Software for E-Assessment of Programming Exercises

Uta Priss, Nils Jensen, Oliver Rod
Ostfalia University of Applied Sciences
www.upriss.org.uk, {n.jensen,ol.rod}@ostfalia.de

Abstract: This paper discusses the use of computer-based assessment (CBA) tools for e-assessment of programming exercises. Because many elearning tools already exist but provide different features, it is useful to integrate such tools. In particular, a combination of a virtual learning environment and a CBA tool provides many advantages (and also some challenges) as discussed in the paper. The paper reports on an implementation of such a tool combination, which was tested in an introductory Java class. So far our results are promising.

1 Introduction

Based on the readily availability of testing technologies in software engineering, there is currently an influx of automated assessment software for computer science exercises. It is now commonplace to find free, web-based pastebins which allow source-code to be executed. Together with lecture notes and exercises, on-line interactive programming tutorials can be build in this manner[1] where students can edit and execute code on-line. We use the term "computer-based assessment system" (CBA) to describe any of these small-scale e-assessment tools. An overview of such tools is provided by Rongas et al. [RKK04]. Ideally CBAs are used in combination with virtual learning environments (VLE) which allow to manage and store teaching resources (lecture materials, timetables, student marks, communication tools, etc). It has been our observation that many computer science departments have at least one CBA tool with some VLE functionality that was written by a faculty member. Considering that it is straightforward but time-consuming to write a tool that has basic CBA functionality but it is difficult to write tools that have complex functionality and a high standard of usability, it seems that it would be preferable if there were established means of reusing, combining and sharing existing CBA and VLE tools instead of everybody writing their own tool. Furthermore considering that it is time-consuming to write programming exercises for such tools, it would also be preferable if there were means for reusing, sharing and exchanging exercises[2].

We are currently involved in a project that intends to combine some of the existing CBA and VLE tools via an XML-based exchange format for exercises and a REST interface for tool interoperability. Figure 1 shows the general architecture of our proposed system. The

[1] For example the Tryit editor at www.w3schools.com or the SQL tutorials at sqlzoo.net

[2] Tools such as Lon-Capa (http://www.lon-capa.org) provide repositories of exercises, but not - so far- for programming exercises.

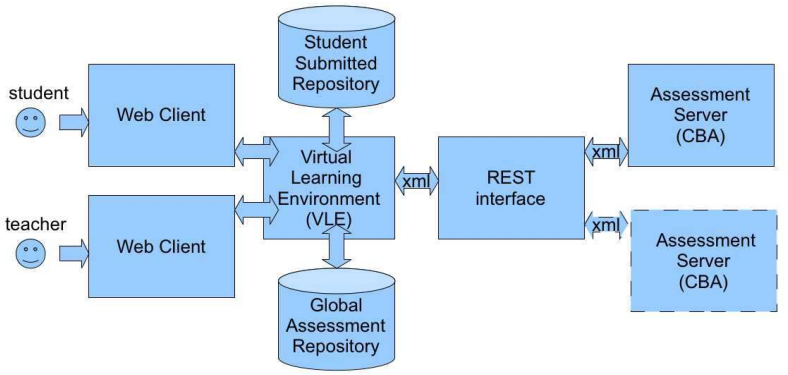

Figure 1: Architecture for combining VLE and CBA tools

purpose of this paper is to discuss the advantages and challenges involved in using such tools for e-assessments and in the deployment of such tools in a typical higher education setting.

2 Advantages and challenges of using e-assessment software

Our main interest is using e-assessment software in STEM subjects (science, technology, engineering and mathematics). In non-STEM subjects, e-assessment is often restricted to certain types of exercises, such as multiple-choice or fill-in-the-blank questions. In STEM subjects, even more complicated exercises often have algorithmically determined solutions and can therefore be assessed using CBA tools. Furthermore, classes in these subjects are often quite large (at least in the first years) which means providing feedback or marking exercises can be tedious. The usefulness and features of CBA tools are also discussed, for example, by [Ze00], [Sp05] and [DW09]. In formative assessments, a major advantage of CBA tools is the provision of detailed and consistent feedback which is available to the students on demand whenever they submit code to the system. Instructors do not need to help students with simple problems and instead can focus on providing feedback which goes beyond the automated feedback. For example, if a student writes a computer program which contains syntax errors, it is much better to have a program identify the errors than for an instructor to read the code. This frees up time for the instructor to help students with more complex problems, for example, for a discussion with the student about the

underlying syntax structures. While it is time-consuming to create exercises for CBA tools, the exercises can usually be reused many times - although if CBA tools are used for exams, it can be more difficult to reuse questions because it would require to keep them secret. Presumably, automatically marked exams have a higher degree of fairness because no human bias is involved in the marking process. Plagiarism is a problem, in particular for homework assignments but plagiarism is a problem in non-STEM or non-elearning homework assignments as well.

