Online Submission of Coursework – a Technological Perspective

Mike Joy, Nathan Griffiths
Department of Computer Science
University of Warwick
Coventry CV4 7AL
UK
{M.S.Joy,N.E.Griffiths}@warwick.ac.uk

Abstract

The development of an online submission system at the authors' Computer Science department over the last few years is described. The changing technologies used by the in-house software are discussed, together with the pedagogic and administrative issues that are affected by the process. The authors conclude by examining the issues that currently direct the future development of the software.

1. Introduction

The use of software to facilitate and add value to the process of student submission of assignments, and the subsequent marking by teaching staff, is becoming common, facilitated by the ubiquity of Internet access, and the relative affordability of computing equipment. Public domain and commercial course management tools are available which include assignment submission and automated assessment as part of the software functionality, for example WebCT [1] and Questionmark Perception [2]. However, such tools do not address the specific needs of computer programming assignments. In particular a system is required to support academics in assessing student submissions, through collecting submissions, performing automatic tests on them, checking for plagiarism, and providing an interface for marking and delivering feedback.

The "BOSS" Online Submission System has been developed over a number of years, as a tool to facilitate the online submission and subsequent processing of programming assignments [3]. In this paper we discuss the development of BOSS from a technological standpoint, and reflect on the issues that have affected it, both technical and process related.

2. Description of the software

The BOSS software permits users to perform six principle tasks.

1. Students are able to run automatic tests on their programs prior to submission (and afterward if they wish to resubmit within the prescribed deadline).
2. Students submit their (programming) assignments online.
3. Staff are able to run automatic tests on the set of student submissions, and as part of the marking process.
4. Plagiarism detection software identifies potential intracorpal source-code plagiarism.
5. Teaching staff can mark a submission online, by viewing the results of the automatic tests, running the submitted program, and viewing the submitted source code.
6. Teaching staff give feedback on each submission, and BOSS collates the feedback from the set of markers of a given submission and provides a mechanism for communicating this back to the student.

The software uses a client-server architecture with separate clients for students and for authorized staff (for security reasons). Each client is provided both as a secure web client and as a stand-alone application, so maximizing the flexibility of the system in terms of users' working environments.

An overview of the system architecture can be seen in Figure 1, showing its primary components. There are four data repositories (represented by rounded boxes), which store information about students, student submissions, the results of automated tests, and the results of plagiarism detection. There is an automatic test server which is responsible for performing tests on students' submissions and storing the results (or

Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT'04)
0-7695-2181-9/04 $20.00 © 2004 **IEEE**

