational Research, 78(1), 153–189. https://doi.org/10.3102/0034654307313795

Souza, Draylson M. de, Oliveira, B. H., Maldonado, J. C., Souza, S. R. S., & Barbosa, E. F. (2014). Towards the use of an automatic assessment system in the teaching of software testing. In 2014 IEEE Frontiers in Education Conference (FIE) Proceedings, 1-8. IEEE.

Souza, Draylson Micael de, Isotani, S., & Barbosa, E. F. (2015). Teaching novice programmers using ProgTest. International Journal of Knowledge and Learning (IJKL), 10(1), 60–77. https://doi.org/10.1504/IJKL.2015.071054

Souza, Draylson M., Felizardo, K. R., & Barbosa, E. F. (2016). A systematic literature review of assessment tools for programming assignments. Proceedings - 2016 IEEE 29th Conference on Software Engineering Education and Training, CSEEandT 2016, 147–156. https://doi.org/10.1109/CSEET.2016.48

Souza, F. C. M., Papadakis, M., Durelli, V., & Delamaro, M. E. (2014). Test Data Generation Techniques for Mutation Testing: A Systematic Mapping. Conference on Software Engineering, (17), 419–432. https://doi.org/10.13140/RG.2.1.3699.9209

Striewe, M., & Goedicke, M. (2014). A Review of Static Analysis Approaches for Programming Exercises. In International Computer Assisted Assessment Conference. Springer, Cham., 100–113.

Taras, M. (2005). Assessment–summative and formative–some theoretical reflections. British Journal of Educational Studies, 53(4), 466–478.

Zougari, S., Tanana, M., & Lyhyaoui, A. (2016a). Hybrid assessment method for programming assignments. In 2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)., 564-569.IEEE. https://doi.org/10.1109/CIST.2016.7805112