View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UCT Computer Science Research Document Archive

Automatic Marking with Sakai

Hussein Suleman
University of Cape Town
18 University Avenue
Rondebosch
+27 21 650 5106
hussein@cs.uct.ac.za

ABSTRACT

Large student numbers often drive teaching staff to consider greater degrees of automation of assessment activities. In introductory Computer Science classes - where submitted programs need to repeatedly be compiled, executed and tested - automation is an obvious route to investigate. This paper reports on an experimental automation system for assessing programming assignments, and its integration with the open source Sakai learning management system. While the system has been an administrative success, feedback from students has identified numerous areas for improvement at the interface of the student and the automatic marker. Furthermore, the use of automation has highlighted the need for teaching software development methodology from an early stage.

Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: Computer Uses in Education – *computer managed instruction*; K.3.1 [**Computers and Education**]: Computer and Information Science in Education – *computer science education, self-assessment.*

General Terms

Economics, Human Factors, Standardization, Verification.

Keywords

Automation, interoperability, Sakai, assessment.

1.INTRODUCTION

Computer Science assignments often involve the creation of one or more computer programs, and assessment of these assignments can and has been automated to various degrees over the last 40 years [1]. In the simplest of cases, the programs can be compiled automatically but submissions may also be executed, tested and scored without human intervention. Experiences by users of automated systems have been largely positive, with benefits to both students and staff involved with courses [2].

Automation may be desirable for a number of reasons:

- There is no longer a need for human markers – this reduces the assessment and management workload, especially for large classes.
- There is greater consistency since all submissions are marked in terms of exactly the same criteria, which are applied evenly to all submissions.
- All assessment is explainable. Every mark awarded or deducted can be explained fully and consistently.
- Feedback from assessment can be instantaneous. This then raises the possibility of multiple submissions and iterative learning.
- There is an electronic audit trail of all assessment activities – both submission and marking.

A significant new factor motivating automation is the increasing acknowledgment that students should be taught good software engineering practices through the mechanism of early assessment and throughout their training [3]. Automatic marking implies automated testing techniques that are usually comprehensive and mimic unit testing. Students thus indirectly learn to program in a more professional manner, favouring robustness and precision over quick-and-dirty solutions.

However, automation of assessment does have significant disadvantages, including:

- Submissions that do not completely meet the criteria for assessment cannot be assigned partial marks without substantial effort or complexity in system design.
- Assessment criteria must be simple so that measurements can be made automatically. For example, it is difficult to check for meaningful variable names automatically.
- Students are required to meet more stringent requirements than would be the case with traditional human marking.

The aim of any automated marking system is therefore to capitalize on the advantages of automation, while adequately dealing with the disadvantages.

In the case of this project, automated marking was introduced in a first year Computer Science course with an emphasis on Java programming. The prime motivator was increasing use of the Sakai Learning Management System (LMS) [4] to support teaching in this course, and the recent development of APIs to link external applications into Sakai. Thus, the aim of the project was to provide students and staff with all the benefits of an automated marking system along with seamless integration with the Sakai LMS.

Given that the course in question was a first year course, assessment was based largely on problem solving and programming style. Performance and efficiency of solutions was not tested for. The Automatic Marker assessed programs for correctness of output, either from the program as a whole or from individual classes/methods, although its design allowed assessment by other criteria as well. Testing for structural elements of programs was avoided because the varied backgrounds of students meant that some students used advanced solutions at an early stage. Output from programs was compared to predefined output to determine marks for submissions.

This paper reports on the design of this Automatic Marker system and evaluates it in terms of its expected benefits and the experiences of students who used it.

The next section discusses LMSes and interoperability. This is followed by the design of the Automatic Marker, how it integrates with the LMS, and evaluation of the tool. Finally, related work and concluding remarks are presented.

2.LEARNING MANAGEMENT SYSTEMS

Learning management systems (LMSes) are computer systems, usually Web-based, that support teaching activities. These are either distance/self-paced learning or blended learning, where interaction with the LMS complements interaction in traditional classroom and laboratory environments.

