Retaining Majors Through the Introductory Sequence

Elise H. Turner, Erik Albert, Roy M. Turner, and Laurence Latour
Dept. of Computer Science, University of Maine
Orono, ME, USA 04469*

Abstract

Retention is an important issue for Computer Science Departments. In many cases students leave the major due to frustrations with programming in the complex languages often used in CS1 and CS2 or because they do not understand that computer science is much more than programming. We have redesigned our introductory sequence to include a rigorous, non-programming introduction to the field and a CS1 course which uses Scheme so that students can focus on the principles of programming instead of the complexities of a particular language. In the first year that we have required these courses in our major, we have had positive results. In this paper, we describe what we have learned through discussions with students and student surveys.

Categories amd Subject Descriptors

[K.3.2][Computer and Information Science Education] [Computer science education]

General Terms

Design

Keywords

Retention, CS1/2, Non-Programming Introductory Computer Science

1. INTRODUCTION

In Academic Year 2005-06 at the University of Maine, we have begun to require a new sequence of introductory courses designed to emphasize conceptual information about computer science and programming over the complexities of a particular language. We expected these courses to better prepare students for upper-level courses in the major, specifically by teaching them to focus on conceptual issues instead of simply completing specific software projects. We also hoped that these courses would help students gain an understanding of what computer science is so that they could decide early if the major was right for them. Although we expected to lose some students in the first year of the program, we expected the new sequence to help us retain talented students, especially women, who were interested in computer science. Since requiring the new curriculum, we have seen a significant increase in the percentage of computer science students who successfully complete the first year and plan to continue in the program. Although we have a fairly small program, we believe that these preliminary results are worth discussing in this time of decreasing enrollments.

In this paper, we discuss why we believe our introductory sequence helps us to retain students. We begin in Section 2 with our reasons for changing the curriculum and a brief description of the courses. In Section 3, we present the retention data for our courses and information gained from students. We conclude in Section 4 with a summary of our work.

*The first author is the contact author and can be reached at eht@umcs.maine.edu

2. DESCRIPTION OF COURSES

Before the new curriculum was required, Computer Science majors were required to take standard, programming intensive CS1 and CS2 courses using C++. Introduction to Computer Science 1 (Old-CS1) was the CS1 course, and Introduction to Computer Science 2 (Old-CS2) was the CS2 course. Old-CS1 was required for our majors and some majors in the engineering school. It could also be used to fulfill a requirment in several different programs. The course was difficult to teach because of the disparity in the skill level of students entering the class. On the one hand, a significant number of students came to the class with significant programming experience, often in C++. On the other, large numbers of students had little or no experience with computers other than using the World-Wide Web, playing video games, and using applications such as word processors and spreadsheets. This was largely due to the disparity in opportunities for studying computing in Maine high schools. So many students, often women, who were talented but inexperienced were put at a severe disadvantage. It was also beginning to appear to faculty advisors that talented problem solvers who were capable programmers were leaving the major because they found programming tedious and had not been shown that there was more to computer science. Through lengthy discussion with our advisees, we came to believe that both talented women and talented men most often left the major for this reason. It was also the case that students who were most interested only in programming were frustrated by upper-level courses that required them to understand the theoretical and conceptual foundations of computer science.

In an effort to help students to choose an appropriate major, and to better prepare them for upper-level courses, we began to teach *Foundations of Computer Science* (CS-Intro). The course was first taught in Fall 2002. At that time it was not required. It has been taught each Fall since then, becoming a requirement in the new curriculum in Fall 2005. When the course was not required, a disproportionate number of students took it in place of our remedial programming course. These students were placed in this course because they had no programming experience and no substantial experience with computers of any kind, and they were not prepared to take Calculus I. As a result, it is difficult to compare information about retention during those early years with information about retention after the course was required. However, the course had a lower drop-out rate than the first semester programming course. Most of the people who dropped CS-Intro were those who reported on beginning of the semester surveys that they were interested in only programming, technical certification, or some other non-computer science, computer-related area. Most women who had taken CS-Intro expected to remain in computer science or other STEM (Science, Technology, Engineering and Mathematics) majors at the end of the course.[1] Although the numbers were small, our discussions with women who had taken CS-Intro [1] suggested that the course would help us to retain women in the major. In addition, discussions with all students and student surveys suggested that students had a better understanding of the discipline and became more committed to their major after taking CS-Intro.

