Journal of Information Technology Education: Innovations in Practice

An Official Publication of the Informing Science Institute
InformingScience.org

JITEiip.org

Volume 17, 2018

INVESTIGATING THE FEASIBILITY OF AUTOMATIC ASSESSMENT OF PROGRAMMING TASKS

Janet Liebenberg* North-West University, Potchefstroom, South Africa janet.liebenberg@nwu.ac.za

Vreda Pieterse University of Pretoria, Pretoria, South Africa vpieterse@cs.up.ac.za

* Corresponding author

ABSTRACT

Aim/Purpose: The aims of this study were to investigate the feasibility of automatic assessment of programming tasks and to compare manual assessment with automatic assessment in terms of the effect of the different assessment methods on the marks of the students.

Background: Manual assessment of programs written by students can be tedious. The assistance of automatic assessment methods might possibly assist in reducing the assessment burden, but there may be drawbacks diminishing the benefits of applying automatic assessment. The paper reports on the experience of a lecturer trying to introduce automated grading. Students' solutions to a practical Java programming test were assessed both manually and automatically and the lecturer tied the experience to the unified theory of acceptance and use of technology (UTAUT).

Methodology: The participants were 226 first-year students registered for a Java programming course. Of the tests the participants submitted, 214 were assessed both manually and automatically. Various statistical methods were used to compare the manual assessment of student's solutions with the automatic assessment of the same solutions. A detailed investigation of reasons for differences was also carried out. A further data collection method was the lecturer's reflection on the feasibility of automatic assessment of programming tasks based on the UTAUT.

Contribution: This study enhances the knowledge regarding benefits and drawbacks of automatic assessment of students' programming tasks. The research contributes to the UTAUT by applying it in a context where it has hardly been used. Furthermore, the study is a confirmation of previous work stating that automatic assessment may be less reliable for students with lower marks, but more trustworthy for the high achieving students.

Accepted by Editor Bronwyn Hegarty | Received: May 28, 2018 | Revised: August 27, October 29, 2018 | Accepted: November 13, 2018.

Cite as: 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. https://doi.org/10.28945/4150

(CC BY-NC 4.0) This article is licensed to you under a Creative Commons Attribution-NonCommercial 4.0 International License. When you copy and redistribute this paper in full or in part, you need to provide proper attribution to it to ensure that others can later locate this work (and to ensure that others do not accuse you of plagiarism). You may (and we encourage you to) adapt, remix, transform, and build upon the material for any non-commercial purposes. This license does not permit you to use this material for commercial purposes.

Findings	An automatic assessment tool verifying functional correctness might be feasible for assessment of programs written during practical lab sessions but could be less useful for practical tests and exams where functional, conceptual and structural correctness should be evaluated. In addition, the researchers found that automatic assessment seemed to be more suitable for assessing high achieving students.
Recommendations for Practitioners	This paper makes it clear that lecturers should know what assessment goals they want to achieve. The appropriate method of assessment should be chosen wisely. In addition, practitioners should be aware of the drawbacks of automatic assessment before choosing it.
Recommendation for Researchers	This work serves as an example of how researchers can apply the UTAUT theory when conducting qualitative research in different contexts.
Impact on Society	The study would be of interest to lecturers considering automated assessment. The two assessments used in the study are typical of the way grading takes place in practice and may help lecturers understand what could happen if they switch from manual to automatic assessment.
Future Research	Investigate the feasibility of automatic assessment of students' programming tasks in a practical lab environment while accounting for structural, functional and conceptual assessment goals.
Keywords	assessment of programs, automatic assessment, UTAUT, assessment goals

INTRODUCTION

Lecturers are struggling to keep up with their daily responsibilities because of an ever increasing workload. In South Africa, university enrollment has almost doubled, increasing from 495,356 in 1994, to 975,837 in 2016 in public universities and universities of technology (Council on Higher Education, 2018). At the North-West University, where this study was conducted, the number of first-time undergraduate students enrolled increased by 57.9% from 5,929 in 2009 to 9,359 in 2015 (Department of Higher Education and Training, 2017).

Manual programming assessment is time-consuming (Buyrukoglu, Batmaz, & Lock, 2016). Due to increased student numbers, more manpower is needed to deal with the increase in the assessment workload. When multiple people are involved, assessment can become inconsistent (Orrell, 2008).

A possible solution to the above-mentioned problems regarding assessment is to use the assistance of Automatic Programming Assessment (APA) methods. According to Romli, Sulaiman and Zamli (2015), APA methods have greatly improved educators' ability to grade and evaluate student programming exercises.

Automatic assessment of programming assignments is not new. It has been used for more than 50 years (Douce, Livingstone, & Orwell, 2005). TRAKLA2 is an example of a successful automatic assessment tool that has been used since 1991 (Korhonen & Malmi, 2000). ArTEMiS (Krusche & Seitz, 2018), as well as applying the token pattern approach (Poon et al., 2018; Yu, Tang, & Poon, 2017) into PASS (Yu, Poon, & Choy, 2006) are recent additions that improve APA. Modern APA systems require greater sophistication than the early systems owing to the programming environment being more complex. For example, students are required to use integrated development environments (IDE) to produce programs that use graphical user interfaces (GUI). Automatic assessment in this environment is challenging compared to simplistic command-line programming environments.

The study reported in this paper was conducted in order to decide if it is feasible to introduce automatic assessment of the programming assignments of students at the North-West University in

South Africa. The following describes the events and problems that led to this research. Pseudonyms Alice and Betty refer to the two researchers who conducted this research.

Alice lectured a C# class of almost 300 students and found the assessment of such a large group extremely challenging. Alice learned about the automatic assessment tool that was used at Betty's university and inquired about the possibility to use it at her university to assess assignments. Betty responded that it was a possibility, but that their system was at that stage configured to assess only C++, C, Java and ASM assignments. Since C# was not already supported, Alice suggested to a colleague who presented a Java course to approximately 250 students, that they should use the opportunity. Betty mentioned that the run-up time to assess an assignment was three weeks. This was communicated to Alice's colleague.

The colleague decided to send her semester test to Betty and expected to receive the marks in three weeks' time. After the semester test was written, the colleague sent the question paper, a model answer, a marking scheme and all the students' answer files to Betty. Betty immediately responded that automatic assessment of this test could not be done in its current form. The colleague should have consulted with Betty prior to the test for the applicable formulation of the questions and the configuration of the automatic assessment tool in order for the students to upload their answers directly to the tool. At that point the colleague realized that she had to assess the test manually. Alice and Betty then decided to use this incident to investigate the feasibility of using Betty's automatic assessment tool to assess programming tasks at Alice's university.

The aims of the study were to:

- compare manual assessment with automatic assessment in terms of the effect of the different assessment methods on the marks of the students; and
- investigate the feasibility of automatic assessment of programming tasks from a lecturer's point of view based on the unified theory of acceptance and use of technology (UTAUT) (Venkatesh, Morris, Davis, & Davis, 2003).
  (UTAUT is described in the section on technology acceptance.)

In the following three sections, a review of the literature regarding the foundational concepts of this research, namely automatic assessment, assessment goals and technology acceptance, is presented.

AUTOMATIC ASSESSMENT

In this section, the context of the research in relation to the body of knowledge regarding automatic assessment of programming assignments is provided.

Douce et al. (2005) categorize the APA systems developed since inception up to 2005 according to age. In each of the three generations they identified, the APA systems adopted more advanced technologies correlating with the state of the art technologies used for program development in each period. 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 in introductory programming courses.

Many benefits of applying automatic assessment of programming assignments have been reported. Automatic assessment is more likely 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; Nordquist, 2007; Poon et al., 2018), and allows for students to submit multiple improved versions of the programs they have written (Del Fatto et al., 2017; English & English, 2015; Malmi, Korhonen, & Saikkonen, 2002; Pettit, Homer, Gee, Mengel, & Starbuck, 2015; Staubitz et al., 2016). Automatic assessment can play a motivational role in engaging students in the educational process (Šťastná, Juhár, Biňas, & Tomášek, 2015; Staubitz et al., 2016). The most appealing benefit seems to be the possibility of saving time. 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 that can automatically identify correct code, reducing manual assessment to involve only code containing errors.

