

What Can Computer Science Learn from a Fine Arts Approach to Teaching?

Lecia J. Barker Kathy Garvin-Doxas
University of Colorado
Boulder, CO 80309
303-735-6004
{barkerl, garvindo}@Colorado.edu

Eric Roberts
Stanford University
Palo Alto, CA 94305
650-723-3642
eroberts@cs.stanford.edu

ABSTRACT

Two pedagogical techniques of IT programs are compared, a traditionally taught computer science (CS) major and an IT certificate program using a fine arts approach to pedagogy. The latter graduates a higher percentage of women than of males. Although the two programs are quite different in the nature of the material and what students are expected to learn, CS instructors can borrow from the certificate program in ways that could increase attraction to and retention of women in CS, especially by allowing students to hear each other articulate what they are learning; mentioning practical applications of theoretical principles; and requiring that students display their knowledge and solutions to their peers.

Categories and Subject Descriptors

K.3.2 [Computers and Education] Computer and Information Science Education

General Terms: Human Factors

Keywords: Gender issues; CS education research; Classroom management; Pedagogy

1 INTRODUCTION

The attrition of women in computer science (CS) in the U.S. is much higher than that of men [3, 4]. Several explanations are offered in the literature. Many women lose confidence in their ability to complete the tasks required for earning acceptable grades, despite evidence that they perform equally to men when lack of experience is factored out [5, 6]. This is partly because the individualized classroom climate of introductory courses makes it difficult for students to accurately gauge their abilities in comparison to their peers [1]. Margolis and Fisher reported that a loss in confidence often precipitated a student's decision to leave the CS major [11]. Other explanations include the low number of female faculty role models in computer science, poor student-mentor relationships caused by impersonal and non-collaborative computer science classes, classroom environments characterized by unchecked one-up-man-ship, and a "chilly" climate

This material is based upon work supported by the National Science Foundation under Grants 0090026 and 0082915.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCSE'05, February 23-27, 2005, St. Louis, Missouri, USA.
Copyright 2005 ACM 1-58113-997-7/05/0002...\$5.00.

where women experience difficulty entering peer study groups and face "pervasive stereotypes" that women cannot be technical or must use their feminine wiles to get good grades [1, 2, 4, 15]. Many women experience the double bind described by Seymour, in which being feminine or attractive is interpreted as being incompetent, while playing down one's femininity is interpreted as unnatural or unattractive [16]. A particularly large, multi-university study showed that leaving a major was related to teaching practices [17].

There is a relationship between attracting students to and retaining them in an established program. When students are considering a program of study, they ask other students, professors, family, and friends about their experiences or what they have heard. For example, when asked what computer science was about or why they had not entered a CS major, ITC students interviewed as part of this study perceived CS as narrow; CS students as focused on learning programming; the major as not permitting them to apply their skills to other fields; as antisocial; and as a major where they could not exercise creativity. This extreme view has some elements of truth, according to our observations and CS majors; thus when CS majors tell other students about their experiences, these are less likely to enter the CS major.

Many interventions have been proposed to increase retention of women in computer science. These include offering introductory or immigration courses with a broader view of the IT industry or which accommodate different abilities at the early stages of the program; increasing student collaboration (e.g., through pair programming); and including projects which are more meaningful to women [7, 12, 14, 15, 18]. Ensuring that visible role models are presented to students, making available faculty and student role models, and improving student-faculty and student-student relationships through mentoring programs are also recommended [2, 7, 10, 12]. A combination of strategies is advised [3].

In this paper, we offer a method of increasing retention by creating a classroom culture in which learning is a social or community process, not private and individualized. We conducted more than 600 hours of classroom observation in a multi-disciplinary information technology certificate program and a traditional computer science major. The certificate program's project courses use a fine arts approach where public critique and collaboration are routine and required; this program, though based in technology, both attracts and retains women at higher rates than does computer science. Below we present the background of the study, including a brief description of each curricular program and our research methods. We then describe two differences across these two classroom settings: knowledge sharing and assessment techniques. We conclude with some ideas on how CS instructors might adapt some of the pedagogical techniques of the certificate program to, in combination with other strategies, improve recruitment and retention of women.