However, there were still problems with retaining students through our previous Old-CS1 and Old-CS2 courses. Students who had done well in CS-Intro still suffered the same kinds of problems in Old-CS1 and Old-CS2 that had been seen at our institution and elsewhere. Average students who had not programmed often found the course overwhelming and could be intimidated by those with more experience [2, 3]. Talented students, even those who had taken CS-Intro, continued to drop the major because they found programming tedious and the Old-CS1 course lacking in intellectual challenge [4]. Our Curriculum Committee had already begun to discuss a major curriculum revision to strengthen our program, and so we tried also to address these concerns when developing our new introductory sequence.

This led us to several changes:

- CS-Intro became a required course so that students would have an early understanding of the discipline, including the types of problems computer scientists address and how they address them.

- Old-CS1 was replaced by *Introduction to Problem Solving using Computer Programming* (CS-Prog). The latter used Scheme as an introductory language so that students could focus on general programming concepts instead of the syntax of a particular language and to level the playing field for students with no previous programming experience.

- Old-CS2 was replaced with *Introduction to Object-oriented Programming and Design* (CS-OO) using Java. Lectures in the course focused on object-oriented programming and design, and Java was taught in recitations. This was done so that students would begin to build the skills needed to learn new languages quickly, and so they would begin to make generalizations about programming languages.

- Courses were named to reflect the focus on general principles to remind students that there was more at stake than simply learning a language. Old-CS1 and Old-CS2 were renamed to indicate that they were primarily programming courses, not general introductions to computer science.

- CS-Intro's enrollment was limited in size so that students and faculty would get to know each other, and so that students would feel comfortable asking questions about the course material or the major both in and out of class.

- Recitation sections were added to the classes. This allowed students to meet in small groups and to get more individual attention. It also allowed students to get to know the graduate teaching assistant so they would feel comfortable asking questions.

- Tips for studying and writing exams (e.g., how to write definitions, how to prepare for an exam) were given in class. And, in the first semester, in-class time was devoted to reviewing for the exams and discussing the exams when they were handed back.

In the remainder of this section, we discuss each of the new courses in more detail.

Foundations of Computer Science

This course is a rigorous, non-programming introduction to computer science, described in detail in [5]. The course is unlike most non-programming introductions because it is neither a high-level survey nor a pre-programming course. Instead, students study core areas of computer science by learning about specific techniques used to solve fundamental problems in the area. The areas currently covered in the course are: digital logic, organization and architecture, programming languages, operating systems, and networks. Students have a brief exposure to professional ethics in a networks topic on privacy. In each section, after a brief overview of the area, students study three to five topics in detail. Topics include Karnaugh maps in digital logic, Booth's algorithm in organization and architecture, Backus-Naur form in programming languages, semaphores in operating systems, and Hamming codes in networks.

Topics are presented as solutions to specific, important problems in the field. Most topics are covered at the same depth that would be seen in an upper-level introductory course in the area. For example, students learn why Karnaugh maps work, how to construct them, and how to use them to minimize circuits. Homework and exam questions

[1]Five out of seven expected to remain in (or, in one case, transfer to) the major at the end of the course. One transferred to Electrical Engineering Technology because she was more interested in hardware than software, and one was interested mostly in programming. One transferred into the Electrical and Computer Engineering Department.

as well as slides for a topic are often taken from the corresponding upper-level course. Because students work with the material at this depth, they are able to understand how computer scientists approach problems and to see that computer science is not just programming.

Although non-majors and more senior students have been successful in the course and have given it positive evaluations, CS-Intro is designed for first-year Computer Science majors. The course is structured to instill good study habits in the students and to help them to succeed [5]. In addition, we explicitly teach students good study habits. Class size is intentionally kept small so that the instructor and the students can get to know each other, and so that students feel more free to participate in class and to contact the instructor or teaching assistant outside of class.

