Tools and Techniques for Large Scale Grading using Web-based Commercial Off-The-Shelf Software

*Robert N. Lass Christopher D. Cera Nathaniel T. Bomberger
Bruce Char Jeffrey L. Popyack Nira Herrmann Paul Zoski

{urlass,cera,ntb22,bchar,jpopyack,nherrmann,pzoski}@mcs.drexel.edu
Drexel University Programming Learning EXperience (**DUPLEX**)
Departments of Mathematics and Computer Science
Drexel University
Philadelphia, PA 19104
http://duplex.mcs.drexel.edu

ABSTRACT

Courseware/Course Management Systems (CMS) such as WebCT or Blackboard are an increasingly popular way to provide a web presence for a course. However, their current web-browser reliance makes it difficult for them to provide functionality that could be useful to computer science instructors. This paper describes our augmentation of a CMS in a large introductory computer science class. It further describes our enhancement of the CMS by client-side software (i.e. residing on the graders computer), written for use by the instructors and graders. Finally, it indicates how conventional CMS architecture can be extended to provide additional functionality that would be desirable for computer science instruction.

Keywords

Course Management Systems, Courseware, WebCT, Plagiarism Detection, Electronic Pen-based Markup, Introductory Programming

Categories and Subject Descriptors

J.1 [**Computer Applications**]: Administrative Data Processing—*Education*; K.3.1 [**Computing Milieux**]: Computer Uses in Education—*Computer-Assisted Instruction (CAI)*; K.3.2 [**Computing Milieux**]: Computer and Information Science Education—*Computer Science Education*

General Terms

Management, Design, Human Factors

1. INTRODUCTION

Web-based Course Management Systems (CMSs) provide an increasingly elaborate environment for instructors and students. These portals provide a central repository for lectures, course materials, discussion and email capabilities, and facilities for online testing and assignment submission. In higher education, these facilities are becoming more and more common for delivering a cost-effective and cooperative learning environment.

We provide introductory computer programming courses to Computer Science (CS), Computer Engineering (CE), Information Systems (IS), and Digital Media (DM) majors. Our teaching staff consists of tenure-track and auxiliary faculty working with graduate teaching assistants and undergraduate lab assistants and graders. For large classes of about $250-300$ students, we typically employ 2 professors, and $10-12$ teaching assistants (TAs). Further discussion of our curricular redesign can be found in [4].

The course format consists of a weekly, one-hour lecture given by faculty and a two-hour lab section where students take an online quiz and complete group exercises under the supervision of graduate and undergraduate assistants. Course management is handled through our institutionally supported CMS, which also serves as a repository of course materials, student submitted work, and grades. This repository is ideal for gathering statistical artifacts useful for administrative evaluations over several iterations of the same course.

Advantages of Course Management Systems include file sharing among the teaching staff or between teaching staff and students; electronic storage of student documents in a central repository, which minimizes the loss of student work; and ease in sharing course materials between terms since course directories can be copied into a new course. There are also blanket facilities for collaborative services such as newsgroup-style discussion threads and chat.

Our experience using a CMS is mixed: its strength as a repository for sharing information in a controlled fashion generally offsets the extra effort students and instructors must exert to learn how to use it. We have also found some aspects, such as its browser-centric user interface to be excessively time consuming to operate. In addition, not all the features we need to make our courses run

*This work supported in part by Drexel University, the National Science Foundation Division of Undergraduate Education through grant DUE-#0089009 and the Pew Learning and Technology Program at the Center for Academic Transformation as part of the Pew Grant Program in Course Redesign.

ITiCSE 2003 6/03 Thessaloniki, Greece

more efficiently are currently available.

A general CMS can not offer features that would only benefit teaching staff in a limited domain. For instance, a general CMS will not provide support for specialized source code plagiarism detection. A general system probably should not try to do so, but it should at least provide interfaces to support interoperability with other systems.

