Labrador: A Tool for Automated Grading Support in Multi-section Courses

Christopher D. Cera[1], Robert N. Lass, Bruce Char, Jeffrey L. Popyack, Nira Herrmann, and Paul Zoski

Drexel University Programming Learning Experience
(**DUPLEX**)
Department of Mathematics and Computer Science
Drexel University,
3141 Chestnut Street
Philadelphia, PA 19104
http://duplex.mcs.drexel.edu

1. Introduction

Web-based Course Management Software (CMS) provides a single portal for students and teaching staff to collaborate in an elaborate interactive environment. This portal provides tools for online submissions, examinations, student performance monitoring, discussion threads, real-time chat, and electronic mail. In higher education, these facilities are becoming more common for delivering a cost-effective and cooperative learning environment.

We provide introductory computer programming courses to Computer Science, Computer Engineering, Information Systems, and Digital Media 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 2-3 professors, and 10-12 teaching assistants.

Our first two introductory programming courses consist 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 WebCT, which also serves as a repository of course materials, student submitted work, and grades. This repository is ideal for gathering statistical artifacts useful for future administrative evaluations over several iterations of the same course. However, the software does require an initial time investment to learn. As well, the time required to perform routine repetitive tasks such as grading student assignments can be very high. Discussions with other instructional users [2], indicates a similar experience.

WebCT's 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. But, we have also found some aspects of its user interface to be somewhat time consuming to operate. We are confident that these issues will be resolved in the future. However, our immediate operational needs for large classes motivated us to create software tools to facilitate bulk downloading coupled with post-processing of student submitted work, which we report on here. We believe

[1] This work supported in part by Drexel University, the National Science Foundation's 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. Email contact: `cera@mcs.drexel.edu`, Website: `http://duplex.mcs.drexel.edu`

that our experience with user-developed, customized add-ons to CMSs illuminates an area that will be of growing interest and importance.

The rest of this paper is organized as follows: Section 2 outlines the general problem and our goals in building software to solve it. Section 3 introduces our cross-platform software solution to the problems. We further elaborate on its use in a practical course environment and the details of its implementation. Section 4 discusses our results, presents our conclusions, and outlines our goals for future research.

2. The Problems and Solution Goals

We began using WebCT Campus Edition 3.5 in September 2000. 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 uncompress files that were deposited by students in various archiving/compression/encoding formats (e.g. .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

Our goal was to be able 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 [3] plagiarism detection system). We addressed this problem by creating a software tool we call Labrador. This tool has also proved extensible and suitable for other uses, such as generating PDF versions of written work and source files that can be graded electronically with digital pen-based markup.

WebCT administrator-level access is typically restricted to a few people because of the need to protect the security and integrity of the WebCT database. Instructional staff members are given more limited ability (Designer 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 username individually, then on each file the student uploaded. Beginning with WebCT version 3.7, users may download all files for a particular student's assignment by using the zip feature. However, the user still must click a checkbox to download each student's submission. Figure 1 depicts the user interface for browsing assignments. The quiz interface is similar in that each student must be individually selected and viewed. This interface was designed so then interactive viewing and grading could take place. To eliminate the breadth of server requests, some teaching assistants at Drexel prefer to use Labrador to download quiz content, and mark grades in a spreadsheet program (ie. Microsoft Excel) for batch upload.

To ameliorate the workload issues, and simplify the handling of section data, we decided to create a software tool, Labrador, which could interact with WebCT and provide the following features:

1. Quick download of assignments and quizzes with minimal clicking.
2. Ability to pick up 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.

5. Ability to deposit submissions on a remote computer.

3. Labrador

Labrador is an online course management supplement developed for use with WebCT that enables users to easily interact with the student files in a single section even when a common class site is used. It is an extensible and modular interface between WebCT and other systems. Written in Perl, it runs on all of today's popular operating systems.

Labrador operates within the limitations of the designer and TA access privileges. Thus, no special privileges are needed to operate the tool.

3.1 Perl

Perl is an interpreted language with powerful facilities for manipulating textual data that allows rapid development of complex programs [4]. Perl is already used within WebCT to conduct backend operations, including database interactions. Since our task consists of parsing WWW documents retrieved from a WebCT system, Perl was a natural choice. Furthermore, by taking advantage of the huge number of Perl modules in circulation on the Internet, we did not have to spend a lot of time "reinventing the wheel".