Introduction to Problem Solving using Computer Programming

This course is very similar to a standard CS1 course. The course introduces students to programming, using Scheme, and to data structures and algorithms. We follow [6] in making the design and use of abstract data types an important part of the course.

The course, and its name, are meant to help students understand the role of programming in computer science. Students complete programming assignments as homework, culminating in a large project that is to be designed (and implemented, if time allows) at the end of the semester. Exams and quizzes cover mostly conceptual information related to programming, data structures and algorithms. Using this approach, we expect students to be able to generalize what they have learned in order later to learn new programming languages and for them to recognize programming's role as a necessary tool for computer science.

The course is required for our first-year students, for first-year students in the University of Maine's New Media program and for students in the Business School's Management and Information Sciences program. Consequently, class sizes are larger than they have been for CS-Intro. However, here again, we try to support their growth as students. All students are required to attend small recitation sections each week so that they will have an opportunity to get to know the teaching assistant. Students are also encouraged to go to the instructor's office hours with questions. As in CS-Intro, the instructor devotes some class time to good study habits, especially those related to programming classes.

Introduction to Object-oriented Programming and Design

CS-OO is a four-credit course that is both an introductory course in the Java language as well as an intensive introduction to the process of designing and implementing programs as systems of interrelated objects. Because of this additional object-oriented design component, it differs from the traditional introductory computer science (programming "in the small") course. Students find it a challenge not only to design and implement small programs, but also to decompose larger problems into modules that can then in turn be implemented independently as small programs. All projects in the course have elements of both small-scale and large-scale programming.

The prerequisite for this course, CS-Prog, uses the relatively syntax-free Scheme programming language in order to facilitate learning basic algorithmic problem solving principles. This allows CS-OO to put greater emphasis on large-scale programming concerns. Students initially find the relatively syntax-heavy Java language to be a challenge, but then, fairly quickly, realize the real problems lie in the process of separating a system properly into concerns and then implementing the resulting objects and object interfaces.

3. FEEDBACK FROM THE FIRST YEAR

In its first year, our new curriculum increased retention in our major as measured by the size of the sophomore class and by the number of students who successfully completed the required, year-long programming sequence. The data shown below include both first-year college students and transfer students who were enrolled in our first semester courses. The data also include students enrolled in both our Bachelor of Science (B.S.) and Bachelor of Arts (B.A.) programs. The programs are very similar in terms of Computer Science requirements, with the B.A. program requiring two fewer (three instead of five) Computer Science electives and less science and mathematics. Students currently are admitted into the B.S. program unless they request to be admitted into the B.A. program. However, students often move between the B.S. and B.A. programs, especially early in the program. So, distinctions between B.S. and B.A. students are mostly meaningless at this stage.

Twenty-five first-year majors entered the program in Fall 2005. In Fall 2006, the program had twenty-five sophomore majors. In only one of the previous five years, had there been as many or more majors in the sophomore class than in the preceeding first-year class. Figure 1 shows the number of students in the first-year and sophomore class enrolled in the major in the fall for each of these years. The number of first-year students entering our program dipped in Fall 2002 and dropped further in Fall 2004, inreasing slightly in the following two years. We believe the decrease in enrollments reflects the national trend, but it is unclear why enrollments increased so dramatically in Fall 2003. It is interesting to note, however, that while the number of first-year majors increased by 150% from Fall 2002 to Fall 2003, the number of sophomores in Fall 2003 increased by only 105% of the number of first-year students in Fall 2002.

Class	2001	2002	2003	2004	2005	2006
First Year	66	36	54	18	25	27
Sophomore	47	44	38	27	15	25
% Retained		67%	105%	50%	83%	100%

Figure 1: Computer science student enrollment and retention data. Numbers reported are for the fall of the year. "% Retained" is the number of sophomores as a percent of the previous year's number of first-year students.

Figure 1 shows the students officially enrolled in the major. This includes students who are waiting to transfer into another major, students who are on a leave of absence, or students who are taking courses abroad or elsewhere. This number also does not reflect the success of students in the major. However, because it does not track individual students, it allows us to recognize students who transferred to our major after taking our first semester course.