WebCT and Blackboard [6] are among the more popular commercial LMSes. Each offers a suite of services to learners and teachers. Each class could set up its own sub-environment with a configurable set of tools and navigation. These tools include resource/file managers, discussion fora, course outline storage, assignment management and mark management.

As LMSes became more popular, users began to move courses between systems and third parties began to produce content for integration into LMSes. To facilitate both of these, the IMS Global Learning Consortium [7] was created to develop standards for educational technology. IMS has since defined standards for learning object metadata, content packaging, question and test interoperability, etc.

In addition to the commercial ventures, numerous Open Source Software LMSes are now available and in use in institutions around the world. Sakai [5] and Moodle [8] are among the most popular.

Moodle [7] presents the now standard LMS tools in a manner that supports constructivist learning. It is easy to install and has minimal resource requirements, while still being able to scale up to support a fairly large university if necessary. Moodle also is popular because of its simple design and extensibility, using the PHP language for all its back-end applications.

Sakai [5] is an alternative to Moodle, which has been produced as a collaborative effort by a team of major universities. The emphasis is on course-based portals rather than individual learning experiences. Sakai also provides all the standard LMS tools, with some degree of configurability of the end-user experience.

Moodle supports many modern interoperability standards, such as RSS feeds for new forum postings and SCORM content rendering. Moodle has been able to export and import all course data in internal XML formats from early incarnations of the software.

Sakai is arguably not as interoperable as Moodle at the time of this writing. However, Sakai provides a mechanism – the linktool [8] – for connecting in externally-hosted tools. This tool has a well-defined API for interoperability with a broad range of tools, with very similar services and usage scenarios to Facebook applications [9]. In addition, Sakai provides a set of Web services to developers of external applications that need to interoperate with Sakai.

While Sakai has some shortcomings (such as inadequate standardisation of data formats), its interoperability interfaces are ideally suited for rapid application development. As such, the Automatic Marker was developed to integrate loosely with Sakai. The sections that follow discuss the design of the Automatic Marker in general, followed by how it integrates with Sakai.

3.SYSTEM DESIGN

3.1Submission and Marking Process

The Automatic Marker provides students with a Web-based interface (see Figure 1) to submit assignments. As each assignment is submitted, the Automatic Marker stores the assignment internally and then initiates the marking process. The Automatic Marker can use different scripts (called graders) for different types of assignments – the following discussion is only about a grader that compares output for correctness.

Each assignment is associated with an XML configuration file that stores information on how to mark the assignment. Figure 2 shows a snippet from such a configuration file.

There is one question XML node per question in the assignment. Each question defines commands to compile and run the program. Then the question defines which files are to be compared for correctness. A series of trials are defined, where each trial specifies files to be created in the filesystem. For each trial, the Automatic Marker first stores the specified files, then executes the “run” command, and finally checks the specified files for equality. Equality is defined as the number of lines in each file being equal and the contents of each pair of corresponding lines being identical after trailing whitespace is deleted.