In our opinion the main advantages of using CBA tools for e-assessment are didactic ones. Electronically delivered formative assessments are not restricted to time-tabled practicals which facilitates flexible learning and improves the diversity aspects of the learning materials. For example, students with learning difficulties and part-time students can work through the exercises at their own pace and in their own time. Students with visual impairment can display computer-based materials using appropriate fonts. International students can use additional translation software if they are having language difficulties, and so forth.

As Ben-Ari [Be98] observes, computer functions are not open to social negotiation. A CBA tool confronts students with the reality feedback from the computer: either something produces the correct result or it does not. Some students who prefer reflective or social styles of learning may have problems with the high level of precision that is required of exercises that are evaluated by a CBA tool. It is a question, however, whether this is only a problem of teaching computer science with CBA tools or simply a feature of the domain itself. In our opinion it is essential that sufficient human tutoring is still available during practicals that use CBA tools in order to help students who are finding this format difficult. The use of CBA tools supports inquiry- or problem-based learning styles or "learning by doing" because students are able to work independently and self-guided. They do not need to worry about having to ask "stupid questions" because to a certain degree they are able to experiment with the tool and find out for themselves whether something works or not. At the same time, of course, lecturers need to be careful to create a culture that encourages students to ask questions and avoids promoting a "geeky" atmosphere in class which intimidates less technically skilled students. Furthermore at least in software engineering, CBA tools often use software testing technology which is common professional practice in that field. Thus ideally CBA tools provide support similar to what is used by professionals in the field. If CBA tools provide additional feedback that is not available by standard tools, it is possible to reduce the amount of feedback throughout the course so that students do not become dependent on an artificial programming environment.

Thus, we believe that there are strong arguments in favour of using CBA tools as long as there is a commitment by the university to support the tools, to allow lecturers sufficient time for the creation of the exercises and to provide adequate numbers of human tutors in practicals that make use of CBA tools.

3 Testing of a prototype of an CBA/VLE system

We conducted a requirements analysis using interviews with programming language instructors and a review of the relevant literature. We then implemented a system that uses a combination of CBA and VLE technology and tested it with 12-16 students in two sessions with about 5 exercises each of an introductory Java course. The CBA-tool performs checks using Checkstyle, JUnit and blackbox tests which are configured by an instructor. Figure 2 shows a screenshot of the system that was used for the testing. At the moment the connection between the VLE and the CBA tools is implemented not via a REST interface (as in Figure 1) but as an IFrame. But this is only because we are using a prototype. We do not think that this has any impact on the usage of the system. The outer frame in Figure 2 (highlighted in green) shows the VLE component. The inner frame (highlighted in red) shows the CBA tool.

Overall the feedback we received from the students and the lecturers was positive. Some of the feedback highlighted technical problems of the software which will be fairly straightforward to fix and will not be discussed in this paper. Students had difficulties with the precision required by the system (which also confirms Zeller's [Ze00] observations). Students were not used to reading instructions literally and to writing code that only produces the output exactly as requested. Presumably it is a useful learning experience for the students to be able to implement precise requirements because they may be required to do so later in their professional lives if they are working in larger teams.

We paid careful attention to the time required to create the computerised versions of already existing exercises. We found that it only took about 2 hours per exercise and we think that this time can be reduced further. However, the exercises were converted by a staff member who had previously been extensively trained in the use of the CBA tool. Someone who has not used the tool before may need significantly more time. A question is therefore whether it will be possible to improve the usability of the CBA tool so that it reduces the learning curve. A major concern of the lecturers whom we interviewed prior to installing the system was about the long-term availability of the tool. They do not want to invest time into learning a tool unless they will be able to use it for many years to come. We believe that this is one reason why many computer science instructors rather use a simple tool that they themselves wrote than a more complex tool with sophisticated functionality which is not directly under their control.

In addition to the question as to what tools lecturers will eventually want to use, the plagiarism issue remains of importance. Since our tool is at the moment aimed at formative assessment, plagiarism may not be a significant problem. Nevertheless even during our brief testing session we observed that different students submitted the same code. It is tempting for students to submit code that they got from their friends by email. Students (and lecturers) may not consider plagiarism a problem if the exercises are not marked. Students may not be fully aware of the lack of learning benefit that they are obtaining if they are cheating. Again this seems to support our claim that human tutoring is still important during practicals that use CBA tools. A lecturer should discuss the code with the students in order to discourage plagiarism. Zeller [Ze00] advocates having students peer review each other's code. This may also have a plagiarism discouraging effect but we have not