The automatic assessment tool used in the investigation applies testing-oriented assessment. It requires that both lecturers and students have expertise in developing test suites. For students, it contributes to their understanding of the results of an assessment and their ability to use the results to improve their programs. Edwards (2003) points to the benefits of expecting students to perform more testing and eventually to appreciate its value for the development process. For lecturers, the development of test suites is the instrument to provide relevant feedback to the students, which is directly related to the code they are writing and to the test case that failed (Combéfis & Paques, 2015). Bey, Jermann and Dillenbourg (2018) compared two automatic assessors that differed in their approach. One assesses algorithmic competencies by looking at the code while the other is testing-oriented and looks at the output. They found a positive correlation between the marks produced by the two automatic assessors.

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 Moodle plug-in VPL for the automatic assessment of programming assignments.

Pieterse and Janse van Vuuren (2015) state that it is important that automatic assessment tools should be able to assign partial marks and claim that this can be achieved with careful weighting of different test cases in a test suite in a way that matches the outcomes that are being tested. Birch, Fischer and Poppleton (2016) propose the use of a system that is able to isolate "almost correct" student submissions. In theory, both manual assessment and automatic assessment can be performed at the same fine-grained level and covering all the assessment goals intended for each practical programming assignment. In practice, however, the granularity as well as the assessment goal of any programming assignment is dependent on the person who specifies the marking schemes (Pieterse & Janse van Vuuren, 2015). Ala-Mutka (2005) claims that automatic assessors are capable of evaluating the functionality of entities smaller than a complete program, such as single classes, methods, and even statements. Staubitz, Klement, Renz, Teusner and Meinel (2015) pointed out that there are automatic assessors that can address both dynamic and static assessment of programming assignments while Moreno-León, Román-González, Harteveld and Robles (2017) propose a tool capable of assessing the level of development of aspects of computational thinking.

Staubitz et al. (2016) describe a number of challenges associated with applying automatic assessment of programming tasks. An important challenge, often overlooked, is that considerable time and effort need to be devoted to the implementation of resources for automated assessment (Ala-Mutka, 2005; English & English, 2015; Pieterse, 2013). 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). These aspects became apparent in the investigation discussed in this paper.

ASSESSMENT GOALS

In this section, different classifications of assessment goals from the literature are discussed followed by conclusions regarding the proposed classification of assessment goals of programming assignments.

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 indica-

tive of inaccurate measures of their ability and knowledge (Lister, 2010; Tew & Guzdial, 2010). Often taxonomies, such as Bloom's cognitive taxonomy (Committee of College and University Examiners & Bloom, 1964; Thompson, Luxton-Reilly, Whalley, Hu, & Robbins, 2008) 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.

Sheard et al. (2011) developed a classification scheme that can be used to investigate the characteristics of introductory programming exams. They concluded that the process of classification is highly subjective, and that there is a great variation in the pedagogic intentions and beliefs of the people who set these introductory programming exams. In addition, the classification depends on lecturers' knowledge of the courses they are teaching, how their students would respond to each specific question and it might also be influenced by features of the culture of the institutions at which the lecturers are employed (Sheard et al., 2011). Sheard, Carbone, D'Souza and Hamilton (2013) also found that the process lecturers follow to develop programming exams is largely based on intuition and experience.

In this paper, the assessment of programming tasks is classified in three categories according to the assessment goals of the measure of student skills and understanding of programming tasks, namely structural, functional and conceptual.

Structural

Structural evaluation may include scrutiny of syntax, control structures (sequential, selection and repetition), program complexity, compliance with coding standards, and so forth. These aspects are usually achieved through manual inspection. However, some authors have endeavored to automate aspects of the structural assessment of programs (Ala-Mutka, Uimonen, & Jarvinen, 2004; Ali, Shukur, & Idris, 2007; Waugh, Thomas, & Smith, 2007). Parsons and Haden (2006) developed a drill and practice computer game for mastering syntax constructs. The game itself serves as formative assessment of mastering these constructs and the marks of students, when playing the game, can be used for the 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 (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 Valgrind (2017), Pin (2012) software and Dr. Memory (2016). The automation of establishing the functional correctness of programs is commonplace (Arifi et al., 2015; Ihantola et al., 2010; Staubitz et al., 2015).

Conceptual

Assessing the programming accomplishments of students on a conceptual level is probably the most difficult of the assessment goals to achieve. It is common to carry out this assessment using code-reading questions or questions asking for definitions or explanations in written exams (Petersen et al., 2011). Algo+ attempts to automatically assess students' solutions on a conceptual level by decomposing the student's program and evaluating the recognized underlying program plan (Bey & Bensebaa, 2011; Bey et al., 2018). Visual programming environments, such as Scratch (Resnick et al., 2009) and Alice (Dann, Cooper, & Pausch, 2008) can be used to promote conceptual understanding. The assessment of conceptual aspects in programs written by students is, however, not easy to automate (Posavac, 2015).

TECHNOLOGY ACCEPTANCE

Since the technology acceptance models are used as a lens to evaluate the feasibility of the automatic assessment of programming tasks, the theories of technology acceptance are reviewed in this section, followed by a discussion of the determinants of technology acceptance.

There have been several theoretical models, primarily developed from theories in sociology and psychology, employed to explain technology acceptance and use. The initial theory was the theory of reasoned action (Ajzen & Fishbein, 1980; Fishbein & Ajzen, 1975) and an extension was the theory of planned behavior that stipulates that behavioral intention is influenced by attitudes and subjective norms that in turn influence actual behavior (Ajzen, 1991).

Drawing heavily from the theory of reasoned action (Ajzen & Fishbein, 1980; Fishbein & Ajzen, 1975), Davis, Bagozzi and Warsaw (1989), in the technology acceptance model (TAM), identified and measured a set of common beliefs that apply across a range of IT tools with two primary direct determinants of intention: usefulness and ease of use. TAM2, an extension to TAM, added subjective norm and voluntariness (Venkatesh & Davis, 2000). TAM/TAM2 is widely used in the IS field for clarifying the acceptance of IT tools. The diffusion of innovation (DOI) theory of Rogers (1995) declares that decisions to adopt or reject an innovation are based on the beliefs users form about the innovation. The DOI theory has been used to study a range of innovations (e.g., World Wide Web, spreadsheets, and teaching methods).

Venkatesh et al. (2003) reviewed and synthesized eight theories/models of technology use and formulated a unified model, named the Unified Theory of Acceptance and Use of Technology (UTAUT). The UTAUT is currently widely used in the literature in various contexts, including educational contexts (Al-Adwan, Al-Madadha, & Zvirzdinaite, 2018; Liebenberg, Benadé, & Ellis, 2018; Nur, Faslih, & Nur, 2017). This study made use of the UTAUT as this model provides suitable foundations to determine the attitude of the lecturer towards the use of an automatic assessor. The UTAUT was developed with four core determinants of intention and three moderators of key relationships. The four determinants are performance expectancy, effort expectancy, social influence and facilitating conditions. Self-efficacy, anxiety and attitude towards using technology are the three moderators that are not direct determinants of Behavioral Intention. A discussion of the determinants follows below.

PERFORMANCE EXPECTANCY

Performance expectancy is the degree to which an individual believes that using the system will help to improve performance and therefore enhance the quality of work (Venkatesh et al., 2003). Davis et al. (1989) state that people form intentions towards behaviors they believe will increase their performance and further assert that beliefs influence attitudes that lead to intentions and therefore generate behaviors. In this study, performance expectancy refers to the degree to which the lecturer expected that using an automatic assessor would improve her quality of work.

