Effective Electronic Marking for On-line Assessment

Mike Joy
Department of Computer Science
University of Warwick
Coventry, CV4 7AL
+44 1203 523368
M.S.Joy@dcs.warwick.ac.uk

Michael Luck
Department of Computer Science
University of Warwick
Coventry, CV4 7AL
+44 1203 523364
Michael.Luck@dcs.warwick.ac.uk

1. ABSTRACT

In response to the demands of increasing student numbers, the BOSS system for submission and assessment has been constructed to enable student programming assignments to be submitted and tested on-line. More recent developments of this system have been concerned with the addition of *electronic marking* facilities that incorporate both automated marking, resulting from the automated testing, and manual marking in a secure environment. This paper briefly reviews the system and describes in detail the *electronic marksheets*, their functionality, and their user-interface.

2. INTRODUCTION

By developing techniques for automating the submission, compilation and testing of student programs, we can support the process of marking, and enable marking tasks to be divided among several individuals while maintaining rigour and consistency. Not only can this stem the tide of an increasing workload, it can also enable other administrative tasks to be automated as part of a coherent approach to full course management.

In response to the demands of ever increasing student numbers and the need to maintain and improve quality of teaching, we have developed just such a system for on-line submission and assessment of student programs. This system, known as BOSS, is described in more detail elsewhere with regard to the philosophy behind it in addressing software development concerns [7], how it fits in with a general approach to the teaching of Computer Science [10], its general functionality [6] and its user-friendliness [8]. In this paper, we review the core system and describe more recent developments relating to the electronic marking aspects of the system. We begin by considering the motivation behind the system.

ITiCSE '98 Dublin, Ireland
© 1998 ACM 1-58113-000-7/98/0008... $5.00

Until recently, it was commonplace for submitted programs to be printed out on paper, and for the printed copies to be read and assessed by an examiner. This approach suffers from several distinct problems. First, there will inevitably be inaccuracies in the marking process, as even the most proficient programmer will find it difficult to check the functionality of a program by hand. Second, the amount of time needed to read and fully understand a program in this way is substantial. Third, paper submission of student programs and test output lead to the possibility of forged output listings. Finally, it is only possible to require students to demonstrate that a program has been tested on a relatively small set of test data, and it therefore cannot be tested on unanticipated data. Consequently, students will often tailor their programs to the test data rather than construct more general programs. Automation of the testing and marking process would, in principle, solve these problems. Partial automation is now not uncommon; students are often asked to provide the examiner with a copy of their program, which the examiner can then run and test. However, this solution also suffers from some technical difficulties.

Extreme care must be taken to ensure privacy of submitted programs. For example, if a networked computer system is used, and a file containing the program is made readable to the examiner, it is undesirable for it to be read by other users of the system. There are also many ways in which the process can be frustrated, such as students failing to make their programs readable by the examiner. Moreover, a program submitted in this way could be dangerous if it contained a *Trojan Horse* which, when run by the examiner, might damage the security or integrity of the system. On stand-alone computers, such as PCs, viruses are an unpleasant fact of life. Finally, it is worth noting that the examiner is still required to spend time locating the program, compiling it and running it on suitable data before examining results.

Although much of the testing and marking process has the potential to be automated, this is not as simple as it might at first appear. We can test whether a program meets its specification (at least, in part – the problem is in general formally undecidable). That is, we can run a program against various sets of data, and then check whether its output matches that required by the specification. Furthermore, we can measure the source code using various metrics and arrive at concrete indicators of programming style (modularisation, commenting, consistency of indentation, and so on). The former tests make up the major part of the testing process, as discussed above. The latter, although valuable, serve to refine the final mark arrived at – and in any event, it can be argued that measuring programming *style* is an inexact science [4]. The relative importance of style against correctness when designing a marking scheme for an assignment is a matter best decided for each individual course – for some programming courses, it may be desirable to emphasise style and readability more strongly.

