A User-Friendly On-Line Submission System

M. Joy and M. Luck
Department of Computer Science
University of Warwick
Coventry, CV4 7AL
United Kingdom
Email: M.S.Joy@dcs.warwick.ac.uk
Fax: +44 1203 525714

Abstract

As student numbers on university computing courses increase dramatically, the difficulty of managing the work involved in the submission and assessment of programming assignments rises correspondingly. Spreading the load among several staff can help, but leads to problems of consistency. At Warwick, we have developed a suite of programs to support automated submission and assessment, and which can also be used to assist in marking. This paper considers the issues involved and outlines the main functionality of the software. It then describes the more recent developments of the windowed *version, detailing the user-interface, and the way in which it enables more effective and efficient marking. Finally, we briefly discuss some possible extensions.*

KEYWORDS: *assessment, programming, automatic submission.*

1 Introduction

University staff are under considerable pressure to deliver computer programming courses using available resources with maximum efficiency. Testing and marking assessed pieces of software is time-consuming and can be unreliable if attempted manually.

In response, we have developed a suite of programs for on-line submission and testing of programming assignments, with the aim of maximising portability, extensibility and usability. We describe the user-friendly software which allows students to submit programming assignments on-line, tests them against given specified data, and allows authorised staff to administer and mark them easily and efficiently.

Several issues are relevant to the development of such a system. First, although complete *security* on a university computer network is probably unattainable, risks introduced by the system (such as susceptibility to *hacking*, programs damaging the system, and the possibility of submitted documents becoming corrupted) should be minimised. The system must be sufficiently *flexible* to cope with different courses using different programming languages (both interpreted and compiled), and should also provide feedback to students, giving some indication of the performance of assessed programs. Finally, the system must be *easy to use* — both for the students and for the lecturer.

2 System Functionality

Our software, named BOSS[1], comprises a collection of programs which run under the UNIX operating system. It is designed for courses which are assessed by means of programming exercises, and requires that assessed work is in a form which can be specified very precisely (so that the output from students' programs can be compared with expected output); it is therefore not suitable for courses involving more generalised software design.

The individual component programs include the following facilities.

- A program is submitted and then stored so that it can be tested and marked at a later date. As part of the submission procedure, an automatically generated acknowledgement of receipt is sent to student by email as proof of submission. This receipt also provides a key to check the integrity of student program (and other) files which may be needed in case of system failure.
- All submissions for an item of coursework are run and tested against a number of sets of data. This allows the lecturer to evaluate programs based on an extensive variety and number of data sets so that all aspects can be considered separately.
- Once the tests are completed, the lecturer and tutors can inspect the submissions of individual students or of the entire course. The system provides the ability to view student programs together with the results of the various tests. If an exceptional case arises that has not been adequately addressed by the automatic testing, the system allows manual compilation and testing. In this way, the submissions are assigned marks quickly and carefully under the supervision of a lecturer or tutor who may intervene as appropriate.
- A separate utility allows a program under development to be tested against a given test data set, before it is submitted. This enables a student to ascertain that the program performs as required by the specification under precisely the same conditions as those under which it will be tested subsequently. In so doing, students gain some confidence that their programs are minimally acceptable.
- Finally, there is an extra utility which will check an entire course of submissions, for possible instances of plagiarism. This utility, called *Poirot*, performs a series of increasingly sophisticated comparisons between submissions, using the original programs, the

[1]BOSS stands for "The Boss Online Submission System.

programs with both whitespace and comments removed, and the programs tokenised. As output, the utility produces a graph showing the relation of individual submissions to each other, enabling suspicious cases to be identified easily and inspected carefully.

More complete details of these programs can be found in [**?**].

3 User Interface

Previously, the software has been available with a text-based interface, but a graphical front-end has recently been developed. The alternative interfaces allow it to be run on several hardware platforms.

The graphical user interface has been constructed using the TCL language and TK toolkit [**?**] to provide a windowed point-and-click environment for both submission and marking of assignments. For submission, this interface provides essentially similar functionality to the text-based interface, allowing students to select the required course, assignment and exercise for both actual and test submission as described above. For marking, this interface allows much more sophisticated marking procedures than were previously possible, such as a marking regime of *at least* double marking plus moderation.

Each marker is presented with a marksheet, designed by the course organiser, which includes sections for correctness and style. The correctness aspects are evaluated through the automated testing procedures by which a program is run on unseen test data and compared with expected output. The results of these tests are fed through to the marksheets which are automatically updated with a default number of marks based on a successful or unsuccessful outcome. However, as with all aspects of marking, a marker may override these defaults if required. The marker can view submitted programs and results and can also compile and run a copy of the program themselves.

Other categories in the style section are typically marked manually with the marker inspecting a program and assigning a numeric score, but it is possible to incorporate optional capabilities for automated marking of certain style aspects into the assessment process. All marks are assigned on a scale of 0-10. Though the course organiser may attach certain weightings to the different categories, these are not visible to the individual markers, so that the scales are consistent and do not skew assessment.

Once all submissions have been double marked, a moderator (typically the course organiser) can review the breakdown of marks for each assignment which are displayed in one window, together with the overall mark. The default *moderated* mark is just the arithmetic mean of the two final marks, but the moderator can adjust these, and deduct marks for lateness, for example. After moderation is complete, the moderator is asked to confirm the final score, and to commit that mark to a database for the construction of class mark sheets.

On the system we have running at Warwick, all of the above implements anonymous marking, so that the only means of identifying a student is by their library card number. Only at the point of committing marks to a database, are they de-anonymised, producing marksheets suitable for returning marks to students. At Warwick, we have tied in the system to the central university database for automatic anonymisation and de-anonymisation, for maintaining lists of students taking a course, and for storing their marks.

The system also provides a feedback facility for markers to comment on the submitted work. When all assignments have been marked, these comments can be emailed automatically to students, providing them with prompt and useful feedback.

4 Evaluation

The system has been used successfully on several courses, including two involving Pascal of about 200 students each, one on UNIX Shell programming of about 150 students, one on the compiler tools, Lex and Yacc, of about 70 students, and another using C++ of about 150 students. There has been a generally favourable student response, which has improved as the culture of automatic submission has established itself within the Department.

Large numbers of students can be handled efficiently by the system, with security of assignment submission being assured. Secretarial time is minimised and the volume of paperwork involved is reduced to (almost) zero. The time needed to mark an assignment is also reduced considerably, while accuracy and consistency are improved, especially if more than one person is involved.

The move to a graphical user interface has further enhanced the software, so that students exposed to a point-and-click environment before they arrive at university will more readily accept the submission process. This has also enabled the development of more sophisticated on-line marking capabilities that directly incorporate the checks necessary for a rigorous yet efficient system.

5 Further Development

This automation of the marking process, albeit under the direction of a lecturer and subject to manual intervention, provides further potential benefits. These arise through the online storage of submitted assignments and their marks, which may themselves be used as a resource for a variety of uses. In particular, this allows the generation of statistical data to enable a detailed analysis of student performance for examination boards, for example, and to enable easy normalisation of marks if required. There are, of course, other possibilities. The key point here, is that not only are processes of automated submission, testing and marking of assignments made more efficient and effective, but the side-effects of collecting and maintaining this data can also be important in themselves, adding significant value to the package as a whole.