EFFORT EXPECTANCY

Effort expectancy is defined as the degree of ease associated with the use of the system (Venkatesh et al., 2003). Davis et al. (1989) refer to this as perceived ease of use and claim that it refers to the degree to which a person believes that using a particular system would be free of effort. People will more likely use an application that is perceived easier to use than others and is more likely to be accepted by users. In this study, effort expectancy refers to the degree to which the lecturer regarded an automatic assessor as easy to use.

Social Influence

Social influence refers to the extent to which a person experiences interpersonal influence to use a system from important people within his or her social milieu. In this study, social influence refers to the degree to which the lecturer experienced the influence of peers and students to use an automatic assessor.

Facilitating Conditions

Facilitating conditions (Compatibility) is defined as "the degree to which an individual believes that an organizational and technical infrastructure exists to support use of the system" (Venkatesh et al., 2003). Rogers (1995, p. 224) defined Compatibility as "the degree to which an innovation is perceived as being consistent with the existing values, past experiences, and needs of potential adopters". In this study, Facilitating Conditions refers to the lecturer's belief regarding the ease of installation and use of an automatic assessor and furthermore, the compatibility with the lecturer's current teaching style.

Self-efficacy

Psychologist Albert Bandura (1995) has defined self-efficacy as one's belief in one's ability to succeed in specific situations or to accomplish a task. Self-efficacy in this study refers to the lecturer's belief in her ability to use the automatic assessor.

Anxiety

Anxiety is a feeling of worry, nervousness, or unease about something with an uncertain outcome. In this study, anxiety refers to the degree of stress and hesitance the lecturer experienced with the use of the automatic assessor.

Attitude Towards Using Technology

Attitude towards using technology is defined as an individual's overall affective reaction to using a system (Venkatesh et al., 2003). In this study, attitude towards using technology refers to the lecturer's positive or negative feelings about using the automatic assessor.

Behavioral Intention

Behavioral intention is the dependent variable in this study and refers to a lecturer's intention to use a specified automatic assessor in the future, whether or not he or she used it currently. According to Ajzen (1991, p. 181) "Intentions are assumed to capture the motivational factors that influence a behavior; they are indications of how hard people are willing to try, of how much of an effort they are planning to exert, in order to perform the behavior. As a general rule, the stronger the intention to engage in a behavior, the more likely should be its performance".

METHODOLOGY

In this section, the methods used to collect quantitative data through assessment, as well as qualitative data through reflection are described. In addition, the analysis of two sets of data is explained.

Data Collection Through Assessment

The participants were 226 first-year students registered for a Java programming module in the second semester at the Potchefstroom Campus of the North-West University in South Africa. Data was collected by assessing student's programming solutions both automatically and manually.

The programming question required the students to write the code for two classes, namely SumDiffQuo and MinMax. The class diagram shown in Figure 1 was provided for the SumDiffQuo class. The MinMax class should read five numbers, and calculate the largest and smallest of the numbers. Additionally, the methods of the SumDiffQuo class should be called using the calculated values.

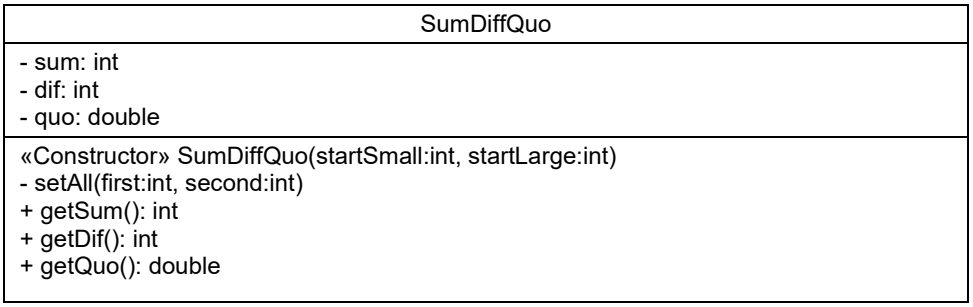


Figure 1. Class diagram

Manual assessment

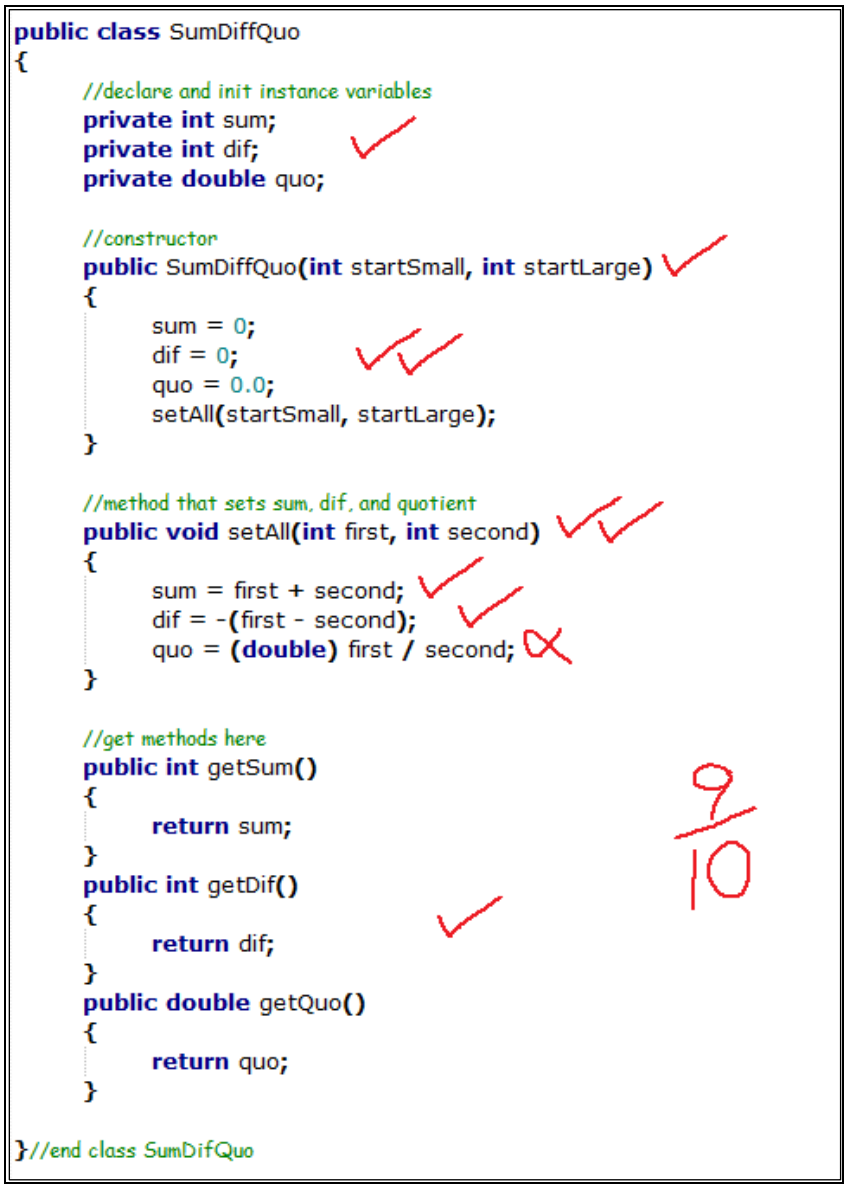
The code the students wrote was saved in files that were given to teaching assistants to evaluate. The assistants were expected to use the rubric shown in Table 1 and they could assign partial marks at their own discretion.

Table 1. Manual assessment rubric

CLASS	OUTCOME	ASSESSMENT GOAL	MAX MARK	TOTAL
	Program not compiling	Structural	-5	-5
SumDiffQuo	declare instance variables	Structural	1	
	constructor heading	Structural	1	
	initialize variables in the constructor	Conceptual	2	
	header of setAll function	Structural	2	
	initialize variables in the setAll function	Conceptual	3	
	implementation of get methods	Conceptual	1	10
MinMax	import libraries	Structural and Conceptual	2	
	main function header	Structural	2	
	declare and initialize variables	Conceptual	2	
	loop header	Structural	1	
	loop increment	Conceptual	2	
	if to replace smallest in loop	Conceptual	3	
	if to replace largest in loop	Conceptual	3	
	create object of class SumDifQuo	Structural	2	
	call functions	Structural and Conceptual	2	
	display output	Structural	1	20
Total				30

This rubric in Table 1 does not correspond to the marking scheme in Table 2 as the assessment methods have different assessment goals. The rubric in Table 1 points to two assessment goals, namely structural and conceptual.

