Redesigning Introductory Computer Programming Using Multi-level Online Modules for a Mixed Audience

**Nira Herrmann, Jeffrey L. Popyack, Bruce Char, Paul Zoski,
Chrisopher D. Cera, Robert N. Lass, Aparna Nanjappa
Departments of Mathematics and Computer Science
Drexel University
Philadelphia, PA 19104**

Nira.Herrmann@drexel.edu, {jpopyack,bchar,pzoski,cera,urlass, uananjap}@mcs.drexel.edu

Abstract

We report here on an extensive redesign and unification of the Introductory Computer Programming sequences offered to computer science, computer engineering, information science and digital media majors. The redesign is intended to improve student learning while reducing costs. The approach makes use of substantial Web-based course material and course management tools, including multi-level online modules that individualize instruction and enable students to self-schedule learning each week. Each module covers a particular aspect of computer programming at different levels of knowledge. Students are assigned work and reading from the module at a level appropriate to the objectives of the long-term goals of their major. This allows students in different majors to acquire the appropriate skill level for each technique and concept. Peer mentors and teaching assistants provide assistance online or in person. In the future, we plan to expand the self-scheduling aspect of the course to allow students to enter the course at different modules, depending on their previous knowledge.

1 Background

The redesign was prompted by problems faced by the department, including (1) a computer science faculty whose modest growth—from 9.5 in 1996 to 13 in 2001—had not kept pace with enrollment growth—from 304 undergraduate computer science majors and 282 undergraduate information systems majors in 1996 to 932 computer science majors and 843 information systems majors in 2001; (2) incoming students whose widely divergent computing experience and skills could not be properly accommodated by the large-lecture format, and (3) a rate of failures, withdrawals, and D grades (DWF) from 25-50 percent, depending on the term and audience for the course.

The goal of the redesign is to provide students with an enhanced, individualized learning experience that reduces the number of DWF grades without overburdening a limited faculty. The solution involves a major change in the method of course delivery, including increased use of online materials, and extensive use of electronic course management tools, facilitated by the availability of a campus-wide wireless network and the requirement that all undergraduates have their own computer.

2 Syllabus and Modules

2.1 Traditional versus Redesigned Courses

The traditional computer programming courses met 3 hours per week, with 2 hours devoted to large (100-150 students) lectures given by faculty and 1 hour to closed computer laboratory sessions (25 students each) supervised by graduate and undergraduate teaching assistants (Figure 1).

Traditional Course	Redesigned Course

• 2 lecture hours + 1 lab hour per week
• Individual assignments only
• Standard lab
• Some online material (instructor created)
	• 1 lecture hour + 2 lab hours per week
	• Individual & group projects
	• Group-work lab
	• Substantial online support, plus
	  ○ Chat & discussion
	  ○ Online submission of assignments
	  ○ Online return of graded assignments

Figure 1: Traditional versus Redesigned Course Structure

The redesigned course consists of 1 hour of lecture per week and 2 hours of laboratory activity (Figure 1). The lecture hour is used primarily to make sure students understand their assignment, know where to get the course materials, and have the opportunity to ask questions about the new topic. Some large group activities are also undertaken (such as "pair and share" [5]) to increase

student understanding of the material. Pedagogical "lecture" material is available online as a series of slides, some with voice-overs, prepared by the instructors.

With the quarter system, classes meet for 10 weeks, and typically one week's worth of class time is devoted to midterm and final examinations: reviewing course material before the exam and going over the exam solutions after the exam. We determined that there would be 9 modules for the redesigned course, with 3 modules comprising 1 credit of instruction so we can more accurately place entering students and offer "partial credit" in the future in the form of one credit for completing three successive modules. The syllabus, presented in Figure 2, covers the CS 1 curriculum for C++:

Week	Topics Covered
1	Course Introduction
2	**Module 1** -- Introduction to C++ First Program, Style, Comments, Variables, Simple I/O
3	**Module 2** -- Numeric Types Basic Arithmetic, Integer Division, cmath library
4	**Module 3** -- C++ Strings string library, char indexing, string methods
5	**Module 4** -- Using Objects Introduction to OOP, Classes, File I/O
6	**Module 5** -- Conditionals if, if ... else, if ... else if conditionals
7	**Module 6** -- Advanced Conditionals and, or, not, nested conditionals
8	**Module 7** -- Introduction to Functions Prototypes, functions, scope, definition, pass by value
9	**Module 8** -- Advanced Functions Pass by reference, const parameters, side effects, simple recursion
10	**Module 9** -- Loops while, do ... while, for loops, nested loops, sentinel controlled loops, EOF controlled loops, Comparing Floating Point Numbers
11	Final Examination

Figure 2: Syllabus for CS 1

2.2 Levels of Mastery

The redesigned course modules will cover the material at different depths of technical content, determined by the needs of the students' majors with respect to levels of subject mastery delineated in Bloom's Taxonomy. [2]

Bloom's Taxonomy is organized in the following way:

- *First level*: students know the terminology of a subject and specific facts about it.
- *Second level*: students gain increased comprehension of the material and are able to explain the material and interpret what they have learned.
- *Third level*: students can apply their knowledge in new situations to solve relatively simple problems.
- *Fourth level*: students can perform deeper analyses of problems to discover component parts and interactions.
- *Fifth level*: students have the ability to apply prior knowledge in original ways to produce things that are new and different, and evaluate the methods used.

Level three is the minimum level of mastery all students in this introductory programming sequence should attain, regardless of major. In a professional setting, students at this level will be able to discuss programming with more technical personnel, understand how programming is used to solve problems, understand the solutions that are found, and relate these to similar problems in other circumstances. This level will be suitable for students who plan to work in technical areas, but not necessarily produce highly technical work themselves.

Computer science and computer engineering students need to reach the fifth level of mastery since they will face highly technical problems that will need to be solved in original ways. Computer science students must develop a deeper knowledge of computing, particularly the top level of Bloom's Taxonomy–the ability to judge the methods used–since many will face complex problems that may not have a single well-defined solution.

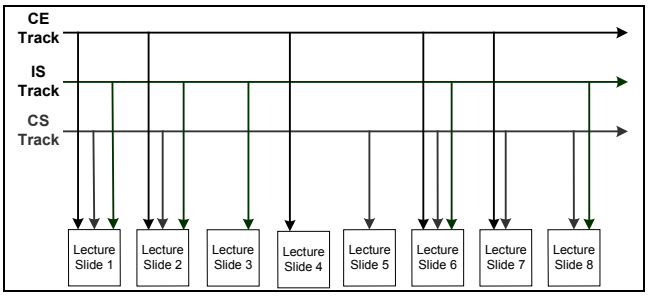


Figure 3: Schematic for Multi-Level Modules

The modules are being designed with material at each of the five levels of mastery and with pathways through the material suitable for each major (Figure 3). Students who have difficulty with the higher levels will be able to change majors and still get course credit without having to drop the course and repeat modules already mastered. This methodology is designed to address a significant resource problem: many students enroll in computer science without understanding the nature of the work. Once in the course, they may find other computing majors more appealing. The redesigned multi-level course will enable them to change majors without losing the work they had invested in a programming course for their previous major.

3 Dedicated Laboratory for Small Group Work

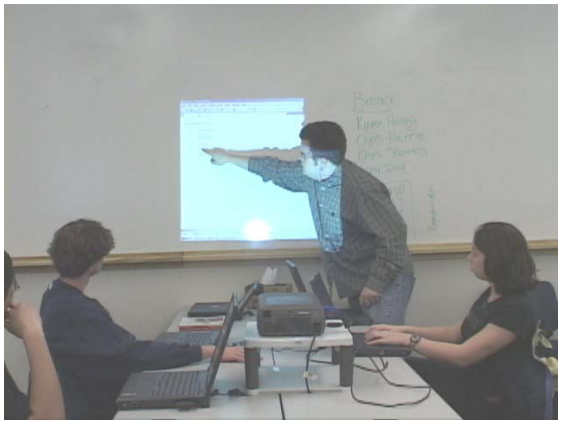
A dedicated computer laboratory has been constructed for the lab sections of the redesigned course, with five clusters where students can work in groups of five. Each cluster has five wireless-networked laptop computers and a projector that can be switched from one computer to another (Figure 4).