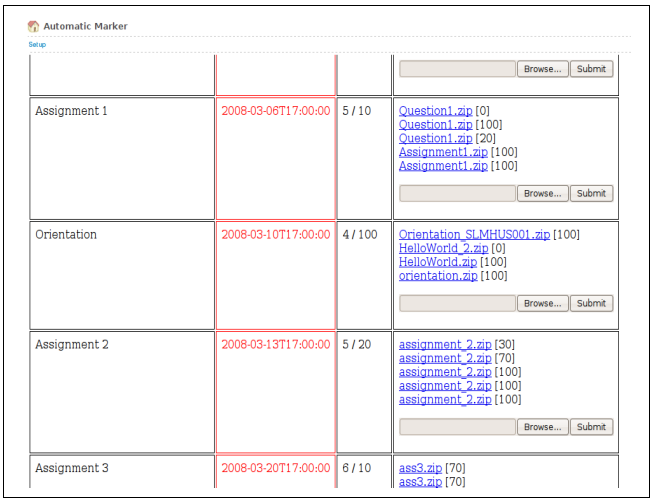

Figure 1: Screen snapshot of main submission interface

```
<markingguide>
   <question>
      <compile>javac Question1.java 2&gt;&amp;1</compile>
      <run>java Question1 &gt; testoutput.out &lt; testinput.in 2&gt;&amp;1</run>
      <check>
         <file1>testoutput.test</file1>
         <file2>testoutput.out</file2>
      </check>
      <marks>20</marks>
      <trial>
         <file name="testinput.in">
5
         </file>
         <file name="testoutput.test">
Enter the height of the triangle:
    *
   **
  ***
 ****
*****
         </file>
      </trial>
      <trial>
         <file name="testinput.in">
10
         </file>
```

Figure 2. XML assignment configuration file snippet

Marks are specified in both the question and trial nodes. The mark in the question node indicates the relative contribution of the question to the total. The mark in the trial node indicates the relative contribution of the trial to the mark of the containing question. Marks are allocated for correct output only. Typically, numerous trials are defined for each question to allow for partially-correct solutions.

Assignments that are submitted late are automatically subject to a late penalty as defined in a system-wide configuration file.

The output of this marking process is streamed to the browser, as shown in Figure 3.

3.2Security measures

Security is a prime concern, especially given that Computer Science students are likely to find and exploit any security holes in a system that runs arbitrary code submitted by students.

Particular features of the BSD/Linux operating systems were exploited to secure the Automatic Marker application. Firstly, when a submission is received, it is decompressed/copied into a randomly-named temporary storage location, thus preventing guessing of directories. Then, any external applications (such as the compiler and JVM) are run via a small mediator application to change the effective user. In a manner similar to Apache's suEXEC [10], the mediator is owned by and always executes as the root user. Since it executes as root, it may switch to another user without performing any additional form of authentication. In this case, the mediator switches to the “nobody” user and then executes the target application e.g., compiler, JVM.

Thus, the user's application executes in a random location on the disk and with minimal privileges. Further, system limits are set to kill the application after a few minutes.

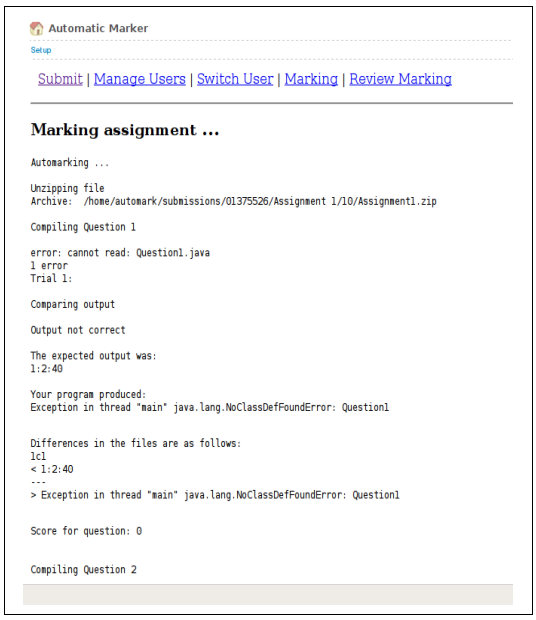

Figure 3: Output from marking of a submission

After each trial of a question in an assignment, the Automatic Marker checks the directory for output and performs output checks. After marking of an assignment is complete, the Automatic Marker deletes the temporary directory.

Figure 4 depicts this sandboxing relative to the other system layers.

3.3Learning Support

The Automatic Marker is designed to support 2 aspects of learning that are typically poorly supported with human-marked assignments.

The first aspect is iterative learning. Every assignment defines how many submissions a student may make. The Automatic Marker uses the highest mark of the set of submissions as the final mark for the assignment. Giving students instant feedback means they have the opportunity to learn from their mistakes and correct them before submitting again. Thus the assignments serve more in the role of formative than summative assessment.