Figure 2 is an example of a program that was manually assessed.



```
public class SumDiffQuo
{
	//declare and init instance variables
	private int sum;
	private int dif;
	private double quo;

	//constructor
	public SumDiffQuo(int startSmall, int startLarge)
	{
		sum = 0;
		dif = 0;
		quo = 0.0;
		setAll(startSmall, startLarge);
	}

	//method that sets sum, dif, and quotient
	public void setAll(int first, int second)
	{
		sum = first + second;
		dif = -(first - second);
		quo = (double) first / second;
	}

	//get methods here
	public int getSum()
	{
		return sum;
	}
	public int getDif()
	{
		return dif;
	}
	public double getQuo()
	{
		return quo;
	}

}//end class SumDifQuo
```


Figure 2. Class diagram

Automatic assessment

The code the students wrote to answer the semester test questions was uploaded to the in-house system at Betty's university. The system is called Fitchfork (Pieterse, 2013). To configure Fitchfork to assess a specific task, one has to write a marking scheme that specifies a number of predefined test cases; therefore the assessment goal is purely functional. The system allows for the specification of marks to be allocated if a test passes, as well as custom feedback per test case in case it passes and in case it fails. In this investigation, the students did not upload their own code and therefore also did not see any of the feedback generated by the system. Nonetheless, the feedback messages were in-

cluded when the marking scheme was prepared. This was done to improve the readability of the marking scheme. It also allowed the researchers to reuse the marking scheme in future. Figure 3 shows typical assessment output that would have been produced by Fitchfork had the students used it in real time.

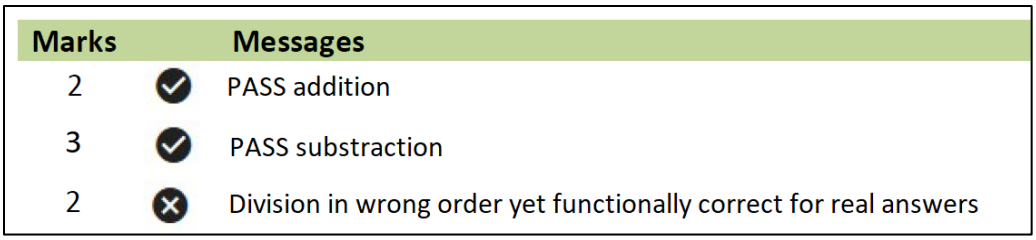


Figure 3. Sample output of Fitchfork

To test the implementation of the functions, Fitchfork was configured to compile their implementation file called `SumDiffQuo.java` along with a driver program that calls each of the functions with selected test cases. Table 2 shows the test cases that were specified by the researchers. To test their driver called `MinMax.java`, Fitchfork was configured to compile their program with a bogus implementation file. When calling the functions from their main program, it should then display the values returned by the bogus functions instead of the correct values.

Table 2. Test cases used for automatic assessment of the functions

FUNCTION	TEST VALUES	EXPECTED OUTPUT	MARK	MESSAGE	MAX MARK	TOTAL
getSum	1, 3	4	2	PASS addition		
		4.0	1	FAIL addition: Result should be an integer		
		other	0	FAIL addition	2	
getDiff	1, 3	2	3	PASS subtraction		
		2.0	2	FAIL subtraction: Result should be an integer		
		-2 or -2.0	1	FAIL subtraction: Values subtracted in the wrong order		
		other	0	FAIL subtraction	3	
getQuo	1, 3 and 7, 22	3.0 and 3.1428	3	PASS division		
		3.0 and 3.0	2	PASS division, but answer should be a real number		
		0.33333 and 0.31818	2	FAIL: Division in wrong order yet functionally correct for real answers		
		other	0	FAIL division	3	8
smallest	3, 1, 5, 2, 4	1	2	PASS Smallest (input random)		
		other	0	Smallest value not identified correctly (input random)	2	

FUNCTION	TEST VALUES	EXPECTED OUTPUT	MARK	MESSAGE	MAX MARK	TOTAL
	1, 2, 3, 4, 5	1 *other*	2 0	PASS Smallest (input ascending) FAIL Smallest value not identified correctly (input ascending)	2	
	8, 6, 5, 4, 3	3 *other*	1 0	PASS Smallest (input descending) FAIL Smallest value not identified correctly (input descending)	1	
largest	3, 1, 5, 2, 4	5 *other*	1 0	PASS Largest (input random) FAIL Largest value not identified correctly (input random)	1	
	1, 2, 3, 4, 5	5 *other*	2 0	PASS Largest (input ascending) FAIL Largest value not identified correctly (input ascending)	2	
	8, 6, 5, 4, 3	8 *other*	2 0	PASS Largest (input descending) FAIL Largest value not identified correctly (input descending)	2	10
Total						18

Table 3 shows how the bogus functions were defined and the output of the students' driver class were assessed. They were awarded 1 mark each for each function call and two marks for formatting the result of the quotient correctly as prescribed with two decimal digits.

Table 3. Automatic assessment of function calls

BOGUS FUNCTION	RETURN VALUE	EXPECTED OUTPUT	MARK	MESSAGE	MAX MARK	TOTAL
getSum	777	777 *other*	1 0	PASS getSum called correctly FAIL getSum not called correctly	1	
getDiff	555	555 *other*	2 0	PASS getDiff called correctly FAIL getDiff not called correctly	2	
getQuo	66.6	66.60 66.6 *other*	3 1 0	PASS getQuo called correctly FAIL Incorrect formatting of answer of getQuo FAIL getQuo not called correctly	3	6
Total						6

Data Collection Through Reflection

The participant was Alice (one of the authors of this paper) who was teaching the C# programming module in the second semester at the Potchefstroom Campus of the North-West University in South Africa. Data were collected through the lecturer's reflections on the feasibility of automatic assessment of programming tasks based on the unified theory of acceptance and use of technology (UTAUT). A structured written reflection was prepared based on the following themes (determinants) of UTAUT: performance expectancy, effort expectancy, social influence, facilitating conditions, self-efficacy, anxiety and attitude towards using technology.

Analysis

To analyze the quantitative data, various statistical methods using SPSS Version 24 were applied. The manual assessment marks of students' solutions were compared with the automatic assessment marks of the same solutions. Since the assessment goals of the two assessment methods did not correspond, only the final marks from the two methods were analyzed. Differences in marks were determined using paired samples T-tests, Pearson correlation analysis and various regression models that were applied to determine correlations.

The qualitative data was obtained in a format that was thematically structured. Further thematic analysis of Alice's reflection could have led to the emergence of additional themes, for example the degree of each determinant demonstrated or the absence of certain elements. Nonetheless, no further analysis was considered necessary.

In the following two sections, the results are discussed and the paper concludes with some recommendations for automatic assessment.

Results

In this section, the researchers report on the results regarding failed APA, a comparison of marks obtained using the different assessment methods and conclude with the lecturer's reflection, based on UTAUT.

Failing to Automatically Assess

Twelve of the 226 students who participated did not submit their electronic documents. Of the 214 submitted documents, only 77 could be automatically assessed, since the remaining 137 documents contained code that did not compile when uploaded to Fitchfork. The remaining 137 documents were further classified based on how the students were penalized for non-compilation during the manual assessment. As can be seen in Figure 4, 112 documents did not compile when tested manually. Sixteen documents that failed to compile automatically were not penalized for non-compilation during manual assessment and nine documents were only partially penalized.