3. THE BOSS ONLINE SUMISSION SYSTEM

The BOSS system for automatic submission of assignments [6], built in an effort to address the problems described above, comprises a collection of programs, each of which performs a different task contributing to the overarching goal of effectively managing the process of submitting programming assignments on-line. BOSS is designed primarily for courses with large numbers of students, assessed by means of programming exercises. The individual component programs of BOSS are designed to be used by two kinds of individual. First, some programs must be used by students so that they can gain feedback and submit their programs. Second, lecturers and any course tutors involved in assisting the lecturer must be able to gain access to the submitted programs in order to test and mark the student submissions. The programs offer the following functionality.

Students may submit programs on-line by means of a user-friendly program that conducts a dialogue with the student to ensure that the correct submission is made. The program is stored and simple checks are carried out (to ensure the correct programming language is used and to verify the student's identity, for example), so that the lecturer can subsequently test and mark it.

In response to a submission, an acknowledgement of receipt is sent to the student by email, which also contains a code, a message digest identifying the contents of their submission. A file only very slightly different (even by just one character) will generate a different code. Thus in the case of a dispute, the code can be used to authenticate that file. An audit file is also maintained with copies of all such receipts issued.

All submissions for a specified item of coursework can be run against a number of sets of data. The output from the students' programs are compared with the expected output for each set of data. Time and space limits are placed on the execution of a program so as to prevent a looping program from continuing unchecked, and other steps are taken to minimise the potential for a program to damage the system.

Submissions and the results of the testing process can be inspected on-line by authorised staff. Anonymity is preserved by storing data by University ID number.

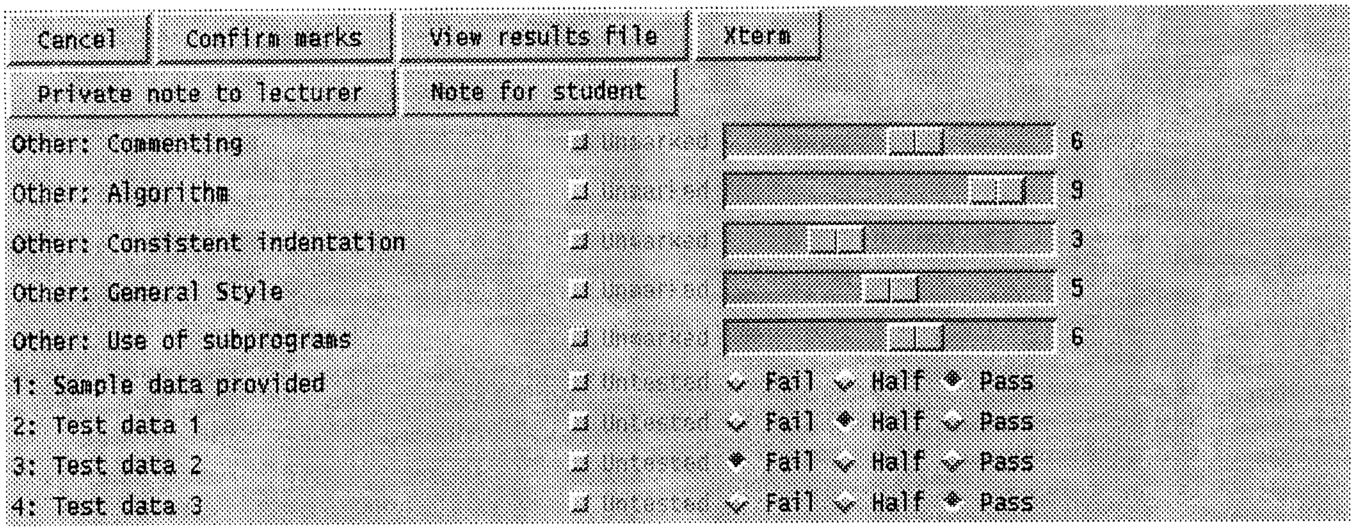
Students can test their programs by running them against one data set on which they will eventually be tested, and under precisely the same conditions. Thus a student can check that their program will run correctly under the final testing environment. This ensures that the program will work as the student expects when being tested and marked. In addition, it provides students with confidence that their submitted work does pass some minimal requirement.