The second aspect is feedback and error reporting. If a solution is not correct, the student is immediately given a listing of the output produced by his or her program as well as the expected output and the differences between the two. This can be scrutinized to help in determining where the errors lie. The Automatic Marker therefore provides the facilities to help students locate their errors as part of the instant feedback they are given.

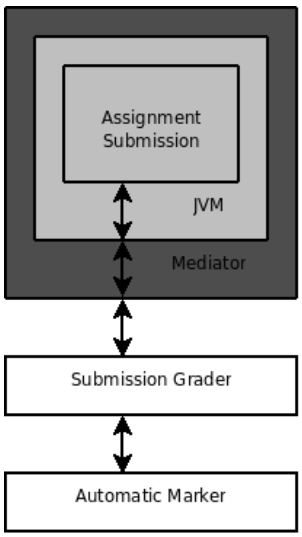

Figure 4: Sandboxing for execution of user code

3.4Management Facilities

Currently, the emphasis of the Automatic Marker has been on correctness and the interface with students, so little effort has gone into administrative tools. However, some tools were necessary for teaching assistants and tutors (support staff).

A useful feature of Moodle, which has not appeared in many other LMSes, is the ability to assume the identity of a student – see what the student sees and be able to act as the student. This functionality was built into the Automatic Marker to help when resolving queries from students.

In addressing some queries from students, it is necessary to also handle special requests and extensions. Invariably, some students begin the course late and need extensions on the first assignments. Then, there are students who fall ill during the semester and need to negotiate multiple deadlines that are different from the rest of the class. There are also those students, especially early in the semester, who exhaust their submission quotas before realising the limits that are in place. In all these cases, and others, the support staff need to make adjustments on a per-student basis. Whenever one of the support staff logs in, they are therefore presented with the option to either extend individual assignments or increase the number of handins allowed for individual assignments. Figure 5 illustrates this interface.

3.5Manual Marking

While most assignments can be marked automatically, correctness was not the only criteria used for assessment. To test for programming style, assignments were marked manually, but using as much automation as possible.