The majority of the work on automation and management of computer science courses focuses on "homegrown" systems that are not intended for wide deployment outside of their department or university [13, 14, 10, 2, 5]. Another family of university-developed courseware focuses on computer-aided assessment in the form of interactive tutoring [10, 8]. We focused our efforts on writing software to interact with the third-party CMS supported by our university. We found that we were able to easily write software to automate repetitive tasks associated with the CMS. This approach conserves our resources, since we only have to administer our client program and not the CMS itself.

Computer science academics, when faced with less-than-ideal software, are tempted to write their own from scratch. However, the task of developing and maintaining a large "mission-critical" system requires serious commitment of resources. Either commercial support by itself (as envisioned by WebCT, Blackboard, and similar commercial systems) or in conjunction with an open-source effort (such as envisioned by the Open Knowledge Initiative (OKI) [9]) is needed. A more limited approach, namely creating software tools to interact with an existing CMS system to provide additional features, is a feasible and useful alternative; one we have embraced. We report here on tools created to support by facilitating course management in large CS courses and extend the capabilities of the CMS.

Our experience with user-developed, customized add-ons to CMSs illuminates an area that will be of growing interest and importance. Indeed, OKI has realized this view and has built-in APIs which support a subset of the enhancements described in this paper.

The rest of this paper is organized as follows: Section 2 discusses the general problems and our goals that include building software to solve them. Section 3 discusses our approach to solving these problems in the context of CS courses. Section 4 introduces our cross-platform software solution which demonstrates our ideas of CS course management. We further elaborate on its use in a practical course environment and the details of its implementation. Section 5 discusses our results, presents our conclusions, and outlines our goals for future research.

2. PROBLEMS AND GOALS

CMSs typically provide facilities for instructors to post assignments, for students to submit electronic copies of assignments or quizzes, and for instructors to post grades for them after electronically downloading the student files for inspection on their own machine. This is convenient for large courses, in that all work is time-stamped and archived, students need not hand in a hard copy or floppies, and password protected grades are available to students whenever and wherever they have access to a web browser. However, we found both "early adopter" and "design limitation" problems with these functions.

Initially, with our CMS, WebCT [15], we found that downloading submitted homework assignment files for a single section or the entire class involved substantial effort: multiple clicks and some typing were required to download each student's program files onto a grader's local file system. It was difficult to distribute or collect student work by lab section while still providing students a common site to pick up and submit material. Once downloaded, additional commands were needed to handle files that were deposited by students that were archived, compressed, or encoded (e.g. `.tar.gz`, `.uue`, `.zip`). Care had to be taken to ensure that all files were downloaded and that the resulting file had a directory structure that made it easy to find a particular student's files.

We decided to create a software tool that would interact with the CMS server software and provide the following features:

1. Quick download/upload of files with minimal clicking.

2. Ability to pick up or deposit student work by lab sections.

3. Organization of downloaded files into coherent directory structures.

4. Extensible bulk post-processing of files: file transformation, submission of class files to spelling, style, and plagiarism checkers, etc.

Our goal was to quickly and easily download only the assignments needed by a specific grader (e.g., just students in one particular section) and have them organized into a systematic format suitable for grading or interfacing with another program for further processing (e.g., the JPlag [12] plagiarism detection system). We also wanted the software tool to be easily extended to other uses, such as generating PDF versions of written work and source files that can be graded electronically with digital pen-based markup [11].

Administrator-level access in CMSs is typically restricted to a few people because of the need to protect the security and integrity of the database. Instructional staff members are given more limited ability (in WebCT, "Designer access" or "TA access") to change or extract information from the database: in order for staff members to download individual student assignments, they must click on each student's user name individually, then on each file the student uploaded. This can be a very laborious and time consuming task for a class of dozens or even hundreds of students, particularly since programming assignments can contain several files.

Recent versions of our CMS allow a staff member to zip each student's assignment files into a single file, but still require clicking on each student's user name to obtain a single section. Downloading the whole class does not entirely solve this problem since the grader still must separate his or her section from the rest of the class. Also, this is not always quicker than downloading the individual files if the server is slow in handling the increased processing required to compress the files.

