NIGHTINGALE PUBLICATIONS
AND RESEARCH INTERNATIONAL]

THE USE OF E-ASSESMENT IN COMPUTER SCIENCE EDUCATION

***MUHAMMED ABDULAZEEZ HASSAN & **SAMIR HMED**
**Federal College of Education, Zaria*
***Shehushagari Colledge of Education Sokoto*

Abstract

Most e-assessment systems are mainly used in simplifying the high manual effort by teaching personnel in assessing the students. This paper will be reviewing an extension proposed for the e-assessment systems that will be requiring some higher-order cognitive skills. The most recent module will allow programming exercises to be assessed in regard to a specific back-to-back testing and test-driven development. Although these e-assessment systems are still a prototype, its usage by the programming lecturers has proved to be practical in performing e-assessments.

Keywords: *Software testing, Assessment, Software tests. Online learning, E-assessment*

Introduction

Assessing a student in the process of learning and teaching is very important as it helps them in measuring their learning performance as well as identifying their individual success. Despite so, assessment actually affects teaching personnel in a whole different way since they are required to put an extensive amount of manual effort to conduct the assessments on the students. Due to this, most universities will try to help and support them by implementing e-learning systems. Nowadays, most e-assessment systems focus more on examinations that are in their simplest forms. This includes tests such as input of short text or multiple-choice questions.

The reason for this is mainly because these systems are not suitable to be used in assessing higher-order cognitive skills (Heywood, 2000). However, such skills are extremely important to be assessed in

Computer science education since memorized knowledge will not suitable to be assessed for these students. There are previous studies and papers that focus on the introduction of e-assessment systems that pointed out its new ability that made it possible to be used in education relating to computer science and mathematics in assessing formal specification techniques (Gruttmann, et al., 2010) as well as formal proofs(Bohm, et al., 2008).

For the purpose of this paper as mentioned earlier, the extension of such e-assessment systems for programming exercises as proposed by (Usener, et. al, 2012) in addition to the type of software testing performed will be introduced. It is also crucial for programming to be aligned with software testing in this paper in order to make sure that the quality achieved is adequate since assessing it is by no means a trivial task. It is also important to note that some contrasting solutions will probably be equal in a semantic manner while there will be a few semantically distinctive solutions that possibly be able to solve the exercise being provided.

In addition to this, in order to introduce different techniques for testing as well as motivating the test on software, the test case generator Muggl (Kuchen & Majchrzak, 2009) will be incorporated in this paper. Muggl will be able to offer the students with a system that generated the test cases individually for them in which can be used in improving their solution prior to the final submission electronically. Throughout this paper, a few contributions will be made by first expanding the body of knowledge in conjunction to the application of e-assessment in the scope of computer science education. Then, this paper will be covering the description of the innovative approach to integrate an e-assessment tool with an automated tool for test case generation as proposed by Usener, Majchrzak and Kuchen (2012). Following to this step, this paper will be highlighting the educational merits that are brought by the e-assessment system specifically for programming as well as for testing exercises. And finally, this paper will be reviewing the effectiveness and acceptance of the approaches among the students.

In order to completely use these approaches, this paper has been organized based on the following structures. Section 1 of this paper, which is this section, introduces the topic to the readers to give a general idea on what is being studied in this paper. Following this is Section 2 that introduces the approach's background while discussing on the previous work by other

researchers that are related to the topic being studied. Section 3 on the other hand explains the tools and their integration with an exemplary scenario being described in Section 4. Next, a review on the effectiveness and acceptance of the approaches among the students of computer science course is provided in Section 5 followed by a conclusion as well as some highlights for future work in Section 6.

Background

The foundation of this study will be highlighted in this section to further deepen the understanding on the approach taken in this study. This study proposes the usage of e-assessment in computer science education while motivating two types of software testing techniques to be used for the system.