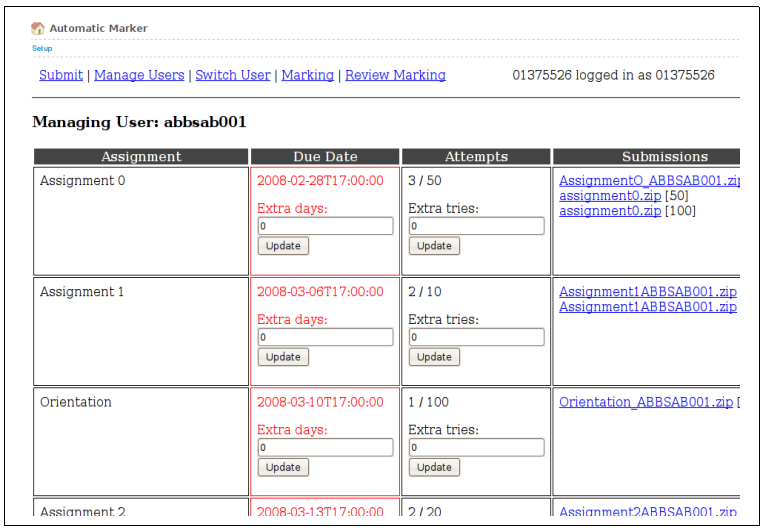

Figure 5: Management of student submissions and extensions

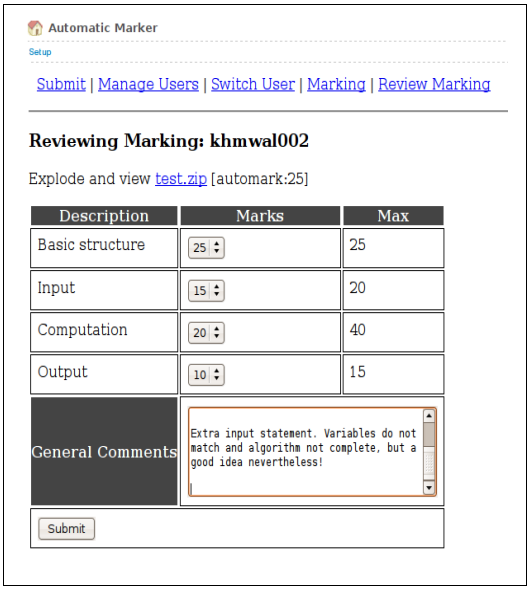

Figure 6: Screen snapshot of manual marking template

Support staff can choose the "Marking" option, whereupon they are presented with a list of assignments that may be marked. On selecting one of these, the Automatic Marker randomly chooses an unmarked assignment and presents it to the marker within a marking template for that assignment (see Figure 6). Files related to the submission are available to be opened in a separate window, which uses AJAX techniques to dynamically update the display of a file without reloading the entire window. Thus, the marker can scan through a submission with a minimum amount of clicking and scrolling.

4.INTEROPERABILITYWITH SAKAI

Interoperability with Sakai is a key issue for the Automatic Marker and occurs at various levels of human and machine interfaces.

In the user interface, a menu item was inserted into the main user menu for the Automatic Marker. This is a special linktool menu item that invokes the linktool Sakai Tool when a user selects the option. Unlike a regular URL linked into the menu, linktool passes various parameters to the application to link it to the course environment, logged-in user and other forms of context.

The Automatic Marker is then loaded into an iframe in the main content area, an HTML page that exists within another HTML

page. This is not ideal but it is how Sakai currently integrates all of its tools.

As the application loads, the parameters passed to it by the linktool indicate the context (course), role and user ID of the user, as well as appropriate encrypted checksums. The application then uses a Web service to re-authenticate itself with the Sakai server using these encrypted details. This prevents third-party spoofing of credentials. Once authenticated, the application proceeds with its regular operation until it needs to communicate with Sakai.

The following Web services are used as an interface between the Automatic Marker and Sakai at various points in its operation:

- Get list of assignments.
- Set mark and feedback comment.
- Get list of students.

The list of accessible assignments, and all dates associated with each assignment, is obtained each time from Sakai. This is done because the list differs from student to student. Additional configuration information about each assignment is stored by the Automatic Marker and these 2 pieces of information are reconciled before presenting the user with a list of assignments.

An additional encrypted token is used when the Automatic Marker needs system-wide information (such as the list of students). This removes the need to store a plaintext password to authenticate as a different user.

5.EVALUATION

The Automatic Marker was used for an introductory Java one-semester (12 week) first year course with approximately 300 students and 20 support staff (tutors and teaching assistants). All 9 assignments done by students, as well as the orientation exercise, practice exercises and practical test were channeled through the automatic marker. Additional manual marking of 2 assignments for stylistic criteria was conducted also using facilities provided within the Automatic Marker.

Some staff perspectives emerged and student opinions were gathered at the end of the course.

5.1Staff Analysis

As expected, the following advantages of automatic marking were apparent:

- There were fewer queries from students about marking or mark transcription in official marksheets.
- There was no need to track down tutors and check on timing and quality of marking.
- Marking was consistent in all assignments.
- Tutors were happier – they did less marking overall and spent more effort on the manual marking.
- Students learnt to build robust code.

A number of issues were highlighted by and noted from interaction with both students and staff during the semester. These include:

- Students and staff found the errors from the Automatic Marker difficult to understand. This was to be expected from first-year students but staff had great difficulty in helping to track down simple errors such as those related to formatting. This highlighted the fact that most support staff were themselves students in the early stages of their studies.
- Many students believed that they could solve the problems but that "satisfying the Automatic Marker" was different from solving the problem – this was raised repeatedly on the discussion forum. The issue of what constitutes a correct solution is not clear – some students were frustrated by spacing of output while others felt that precision of floating point output should not be checked for.
- Many students blamed the Automatic Marker as a first resort and complained to the department when their programs did not work. The Automatic Marker was never the problem although each such complaint was investigated. This raises the issue that students trust human markers but do not trust a machine to assess their work.