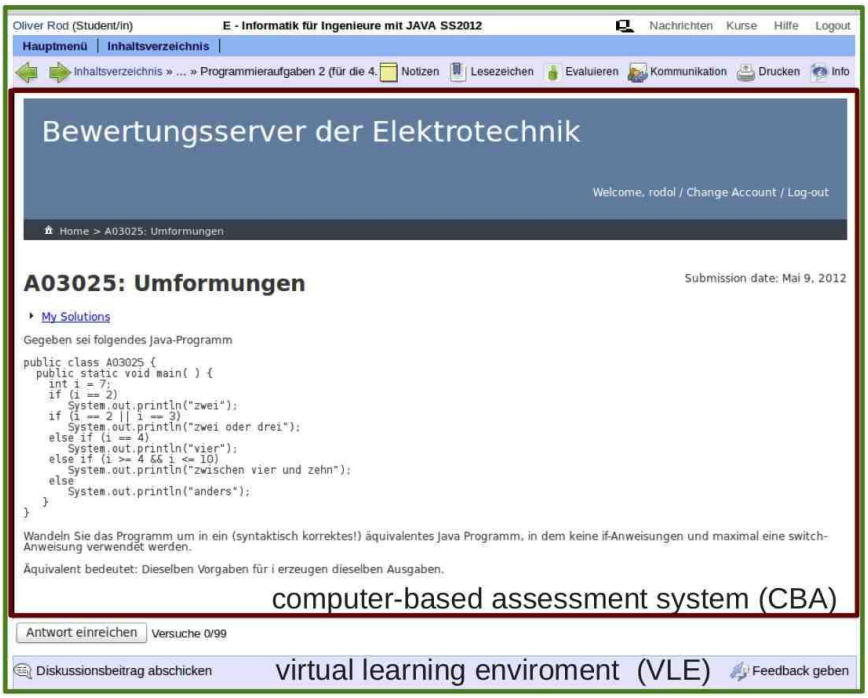

Figure 2: Screenshot: combining VLE and CBA

yet tested this.

We believe that the automated feedback for the students currently generated by our system is only useful for small exercises where the code submitted by the students is fairly short. For longer code submissions, the students would be better advised to check their code carefully with standard development tools before they upload it to our tool. This is because standard development tools are capable of highlighting problems directly in the code whereas our tool lists errors by line numbers. Providing timely and comprehensible feedback is a major challenge that we still need to work on.

We think that there is a danger that automated tests overlook certain errors that would be picked up by a human tutor. This is because the person who implements the exercises needs to predict possible errors which is more difficult than detecting errors in student code during the marking process. This problem could be solved by using the same exercises for a number of years and manually checking of a sample of the exercises in the first instance. From a pedagogical viewpoint it would be interesting to investigate the students' attitudes, though. Does the use of an automated tool affect how carefully the students check their code before they consider it complete? As mentioned before, the amount of feedback provided by a CBA tool might need to be reduced throughout the course so that students are not dependent on tools which are not available outside the teaching environment.

4 Conclusion

We successfully implemented a prototype of a combination of a VLE and a CBA tool. It demonstrates the general feasibility of this kind of tool combination. Nevertheless, many challenging aspects are still left for future research and development. We believe that there are strong didactic arguments for using CBA tools for programming exercises, in particular, the provision of detailed and consistent feedback which is available to the students on demand whenever they submit code to the system. It is not clear at the moment what the cost is of implementing such systems, whether the time-required to create exercises for the system equals the time-saved by reducing the manual feedback and marking that is required. In any case, our analyses so far demonstrate that it is important to provide for sufficient amount of human tutoring during practicals that use CBA tools. In our opinion it is not pedagogically sound to replace human tutoring by CBA tools no matter how sophisticated they are because some students' learning styles require more human input and because of potentially increased plagiarism.

Acknowledgements

This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) under grant number 01PL11066H. The sole responsibility for the content of this paper lies with the authors.

References

[Be98] Ben-Ari, M.: Constructivism in computer science education. SIGCSE '98, Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, ACM, 1998.

[DW09] Demuth, B.; Weigel, D.: Web Based Software Modeling Exercises in Large-Scale Software Engineering Courses. CSEET'09, Software Engineering Education and Training, 2009; p. 138-141.

[RKK04] Rongas, T.; Kaarna, A.; Kalviainen, H.: Classification of Computerized Learning Tools for Introductory Programming Courses: Learning Approach. In Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT '04), IEEE Computer Society, 2004; p. 678-680.

[Sp05] Spacco, J.; Strecker, J.; Hovemeyer, D.; Pugh, W.: Software repository mining with Marmoset: an automated programming project snapshot and testing system. SIGSOFT Softw. Eng. Notes 30, 4, 2005; p. 1-5.

[Ze00] Zeller, A.: Making students read and review code. Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education, ACM, 2000; p. 89-92.