passing feedback to the student if the test is being run

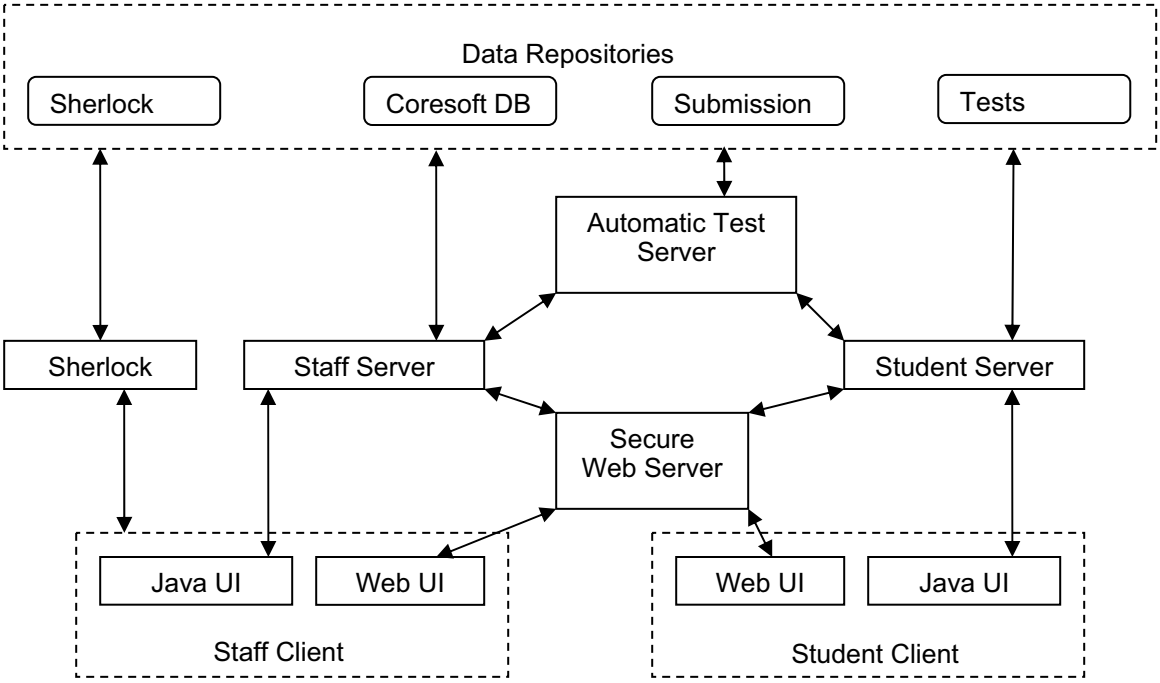

Figure 1. The BOSS architecture

prior to submission). The staff and student servers provide appropriate functions and data access to staff and students respectively. Both the staff and student servers have a web-based interface and a standalone Java application interface. The web interfaces communicate with the other system components via a secure webserver using SSL. The staff interface also provides access to the plagiarism detection software (called Sherlock) which analyses the stored submissions and stores various metrics for assessment by teaching staff [4].

The software has been made available as Open Source under the GNU General Public License. There are three primary reasons for taking the Open Source route. Firstly, the development of BOSS is not commercially viable given the level of commitment to support and on going development that would have to be made locally. Secondly, making the software Open Source encourages take-up by other institutions and the subsequent community support and development that naturally follow. Finally, placing the source code of the system in the public domain enables other institutions not only to use the system, but to customize and extend it without any license restrictions (and hopefully feed back their extensions to the user community).

3. The technological dimension

The initial software package was developed in the mid 1990s, when the majority of terminals were still text-only, and students would normally interact with the University computer systems from on-campus.

The technology initially deployed was an application with a text interface, which ran on a central UNIX server. Coding was in ANSI C, and designed in as re-usable and modular a fashion as the language would easily allow. Security was achieved by means of standard UNIX file permissions, and judicious use of the "setUID" mechanism. The Snefru [5] hash algorithm was used to sign each submission and ensure the integrity of submitted data.

This solution was successful, but the rapid advent of higher-quality terminals with graphic capability suggested that an improved user interface was desirable. Not only would staff productivity increase with a "point and click" interface, but student perception of the software would improve, since it would appear more "up to date". The immediate solution was to add a front-end coded in Tcl/Tk, which was relatively easy to implement due to the modular structure of the underlying code [3]. Whilst this solution was effective, it exposed a fundamental weakness in the choice of technology – the scalability was poor. For example, the modular constructs of

Tcl/Tk are few and primitive, and the Tcl/Tk scripting language is weakly typed. It was felt that the software was not amenable to significant development in its current state, and a permanent solution was sought.

The relatively novel (at the time) Java language was chosen to form the basis of a complete re-write, Not only was the language seen as suitable for large scale projects, but its platform-independence would shield us from future decisions about hardware purchase. Moreover, the Computer Science department uses Linux and Solaris based machines, whilst the rest of the University uses Windows based solutions and so it was desirable for BOSS to be usable on each of these platforms. The use of Java made this relatively simple (in comparison to the use of alternative languages such as C++), by providing the same class files to each platform via a platform specific bootstrap mechanism.

In 2000, a small team of undergraduates was employed over the summer vacation to implement the rewrite, and the product – a client-server architecture using RMI for data transport – forms the basis of the current system. At an early stage in the development of the Java code, we decided that any maintainable and robust solution required a modular approach. Both CORBA and RMI were considered, the latter chosen on account of its Java base and consequent ease of coding. The use of Applets was ruled out, since correct functioning of Applet-based clients is dependent on the browser support for Java and the power of the client machine. Not only do some proprietary browsers not support Java fully (and this has been the subject of litigation both in the US and the EU), but at the time students' personal machines were unable to run complex Applets acceptably fast.

3.1. Plagiarism detection

The department's plagiarism detection software, known as "Sherlock" [4], has been developed in parallel with BOSS, and until 2002 was a separate tool. Sherlock reports on a collection of documents and reports instances of pairs (or larger clusters) of documents that contain similarities. Initially written for use with Pascal (and now Java) programs, Sherlock has been extended to use freetext data files. Both its source code and free text facilities compare well, both in terms of accuracy and of ease of use, with other plagiarism detection tools such as CopyCatch [6].

3.2. HCI issues

The development of both web-based and application clients – which may appear a duplication of effort – is motivated by two main factors.

Students demand a simple to use product to submit their work, both from the campus and when working at home, suggesting a web client as being appropriate.