After all assignments were complete for the semester, student performance was compared with the previous year. In 2007 the average assignment mark was 79.43% and in 2008 it was 82.91% - thus it can be argued that student performance is at least comparable with the Automatic Marker.

5.2Student Feedback

At the end of the semester, students were asked to complete a survey as part of their evaluation of the course. Questions specific to the Automatic Marker were posed in a manner similar to what would have been asked of human markers, for a degree of comparability. The results from this survey are presented in Table 1, where N="No answer", 1="Poor", 2="Below average", 3="Average", 4="Good" and 5="Excellent".

Table 1: Results from student survey

Question	**N**	**1**	**2**	**3**	**4**	**5**
How well did this support the notion of iterative learning?	1	3	6	22	23	16
How fair and even was the marking?	0	8	15	18	22	8
Rate the speed of obtaining a response from the automatic marker.	1	1	2	3	13	51
Rate the feedback provided.	2	7	10	17	21	14
Rate your overall experience.	0	6	7	23	25	10

The majority of students believed that the system supported iterative learning. A substantial number of students felt the support for iterative learning was poor or below average. This may mean students did not understand the concept of iterative learning, as presented in the question. All students were allowed at least 10 submissions for all assignments, and in some cases more, and almost without exception students used this facility often.

While most students felt that the marking was fair and even, 22 students rated the system as poor or below average. This implies that students feel they are being treated differently by the Automatic Marker, which does not make sense. From personal interaction with students, it is likely they simply do not understand the meaning of "fair".

The vast majority felt that the system response time was excellent. A small number did not agree. This could be because students interpreted this as the speed of a response from the Web server in general – this is dictated by various factors including the speed of computers (or lack thereof) in the student laboratories. In terms of the Automatic Marker itself, marking of submissions rarely took more than 10 seconds, even close to the submission deadlines. The response to this question stresses the importance of the performance of multiple components in a seamless solution – if end-users are not able to detect the boundaries among components, they cannot attribute failures correctly.

17 of the respondents felt that the feedback was average while another 17 felt it was below average or poor. There are many reasons that contribute to this. Firstly, the output produced by the Automatic Marker is terse – it was expected that students would correctly interpret numbers such as character positions of errors but few students ever looked at those. Then, during the marking phase students were given only the output but not the input to prevent solutions that were not generalisable – unfortunately this often led to confusion over the cause of errors. Feedback from the Automatic Marker has been confirmed by students as an area where much improvement is needed.

Finally, in the overall ratings the majority of students rated their experience with the Automatic Marker as either average or good. Comparatively fewer students indicated the experience as being poor or below average, which seems reasonable in light of responses to prior questions.

At the end of the survey students were asked for general comments on the course and a number of students provided input on the Automatic Marker. Some comments confirmed the mismatch between what students thought of as correct and what the Automatic Marker used as its internal notion of correct:

- "The automatic marker is always faulty and even though my program works at home, the A.M doesn't compile and run properly on [Sakai]."
- "Sometimes it is right to you but wrong to the automark."

Some comments from students point to future improvements that can be made so assessment is not purely based on output:

- "The auto-marker is really bad because [it] prevents [you] from learning from your mistakes. It's over-sensitive."
- "The idea of the automatic marker is good and would be excellent if there can be a way of improving its capabilities such as the general marking techniques, that is not to only be based on program output but the general code structure."

A few students wanted more exercises for those who are having difficulty and those who are ahead of the class. Both of these are immediately possible with the automatic marker.

- "I wish there were more practice assignments instead of just assignments."
- "Something [that] would be nice would be some optional assignments, as a challenge to those who completed their assignments early."

5.3Feedback from Tutors

Tutors were asked for feedback as well but there were only 5 responses. For the most part, the responses from tutors confirmed the comments made by students. In their comments, tutors highlighted the issue of better feedback for students and the increased possibility of plagiarism in the absence of human marking. Plagiarism detection was not integrated for logistic reasons but this is planned for the future.

5.4Sakai Integration

As indicated in the student feedback, students were often not able to separate the Automatic Marker from Sakai – thus, a reasonable level of integration was achieved.