E-assessments for Computer Science Education

Assessments are important for every teaching and learning scenario since it can help in identifying and measuring an individual's learning achievements (Usener & Majchrzak, 2011) as well as acting as the indicator for lecture improvements (Chudowsky, et al., 2001). These assessments will be able to show any clarifications are needed in certain parts of a learning unit. Hence, it is important for both the teachers and learners. Thus, it is critical for computer science students to be taught with analytic, creative and constructive skills in addition to the basic knowledge so as to enable them being capable in developing and enhancing software systems. This is supported by the fact that almost all the relevant learning objectives in computer science actually need constant practice and intensive participation. By solving the exercises that are based on contents from the lecture, students will be able to passively transfer consumed information to active knowledge (Rohde, et al., 2008). However, because the students need to attend a mass number of lectures, the formative assessment's organization will be almost impossible, especially since there is a decrease in resources and low personnel capacities (Wannemacher, 2007) although e-assessment will be able to solve this issue. Not to mention that it can also be considered as an opportunity in providing formative assessments in education under computer science courses despite the facts that most common e-assessment systems are not suitable in examining any skills that are constructive, creative and analytic. The reason for this is mainly because these systems will only be

providing simple question types most of the times that include multiple choice questions or questions with short text answers (Jenkins & Cook, 2010). Thus, a more opened question type is crucial to have in examining skills that are constructive, creative and analytic.

Test-driven Development

Usually, modules will first be coded and then the test cases will be created. However, test-driven development (TDD) or also named as test-first development (Beck, 2002), actually proposes a reverse process of this (Astels, 2003) where a test case is first written for the intended functionality using a unit testing tool(Koskela, 2007)before the module is being programmed. This resulted in the process to be able to keep the system simple since it will only be implementing functionalities that are needed. According to (Beck, 2002) test-driven development also is able to improve the maintainability of the codes as well as code re-usage in addition of having high level tests' functional coverage. Writing codes that will only corresponds to the test cases will needs some discipline, however it will still facilitates finding of programs' defects right after the codes were introduced (Koskela, 2007). According to Beck (2002), it is better to remove any defects in the early stages of development as lesser costs will be involved.

Back-to-Back Testing

Back-to-back testing refers to the technique selected for fields where system failure is inacceptable. A program' prototypes, which are supposed to be semantically equal (Roitzsch, 2005)are built based on the same specifications by independent development teams. The yields of the prototypes executed after an emphasis of programming is then checked and any differences detected will be directly reported to the development groups since they imply that one or more prototypes contains at least a defect. Testing is then iterated until the point that every prototype carries on similarities. Although the number of defects can drastically be reduced this method of testing is very costly (Roitzsch, 2005). As a test that diversifies, it tries to conquer the blemishes that most other testing strategies have because of their heuristic nature. Hence, back-to-back testing is picked when the test required a significant justifies on the increased efforts and can be considered as cost effective if utilized under appropriate situations (Vouk, 1990).

Previous Literature

This study will discuss on related work from three perspectives: programming exercises' e-assessments, automated generation of test cases and the combinations of both approaches. As to e-assessment solutions for programming exercise, there are already some research projects with different evaluation techniques exist at the moment such as DUESIE, ELP and Praktomat. Each of the frameworks offered exercises' assessment with respect to Java programming language although both DUESIE and Praktomat still support other languages.

The development of Praktomat was for the purpose of aiding a programming workshop for second year students of computer science courses. According to Storzer, et al. (2002), Praktomat's key features are solution testing that is both static and dynamic in addition to the students' capacity to comment and view different solutions after the initial submission. Tests that are dynamic and static are usually utilized for assessment although they can also be utilized by the students to review their work before submission. DUESIE on the other hand is relatively similar to Praktomat as it also utilizes static and dynamic tests to check the style of coding as well as the functionality of the students' program code (Wismuller, et al., 2008).Interestingly, DUESIE doesn't check students' answer heretofore to spare them from understanding their activity by trial and error method. Thus, static and dynamic test in both Praktomat and DUESIE frameworks depend on instruments for style checking as well as unit testing that took after the tutor'smanual amendments.

In contrast to the previous two systems, ELP is basically intended for self-assessment and designed to help the students with almost no programming background to learn fundamental features of Java (Bancroft, et al., 2004).

The framework gives brief exercises based on the "fill in the gap" concept with each concentrating on one particular coding issue. A solution being turned in is contrasted with stored sample solutions and a quick input by the auxiliary similitude is then provided (Bancroft, et al., 2004). The input should empower the students to reconsider and redress the answer. Despite this, ELP only provides prompt criticism without human cooperation that is suited for small exercises that are well defined.

The automated test case generation (TCG) is a dynamic field of research that infers a few impediments regardless of its merits (Graham & Fewster, 1999). For reasons of study's scope, only the features that are directly linked with

Muggl will be highlighted such as shown in the list below. Despite these, neither of the e-assessment tools described has the capacity to create test cases for students' programs as most of the e-assessment tools usually will only be providing simple exercises.