Figure 2 shows the percentage of students who have successfully completed the required programming sequence in one year. We consider a student to have successfully completed a course if he or she received the required grade of C or better. These numbers reflect the percentage of students who continued to pursue the major after taking the first semester course. These numbers include only computer science majors and undecided students. They do not include students who were in the first semester course in a different major and then transferred into computer science to continue the sequence. The figure also shows the students who successfully completed the first semester of the sequence. There were 24 computer science students entering the new curriculum in Fall 2005 and enrolled in CS-Intro/CS-Prog. There were 19 (79%) who enrolled in CS-OO in Spring 2006, with 15 (79%) who completed it, all with a grade of C or better.

It is more difficult to obtain comparable data for our old curriculum, partly because our data include students retaking the courses. In addition, since students often transfer into the major during their first or sophomore year, and because a student's class is listed in enrollment data based on credits earned, not time in or progress through the major, some students take first-year computer science courses even though they have sophomore, junior or senior standing. We can, however, look at sequences of Old-CS1 and Old-CS2 in sequential semesters.

	Curriculum	Contiguous Sequence
F04–S05	Old	6.94
S05–F05	Old	7.4
F05–S06	New	28.6

Figure 2: Percent of COS and undecided students successfully (grade $\geq$ C) completing, by starting semester, the first course of the old and new curricula as well as percent successfully completing the sequence in contiguous semesters.

We look at Old-CS1 and Old-CS2 for the past two years. Because these courses served other majors as well as our own, they were taught each semester. These courses had been taught for many semesters before those shown, and some students repeated Old-CS1 and/or Old-CS2 several times. We count as Old-CS1 students any student in the Computer Science major who enrolled in Old-CS1 during the semester being described who had not previously taken Old-CS1 within the past two years. We count as Old-CS2 students any student counted as an Old-CS1 major who took Old-CS2 the following semester. As with students in the new sequence, this does not include students who left the program but will return.

With such small numbers of students, student surveys are our best source of information about success in these courses regarding our objectives related to retention. Students completed these surveys during the last class period of the semester. They also completed University course evaluations on the same day. Students were given the option of including their names on the surveys so that we could correlate feedback with grades received in the courses and other information that we had gleaned from discussions with these students. Of the 25 students remaining in CS-OO at the end of the semester, nineteen completed surveys. Of these seventeen were Computer Science majors.

Most students expected to continue in the major after the first year. Few students were not successful in the course. Most who seriously considered dropping the major did so because of a lack of interest in computer science instead of a misunderstanding about the major. Of the seventeen Computer Science majors who completed surveys, only one had transferred out of the program and only four others reported considering dropping the major. One of these three did not successfully complete any of the required courses in the first year. The student who planned to transfer reported that in response to "The thing that keeps me in the computer science major is." The other students circled "do" in the statement "When I get really frustrated I (do/do not) think of dropping the major..." of the three that gave a reason, only one gave the reason of not being interested in building systems. Another said that he was not sure of his life goals. The final student's frustration stemmed from not feeling comfortable with Java, despite a high grade in CS-OO. However, through discussions with this student we know that she is not seriously considering dropping the major, and has returned as an enthusiastic Computer Science major in Fall 2006.

CS-Prog was meant to give students an introduction to the principles of programming so that they were prepared to handle the complexities of the language when programming in Java. This was because, from discussions with advisees, faculty believed that many students felt overwhelmed in Old-CS1 because of the language. This was particularly true of women, but was true of many men as well. These students often were frustrated enough by the complexities of the programming language that they dropped the major, even if they were doing well enough to successfully complete the course.

Although six students said syntax was the most difficult thing that they learned in CS-OO, only three of these students gave a syntax-related response when asked to fill-in the blank in "I get frustrated in class ________." Of the sixteen students[2] who responded, only seven filled in the blank with a response related to programming (including syntax); the other nine gave answers related to how the class was managed. This suggests that allowing students to understand programming before tackling complex languages helps students to overcome the frustrations of programming in these languages.

Students also appear to understand that CS-Prog was intended for this purpose. Of the eighteen people who responded to, "What do you think is the main purpose of [CS-Prog]?" fifteen indicated that it was to introduce students to general principles of programming, such as "programming basics" or "object-oriented perspective." Only three students mentioned the language specifically or the sytax of the language.