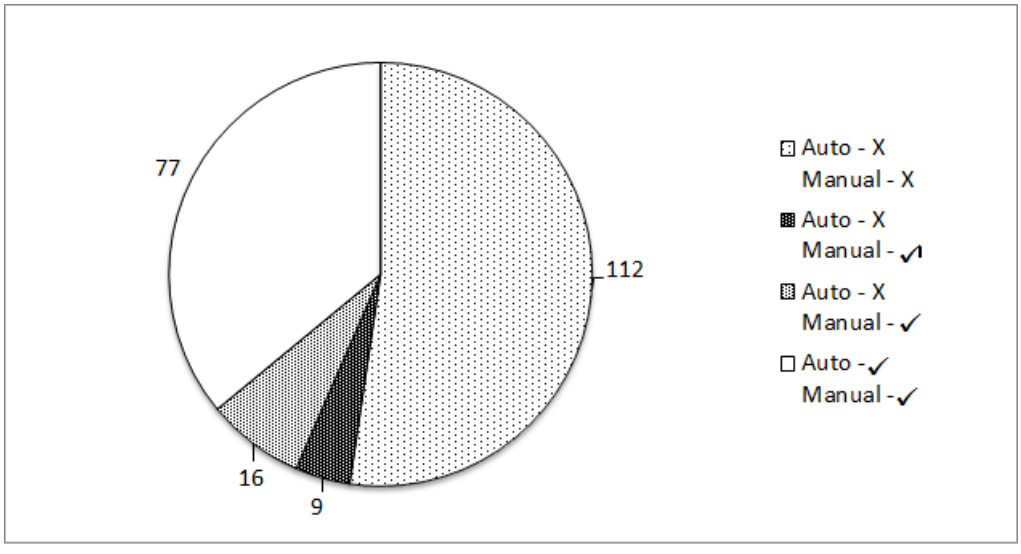


Figure 4. Compile outcomes

The researchers scrutinized the 25 documents that were not penalized for non-compilation during manual assessment although they did not compile when tested automatically. Table 4 shows the types of error that caused the code in these documents not to compile. The number of cases of each error

that was not penalized or only partially penalized during manual assessment is shown. The totals are larger than the number of documents as some students had more than one error in their code. Most of the problems are related to spelling errors in function names or parameter mismatches when functions are called.

When a student program has some of these errors, it may compile when only the student's code is used. The student may, for example, have the same misspelling of a function name in both the main program and in the file containing the function definitions. In the automatic assessment program, the student functions are called using a test harness that will compile only if all the functions in the student's program are spelled exactly as specified and can be called with the parameter types as specified.

Table 4. Compile errors tolerated during manual assessment

ERROR	NOT PENALIZED	PARTLY PENALIZED	TOTAL
Spelling differences	11	3	14
Parameter mismatch	6	7	13
Undeclared variables	0	2	2
Undefined function	0	1	1
Scope errors	0	2	2
Total	17	15	32

Marks

Table 5 shows the descriptive statistics of the marks of the 77 scripts that were assessed manually, as well as automatically. The mean mark for assessments marked manually are higher than those marked automatically.

Table 5. Descriptive statistics (n = 77)

	MEAN	STD DEV	MODE
Automatic assessment	49.84	38.91	0
Manual assessment	65.46	24.76	100

Figure 5 shows the mark distribution for the automatic assessment, as well as the manual assessment of the 77 documents that could be automatically assessed. The large cohort of students who achieved low marks when automatically assessed might be explained by the fact that these students were not accustomed to the strict requirements when automatic assessment is applied and may have been careless when writing their code.

The distribution of the marks students achieved when they were assessed automatically, when compared with the marks they achieved when assessed manually, shows that more students were getting high marks, but also that more were failing. A similar observation was reported by Matthíasdóttir and Arnalds (2015) when they compared the differences in marks between manually assessed programs and automatically assessed programs. Automatic assessment seems not to have the same distinguishing power as that of manual assessment. The marks achieved with manual assessment were closer to normal while those achieved with automatic assessment were closer to binary. Full marks is the mode for manual assessment while zero is the mode for automatic assessment. The fact that the mode values are the opposite extremes is telling of the difference between these methods of assessment. Very few students were awarded extremely low marks during manual assessment while the opposite is true for automatic assessment. When a program contains errors, the manual marking seems to be much more lenient than the automatic marking.

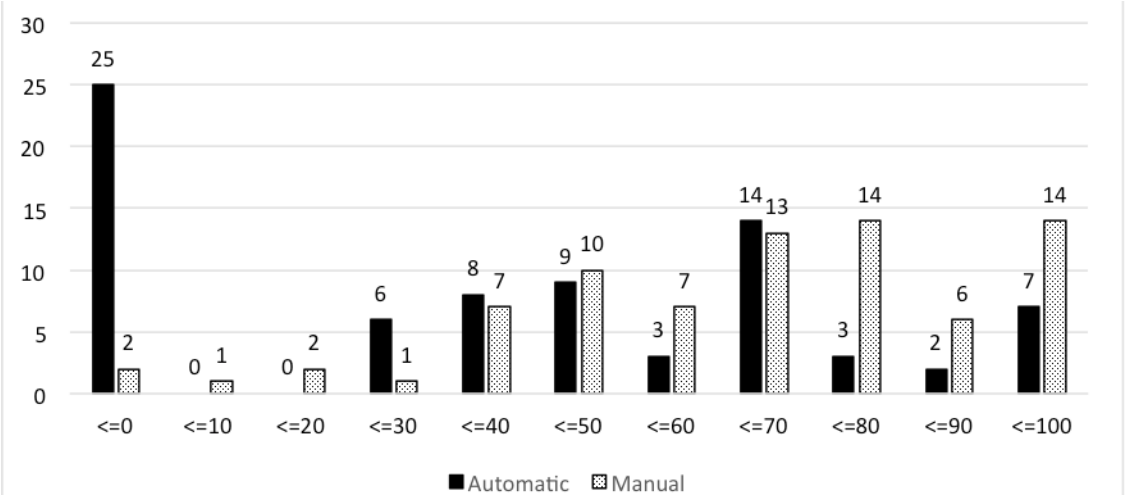


Figure 5. Mark distribution per assessment method (n=77)

Table 6 shows the types of error that occurred in the 18 cases where the students' code compiled, but were awarded zero marks when automatically assessed.

Table 6. Run-time errors (n = 18)

ERROR	TOTAL
Blank output	8
Use of uninitialized variables	6
Endless loop	3
Division by zero	1
Total	18

Figure 6 is a Venn diagram showing the number of students awarded full marks per assessment method. More students (12 documents) were awarded full marks during automatic assessment compared to the number of students who were awarded full marks during manual assessment (9 documents). Only seven were awarded full marks regardless of assessment method.

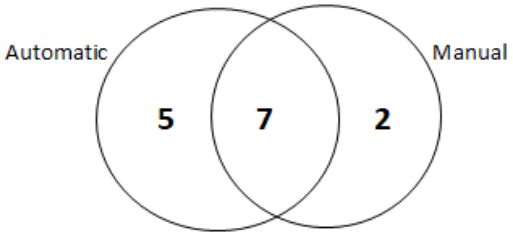


Figure 6. Number of students awarded full marks

Close investigation of the five cases where students got full marks during automatic assessment, but not during manual assessment revealed that the manual assessment was faulty. Full marks should have been awarded according to the assessment rubric. It was found that in the two cases where documents were awarded full marks during manual assessment, but not during automatic assessment, marks were lost owing to ill-formatting in both cases.

The difference between the manual and automatic assessment marks was analyzed using a paired samples T-test. A medium, practically visible difference was found ($d = 0.401$, $p < 0.001$). The differ-

ence can be explained by the fact that the granularity of marks for manual assessment in this particular case was much higher. When comparing the assessment rubric in Table 1 with the test cases in Table 2, it is obvious that a higher level of detail was considered when assessing manually. Test cases for automatic assessment are generally not proficient for fine-grained assessment.