Perl has been ported to almost every platform including Windows, Macintosh, and Unix operating systems. Labrador has been designed to take advantage of its cross-platform capabilities.

3.2 User Interface

We provide a variety of methods for interfacing with Labrador to satisfy the varying preferences of different users. Currently, there are three different methods with a fourth under design:

1. *Command-line*. This method involves typing all of the information needed to retrieve the proper files from WebCT at the command-line prompt, using various options.

2. *Interactive.* If the needed information is not supplied at the command-line, Labrador will prompt the user for it. This method is often used by Windows users: Double-clicking on the Labrador icon is equivalent to running the program with no command-line options and results in a series of prompts.

3. *Configuration File*. A user can include a configuration file which defines several variables that would normally be expected at the command-line. This file can be generated any time the program is run, and then serves as the default for future uses of the program unless over-ridden with a command-line option.

4. *GUI.* The authors are currently planning to build a Graphical User Interface using Java Swing, which should also work on all popular platforms.

An example download, using the command-line and interactive interfaces, is shown in Figures 2 and 3, respectively. Below we discuss the components of Labrador in detail. The components are decoupled such that they can be invoked independently, or in sequence.

3.3 Bulk Downloading of Assignments and Quizzes

The underlying technology behind this component is essentially a Web-crawler. It simulates the Web browsing that a staff member would normally undertake to download only the submissions for students of interest. In Section 2 we discussed some inefficiencies with the normal downloading process and the labor it involves. By automating the series of clicks required for the standard downloading process, we can considerably diminish the time and effort to complete the task. Another advantage of using Labrador for this process is that the user does not have to interact with a Web browser, and downloads can be done remotely. The bulk downloading process for assignments has been addressed in WebCT version 3.7, but this functionality is still needed for quizzes.

This is the only component of Labrador that interacts directly with the WebCT system. It downloads student-submitted files to a directory on the Labrador user's computer. All other components of Labrador use the downloaded files in this directory.

In order to download assignments in bulk from WebCT, the user must first provide access authorization to Labrador. Depending on the chosen interface method discussed in Section 3.2, the user must provide a username, password, course number, and the name or identification number of the assignment or quiz to be downloaded. The user may optionally provide a list of desired students, and Labrador will selectively download only the submissions of those students. This list can be easily saved and re-used for future submission retrievals.

3.4 Post-Processing Student Data

After a file is downloaded it may undergo some filtering to facilitate subsequent steps of the grading process.

3.4.1 Decompression and Archive Extraction

Labrador can optionally decode, de-archive, and decompress work, which was encoded by either the student or WebCT. Other systems, such as source code plagiarism detection systems, will expect documents to be in text form. The original files may be in a combination of formats (for example, both gzipped and tarred), and Labrador will properly decipher them. Thus, subsequent components of Labrador do not have to perform this filtering.

Some of the types of files and formats submitted by students include:

1. `tar` format, used to archive directories of files into a single file.
2. `gz` format, used to compress files sent over networks, to reduce the bandwidth.
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.

3.4.2 Section Sorting

The first prototypes of Labrador simply downloaded all student submissions for a particular assignment or quiz. This wasted server bandwidth as each TA downloaded unnecessary assignments. Now Labrador will sort assignments and quizzes by creating a directory for each

section with subdirectories for each student in their respective section. A depiction of the results is shown in Figure 4.

Since WebCT does not provide an easy process for representing sections, Labrador needs the user to supply this information. This can be done in one of two ways:

1. Creating a "Section" column in the grade database on WebCT and populating that column with each students' section.
2. Supplying a comma-separated (.csv) file, containing *"username, section"* pairs, as input to Labrador.

At many colleges and universities, it is possible to automatically generate these files for large courses using the central student information system and exporting to Microsoft Excel, which can in turn be used to generate the .csv file.

3.4.3 Portable Document Format (PDF) files for Electronic Pen-based Markup

The most recent component added to Labrador is the ability to transform source code from submitted student assignments into Adobe's Portable Document Format (PDF) files. An advantage of PDF files is that they can be viewed with free software available on almost every platform.