Each group can project its shared work onto the white

board "wallpaper" that covers all of the walls. The white board wallpaper allows students to annotate the projected screen image or write notes to the side as they work through the assignment.

Group assignments are downloaded from a central server using the wireless network. Assignments include a series of questions for students to answer that require coding as well as a deeper understanding of the language.

Figure 4: Group-work Cluster in Computer Laboratory



Intermediate checkpoints require students to get an instructor or teaching assistant to examine and grade the work completed so far before students continue to the next stage of the assignment.

4 Automating Course Management

Managing the paperwork for these large classes was a major problem as enrollments grew. Students would submit assignments on paper and a diskette with their programs. Grading required reorganizing assignments, quizzes and examinations turned in during lectures so each teaching assistant received the work for his/her roster of students. When students claimed that their assignments or quizzes were lost, there was no reliable mechanism for ascertaining that the work had ever been turned in.

Adopting course management software (CMS), in our case, WebCT, helped alleviate some of these problems. Electronic submission of assignments and online quizzes and examinations improved the flow of student work. However, problems remained in processing the large numbers of assignments and in sorting student work by laboratory section or lecture group as needed for grading purposes. CMS technology is new and evolving so some necessary features are still not implemented elegantly [6]. The structure of student programming assignments, for example, necessitates inspection of many files (possibly in programming language specific file browsers), as well as compilation and execution of programs. A typical CMS is not designed to handle this level of specialization.

We have developed several software tools, described below, to assist in further automating the processes of handling student assignments and quizzes online.