1. IBIS representatively performs Java byte code while utilizinga constraint solver regardless of its auspicious approach. The work by Meudec & Doyle (2003) is in its initial state and there are no recent papers have been presented on IBIS.
2. Fischer and Kuchen (2007) introduce the TDG approach for functional logic programming specifically for Curry with the ideas being very similar for their approach and Muggl although a constraint solver is not included in their tool.
3. The counterpart of Muggl,Pex (de Halleux & Tillmann, 2008), is for .NET based programs and it utilizes the Microsoft Common Language Infrastructure. It is reported to be performing very well with its constraint solverZ3 (Bjoerner & de Moura, 2008) being a satisfiability module solver for the ories that is different to the concept of Muggl's solver.

Incorporation of EASy and Muggl

This section will be introducing EASy and Muggl before highlighting the synergy of both tools.

EASy

The development of EASy was meant to offer a program that will be able to help assessing various exercise electronically in computer science education (Gruttmann, 2010). EASy specifically focus on giving exercises that are complex that will allow students to hone their constructive, creative and analytic skills. Hence, EASy is currently giving the practice modules for Java programming exercises, software verification proofs, a multiple-choice as well as mathematical proofs module.

Muggl

Muggl (Kuchen & Majchrzak, 2009)on the other hand refers to a tool for the automated test cases generation. Unlike most tools for automated test cases generation that are random or relying on some form of input that is pre-

existing, Muggl actually uses the structure of a program in drawing the conclusions regarding the test cases needed. Besides, Muggl only processes Java byte-code instead of utilizing any source code.

The Tools' Ideal Interaction

Extending EASy with the functionality of Muggl is relatively easy mainly because it has a modular structure. In order to make Muggl able to be invoked, EASy had to be extended so that it can have the functionality to adequately present the results obtained from Muggl especially since the generation of test case needs a lot of computing power. Subsequent to a short test generation time, the result from Muggl will be provided as either a feedback with explanation on Muggl's interruption or a test suite of JUnit with test scenarios being generated. Students should be motivated from viewing the generated tests and allow them to rethink of the solutions.

Model Application

All exercises related to programming are to be submitted through EASy for them to be for any correctness in terms of both syntactic and functional before being corrected by the tutor. Hence, in order to motivate software testing's inclusion; discrete techniques for testing were incorporated into the exercise. Based on a reasonably simple task that still offers some challenges in conjunction to the style of programming and specifications restrictions, the students were given a small program's textual specification that they need to implement and proceed in a test-driven manner. The application of the research's model did not only allow the students to learn how to use back-to-back testing and TDD, but also to make a conclusion or decision based on the interpretation of test cases.

Evaluation

With a specific end goal of getting impressions about the use and advantages of EASy, students were inquired to fill in a survey to impart their experience while solving the exercise through the framework utilization. Since the status of the approach taken in this study would not simply be evaluated but concluded for further improvement, the survey was amended with fields that allow free text to not only present the quantitative outcomes but also qualitative discoveries.

Main Results

After solving the exercise on TDD, the student's answered few questions related to the approach taken by the study to combine EASy and TCG. They were asked to describe whether the approach is informative by comparing the test case being generated with the one being given. In total, 50 students answered the surveys and Figure 1 below shows the results of the helpfulness of the approach as judged by the students through the comparison of the test case being generated with the one being given.

Although its only 2%of the students found that the comparison is very helpful, a significant34%stated that it is actually helpful. However, 36%of the students remained neutral and stated that they at least gained some help from the comparison. These results show that the approach is relatively promising especially since the test scenarios' interpretation is manageable for the students while making their learning curve steeper.