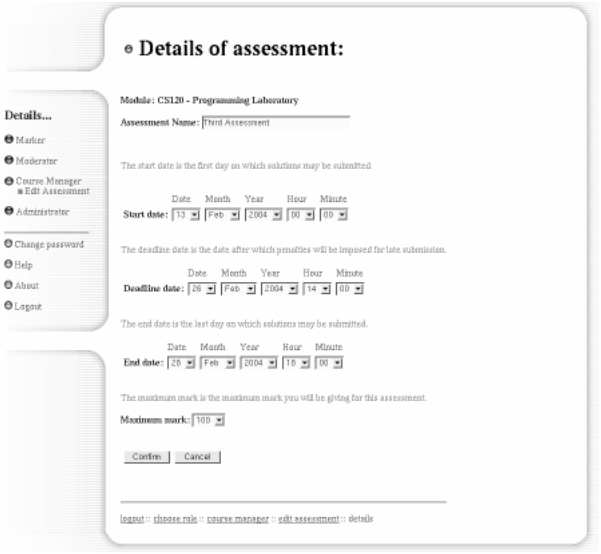

Figure 2. Web client screenshot

Figure 2 shows screenshot of a dialogue from the web-based client for staff.

Staff who are marking assignments for large classes desire an interface which is quick to use, and minimizing of key strokes. This type of interface is simpler to code as an application, and when used on a machine directly connected to the campus network avoids the delays inherent in the web-based solution. The corresponding screenshot is presented in figure 3.

Both interfaces have been coded to take account of appropriate "good practice" [7]. For example, the web interfaces are structured as collections of relatively small individual pages with many navigation aids and shortcuts, and are appropriate for remote access to the server where the connection may be slow or unreliable. The application interfaces maximize the amount of relevant information available on each screen, to enable the user to navigate through the dialogues and complete their task, and is appropriate for local high-speed connections normally available to staff. Both student and staff clients have been coded with both types of interface, and evaluation of the usage patterns is ongoing.

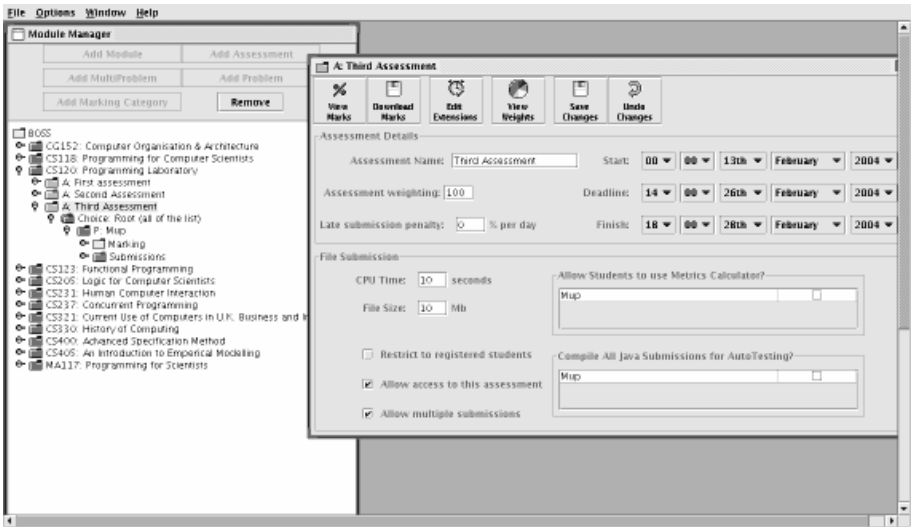

Figure 3. Application client screenshot

3.3Data storage

Central to a data-bound application such as BOSS is the storage and management of the data. In addition to storage of submitted assignments as archives on secure backed-up file systems an SQL database is used for other data, such as times of submissions, basic student identity information, and marks awarded. The initial deployment of a proprietary database was found to be unsuccessful (due to the repeated requirement of systems staff to manage the database), and free databases such as MySQL, MSQL and PostgreSQL have since been used. Differences between the dialects of SQL used are a continual source of frustration, though the latest versions of MySQL and PostgreSQL allow interchangeability with minimal intervention, assisted by the use of JDBC to connect with the Java servers.

In order to facilitate the import of data from external sources (such as the University's Student Record System, or SRS), an "institution-independent" database schema – CoreSoft – was developed [8]. The aim of CoreSoft was to present the minimum data required for BOSS (and other related applications requiring similar data) in a format that would be compact, and use appropriately normalized tables with easy to remember names. The translation of data from external databases to the CoreSoft schema (and vice-versa) is – at least in principal – a straightforward task.

4. University process

In order for a system such as BOSS to be used effectively, it must interact with institution processes efficiently and accurately, and several issues have arisen during the deployment of BOSS that may well apply to many other institutions.