4.1 Labrador

Labrador [3] is a client-side WebCT supplement for retrieving submitted lab assignments, quizzes, and other student work; unpacking it, when needed, from several compression or archival formats; distributing it to the appropriate grader; and submitting it for further processing. Labrador is implemented in Perl and is compatible with several platforms (Windows, Mac, Linux and other Unix variants). The software interacts with WebCT to perform functions available to users with either TA or Designer access to WebCT.

```
$ 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. 1
[2]    Asst. 2
[3]    Asst. 3
[4]    Asst. 4

--> 4

...

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 5: Screenshot of Labrador in Interactive Mode

Users can specify Labrador commands using several modes of interaction, individually or in combination (Figure 5):

1. *Command-line* mode allows the user to invoke Labrador by a single DOS or shell command using command options.
2. *Interactive* mode is invoked when information is not supplied at the command-line, with Labrador prompting the user for the needed information.
3. *Configuration File* mode allows the user to define Labrador options in a configuration file which is read in lieu of having the user supply each command individually.
4. *GUI* mode is currently under development.

Some of the original functionality of Labrador has been subsumed by newer releases of WebCT, but Labrador is evolving in parallel, acquiring useful new features as our needs change. One of the most important features is the ability of Labrador to configure student files for subsequent

processing, either by external software packages (e.g., computer language or plagiarism detection software) or by the graders using pen-based mark-up.

4.2 Electronic Pen-based Mark-up

In the traditional course, students submitted assignments on paper; graders would read through the code or homework problems, marking errors as they found them, and inserting comments, advice, and corrections by hand. One current drawback to using CMS's is the mechanism for providing feedback to students on electronically submitted assignments. WebCT provides a text box where the grader can type in summary comments about the assignment along with the grade. This is time-consuming and not as communicative as freehand markup of papers to show exactly where errors have occurred.

We have developed a technique for emulating "freehand" feedback on electronically submitted work: Labrador converts students' program files into Portable Document Format (PDF) and graders then use a pen tablet to mark each assignment as if it were on paper (Figure 6, done on a Sony Vaio LX - 920, a desktop system with a Pen Tablet). The latest release of WebCT, version 3.8, allows return of PDF files to each student, facilitating the routine use of this type of feedback.

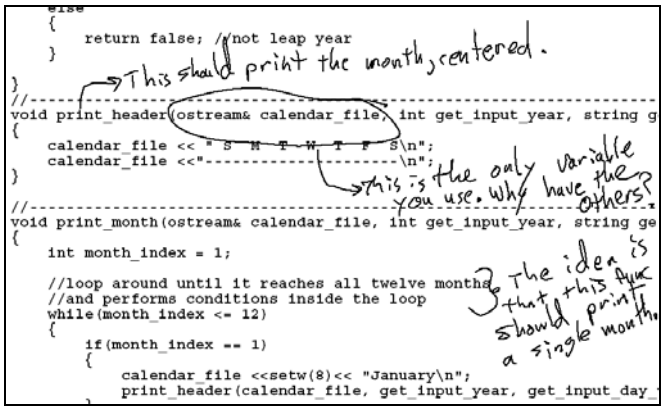


Figure 6: Pen-based Mark-up of Student Code

The primary drawback to using pen-based mark-up of assignments is the expense of obtaining a sufficient number of pen-based tablets for large courses with multiple graders. However, pen-based tablets are becoming more ubiquitous and software to support their use is moving into the mainstream (e.g., [4]) so this technology should be within reach of academic budgets shortly.

4.3 Quiz Question Database

Over 300 students may take the introductory programming courses in a single term, requiring several parallel lecture sections and numerous laboratory sections. Providing weekly quizzes for each laboratory section that differ enough to prevent cheating yet provide a consistent level of difficulty is a challenge.

Our approach has been to develop a database of questions for each quiz, grouped by topic and level of difficulty. Different quizzes are generated for each laboratory section using random selection according to pre-specified decisions by the faculty on the make-up of the quiz with respect to topics covered and level of difficulty. Despite the limited size of the database to date, students taking the quizzes later in the week perform similarly to students taking the quizzes early in the week, indicating that widespread cheating is not an issue.

We are working on expanding the database to increase the number of questions and plan to explore the use of digital library techniques to tag each question with respect to its difficulty and the level of knowledge required to answer it. This is a more difficult problem than it seems on the surface, particularly with respect to determining the level of knowledge needed to solve each question, which requires a learning theory analysis for computer programming.

4.4 Individualized Feedback to Students

We are developing an automated system to provide individualized feedback to students, particularly those whose submitted work indicates they are not mastering the material. Early intervention gives students a chance to improve their performance or decide to drop the course. The automated system will use email to contact students and inform them that there performance is substandard and they are in danger of failing, provide information on what they need to do to pass, and invite them to meet with the instructors and teaching assistants to discuss their status. This approach has worked well in other courses where it is handled manually, and we would like to incorporate a similar, automated approach.

5 Plagiarism Detection