The correlation between the different methods of assessment was determined using Pearson's correlation coefficient. This correlation proved to be a high practical significant relationship ($r = 0.789$, $p < 0.001$). The regression model for the correlation between automatic marking (x) and manual marking (y) is $y = +40.44 + 0.52 * x$. At face value, it seems feasible to substitute x in this formula with the automatic assessment mark, in order to calculate a value comparable with the manual mark. However, $R^2 = 0.623$, thus only 62.3% of the variance is explained by this model.

Bland and Altman (1986) point out that the use of correlations may be misleading and propose that the correlation between the mean and difference of the value pairs be considered. Figure 7 shows the application of their technique using our data.

Ideally, the regression line of such a scatter plot should be horizontal. In our case, the formula $y = 41.09 – 0.33 * x$ defines this line. The line has a visible downslope of -0.33 that indicates that the correlation between automatic assessment and manual assessment is not reliable, which is not surprising since the assessment goals of the methods were different. However, Bey et al. (2018) did find a positive correlation when they compared two automatic assessors with differing assessment goals. Although the researchers could not establish a correlation, the gradient of the regression line in Figure 7 indicates that automatic assessment may be more reliable for students with higher marks since the correlation becomes stronger as the mean of the marks increases. A possible explanation for this is that correct programming solutions are awarded good marks regardless of assessment goals while the marks assigned to partially correct solutions might differ for different assessment goals.

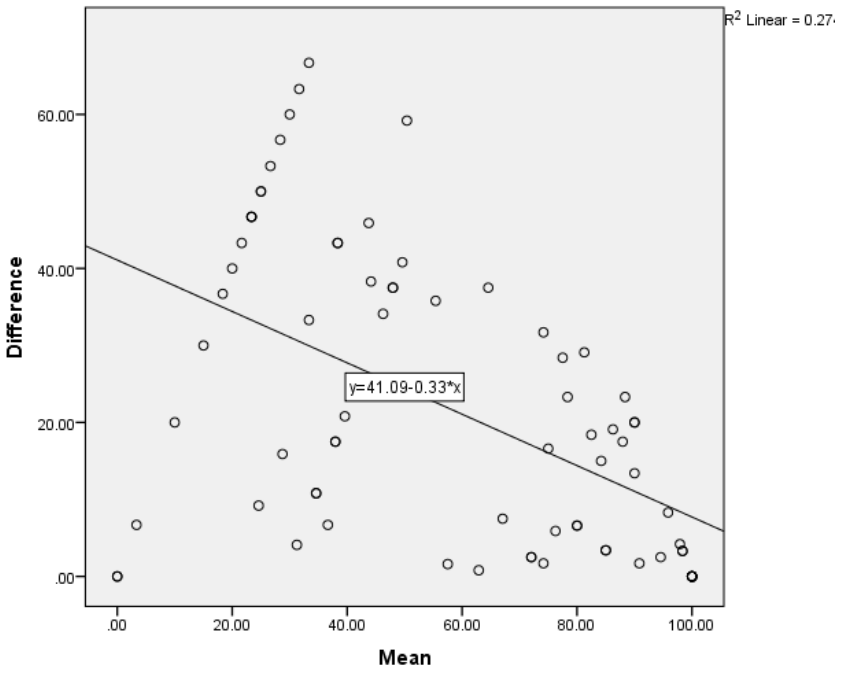


Figure 7. Difference plot (n=77)

To further investigate the above matter, the students are classified into three categories based on their manual assessment marks. The categories are Fail (< 50%), Pass (>= 50% and < 75), and Pass with distinction (>= 75). Table 7 shows the results of the T-test. Since the effect size (d) for the failing students ($d = 1.303$, $p < 0.001$) is greater than 0.8, the results show a very large, practically significant difference, thus supporting the findings in Figure 7 that automatic assessment may be less reliable for

students with lower marks. Similarly, for the students who passed, the large effect size ($d = 0.747$, $p < 0.001$) implies that the automatic assessment is not reliable. However, the small effect size ($d = 0.151$, $p = 0.409$) for the distinction candidates indicates that automatic assessment is likely to be more trustworthy for the high achieving students.

Table 7. Differences per performance categories

PERFORMANCE CATEGORY	AUTOMATIC		MANUAL		EFFECT SIZE (d)	p
	Mean	StdDev	Mean	StdDev		
Fail (n=19)	12.07	14.70	32.64	15.78	1.303	< 0.001
Pass (n=30)	38.62	32.62	62.99	7.74	0.747	< 0.001
Pass with distinction (n=28)	87.49	19.02	90.37	9.31	0.151	0.409

Lecturer Written Reflection

In this section, Alice's reflection on her intentions to use the automatic assessment tool used at Betty's university is described. This reflection is structured using the UTAUT model and the determinants and moderators discussed in the section on Technology Acceptance. Alice's reflection is based on her subjective feelings engendered by personal observations and experiences towards the feasibility of automatic assessment of programming tasks at her university.

Performance expectancy

My initial expectation was that an automatic assessor would increase my performance. I expected an automatic assessor to be a useful tool in my work as a lecturer and that automatic assessment would be more reliable than manual marking by teaching assistants. I expected automatic assessment to save me a lot of time and therefore increase my productivity. However, the actualization was entirely different from this expectation. In the first place, the usefulness of the tool proved to be a disappointment since I could not use the tool for the C# course. The reliability of the automatic assessment compared to the manual assessment also proved to be questionable, since the analysis showed that it might only be reliable for high achieving students. Furthermore, Fitchfork turned out to be a cumbersome tool to use even for the Java course. The compilation of the marking scheme proved to be quite complicated and very time-consuming and in fact reduced my productivity.

Effort expectancy

I expected that the process to apply automatic assessment is clear and understandable and it would be easy to learn to use the automatic assessment tool and to apply it in my course. I further expected that it would be easy for me to become skillful at using the tool. However, the automatic assessment process proved to be anything but effortless. I learned that to set a marking scheme for the tool required multiple additional skills, such as competency in XML and regular expressions, as well as in the compilation of effective test cases.

Social influence

I was definitely influenced by people in my social environment to start using an automatic assessment tool. My colleagues and I increasingly experienced the pressure of assessing programming exams combined with ever increasing student enrollment and this encouraged me to investigate alternative methods of assessment. My colleagues, academic peers and even my family supported the idea that some of my work could be automated and that I should use the system.

My proposal to implement the automatic assessor in my department was enthusiastically accepted by the management of the university and they granted financial support for the initiative.

Facilitating conditions

I realize that the current investigation made use of the resources at Betty's university, but at this stage I believe that my university has the resources necessary to use the tool and the tool is compatible with our facilities. I have confidence that Betty's university will be available for assistance with system difficulties.

Self-efficacy

I cannot complete an automatic assessment task using the tool if there is no one around to tell me what to do as I go and, furthermore, no help facility for assistance exists. I realize that the process is complicated and I still have a lot to learn before I will be skillful to use the automatic assessment tool and apply it in my course.

Anxiety

Since I am a power user of technology, I am not anxious to use the automatic assessment tool, despite the fact that it is an unfamiliar system. I am positive that it will be more fun to rise to the challenge to compile an effective marking scheme for the tool than to wrestle through the manual marking of hundreds of programs.

Attitude towards using technology

Initially, I felt positive about the prospects of using the APA, but gradually my enthusiasm dwindled as my journey through the process of using the APA progressed.

Behavioral intention

I intend to use the automatic assessor in future during practical lab sessions despite the fact that very little of my initial expectations were met.

DISCUSSION

Comparative analysis uncovered that the marks do not correlate between the different assessment methods; automatic assessment seems to be useful for high achieving students. However, the lecturer would not know in advance who the high achievers would be and, in any case, manual marking of the high achievers' tests is the easiest and takes the least time – it is the low achievers whose programs often do not even compile who pose the greatest challenge. The quality of the marking scheme improves the quality of assessment for both methods of assessment; however, the manual method has human intelligence for interpretation that the automatic method does not have. The lecturer could use a similar system reported by Del Fatto et al. (2017) whereby the system automatically identifies correct code and therefore time is saved since only the erroneous programs have to be manually assessed.