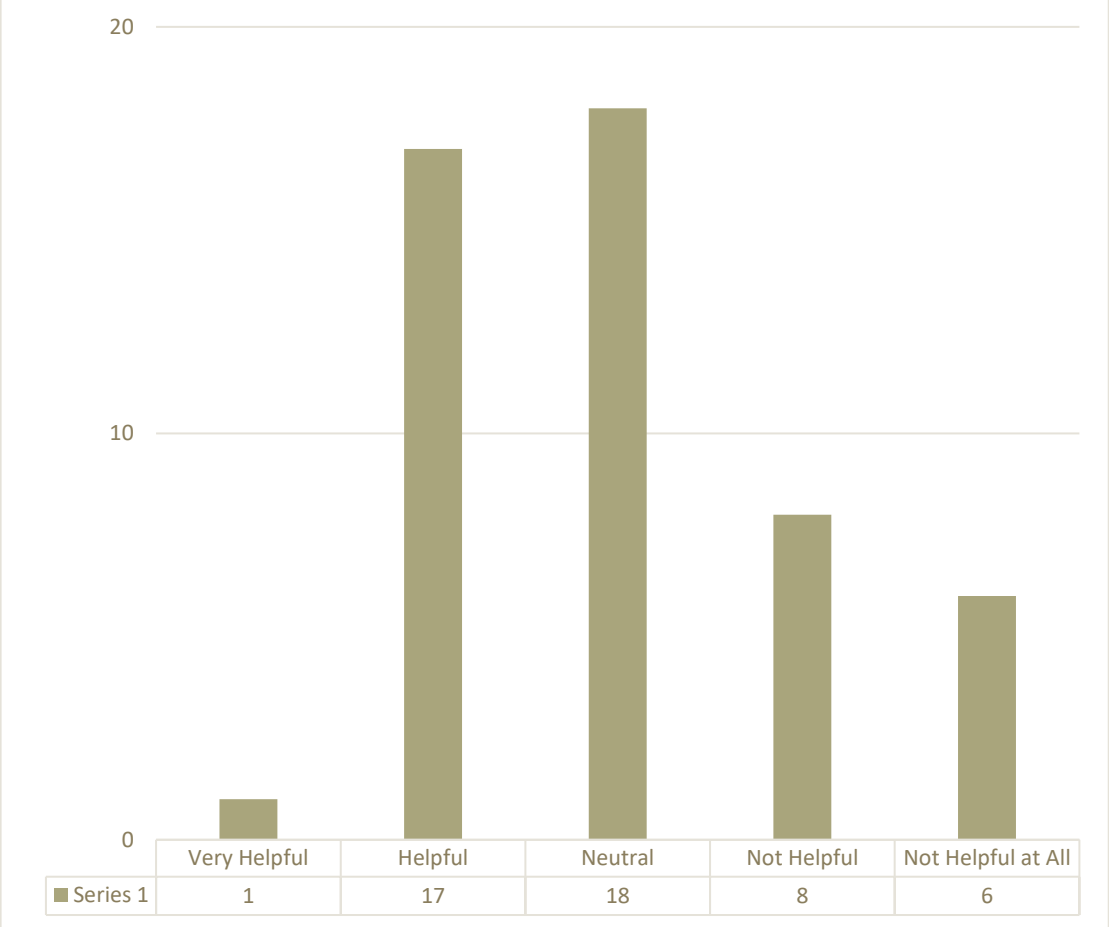

Figure 1: *Result for the helpfulness of comparing the test cases being generated for the students.*

Evaluation - Qualitative

Even though quantitative results are already enough to judge the approach, qualitative feedback will be able to help discussed how EASy is perceived by the students even further and offers some hints on suitable improvements. For reasons of scope for this study, students' comments are not being repeated but rather summarized.The results were so diverse when they provide suggestions on the integration of EASy with TDG. Some students expected the exercise to be less restrictive and inquired for shorter specification to be presented while others found it to be too complex for them and asked for a longer description. However, this assortment can be clarified through the variances in skills of the students.

There are some students that recommended the explanations on how test cases are being generated by Muggl to be provided as it can help them understand the concept further to make improvements for their programs. Apart from this, students also suggested that EASy should be able to automatically compare student test cases with the exemplary test case although this would relieve them of understanding the reason of utilizing back-to-back testing.

Although most of the students did not know about the usage of EASyin applicable courses, their recommendations still supported the positive findings in utilizing EASy for proofs (Gruttmann, 2010). Despite this, there are still a number of students who expressed scepticism while other perceived EASy as being adequate for almost all course that excluded purely writing texts examination. Thus, the usage of this approach in numerous natural sciences courses as well as for some economics courses was suggested by the students.

Conclusion and Recommendations

In this paper, an extension of a tool for e-assessment, EASy as proposed by Usener, Majchrzak and Kuchen (2012), has been presented by adding the functionality for programming exercises that is a part of computer science education. This will be able to replace the paper based processing of handed in solutions by checking them functionally and syntactically through the integration of Muggl that automatically generate test cases. This generation is based on the constraint solving in addition to the symbolic execution. The tool

will be able to help in improving students' solutions by generating test cases while at the same time extending their familiarity with software testing.