Final marks are stored in a SQL database and correlated with information from the University database (names and courses versus ID numbers and course registration, for example) to produce final marksheets for examination secretaries.

The BOSS system is a tool to allow students to submit assignments, and for those programs to be tested automatically. It is not an automated marking system. It is the responsibility of the individual lecturer to provide a marking scheme which takes account of the results produced by BOSS, together with all other factors which may be regarded as important (such as program style, commenting, etc. [9])

Action that should be taken when a student's program does not pass one or more of the tests on which it is run is, again, the lecturer's responsibility. It may be desirable to award marks for a partially working program, but BOSS does not address that problem. We do not aim to remove the instructor from the teaching loop, but instead simply to assist the instructor in achieving a quicker, more accurate and more consistent assessment of programming assignments. This is important, and should be made clear to students to avoid any misconceptions about the extent and scope of the automated system. It is our experience that students gain confidence from the system, but they are also uneasy about the possibility of its unlimited significance in the assessment process.

This software is available with both a text-based interface and a graphical front-end, allowing it to be run on several hardware platforms. It is also configurable to particular



```
Other: Commenting              Unmarked                6
Other: Algorithm               Unmarked                9
Other: Consistent indentation  Unmarked                3
Other: General Style           Unmarked                5
Other: Use of subprograms      Unmarked                6
1: Sample data provided        Untested  Fail  Half  [✓ Pass]
2: Test data 1                 Untested  Fail  [✓ Half]  Pass
3: Test data 2                 Untested  [✓ Fail]  Half  Pass
4: Test data 3                 Untested  Fail  Half  [✓ Pass]
```

Figure 1: Electronic Marksheet

course requirements, and has optional capabilities for incorporating automated marking into the assessment process.

4. ELECTRONIC MARKSHEETS

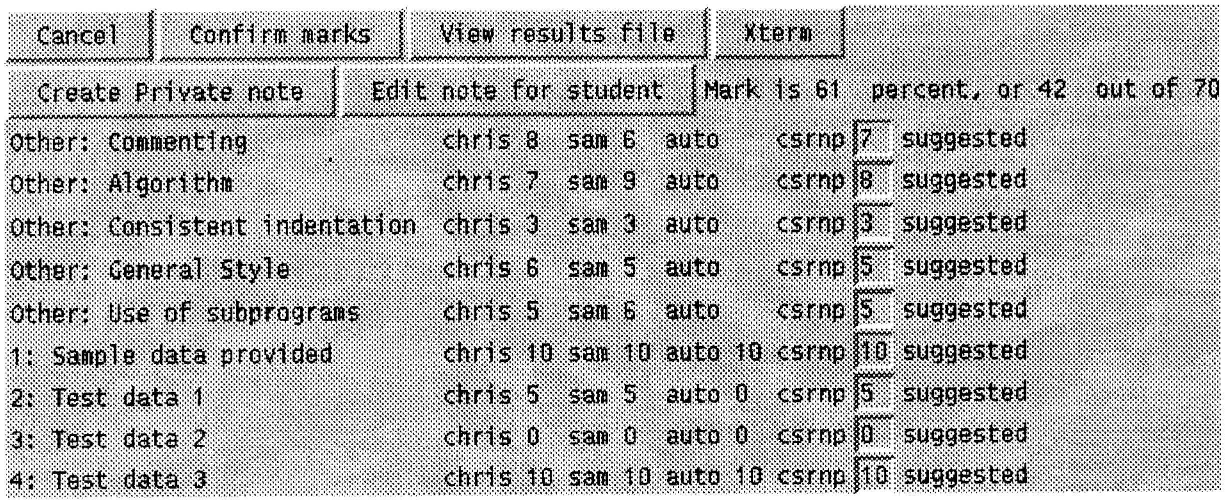
In more recent developments, the process of marking has been even more closely integrated into the system through the use of graphical *electronic marksheets*, constructed using the Tcl/Tk toolkit [11]. Lecturers simply need to specify the categories for which marks are awarded, and the weight attached to these categories, and a graphical marksheet is constructed. The marksheet integrates marks resulting from running and testing the program with those relating to other aspects of the program (such as style, for example), and the interface includes buttons that provide a range of functionality as described below. Figure 1 shows a screen dump of a completed electronic marksheet with the buttons grouped at the top and the categories of marks below.