An automatic assessment tool verifying functional correctness might be feasible for assessment of programs written during practical lab sessions but could be less useful for practical tests and exams where both functional and structural correctness should be evaluated. It is feasible to use an automatic assessor in practical lab sessions, since accurate calibration of the competence of the students is of secondary importance and the benefits of rapid feedback outweigh the drawback of low accuracy.

The creation of automatically assessable programming assignments, along with the test cases to cover the required assessment goals at the desired granularity for these assignments, is considered to be a

challenging exercise (Ala-Mutka, 2005; Pieterse, 2013; Staubitz et al., 2015). This reality, experienced in this study, plays an important role in the inclination of instructors to automatically assess student submissions at a courser granularity and covering fewer assessment goals as they would when using manual assessment of programming tasks. It can be explained by the fact that instructors may find it tedious and difficult to design the required large number of test cases to satisfy the pedagogical requirements when having to apply automatic assessment of the tasks (Cerioli & Cinelli, 2008).

Alice did not expect in terms of the effort and performance determinants that the use of the assessment tool would be difficult and time-consuming. In terms of social influence and facilitating conditions, Alice did not experience these as inhibiting factors. Alice's attitude towards technology is the essence of her behavioral intention. Her initial positive attitude can be linked to the fact that she was generally not anxious to use new technologies. However, her misconceptions regarding effort expectancy may have caused her positive attitude to wane.

Alice's reflection did not mention assessment goals at all. It appears that Alice did not consider assessment goals and consequently did not expect her assessment goals to differ so significantly from the assessment goals of the automatic assessor. She came to the realization that her educational style and beliefs regarding assessment of programming are in sharp contrast to the assessment style and principles of the automatic assessor. This realization corresponds with the findings of Sheard et al. (2011) and Sheard et al. (2013) that there is a great variation in the pedagogical intentions and beliefs of people who set programming exams and the process is based largely on intuition and experience.

Based on Alice's testimony on her behavioral intent it seems as if she no longer believes that the automatic assessor is feasible for assessment of tests and exams, but she intends to use it for assessment of programming assignments written during practical lab sessions. We envisage that the advantages of using the automatic assessor would outweigh the disadvantages in a situation where it is of secondary importance that the assessment should accurately determine the proficiency of the students.

Limitations

The researchers recognized some limitations to the study. The programming assignment was small, it had only two classes, and a larger assignment may have produced different results. The students did not have experience in being automatically assessed and with more familiarity using this method the difference between the two assessment results may be less pronounced. The fact that only 77 of the 214 submitted documents could be analyzed presented a small sample; a larger sample would be beneficial. The observations made in the study are dependent on the assessment rubric that was used with manual assessment, as well as the test cases that were used in the automatic assessment. The researchers are not in the position to claim that either the rubric or the test cases are perfect, and the use of other rubrics and test cases might not present similar results. The qualitative data presented the view of only one lecturer and other lecturers may have a different experience with the automatic assessor. Furthermore, this experience is based on one automatic assessor and it is likely that the use of other automatic assessors may be viewed in a more positive light.

Lessons Learned

It was discovered that different assessment goals exist, and it is important in the design of assessment of programming assignments, tests and exams. In addition, automatic assessment may pose numerous challenges not often reported in the literature. The researchers realized that the introduction of an automatic assessor is likely to require an adjustment period before the full benefits could be realized.

CONCLUSION

In view of the increasing enrollment of students and in the light of the availability of automatic assessment tools, automatic assessment seems feasible for assessments in lab sessions, but may be challenging to use for tests and exams where it is important to verify functional, structural and conceptual correctness. In addition, the researchers found that automatic assessment seemed to be more suitable for assessing high achieving students.

This study would be of interest to lecturers considering automated assessment. The two assessments used in the study are typical of the way grading takes place in practice, and this may help lecturers understand what could happen if they switch from manual to automatic assessment. Although, in our study, comparative analysis revealed that the marks do not correlate between the different assessment methods, automatic assessment seems to be useful for high achieving students.

Being mindful of assessment goals in marking schemes, test cases and formulation of questions may improve the overall quality of assessment for both methods of assessment.

RECOMMENDATIONS

It is recommended that lecturers identify the assessment goals they want to achieve and choose the appropriate method of assessment wisely. In addition, lecturers should be aware of the drawbacks of automatic assessment before choosing it.

FUTURE RESEARCH

Future research may include an investigation of the feasibility of automatic assessment of student programs in a practical lab while accounting for different assessment goals. This can be achieved by repeating the study reported in this paper, but with the following differences:

- Conducting the automatic assessment in a practical lab instead of after the fact in a practical test. In this situation, the students can benefit from real-time feedback. The manual assessment will be conducted after the fact.
- Aligning the assessment goals between the two assessment methods. This can be achieved by synchronizing the manual assessment rubric with the test cases used for automatic assessment.
- Gathering data from students and the lecturer on the feasibility of automatic assessment based on UTAUT, instead of the reflection of one lecturer.

ACKNOWLEDGMENT

The authors would like to thank Annette van der Merwe at the North-West University for her cooperation in using the data obtained from the manual and automatic assessment of the solutions submitted by her students.

REFERENCES

Ajzen, I. (1991). The theory of planned behaviour. *Organisational Behaviour and Human Decision Processes, 50*(2), 179-211. https://doi.org/10.1016/0749-5978(91)90020-T

Ajzen, I., & Fishbein, M. (1980). *Understanding attitudes and predicting social behaviour.* Englewood Cliffs, NJ: Prentice-Hall.

Al-Adwan, A. S., Al-Madadha, A., & Zvirzdinaite, Z. (2018). Modeling students' readiness to adopt mobile learning in higher education: An empirical study. *The International Review of Research in Open and Distributed Learning, 19*(1). https://doi.org/10.19173/irrodl.v19i1.3256

Ala-Mutka, K. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education, 15*(2), 83-102. https://doi.org/10.1080/08993400500150747

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. https://doi.org/10.28945/300

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. https://doi.org/10.1109/ICoCS.2015.7483289

Bandura, A. (1995). *Self-efficacy in changing societies*. NY: Cambridge University Press.

Bey, A., & Bensebaa, T. (2011). *Algo+, an assessment tool for algorithmic competencies*. Paper presented at the 2011 IEEE Global Engineering Education Conference (EDUCON), Amman, Jordan. https://doi.org/10.1109/EDUCON.2011.5773260

Bey, A., Jermann, P., & Dillenbourg, P. (2018). A Comparison between two automatic assessment approaches for programming: An empirical study on MOOCs. *Journal of Educational Technology & Society, 21*(2), 259-272.

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

Birch, G., Fischer, B., & Poppleton, M. (2016). Using fast model-based fault localisation to aid students in self-guided program repair and to improve assessment. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, 168-173. https://doi.org/10.1145/2899415.2899433

Bland, J. M., & Altman, D. (1986). Statistical methods for assessing agreement between two methods of clinical measurement. *The Lancet, 327*(8476), 307-310. https://doi.org/10.1016/S0140-6736(86)90837-8

Buyrukoglu, S., Batmaz, F., & Lock, R. (2016). Increasing the similarity of programming code structures to accelerate the marking process in a new semi-automated assessment approach. *Proceedings of the 11th International Conference on Computer Science & Education (ICCSE)*, 371-376. https://doi.org/10.1109/ICCSE.2016.7581609

Cerioli, M., & Cinelli, P. (2008). GRASP: Grading and Rating ASsistant Professor. *Proceedings of the ACM-IFIP IEEIII 2008 Informatics Education Europe III Conference*, 37-51.