Sending the appropriate student files to graders in a large course was also a problem with earlier versions of WebCT. In order to share common content files (e.g. lecture notes), all students were enrolled in a single class. But then it became difficult to identify students by section so that grading work could be divided up by section. Furthermore, even if section information is available in the gradebook, as with version 3.8 of WebCT, in some classes at Drexel grading is assigned to TAs through other criteria, such as the last two digits of the student ID number. The CMS should allow the client to specify the criteria for assigning grading work to TAs.

3. TECHNICAL APPROACH

This section will describe our approach to the management of CS courses. This approach is broken down into a series of stages, where each stage is dependent upon all previous stages. Most of the operations described here will be invoked by a staff member, and the processes executing those operations will reside on the host of that staff member.

3.1 Bulk Downloading of Student Submissions

All student submissions, whether assignments, quizzes or examinations [1], will be in a centralized repository. An assignment will consist of a set of files, while an exam will typically be an ASCII file consisting of questions and answers. A grader should have the ability to automatically download submissions in bulk (i.e. for a particular submission, download every student who submitted that assignment), sort them by section, and ideally they should be unarchived and uncompressed after transmission. The range of queries invoked by a staff member will vary greatly. For instance, there might be only a subset of students to grade (i.e. just a particular section) and for only a subset of their submissions.

Most systems enforce an "interactive grading" approach. This is where a grader views a submission using their web browser, and performs the actual grading online. Hence, the grader can view the submission, but never actually save it on their machine unless they trigger the browser to do so. This approach is inadequate since it requires a grader to have network connectivity whenever they wish to grade submissions. It is much less restrictive and more practical to have a system which allows graders to download submissions to their machines when they have network connectivity (in bulk), and grade the assignments without being connected to the Internet.

Regardless of the interface provided by the repository, sophisticated functionality can still be developed using software tools that operate at the same level as the course staff (faculty and teaching assistants). It is often the case that the repository is not developed or even maintained by the staff members. Most of the functionality that is available through the web-browser interface can be automated by a simple HTTP program. The authors believe that the investment of time in automating this process is warranted since the labor savings in later stages of the grading process outweigh the initial investment in building the automated extraction functionality. Those savings will be addressed in the following sections.

This is the only aspect of developing software tools which depends upon the actual repository interface. Commands used by each CMS and storage locations for files differ, which means that porting the software tools to other systems initially requires a certain amount of customizing. In addition, when the CMS interface changes, preserving the software tools ability to access the system will require some maintenance activities. However, the remaining functionality of the software tools can be decoupled from the details of how the repository is maintained, so changes to these sections of the software are not needed unless additional information from the repository is required. Generally, all additional processing is done on the user's machine rather than in the actual repository, an advantage if the repository interface is subject to frequent changes in its access interface.

3.2 Post-Processing Assignments

After submission, a file may be further processed to help grade it. This processing can occur on the grader's computer as opposed to the CMS server.

The original files submitted to the repository may be in a combination of formats, and therefore must be properly filtered, decoded, dearchived, and decompressed. Thus subsequent stages of the pipeline do not have to perform this filtering. A subset of the types of file formats submitted by students include:

1. `tar` format, used to archive file system hierarchies into a single file.
2. `gz,bz2` formats, used to compress files.
3. `zip` format, the most widely used archive and compression format.
4. `uue` format, a more arcane format used to transfer binary data between systems that do not support receipt of binary data.
5. `pdf,rtf,doc` formats, binary encoded formats used to submit typeset textual data.
6. `txt` formats, used to submit ASCII data which could be documentation, or source code.

Some binary formats, such as that of Microsoft Word are proprietary. Limited reverse-engineering does exist for conversion of such files to text.

3.3 PDF files for Electronic Pen-based Markup

Using Adobe Acrobat software, PDF documents can have sections of text highlighted, underlined, and further annotated before being returned to students. Pen Tablets, such as the Wacom Graphire or the Sony Vaio Slimtop Pen Tablet PCV-LX920, provide an intuitive interface for such marking of PDF documents. This feedback method has the closest resemblance to the handwritten markings of traditional pen and paper grading while keeping the advantages of being represented digitally.