2 BACKGROUND

2.1 University, Students, and Programs

The two programs observed were a traditional computer science (CS) major and a six-course, multimedia-oriented IT certificate program (ITC). We are not suggesting that these two programs of study are comparable; we suggest only that pedagogical techniques from the ITC's fine arts approach could be introduced into CS to improve learning and retention for all students, and in particular, women. Both curricular programs were observed in the same university, a public, doctoral-extensive university enrolling 24,000 undergraduates in 2003.

2.1.1 CS Major

The CS department, housed in a school of engineering, had ~700 majors in 2000 when the study was begun, but like most CS departments across the country, these numbers have decreased over the past few years. The CS major is consistent with Computing Curricula 2001's description of a "programming first" approach [9], in which early instruction concentrates on programming. Instruction tends to occur in a large lecture, with students independently learning skills in labs [1].

This CS program typically graduates fewer female undergraduate majors than the U.S. national average, at about 17 percent. Between 1994-2000, its attrition rate for women was 11.4%, higher than men at 10.5%; this was not a statistically significant difference. (Note, however, that this may be due to very low power, given the very small population of women on which to base the analysis.) The attrition rate for CS women at this institution is not as high as the national average (cf. [4]), but entry is lower. As Cohoon has demonstrated in her research, the problems of attrition and entry are not experienced uniformly across programs [4]; the cause of low graduation rates of women in CS can be low entry, high attrition, or a combination of these.

CS students interviewed as part of this study¹ reported many instances in which males and females are treated differently in class. As has been reported in other studies (e.g., [11]), women believed they were held to higher standards than were men, such that if they were called upon, they could not afford to answer incorrectly for fear it would reinforce a belief that women did not belong. Not surprisingly, about 2/3 of the men interviewed never felt discomfort asking questions in class, while only 1/3 of the women claimed they experienced no discomfort. Some women reported feeling isolated, but at the same time conspicuousness in class, and felt that they had to hide their femininity. One student said, "heaven forbid I talk about nail polish when I'm in that programming class."

2.1.2 ITC Program

In the ITC program, students acquire in-depth skill with high-end, software packages (e.g., Photoshop, Flash) as well as some HTML programming, with the goal of designing multimedia materials and interactive programs. Programming courses (e.g., C++ or Java) are optional. The certificate program, open to all undergraduates, requires that students take six courses; three are

¹ 141 (89 women, 52 men) students were interviewed in another part of this study, including 41 ITC students, 85 CS students, and 15 students who switched out of the CS major. The report is not yet published; acknowledgment to Liane Pedersen-Gallegos and Elaine Seymour will be made in the final version of the paper.

taught exclusively by ITC core faculty. The introductory course teaches basic HTML with course content centering on societal aspects of communication technologies; three are project courses with both individual and group assignments; and the remaining two courses focus on social or theoretical implications of technology from the viewpoint of any of several disciplines.

In ITC, student enrollment is slightly more than half women (52%), and 62% of those who have completed the certificate are women. Both ITC males and females complete the certificate at 92%. ITC students interviewed described a supportive and collaborative learning environment. Students interviewed did not perceive that women were treated differently from men in their ITC classes.

2.2 Research Method

Our data collection and analytical methods fall within ethnographic inquiry, the goal of which is to systematically document and understand interactions and their meanings within a particular social context, such as a classroom or curricular program. Using this approach, social scientists hope to articulate what is often implicit and unarticulated in social settings: both typical behavior and the shared norms and beliefs prescribing what is appropriate. Ethnographic research leads to in-depth knowledge of particular situations and should not be generalized, but used by analogy. When the reader believes their situation is similar in essential features, then ethnographic research can help to shine a light on the interactional details of their classroom.

During two sequential academic years, we observed seven CS courses, eight ITC courses, and six courses cross-listed between the two programs, totaling more than 600 hours of classroom observation (see Table 1). The results we present here are not isolated in particular classrooms, but typical features of learning and assessment in the two programs.