Plagiarism has become a common problem. Plagiarism detection systems (PDS) such as Moss [1] or JPlag [7] have been developed that apply sophisticated techniques to detect similarities in source code structure. Both systems are designed to handle C++ programs, as well as other languages. JPlag handles general English text as well. To use such systems, one typically prepares a zip file of the class' source code files, and transmits it to the PDS Web site where it is analyzed. Labrador automates the process of extracting the student files from WebCT and placing them in the form needed by the PDS.

6 Assessment

The redesign involved considerably more work for faculty than the traditional course: learning the intricacies of the CMS, developing supporting software, reworking course materials into modules, and adapting to a different style of course delivery. However, having all student work online made it easier to look over, know what was turned in and

what was not, and assess students' mastery of it. In addition, the redesign has resulted in a shift in faculty activity from delivering content in a lecture format to interacting with students in both the lecture and lab. Since content is handled online, we have been able to experiment with large group activities even in the lecture sessions. Faculty perceived that students were more engaged and alert during the classes than in typical large-lecture situations.

Discussion and chat groups within WebCT provided good opportunities for mentoring. Students could take the time to think about their questions before submitting them, knowing that they could get an answer at any time. However, the time spent answering email has grown enormously: students did not come to faculty offices. Instead they would find a nearby computer and send email, most of which had to be answered individually. This is an area that needs further work, possibly by automating responses, automatically creating an FAQ page, or finding other mechanisms for handling the volume.

	Lecture Section			
Grade	**Trad 1**	**Trad 2**	**Redesign**	**Total**
A	12%	13%	24%	14%
B	23%	24%	26%	24%
C	14%	17%	12%	15%
D	17%	15%	12%	15%
F	23%	18%	24%	20%
W	11%	15%	3%	13%
Total	**100%**	**100%**	**100%**	**100%**
# Students	65	131	34	232

Figure 7: Final Grades for Students in the Traditional and the Redesigned Courses

The redesign was first tried on an experimental group taught simultaneously with two traditional lecture sections. Preliminary results show that the grades of the students in the redesigned course (Redesign in Figure 7) were generally higher than those in the two traditional lecture sections (Trad 1, Trad 2 in Figure 7).

Both faculty and students expressed concern that the one hour large group/lecture session per week did not provide enough time for faculty-student interactions. Students requested that the number of in-class hours be increased to provide more face-to-face time with the faculty. This is an issue we need to address, possibly by changing the nature of the lecture session or adding other activities.

7 Future Work

We plan to continue developing course materials and software tools to improve course delivery and student learning and to address student and faculty concerns about the need for additional interactions with each other: we need to explore whether faculty and students really need more formal lecture time to interact or are simply used to them and uncomfortable with change. We also plan to extend this approach to other courses in the programming sequence and in other areas, particularly in mathematics where we have close links and where the demand for courses is equally high.

Acknowledgements

This work is supported by the Pew Center for Academic Transformation under The Pew Grant Program in Course Redesign, by the National Science Foundation Division of Undergraduate Education under award number DUE-0089009, and by the Ramsey McCluskey Family Foundation. 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] Aiken, Alex. Moss: A system for detecting software plagiarism, Online, Internet. Available WWW: http://www.cs.berkeley.edu/~aiken/moss.html.

[2] Bloom, B.S. (Ed.) *Taxonomy of Educational Objectives: The Classification of Educational Goals: Handbook I, Cognitive Domain*. (1956) New York; Toronto: Longmans, Green.

[3] Cera C, Lass R, Char B, Popyack J, Herrmann N, Zoski P. Labrador: A Tool for Automated Grading Support in Multi-Section Courses. *Proceedings, WebCT 2002 IMPACT: 4th Annual WebCT Users Conference*, Boston, Massachusetts, July 24-26, 2002

[4] Microsoft Corporation. Microsoft Showcases Tablet PC, First Wave of Products Targeted At Increasing Information Worker Productivity at TECHXNY. Online, Internet. [June 25, 2002]. Available WWW: http://www.microsoft.com/presspass/press/2002/Jun02/06-25TechXUmbrellaPR.asp

[5] McDowell C, Werner L, Bullock H, and Fernal J. The effects of pair-programming on performance in an introductory programming course. *Proceedings of the 33rd SIGCSE technical symposium on Computer science education* . ACM Press, 2002, pp. 38 – 42.

[6] Popyack JL, Char B, Zoski P, Herrmann N, and Cera C. Managing Course Management Systems, Birds-of-a-Feather Session, *The Thirty-Third SIGCSE Technical Symposium on Computer Science Education*, February 27-March 3, 2002, 423.

[7] Prechelt L, Malpohl G, and Philippsen M. JPlag: Finding plagiarisms among a set of programs. *Technical Report 2000-1, Fakultat fur Informatik, Universitat Karlsruhe,Germany*, (March 2000). Available WWW: http://www.ipd.uka.de:2222/