Once we have transformed source code from submitted student assignments into Adobe's Portable Document Format (PDF) files, we can load them into Adobe Acrobat, and annotate them using a pen tablet. The documents and grader markup can then be saved and returned to students. A similar effect could be achieved by loading the document into Windows Journal on a Tablet PC and writing over it. An advantage of returning PDF files to students is that they can be viewed on almost every platform.

The electronic grading interface that WebCT provides consists of text-box where a grader enters comments, and another text-box where they enter the grade. This is extremely inadequate, and forces the grader to resort to citing line numbers with every comment. When assignments were submitted on paper in the traditional fashion, graders were able to circle blocks of code, draw arrows, and so on. A text-box interface increases the amount of work required to make comments, and makes the resulting comments much more difficult for a student to understand.

A standard collating sequence is used to concatenate multiple text files into a single PDF. The "bookmark" feature of Adobe Acrobat can be used to allow the grader to jump easily between different sections of the concatenated result.

3.4 Interfacing with Heterogeneous Systems

There is a need for course repositories to inter-operate with other heterogeneous systems. Every system has at least one format or protocol it accepts as input. Unfortunately, these formats are usually different and often incompatible with one another for different reasons. Our approach to interfacing with heterogeneous systems involves downloading submissions to the graders machine as an intermediate step.

Plagiarism is a troubling problem, and especially laborious to detect in large classes. An automated solution aids appropriate and timely detection of the problem. here are several free and commercially available programs and services which perform batch similarity measures on a collection of documents [1, 12, 16]. In introductory programming, we require similarity detection of source code as well as written documents.

[1] Our weekly laboratories are implemented as examinations since these are also conducted in a question and answer format.

The referenced systems will check student files placed in a file directory of the user for similarity. These systems report overall similarity metrics based on a single document or a collection of documents. Interfaces providing document by document comparisons are also available to aid in human inspection and confirmation of plagiarism.

Moss. We have made extensive use of the Moss program to detect plagiarism in C++ programming assignments. Moss handles several programming languages, including C, C++, Java, ML, Lisp, Scheme, Pascal, and Ada. The diversity of language support can be utilized in a variety of programming courses.

Output from Moss provides a list of HTML links to flagged regions of suspect documents as well as a measure of percent similarity. Moss also has a "common code" feature that allows the user to specify code that should be removed from the similarity detection. This is useful, for example, when the instructor provides computer code for students to use as part of their solution to an assignment; thus, all students would be expected to have this code in common and it should be eliminated when checking for plagiarism.

JPlag. JPlag is another plagiarism detection system that we started using more recently. The programming languages supported by this system include C, C++, Scheme, and Java. It computes a similarity percentage among two documents, or groups of documents if applicable.

This program has an added advantage that it works on written English text, so it can detect plagiarism within the internal and external documentation that accompanies student programs. This system could also be utilized in liberal arts courses to detect plagiarism in written documents, reports, etc.

4. IMPLEMENTATION

The following subsections detail our operating environment, and the tools we've developed to provide the services outlined in this paper. This tool provides bulk downloading of submissions, post-processing of files into an organized directory structure, and PDF generation to facilitate pen-based markup. Our goal is to have a client-side application which runs on multiple platforms and that creates a more efficient grader experience when interacting with a course repository, and providing a platform for the specific tasks which occur thereafter.

4.1 Labrador

Labrador[2] is, a cross-platform utility which combines "ease of use", and the ability to handle a large number of students efficiently. We have found that use of Labrador has significantly reduced the time spent grading under WebCT. Overall, this has allowed our staff to spend more time with students, and less time grading.

4.1.1 Pipeline

Here we present our implementation of the approach defined in Section 3.