When Sakai was offline the portal was not accessible but when the Automatic Marker was offline independently of Sakai students would be able to use Sakai but not the Automatic Marker. Thus, while distributed applications have advantages, they can result in less cohesive systems.

Corruption of marks occurred only once (and was fixed by re-sending marks to Sakai) – because of a local Sakai software upgrade rather than an inherent problem with the interoperability interfaces. Thus, the interoperability interfaces were reasonably stable and robust.

There were no known security breaches at the interface between the 2 systems.

5.5Summary of Results

In general, most students were satisfied with the Automatic Marker but some students were unable to relate to this change in assessment methodology. The feedback from students highlighted many areas of possible improvement.

From an administrative perspective, the Automatic Marker has been very successful.

From an interoperability perspective, the Automatic Marker successfully integrated with Sakai, providing a nearly seamless experience to end users.

6.RELATED WORK

Numerous attempts have been made to construct similar automatic marking systems, both for computer programs and other forms of assignments.

Marking of free-form text by matching phrases has been suggested as an effective technique for courses with text-intensive tests and examinations [11]. Thomas [12] and Waugh [13] extended this idea to mark ER diagrams automatically by extracting the equivalent of key phrases from these diagrams.

Early efforts at automation demonstrated the feasibility of the general approach for various programming languages, while investigating different techniques for assessment. The early Ceilidh system marked student assignments in languages that included Standard ML [14], in a manner similar to this work, but without the high level and seamless integration with a modern LMS. Foubister [14] confirmed that there was no change in student performance with automated testing, as is borne out by this study. Jackson and Usher [15] used a wide range of assessment criteria applied to offline marking of Ada programs in their ASSYST system. Saikkonen et al [16] used a fully automated assessment tool for Scheme submissions which avoided the output matching problem by testing the return values instead.

More recent work has been focused on the specifics of Java assessment and interactive learning. Truong et al [17] attempted to semi-automatically assess Java programs using static analysis, that is without compiling and executing the programs. Tremblay et al [18] assessed Java programs using a command-line tool available to students using a Unix-based system – and mentioned the possibility of a future Web-based application. Blumenstein et al [19] developed the generic GAME system that is a framework for automated grading of assignments in programming languages that include Java.

Many of these systems were similar, but with varying goals that were relevant to local contexts and current technology. None emphasized the integration with a modern LMS as was done in this project.

7.CONCLUSIONS

This project has discussed a proof-of-concept application to demonstrate the feasibility of integrating advanced computer-assisted instruction/assessment tools into a modern LMS. Feedback indicates that such an application can meet its pedagogical objectives while providing a seamless user experience.

The evaluation suggests that some students failed to fully appreciate the different assessment model of the Automatic Marker, with its stronger focus on software testing and robustness. Thus, human factors play a greater role in migrating a user community to new forms of educational technology, and users will need more up-front information in future.

This work also provides further evidence in support of the effectiveness of component-based development based on service oriented architectures. If more educational technology was made available as services, learning environments could be transformed into a rich blend of configurable tools.

8.FUTURE WORK

An administrative configuration interface will be added to the Automatic Marker's user interface to eliminate the need for editing of XML configuration files.

Further generalisation of the marking criteria will enable less strict marking in future. Possibilities under consideration are regular expressions and canonicalisation of text by removing whitespace and punctuation.

More feedback will be provided to students through an interface that is more informative. Students will also be provided with input where it is necessary to help understand how and where their programs have failed.

A plagiarism detector may be integrated into the system in future.

Finally, from a pedagogic perspective, some work is needed to investigate if there are significant learning differences as a result of automatic marking, as opposed to human marking. Specifically, a potential advantage of automated marking is rigorous and iterative feedback and the effect of this on the learning process could be measured.

9.ACKNOWLEDGMENTS

Thanks go to the students in CSC1015F who are regularly subject to experimental teaching and assessment techniques.