Using Adobe Distiller 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.

3.5 Interfacing with Heterogeneous Systems

There is a need for systems such as WebCT to interoperate 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. In this section we explain how Labrador is used to interface with two separate systems.

Plagiarism is a common problem in large classes. To address this, there are several free and commercially available programs and services which perform batch similarity measures on a collection of documents [1,3,5]. In our domain, all submitted assignments are computer programs so we require a somewhat specialized system.

Labrador can quickly download and package all student work in a hierarchical structure on the user's local file system. This structure can then be integrated with a plagiarism detection system for either computer programs or written assignments. 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 the discovery of plagiarism.

It would be desirable to have WebCT provide 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 eliminate the long-term plagiarism problem for frequently offered courses.

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. A screenshot of the browsable Web interface from Moss, with corresponding similarity values, is given in Figure 5. 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 has recently come to our attention. In experimental runs, it has been more effective in detecting plagiarism in computer programs. The programming languages supported by this system include C, C++, Scheme, and Java. It computes a similarity value among two documents, or groups of documents if applicable. A screenshot of JPlag's Web interface, depicting similarity measures, is shown in Figure 6.

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 can also be easily utilized in liberal arts courses to detect plagiarism in written documents, reports, etc. Labrador quickly and easily provides the middle-ware interface between WebCT and JPlag.

4. Discussion

We have found significant advantages to maintaining a WebCT-based course implementation. 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, and fewer student papers misplaced. 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 unsuitable to offer to users who would find such functionality confusing or irrelevant to their disciplines.

We have focused on particular tools 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 WebCT, they were more open to considering its benefits and making use of its strengths in course management.

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

5. 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.

References

[1] Alex Aiken. Moss: A system for detecting software plagiarism (unpublished), http://www.cs.berkeley.edu/~aiken/moss.html.

[2] Jeffrey L Popyack, Bruce Char, Paul Zoski, Nira Herrmann, and Christopher D. Cera. Managing course management systems. In *Proceedings of the thirty-third SIGCSE technical symposium on Computer Science Education, Birds-of-a-Feather Sessions*, page 423, 2002.

[3] 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.

[4] Larry Wall, Tom Chrisiansen, Jon Orwant. *Programming with Perl*. O'Reilly & Associates, 3rd edition, 2000.

[5] 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.

Submissions: Asst. 1

Page: All | Previous page | Next page | Select all | Select none

	User ID	Name	Grade Out of 150	Date	Status
☐	student1	Student #1	146	Apr 3, 2002 16:11	Graded
☐	student2	Student #2	112	Apr 4, 2002 00:00	Graded
☐	student3	Student #3	120	Apr 3, 2002 23:32	Graded
☐	student4	Student #4	129	Apr 3, 2002 21:43	Graded
☐	student5	Student #5			Not Submitted
☐	student6	Student #6	150	Apr 3, 2002 18:30	Graded
☐	student7	Student #7		Apr 5, 2002 13:44	Not Graded
☐	student8	Student #8		Apr 5, 2002 00:56	Not Graded

Figure 1: A screenshot of WebCT's assignment browser

```
$ perl labrador.perl -u rnl22 -c CS172BB -b 19283 -a -e -p -w -f list.txt
This program is in C++ mode.

checking -->student1<-- ....
    has submitted -->Assignment 4.zip<--
checking -->student2<-- ....
checking -->student3<-- ....
    has submitted -->writtenproblems.doc<--
    has submitted -->LargeInt.cpp<--
    has submitted -->LargeInt.h<--
    has submitted -->mainprog.cpp<--
    has submitted -->test.cpp<--
    has submitted -->External_documentation.doc<--
checking -->student4<-- ....
```