A survey on 50 computer science students in Malaysia have been conducted privately to review the feasibility of this study with the result showing that students were more motivated in using this approach and improved in mastering the testing and programming techniques. However, some improvements were recommended for this study although the result is rather promising. First, an improvement needs to make on the integration ofMuggl in EASy. The students will also need to be briefed on how test case generation by Muggl actually work. These test cases also need to be extended so that it will be able to support the inelegant implementation of solutions that are syntactically correct. Besides, EASy will also need to be extended even further to support exercise types available in computer science education as a baby step to ultimately support complex exercise types found in other fields of study.

Reference

Astels, D., 2003. Test Driven Development: A Practical Guide. Upper Saddle River: Prentice-Hall.

Bancroft, P., Roe, P. & Truong, N., 2004. Static analysis of students' Java programs. Darlinghurst, Australian Computer Society, pp. 317-325.

Beck, K., 2002. Test-Driven Development by Example. Boston: Addison-Wesley.

Bjoerner, N. & de Moura, L., 2008. Z3: An Efficient SMT Solver. Lecture Notes in Computer Science, Volume 4963, pp. 337-340.

Bohm, D., Kuchen, H. & Gruttmann, S., 2008. E-assessment of mathematical proofs – chances and challenges for students and tutors. Wuhan, IEEE Computer Society.

Chudowsky, N., Glas, R. & Pellegrino, J., 2001. Knowing What Students Know: The Science and Design of Educational Assessment. Washington D. C.: National Academy Press.

de Halleux, J. & Tillmann, N., 2008. Pex-white box test generation for .NET. Prato, s.n., pp. 134-153.

Graham, D. & Fewster, M., 1999. Software Test Automation. New York: ACM Press.

Gruttmann, S., 2010. Formatives E-Assessment in der Hochschullehre. Munster: Monsenstein und Vannerdat.

Gruttmann, S., Kuchen, H., Majchrzak, T. & Usener, C., 2010. Computer-supported assessment of software verification proofs – towards high quality e-assessments in computer science education. Los Alamitos, IEEE CS, pp. 115-121.

Heywood, J., 2000. Assessment in Higher Education. First ed. London: Jessica Kingsley.

Jenkins, V. & Cook, J., 2010. Getting started with e-assessment, Bath: University of Bath.

Koskela, L., 2007. Test Driven: Practical TDD and Acceptance TDD for Java Developers. Greenwich: Manning Publications.

Kuchen, H. & Fischer, S., 2007. Systematic generation of glass-box test cases for functionallogic programs. New York, s.n., pp. 63-74.

Kuchen, H., Gruttmann, S., Majchrzak, T. & Usener, C., 2010. Computer-supported assessment of software verification proofs – towards high quality e-assessments incomputer science education. Los Alamitos, IEEE CS, pp. 115-121.

Kuchen, H. & Majchrzak, T., 2009. Automated test case generation based on coverage analysis. Tianjin, s.n., pp. 259-266.

Majchrzak, T., 2010. Improving the technical aspects of software testing in enterprises. International Journal of Advanced Computer Science and Applications (IJACSA), 1(4), pp. 1-10.

Meudec, C. & Doyle, J., 2003. IBIS: an interactive bytecode inspection system, using symbolic execution and constraint logic programming. New York, s.n., pp. 55-58.

Rohde, P., Dyckhoff, A. & Stalljohann, P., 2008. An integrated web-based exercise module. Crete, ACTA Press, pp. 244-249.

Roitzsch, E., 2005. Analytische Softwarequalitatssicherung in Theorie und Praxis. Munste: Monsenstein und Vannerdat.

Storzer, M., Krinke, J. & Zeller, A., 2002. Web-basierte Programmierpraktika mit Praktomat. Softwaretechnik-Trends, 22(3).

Usener, C. & Majchrzak, T., 2011. Evaluating the synergies of integrating e-assessment and software testing. Edinburgh, Springer.

Vouk, M., 1990. Back-to-back testing. Information and Software Technology, 32(1), pp. 34-45.

Wannemacher, K., 2007. Computergestutzte Prufungsverfahren–Aspekte der Betriebswirtschaftslehre und Informatik. In: M. Breitner, B. Bruns & F. Lehner, eds. Trends im E-learning. Heidelberg: Physica-Verlag.

Wismuller, R., Quast, A. & Hoffmann, A., 2008. Online-Ubungssystem fur die Programierausbildung zur Einfuhrung in die Informatik. Bonn, GI e.V..

Yellin, F. & Lindholm, T., 1999. The Java Virtual Machine Specification. Second ed. Upper Saddle River: Prentice-Hall.