Downloading Submissions. Our classes make use of the WebCT CMS as the course repository. WebCT allows an authorized grader to download student submissions after clicking through a laborious series of web-based menus. Given a query consisting of a course name, grader's name and password, the submission's title, and a set of students, Labrador masquerades as a human user and crawls through the series of menus until it retrieves the submissions. WebCT's authentication scheme require our software to manage cookies. In addition, the web-crawling mechanism requires a combination of `HTTP-Get` and `HTTP-Post` operations. The extraction interface provided to us is unfortunately HTML. Hopefully future repositories will support better protocols designed specifically for general retrieval of data, as opposed to data designed for merely the display of content (i.e. HTML). The IMS standards [6] and those proposed by OKI[9] are presently addressing these needs. IMS has proposed standards for XML representation of quizzes or tests and student responses and OKI has an API built in that would assist people in building applications such as Labrador.

Post Processing. This optional stage examines each individual file, identifies the encoding format of the file, and extracts the original contents of the file. If any directories are created during this process, the files contained in those directories recursively evaluated in a similar fashion.

PDF Generation. If a student submits multiple files, they are all put together into a single PDF document in a reasonable order. Bookmarks are also created that the grader can click on to easily jump between files while grading.

User Interface. We provide a variety of methods for interfacing with Labrador to satisfy the varying preferences of different graders. Currently, there are four different methods:

1. *Command-line*. All of the information needed to retrieve the proper files from WebCT is provided in various options of the command-line prompt. While user-unfriendly, it makes it easy for Labrador to be invoked in in shell scripts, or through a remote network terminal session.

2. *Configuration File*. A user invoke Labrador with a configuration file in lieu of command-line options. A configuration file can be generated during a Labrador run, to be used in the future. For security reasons, the password cannot be stored in the configuration file, and must be entered at run time.

3. *Interactive*. If the needed information is not supplied at the command-line, Labrador will prompt the user for it. In the Windows environment, double-clicking on the Labrador icon is equivalent to running the program with no command-line options and results in a series of interactive prompts.

4. *Graphical User Interface.* A fully functional Graphical User Interface (GUI) has been developed using Perl/Tk [7].

5. IMPACT

We have found significant advantages to maintaining our course using a Course Management System. Prominent among them are less time spent with HTML layout for course websites, a technology learning curve amortized over many courses instead of just one, fewer student papers misplaced and secure authenticated access to a campus-wide portal. Early adopters, however, will continue to wish for additional functionality for their domain to achieve significant time savings or facilitate additional pedagogical goals. More significantly, we see it inevitable that certain courses or disciplines will desire additional operations or post-processing that would be

[2] The name refers to the canine breed Labrador Retriever. Its name was chosen to emphasize the "fetching" capabilities of the application.

unsuitable to offer to users who would find such functionality confusing or irrelevant to their disciplines.

We have focused on particular services that could be feasibly developed over a modest period of time and that brought substantial value to our teaching staff. The goal of our software solution is to make staff more productive by allowing them to spend their time grading and interacting directly with students instead of operating user interfaces. In addition, since Labrador removed a major source of irritation from graders when using the CMS, they were more open to considering its benefits and making use of its strengths.

6. FUTURE WORK

We are continuing to develop software tools to facilitate course management, interact with other software systems, and provide an easier interface for our students and faculty.

Some areas of interest in the future are automatically compiling student program submissions and running them through a test suite [3, 13]. We are also hoping to begin work on a system for automatically checking the coding style of student homework.

It would be desirable to have a way of easily retrieving data (in the form of previously completed assignments) from earlier offerings of the same course in the event that a particular assignment were reused. This would further eliminate the long-term plagiarism problem for frequently offered courses.

Finally, we have begun work on completing the cycle: pushing the grades back into the CMS.

7. CONCLUSION

The primary goal of our work here is to improve instructional productivity by reducing effort by staff, as well as allowing instructors more sophistication in the management of CS courses. We see considerable growth at providing customized services based on information available through CMSs and other web-based sources. CMSs must provide application programmer interfaces to allow programmed, secured interactivity with clients and other servers for this to become feasible.