Combéfis, S., & Paques, A. (2015). Pythia reloaded: An intelligent unit testing-based code grader for education. *Proceedings of the 1st International Workshop on Code Hunt Workshop on Educational Software Engineering*, 5-8. https://doi.org/10.1145/2792404.2792407

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. https://doi.org/10.1145/2993270.2993271

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

Council on Higher Education. (2018). *VitalStats public higher education 2016*. Retrieved from http://www.che.ac.za/sites/default/files/publications/CHE_VitalStats_2016%20webversion.pdf

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

Davis , F. D., Bagozzi, R., & Warshaw, P. (1989). User acceptance of computer technology: A comparison of two theoretical models. *Management Science, 35*(8), 982-1003. https://doi.org/10.1287/mnsc.35.8.982

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.

Department of Higher Education and Training. (2017). *Statistics on post-school education and training in South Africa: 2015* (pp. 84). Retrieved from http://www.dhet.gov.za/DHET%20Statistics%20Publication/Statistics%20on%20Post-School%20Education%20and%20Training%20in%20South%20Africa%202015.pdf

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. https://doi.org/10.1145/1163405.1163409

Dr. Memory. (2016). Memory debugger for Windows, Linux, and Mac [Computer Software]. Retrieved from http://www.drmemory.org

Edwards, S. H. (2003). Improving student performance by evaluating how well students test their own programs. *Journal on Educational Resources in Computing (JERIC), 3*(3), 1. https://doi.org/10.1145/1029994.1029995

English, J., & English, T. (2015). Experiences of using automated assessment in computer science courses. *Journal of Information Technology Education: Innovations in Practice, 14*, 237-254. https://doi.org/10.28945/2304

Fishbein, M., & Ajzen, I. (1975). *Belief, attitude, intention, and behavior: An introduction to theory and research*. Reading, MA: Addison-Wesley.

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. https://doi.org/10.1145/1930464.1930480

Korhonen, A., & Malmi, L. (2000). Algorithm simulation with automatic assessment. *ACM SIGCSE Bulletin, 32*(3), 160-163. https://doi.org/10.1145/353519.343157

Krusche, S., & Seitz, A. (2018). ArTEMiS: An automatic assessment management system for interactive learning. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education, Baltimore, Maryland,* 284-289. https://doi.org/10.1145/3159450.3159602

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., Benadé, T., & Ellis, S. (2018). Acceptance of ICT: Applicability of the Unified Theory of Acceptance and Use of Technology (UTAUT) to South African students. *The African Journal of Information Systems, 10*(3), 1.

Lister, R. (2010). Computing education research - Geek genes and bimodal grades. *ACM Inroads, 1*(3), 16-17. https://doi.org/10.1145/1835428.1835434

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. https://doi.org/10.1145/2960811.2967155

Malmi, L., Korhonen, A., & Saikkonen, R. (2002). Experiences in automatic assessment on mass courses and issues for designing virtual courses. *ACM SIGCSE bulletin, 34*(3), 55-59. https://doi.org/10.1145/637610.544433

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. https://doi.org/10.1145/2812428.2812470

Moreno-León, J., Román-González, M., Harteveld, C., & Robles, G. (2017). On the automatic assessment of computational thinking skills: A comparison with human experts. *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems, Denver, Colorado,* 2788-2795. https://doi.org/10.1145/3027063.3053216

Nordquist, P. (2007). Providing accurate and timely feedback by automatically grading student programming labs. *Journal of Computing Sciences in Colleges, 23*(2), 16-23.

Nur, M. N. A., Faslih, A., & Nur, M. N. A. (2017). Analysis of behaviour of e-learning users by Unified Teory of Acceptance and Use of Technology (UTAUT) model: A case study of vocational education in Halu Oleo University. *Jurnal Vokasi Indonesia, 5*(2).

Orrell, J. (2008). Assessment beyond belief: The cognitive process of grading. In A. Havnes & L. McDowell (Eds.), *Balancing dilemmas in assessment and learning in contemporary education* (pp. 251-263). London: Routledge.

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. https://doi.org/10.1145/1953163.1953340

Pettit, R., Homer, J., Gee, R., Mengel, S., & Starbuck, A. (2015). An empirical study of iterative improvement in programming assignments. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 410-415. https://doi.org/10.1145/2676723.2677279

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

Pieterse, V., & Janse van Vuuren, H. (2015). Experience in the formulation of memoranda for an automarker of simple programming tasks. *Proceedings of the 44th Annual Southern African Computer Lecturers' Association (SACLA)*, 210-214.

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.

Pin. (2012). A dynamic binary instrumentation tool [Computer Software]. Santa Clara, CA: Intel Corporation. Retrieved from http://software.intel.com/en-us/articles/pintool

Poon, C. K., Wong, T.-L., Tang, C. M., Li, J. K. L., Yu, Y. T., & Lee, V. C. S. (2018). *Automatic assessment via intelligent analysis of students' program output patterns.* Paper presented at the International Conference on Blended Learning. https://doi.org/10.1007/978-3-319-94505-7_19

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., Brennan, K., . . . Silverman, B. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60-67. https://doi.org/10.1145/1592761.1592779

Rogers, E. M. (1995). *Diffusion of innovations* (4th ed.). New York: Free Press.

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

Sheard, J., Carbone, A., Chinn, D., Laakso, M.-J., Clear, T., De Raadt, M., . . . Philpott, A. (2011). Exploring programming assessment instruments: A classification scheme for examination questions. *Proceedings of the Seventh International Workshop on Computing Education Research*, 33-38. https://doi.org/10.1145/2016911.2016920

Sheard, J., Carbone, A., D'Souza, D., & Hamilton, M. (2013). *Assessment of programming: Pedagogical foundations of exams.* Paper presented at the Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education. https://doi.org/10.1145/2462476.2465586

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. https://doi.org/10.18267/j.aip.71

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 onTeaching, 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 excercises in online environments. *Proceedings of the 2016 IEEE Global Engineering Education Conference (EDUCON)*, 314-323. https://doi.org/10.1109/EDUCON.2016.7474573

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. https://doi.org/10.1145/1734263.1734297

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.

Valgrind. (2017) (Version 3.13.0) [Computer Software]. Retrieved from http://www.valgrind.org/

Venkatesh, V., & Davis, F. D. (2000). A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science, 46*(2), 186-204. https://doi.org/10.1287/mnsc.46.2.186.11926

Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User acceptance of information technology. *MIS Quarterly, 27*(3), 425-478. https://doi.org/10.2307/30036540

Waugh, K., Thomas, P., & Smith, N. (2007). Teaching and learning applications related to the automated interpretation of ERDs. *Proceedings of the 24th British National Conference on Databases (BNCOD'07)*, 39-47. https://doi.org/10.1109/BNCOD.2007.19

Yu, Y., Poon, C., & Choy, M. (2006). Experiences with PASS: Developing and using a programming assignment assessment system. *Proceedings of the 6th International Conference on Quality Software(QSIC), Beijing,* 360-368.

Yu, Y., Tang, C., & Poon, C. (2017). Enhancing an automated system for assessment of student programs using the token pattern approach. *IEEE 6th International Conference on Teaching, Assessment, and Learning for Engineering (TALE),* 406-413.

BIOGRAPHIES

Janet Liebenberg is a senior lecturer at the Potchefstroom campus of the North-West University, South Africa. She obtained a PhD in 2015 from the North-West University and she teaches graphical interface programming. Her research interests include the teaching and learning of programming and minorities in Computer Science and Information Systems.

Vreda Pieterse is a lecturer in Computer Science at the University of Pretoria, South Africa since July 2001. Her teaching experience includes computer literacy courses and programming courses using various programming languages. She has also taught undergraduate modules in operating systems, design patterns and software engineering and postgraduate courses in software engineering and software architecture. She completed her PhD in Computer Science at the University of Pretoria. Her research endeavors focus on algorithmics and the construction of algorithm taxonomies. She also has an active interest in Software Engineering, effective teaching practice and teamwork.