For each class, we systematically recorded: number of students attending, sex, and appearance; physical layout of classrooms and seating arrangements; and descriptions of teaching and interaction (student-student, student-instructor) and interactants (male/female; major if known). Observation notes were typed and the resulting text bound into 21 "books."

Table 1 Observation Details

Term	Program	Courses	Hours
Fall 2000	CS	2	46
	IT	3	50
Spring 2001	CS	2	70
	CS/IT*	2	38
Fall 2001	CS	2	103
	CS/IT*	2	48
	IT	3	95
Spring 2002	CS	1	21
	CS/IT*	2	60
	IT	2	73
Total		21	604

*Cross-listed in CS and ITC.

3 KEY DIFFERENCES OBSERVED

3.1 Knowledge Sharing

3.1.1 Knowledge Sharing in CS

In all the computer science courses we observed (except for the cross-listed courses), lecture and lab were separate. In lecture, students tended to sit and listen, looking forward at their professor. When a teacher stands at the front of the room, attention is focused on one person, object, or action, implying that

the learning or knowledge will be imparted by that person and the contributions of others is of lesser value. Teachers have the right to move around the class, while students have no such right without permission from the teacher [13]. Further, the lectern is a signal to students that one-way transmission of information will predominate. Spatial arrangements have additional meaning for participants in a classroom; for example, “in a non-circular configuration . . . in which there is one member at the apex of a triangle, . . . the member at the apex typically has the right (and obligation) of sustained speech” [19, p. 184].

The subject matter of lectures was often taught with general principles preceding reasons why a person might need to know the theory or method. In one class, when a student asked why they needed to know a particular algorithm, the professor replied that he was proving its efficiency. Thinking he had not been clear about his question, the student asked again. After probing the student to better understand the question, the professor apologized for not understanding the question and moved forward with the lecture. From the perspective of an outsider (the observer), it felt like an embarrassing moment, where people simply don't understand each other's perspectives. The professor's focus was theoretical (algorithmic efficiency); the student's focus was on specific application contexts where the algorithm might be used.

The professor was thinking (and teaching) in terms of general principles that could be applied to specific contexts, while the student was hoping to understand a specific context in which the general might apply in order to learn. Deductive instruction was similarly observed across many classes, where abstract principles and reasoning were described prior to mention of any practical application of that knowledge. Because students may be trying to understand computer science in quite different ways from how professors teach, it can be very helpful for students to hear other students articulate what they are learning.

CS professors occasionally experimented in lecture, demonstrating the trial and error process of computing. Once, for example, a professor worked through a program to produce a graphic image, using trial and error and eliciting the knowledge of students. Though it was the end of the class period, when students in most classes are noisily zipping their notebooks into their backpacks as a signal to professors, on this occasion they were too engaged to behave like typical students. More often, professors came to class with working programs and complete lectures, giving the illusion to their less experienced students that they always had the right answer and that programming ran smoothly for them. Thus CS professors were perceived as experts and students as novices. If professors were to teach in a lab or work through problems in class (e.g., ask students for problems that the professor could then try to program on the spot), students would have more opportunities to learn from each other and experience the process of programming through the eyes of their professors.

In lab, students faced their computers working independently on their assignments. Most talk among students was subdued and brief. Cheating in computer science is a significant and possibly increasing phenomenon.² Like most CS departments, this takes seriously its responsibility to reduce cheating. When a student is detected using another student's code, both are punished. We learned from students that the reason they avoided talking to other

students in labs and blocked their screens was fear that someone else would see their code. As documented elsewhere, when we began this research, collaboration was equated with cheating in non-subtle ways on course syllabi in an effort to reduce cheating [1]. The byproduct was that students were not clear about the difference between cheating and collaborating, erring on the side of caution by not talking to each other.