The marks resulting from the automatic tests that are performed, by which a student's program is run on several sets of data and the output compared with expected output, are incorporated into the marksheet directly. If the output is correct and the program passes, full marks for that category are declared on the marksheet. If the program fails, then no marks are awarded, but the tutor or lecturer may subsequently adjust the automatically assigned marks to give either full, half or zero marks. This is shown by the bottom four mark categories on the marksheet of Figure 1. The buttons provide the means by which the marker can inspect the results of testing to discover how marks were awarded automatically, and also allow the marker to run the tests manually in problem cases, if necessary.

The remaining categories of marks are awarded by the tutor or lecturer interacting with the marksheet and moving the slider along on a scale of zero to ten. Only when this mark is combined with the weight (that is not shown to the marker), is the final mark calculated. This allows independent assessment of various aspects of the program without the marker being biased by the number of marks to be awarded. Before a category is assigned a mark, the *unmarked* box is highlighted so that it is obvious which parts of the marksheet need addressing. At present, these marks are awarded manually, but it is possible for various automated measurements of source code to be made to arrive at concrete indicators of programming style (modularisation, commenting, consistency of indentation, and so on) [3, 4, 13]. The incorporation of such metrics are intended to be the subject of the next step in the system's development.

Several further checks are built into the systems to ensure consistency and preserve integrity by anonymous marking. All assignments are identified using student identification numbers alone so that marking is entirely anonymous. In the current version of the system, an extra utility has been developed by which these anonymous marks are linked into the central university database so that reintegration can take place to produce a list of final marks by name, once marking is complete. The system provides for double marking on the same marksheets, and for a moderator to view both sets of marks and the original submission in arriving at a final judgement, with suggested final marks being offered by the system as the mean of the two sets.

Finally, an extensive commenting facility is also included, and is invoked by the buttons at the top of the marksheet. This allows each marker to comment on the submission either for the moderator who fixes on the final mark, or as feedback for the student. At the point of finalising marks,



```
Mark is 61  percent, or 42  out of 70
Other: Commenting              chris 8  sam 6  auto     csrnp 7  suggested
Other: Algorithm               chris 7  sam 9  auto     csrnp 8  suggested
Other: Consistent indentation  chris 3  sam 3  auto     csrnp 3  suggested
Other: General Style           chris 6  sam 5  auto     csrnp 5  suggested
Other: Use of subprograms      chris 5  sam 6  auto     csrnp 5  suggested
1: Sample data provided        chris 10 sam 10 auto 10 csrnp 10 suggested
2: Test data 1                 chris 5  sam 5  auto 0  csrnp 5  suggested
3: Test data 2                 chris 0  sam 0  auto 0  csrnp 0  suggested
4: Test data 3                 chris 10 sam 10 auto 10 csrnp 10 suggested
```

Figure 2: Electronic Moderation Sheet

the moderator may edit the student feedback and the system will email these comments to the student directly, while ensuring that private comments by the markers are kept confidential. All of the comments and each set of marks are retained so that if students query their marks, it is a simple matter to re-examine and justify them.

The moderation window is illustrated by Figure 2, which shows the relevant buttons at the top as before, but now displays the marks of the individual markers (here named by usercodes `sam` and `chris`, as well as the automatically assigned marks (indicated by `auto`) On the right-hand side of the window, the system offers a suggested average as the final mark, which the moderator (with usercode `csrnp`) can adjust if appropriate. The final marks are shown in the top right corner.

In summary, the electronic marksheet not only enhances the usability of the system, but it also increases its functionality, and contributes to a marking regime of high integrity and consistency. These are clearly vital qualities that are to be demanded of any course, but the system makes this visible externally.