All of the Computer Science majors, except the one who transferred, gave reasons for staying in the major. Two students who had discussed their frustration with programming with their academic advisors cited CS-Intro as their reason for staying in the major. Eight students in the survey listed as their reason for staying factors not related to programming such as general interest in the subject, being intrigued

[2]Both majors and non-majors

by computer science, or the desire to take advanced courses. So, CS-Intro seems to play a role in helping students put the frustrations of programming in perspective.

It is important to note that of the three students who gave programming as their reason for entering the major, one was transferring to a department which better matched his interests. Hopefully, CS-Intro allowed this student to make an informed decision about his major. Since he was successful in CS-Intro, but did not do as well in that course as in CS-Prog or CS-OO, it is quite possible that he made a good decision. The other two students responded that they are remaining in the program because of programming.

We should see the story of the two students who remain in the major because of their interest in programming as a cautionary tale. Some students simply love to program and want to learn more about computers so they can be more capable programmers working on more interesting problems. The one student who identified himself on the survey did well enough in CS-Intro to understand what will be expected of him in upper-level courses in the major. However, if he did not take a programming course in addition to our non-programming course, he may not have remained interested in the major. We must be careful that we do not lose students who are most interested in programming, but who are capable problem solvers, in order to retain students who are most interested in problem solving.

Many studies indicate that women, disproportionately, have difficulty in courses like Old-CS1 and Old-CS2. From our discussions with women who have taken Intro-CS, it appears that most develop a great deal of enthusiasm for computer science. This seems to help them get through the frustrations of programming in languages like Java and C++. However, this also appears to be true of most of the men who take the course. In the future, we hope to look more closely to see if there are gender-based differences in students' reactions to the new curriculum or in retention rates.

4. CONCLUSIONS

After reading student surveys and talking with students, we believe that our new introductory sequence will help us retain students in the Computer Science major and help students determine if the major is right for them. We will continue keeping data about retention through the major, and add information that will allow us to differentiate continuing majors from transfer students. In the future, we will be interested in how our new curriculum affects recruiting students into the major, recruiting and retaining women in the major, the quality of students that are recruited and retained in the major, and the students' performance in upper-level courses.

5. ACKNOWLEDGEMENTS

Thanks to members of our Curriculum Committee who are not authors of the paper: Tom Byther, our chair, Phillip Dickens, Edward Ferguson, and Carol Roberts. Many thanks to Ellen Johndro for helping us gather enrollment data and for making all of our lives easier while this paper was being prepared. We also thank the anonymous reviewers for their helpful comments.

6. REFERENCES

[1] E. H. Turner, C. Emerton, R. Ray, and C. Logan. A rigorous introduction to computer science without programming: Three women's perspectives. Technical Report UMCS–TR–2004–2, Department of Computer Science, University of Maine, 5752 Neville Hall, Orono, ME 04469, 2004.

[2] L. J. Barker, K. Garvin-Doxas, and M. Jaskson. Defensive climate in the computer science classroom. In *The Proceedings of the Thirty-Third SIGSCE Technical Symposium on Computer Science Education*, pages 43–47, New York, NY, 2002. Association for Computing Machinery, Inc.

[3] B. C. Wilson and S. Shrock. Contributing to success in an introductory computer science course: A study of twelve factors. In *The Proceedings of the Thirty-Second SIGSCE Technical Symposium on Computer Science Education*, pages 184–188, New York, NY, 2001. Association for Computing Machinery, Inc.

[4] M. Barg, A. Fekete, T. Greening, O. Hollands, J. Kay, and J. H. Kingston. Problem-based learning for foundation computer science courses. *Computer Science Education*, 10(2):109–128, 2000.

[5] E. H. Turner and R. M. Turner. Teaching students to think like computer scientists. In *Proceedings of the Technical Symposium on Computer Science Education (SIGCSE'05)*, St. Louis, MO, February 24–27 2005. Association for Computing Machinery (ACM).

[6] H. Abelson, G. J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, Cambridge, MA, second edition, 1996.