4.1. Databases

The quality of data received from the SRS is sometimes poor. For example, delays in updating student data centrally often preclude the automatic generation of accurate lists of students registered for a given module.

The schema used by the SRS is required for generation of accurate statistical data for government agencies, in addition to the more central management functions. The types of statistics required affect the table structure of the database – for example, separate module codes are used for students resitting a module – and cause the import of data into BOSS (through the generic CoreSoft database schema) to be more difficult than expected.

4.2. Funding and support

Both students and staff have undertaken development of BOSS. Several final year undergraduate projects have addressed specific sections of the code, and from time to time students have been employed during summer vacations to work on the software. Funding for the latter has always been internal to the University, both from the Department's own resources, and from centrally managed funds

(such as the University's "Teaching Development Fund").

The Department's staff have normally provided support for the software. Central support, through the University's IT Services Department, has usually been inappropriate, due to the concentration of expertise in the technologies employed being within the Department.

5. Pedagogy and quality assurance

BOSS has been conceived as a tool targeted at a single task – management of online programming assignments. It is *not* intended to provide a suite of learning materials, and contains no functionality to support students' learning other than that which directly arises from the activity of assessment. The support for learning provided by BOSS is encapsulated by the process of a student getting feedback from automatic tests prior to submission, followed by feedback from markers after submission. Thus the learning benefits to students of using BOSS are similar to other assessment methods, and are primarily dependent on the *academic* design of the assessment (or preferably a sequence of both formative and summative assessments) and the quality of feedback given by markers.

It is interesting to compare BOSS' approach with that of *CourseMarker* [9], a tool developed at the University of Nottingham (previously known as *Ceilidh*). The approach taken in CourseMarker is to allow students to present solutions to programming problems frequently as a formative process. Each solution is then marked against a "template" and against a variety of metrics, allowing the student to improve their solution prior to submission by assimilating the frequent feedback presented by CourseMarker. This was an approach that we chose not to follow, since we wished to focus on the process, and measuring the correctness of students' code. Furthermore, the CourseMarker approach prescribes a style of programming, which it might be argued is not always appropriate, and we decided that such functionality would be inappropriate for BOSS. Our emphasis is on providing a tool to assist teaching staff and encourage best practice in teaching programming rather than to provide an online learning environment.

The use of Sherlock to assist in plagiarism detection has been successful, and has reduced the instance of plagiarism on large programming modules to less than 5% [4].

It should be noted that although our primary aim is to support the teaching of programming, BOSS is also useful as a submission and marking tool for other types of assessment, such as essays. It provides an effective means for the collection of submissions, since students can submit using computers across the campus or even from home via the web interface. The Sherlock plagiarism tool allows teaching staff to detect intracorpal plagiarism in the essays submitted by students. BOSS can also be used as a repository for a marker (or group of markers) to store feedback on each submission. At the end of the marking process this feedback can be collated and moderated for each submission and then returned to the student.

6. Future directions

Any initiative that is dependent on technology is also at risk from changes in technology, and it would be unwise to speculate what those changes will be. However, the paradigm used by BOSS appears to support a variety of courses successfully, and significant changes are not currently envisaged. The underlying technology will be upgraded as and when suitable new technologies and standards are in place (for example, the use of the Simple Object Access Protocol – SOAP – or other XML-based standard for encapsulating the data used by BOSS, is under investigation).

Since BOSS is now Open Source, it is hoped that colleagues at other institutions will identify (and code!) additional functionality.

7. References

[1] WebCT. http://www.webct.com/ (accessed 29 February 2004)

[2] Questionmark Perception. http://perception.questionmark.com/ (accessed 29 February 2004).

[3] Michael Luck and Mike Joy, "A Secure On-line Submission System", *Software - Practice and Experience* **29**(8), 1999, pp. 721-740.

[4] Mike Joy and Michael Luck, "Plagiarism in Programming Assignments", *IEEE Transactions on Education* **42**(2), 1999, pp. 129-133.

[5] R.C. Merkle, "A fast software one way hash function", *Journal of Cryptology*, **3**(1), 1990, pp. 43-58.

[6] CopyCatch Gold. http://www.copycatchgold.com/ (accessed 29 February 2004).

[7] Ben Shneiderman, *Designing the User Interface (third edition)*, Addison-Wesley, 1998

[8] Mike Joy, Nathan Griffiths, Mary Stott, Jon Harley, Cathy Wattebot and Derek Holt, "Coresoft: A Framework for Student Data", *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, Loughborough, LTSN Centre for Information and Computer Sciences, 2002. pp. 31-36.

[9] http://www.cs.nott.ac.uk/CourseMarker/