5. RELATED WORK

These include packages from Isaacson and Scott at the University of Northern Colorado [5], Reek at RIT [12], and the `Submit` system developed by Cameron Shelley at Waterloo. All of these packages interested us, but were inappropriate due to reasons of security. We were especially concerned about opportunistic attempts by students to exploit loopholes in the systems, given that the learning environment at Warwick is intended to encourage and stimulate experimentation.

One type of package in particular deserves some discussion because of their size and distinct approach. Systems such as Ceilidh [1, 2] are packages which provide a full programming environment, handling not only submission and testing of assignments, but also providing tutorial material and a user-friendly interface to the machine. Students are then artificially isolated from the underlying operating system. There are strong arguments in favour of such an approach, which we do not discuss here, and it is superficially very attractive to install and use such a system. However, the more complex a software package becomes, the more complex and time-consuming it becomes to *maintain* and *update*. Furthermore, if that system does not match exactly the requirements of the course on which it is intended to use it, it may not be possible to *customise* its functionality.

More importantly, our system targets the particular areas of concern to us in providing us with a *secure* system that can easily interface with the University databases so that the electronic marksheets are integrated into the broader process of assessment administration. Thus, following marking and moderation, marks are finalised and entered into the University database. When all marks have been entered, a mark list can then be produced ready to be sent to examination secretaries.

6. CONCLUSIONS

Over the course of the four years the systems has been used on our courses, we have enhanced the software in several ways. The original command-line interface to the system has been replaced with a graphical user-interface for both students and tutors, further extending the user-friendliness and functionality of the system as a whole. Not only does this provide student with a more intuitive means of submitting their assignments on-line, it also enables more aspects of the administration of assignments to be integrated into the system. The inclusion of electronic marksheets, for example, enables the provision of a system that supports both anonymous marking and double marking in a coherent, secure and efficient way.

More importantly, perhaps, the time needed to mark an assignment is reduced considerably, while the accuracy of marking, and consequently the confidence enjoyed by the students in the marking process, is improved. In addition, consistency is improved, especially if more than one person is involved in the marking process.

More information can be found on the Web at URL http://www.dcs.warwick.ac.uk/cobalt/

7. REFERENCES

[1] Benford, S. D., Burke, K. E. and Foxley, E. A System to Teach Programming in a Quality Controlled Environment. The Software Quality Journal, 2, 177-197, 1993.

[2] Benford, S. D., Burke, K. E., Foxley, E., Gutteridge, N. H. and Mohd Zin, A. Experience using the ceilidh system. Monitor, 4:32-35, 1993/94.

[3] Berry, R. E. and Meekings, B. A. E. A style analysis of C programs. Communications of the ACM, 28(1):80-88, Jan 1985.

[4] Hung, S. Kwok, L. and Chan, R. Automatic programming assessment metrics. Computers and Education, 20(2):183-190, 1993.

[5] Isaacson, P. C. and Scott, T. A. Automating the execution of student programs. ACM SIGCSE Bulletin, 21(2):15-22, 1989.

[6] Joy, M. and Luck, M. On-line submission and testing of programming assignments. In J. Hart, editor, Innovations in Computing Teaching. SEDA, London, 1995.

[7] Joy, M. and Luck, M. Software standards in undergraduate computing courses. Journal of Computer Assisted Learning, 12:103-113, 1996.

[8] Joy, M. and Luck, M. A user-friendly on-line submission system. In R. O'Connor and S. Alexander, editors, Proceedings of the Fourth Annual Conference on the Teaching of Computing, pages 92-95, Dublin, 1996.

[9] Kernighan, B. W. and Plauger, P. J. The Elements of Programming Style. McGraw-Hill, New York, 1974.

[10] Luck, M. and Joy, M. Automatic submission in an evolutionary approach to computer science teaching Computers and Education, 25(3):105-111, 1995.

[11] Ousterhout, J.K. Tcl and the Tk toolkit. Addison-Wesley, 1994.

[12] Reek, K. A. The try system - or - how to avoid testing student programs. ACM SIGCSE Bulletin, 21(1):112-116,1989.

[13] Rees, M. J. Automatic assessment aid for Pascal programs. SIGPLAN Notices, 17(10):33-42, Oct 1982.