We have discussed how we designed and implemented Labrador to allow more sophistication use of a particular CMS in the management of CS courses. We have seen that with WebCT at least, greater sophistication led to reduced time spent operating the CMS by instructional staff. In our introductory class alone, we estimate that Labrador saves us a total of five hours with each assignment.

Conceptualizing and modularizing the actions Labrador takes made it easier to conceive of and providing additional functionality in support of CS instructional goals. Eventually, we see CMSs uniformly providing APIs to allow local computing staffs to provide customized connections to academic web services such as plagiarism detection, program testing, or instructor-generated grading feedback.

8. ACKNOWLEDGMENTS

We wish to acknowledge the assistance of Mr. John Morris, Dr. Janice M. Biros, and their colleagues in the Office of Information Resource and Technology at Drexel University for their help and support with WebCT.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, the Center for Academic Transformation, or the other supporting organizations.

9. REFERENCES

[1] Alex Aiken. MOSS: A System for Detecting Software Plagiarism (Unpublished), `http://www.cs.berkeley.edu/~aiken/moss.html`.

[2] S. Benford, E. Burke, E. Foxley, N. Gutteridge, , and A. M. Zin. A Course Administration and Marking System. In *Proceedings of the International Conference of Computer Based Learning*, 1993.

[3] Michael H. Goldwasser. A Gimmick to Integrate Software Testing Throughout the Curriculum. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, pages 271–275. ACM Press, 2002.

[4] Nira Herrmann, Jeffrey L. Popyack, Bruce Char, Paul Zoski, Christopher D. Cera, Robert N. Lass, and Aparna Nanjappa. Redesigning Computer Programming Using Multi-level Online Modules for a Mixed Audience . In *Proceedings of the Thirty-Fourth SIGCSE Technical Symposium on Computer Science Education*, pages 0–0. ACM Press, February 2003.

[5] J. Hyvonen and L. Malmi. Trakla – A System for Teaching Algorithms Using Email and a Graphical Editor. In *Proceedings of HYPERMEDIA*, pages 141–147, 1993.

[6] IMS Global Learning Consortium, Inc. Welcome to IMS Global Learning Consortium: Specification: Question and Test. `http://www.imsproject.org/question/index.cfm`, March 2003.

[7] Stephen Lidie and Nancy Walsh. *Mastering Perl/Tk*. O'Reilly, 2002.

[8] Thomas Lozáno-Ṕerez, Eric Grimson, Leslie Kaelbling, Chris Terman, and Patrick Winston. Technologically Enhanced Education in Electrical Engineering and Computer Science, `http://www.swiss.ai.mit.edu/projects/icampus/projects/eecs.html`.

[9] Open Knowledge Initiative. `http://web.mit.edu/oki`.

[10] Abelardo Pardo. A Multi-agent Platform for Automatic Assignment Management. *ACM SIGCSE Bulletin*, 34(3):60–64, 2002.

[11] Jeffrey L. Popyack, Bruce Char, Nira Herrmann, Paul Zoski, Christopher D. Cera, and Robert N. Lass. Pen-Based Electronic Grading of Online Student Submissions. In *Syllabus fall2002, Technology for Higher Education Conf.*, November 2002.

[12] L. Prechelt, G. Malpohl, and M. Philippsen. JPlag: Finding Plagiarisms Among a Set of Programs. *Technical Report 2000-1, Fakultat fur Informatik, Universitat Karlsruhe, Germany*, March 2000.

[13] Kenneth A. Reek. The TRY System -or- How to Avoid Testing Student Programs. In *Proceedings of the Twentieth SIGCSE Technical Symposium on Computer Science Education*, pages 112–116. ACM Press, 1989.

[14] Michael Richichi. ATTIC: A Case Study of Directory-enabled Course Management. In *Proceedings of the 29th annual ACM SIGUCCS conference on User services*, pages 258–261. ACM Press, 2001.

[15] WebCT. `http://www.webct.com`.

[16] Michael J. Wise. Yap3: Improved detection of similarities in computer program and other texts. In *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*, pages 130–134. ACM Press, 1996.