Figure 2: A screenshot of the Labrador command-line interface

```
$ perl labrador.perl
This program is in C++ mode.
The configuration file was not found.  Would you like to create one now?
(y/n) y
Your WebCT username: rnl22
WebCT name of the course (e.g.: CS172BB): CS172BB
File of list of usernames: list.txt
Would you like verbose printing (y/n)? y

----------Config File Written----------

Would you like to get (q)uizzes or (a)ssignments?a
Would you like post processing (y/n)?y
Would you like to generate source code PDFs (y/n)?y
Password:
[1]     Asst. 6
[2]     Asst. 1
[3]     Asst. 2
[4]     Asst. 3
[5]     Asst. 4
[6]     Asst. 5

--> 5

...

checking -->student1<-- ....
    has submitted -->Assignment 4.zip<--
checking -->student2<-- ....
checking -->student3<-- ....
    has submitted -->writtenproblems.doc<--
    has submitted -->LargeInt.cpp<--
    has submitted -->LargeInt.h<--
    has submitted -->mainprog.cpp<--
    has submitted -->test.cpp<--
    has submitted -->External_documentation.doc<--
checking -->student4<-- ....
```

Figure 3: A screenshot of Labrador prompting the user for information

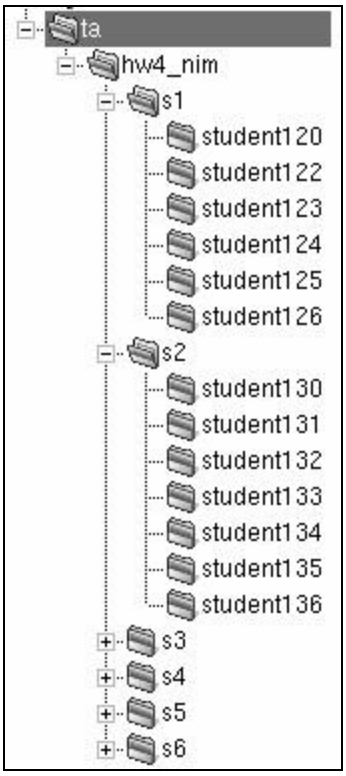

Figure 4: A screenshot of a grader's file system after sorting by sections

student75 (74%)	student146 (96%)	438	136
student68 (43%)	student52 (21%)	370	101
student15 (55%)	student147 (56%)	344	146
student44 (87%)	student20 (51%)	295	143
student66 (57%)	student140 (49%)	264	116
student35 (43%)	student107 (40%)	232	103
student92 (33%)	student24 (33%)	141	47
student67 (12%)	student35 (26%)	140	37
student50 (8%)	student131 (13%)	84	34
student53 (11%)	student17 (10%)	80	28
student22 (24%)	student146 (17%)	80	21
student141 (20%)	student119 (9%)	80	15

Figure 5: Moss' similarity browser

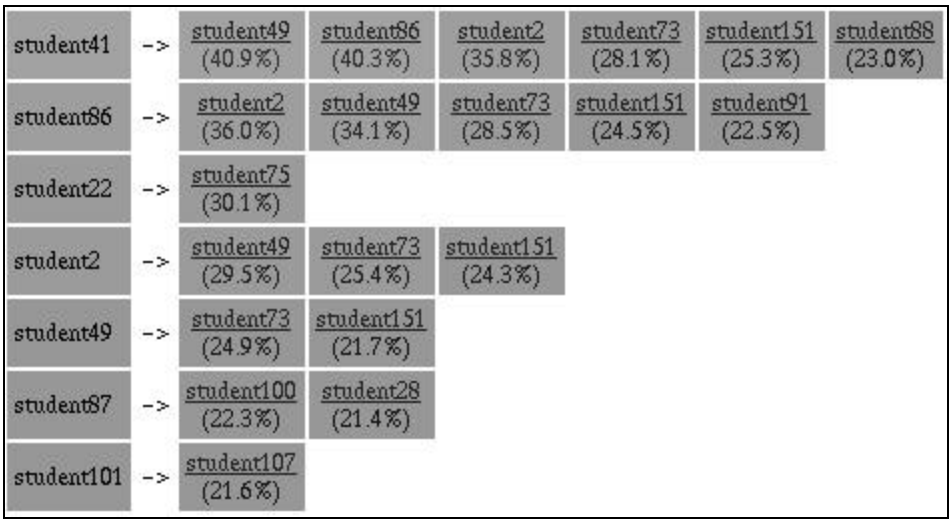

Figure 6: JPlag's similarity browser