10.REFERENCES

[1] Douce, C., Livingstone, D., and Orwell, J. 2005. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.* 5, 3 (Sep. 2005), 4. DOI= http://doi.acm.org/10.1145/1163405.1163409

[2] Malmi, L., Korhonen, A., and Saikkonen, R. 2002. Experiences in automatic assessment on mass courses and issues for designing virtual courses. *SIGCSE Bull.* 34, 3 (Sep. 2002), 55-59. DOI= http://doi.acm.org/10.1145/637610.544433

[3] Edwards, S. H. 2003. Rethinking computer science education from a test-first perspective. In *Companion of the 18th Annual ACM SIGPLAN C*onference on Object-Oriented Programming, Systems, Languages, and Applications (Anaheim, CA, USA, October 26 - 30, 2003). OOPSLA '03. ACM, New York, NY, 148-155. DOI= http://doi.acm.org/10.1145/949344.949390

[4] Sakai Project. 2008. http://www.sakaiproject.org/

[5] Blackboard. 2008. http://www.blackboard.com/

[6] IMS Global Learning Consortium. 2008. http://www.imsglobal.org/

[7] Dougiamas, M. and Taylor, P.C. 2003. Moodle: Using Learning Communities to Create an Open Source Course Management System. In Proceedings of the EDMEDIA 2003 Conference, Honolulu, Hawaii.

[8] Sakai. 2008. Linktool. https://source.sakaiproject.org/svn/linktool/trunk/linktool.txt

[9] Facebook. 2008. Developers Documentation. http://developers.facebook.com/documentation.php

[10] Apache HTTP Server Documentation Project. 2008. Apache suEXEC Support. http://httpd.apache.org/docs/1.3/suexec.html

[11] Parkinson, J. 2005. Essays marked by computer program. BBC News. http://news.bbc.co.uk/2/hi/uk_news/education/4425423.stm

[12] Thomas, P., Waugh, K., and Smith, N. 2005. Experiments in the automatic marking of ER-diagrams. *SIGCSE Bull.* 37, 3 (Sep. 2005), 158-162. DOI= http://doi.acm.org/10.1145/1151954.1067490

[13] Waugh, K., Thomas, P. and Smith, N. 2007. In Proceedings of the 24th British National Conference on Databases. BNCOD'07. IEEE Computer Society.

[14] Foubister, F.P., Michaelson, G. J., and Tomes, N. 1997. Automatic assessment of elementary Standard ML programs using Ceilidh. J. Computer Assisted Learning 13 (2). doi:10.1046/j.1365-2729.1997.00012.x

[15] Jackson, D. and Usher, M. 1997. Grading student programs using ASSYST. In *Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education* (San Jose, California, United States, February 27 - March 01, 1997). J. E. Miller, Ed. SIGCSE '97. ACM, New York, NY, 335-339. DOI= http://doi.acm.org/10.1145/268084.268210

[16] Truong, N., Roe, P., and Bancroft, P. 2004. Static analysis of students' Java programs. In *Proceedings of the Sixth Conference on Australasian Computing Education - Volume 30* (Dunedin, New Zealand). R. Lister and A. Young, Eds. ACM International Conference Proceeding Series, vol. 57. Australian Computer Society, Darlinghurst, Australia, 317-325.

[17] Saikkonen, R., Malmi, L., and Korhonen, A. 2001. Fully automatic assessment of programming exercises. *SIGCSE Bull.* 33, 3 (Sep. 2001), 133-136. DOI= http://doi.acm.org/10.1145/507758.377666

[18] Tremblay, G., Guérin, F., Pons, A., and Salah, A. 2008. Oto, a generic and extensible tool for marking programming assignments. Software: Practice and Experience, 28 (3), John Wiley & Sons. doi:10.1002/spe.839

[19] Blumenstein, M. M., Green, S., Nguyen, A. T., and Muthukkumarasamy, V. 2004. GAME: A Generic Automated Marking Environment for Programming Assessment. In Proceedings ITCC 2004 International Conference on Information Technology: Coding and Computing. http://hdl.handle.net/10072/2110