Although the tone of anti-collaboration statements has softened, syllabi typically continue to include statements like this:

“Some of the homework that you do in this class may be done with other students. However, other work must be done on your own, with no help except from the instructors. For your assignments, you will be asked to accept the student honor pledge: *‘On my honor, I have neither given nor received unauthorized assistance on this work.’* In your submission you should clearly indicate which students helped you in the assignment and on what questions. Breaking this rule for homework or exams will result in an F for the entire course (not just for the assignment).”

It is clear that students had trouble determining the difference between authorized and unauthorized assistance. In interviews conducted by one of our graduate students studying an experimental, group-based introductory course, CS students said that in this course, they could do things that their friends had told them was considered cheating in their traditionally taught courses (e.g., discussing how to write an algorithm as a group). These kinds of messages are the beginning of students' socialization into their chosen discipline, as shown by a comment from CS senior, who said, “you shouldn't have to collaborate to get a job done.”

The combination of the instructor-centered learning environment where the instructor is the principle source of knowledge, subject matter abstracted from the world of human experience, and fear that talking to another student might be construed as cheating resulted in very little observable knowledge sharing among students. A substantial tradition of research shows that students benefit from cooperative learning and hearing each other articulate what they are learning and that women prefer collaborative learning environments. Thus it is no surprise that ITC students interviewed describe their classroom environments as one of the most attractive features of the program.

3.1.2 Knowledge Sharing in ITC

The introductory ITC course is, like the CS courses, taught using lecture and lab, but with the additional requirement that students discuss what they are learning. In addition to the professor, this course includes a series of guest speakers who speak about their experience (e.g., e-commerce or other domains). We observed faculty calling on students frequently and encouraging them to ask questions during lecture. They also were required to analyze what they had heard orally by means of a conceptual framework, defined early in the term and threaded through the semester. The nature of the material was meaningful in and of itself, since students can look around and see the subject matter, the increasing prominence of technology, as part of their everyday lives.

The project-based courses were taught using a fine arts or studio approach. Faculty used both lecture and hands-on activities, but lecture occurred in the lab and directly preceded the hands-on experience. For example, the professor would teach a concept, show a concrete example, and then students would put their hands directly on their keyboards and practice it. The professor was always in the lab with the students, walking around, helping them

² That is, the rise in cheating incidents could be a function of cheat-detecting software or the easy availability of others' code is available on the Internet. Most likely, it is a combination of these.

learn. When he was satisfied that most in the class had learned, he would ask them to look away from their monitors to hear more lecture. In addition, the professor was not always the only person conveying knowledge about design principles, software, or subject matter. As a multi-disciplinary program, it should be expected that students bring a variety of skills and perspectives with them. So, for example, we observed some environmental design students whose skill with certain software, such as Form Z, far exceeded the instructor's. In such a case, we sometimes heard students explaining certain concepts to the professor and to other students; in this way students could come to count on each other to help them learn.

In project courses, initial assignments were personalized. Students were asked to perform general tasks or produce web sites, but to do so by means of their own interests or experiences. The learning objective was for students to understand how to manipulate features and capabilities of the software they are using. To do in-class and after-class assignments, students were told early that there was no reason for them to re-invent the wheel; that is, professors explicitly told students that they should search for and find source code before creating it themselves. Instructors demonstrated web sites with HTML tips, samples of JavaScript and java code, etc., much like computer science professors refer to libraries. This created an expectation that students would either teach themselves a large part of what they were supposed to learn (a sometimes unwelcome expectation, according to interviews) or to seek out the knowledge of their peers.

Student interaction was not forbidden in any way: assignments were personalized, so there was no real way to "cheat." It was not uncommon in labs for a student to call out the professor's name to ask for help; indeed, labs were noisy with student talk. When the professor was busy, which was often the case, students came to expect help or information from anyone, calling out, for example, "does anyone know what the image needs to be saved as to open in Flash? A jpeg?" Anyone who already knew how to do something would guide the student to help, often standing up and looking over the less-experienced student's shoulder to look at the monitor and give very specific instructions. Page after page of observation notes taken from the ITC project courses is filled with students helping each other in a variety of ways.

Students become socialized in ITC to expect learning with each other and professors in the lab, which became evident from student reactions to the way CS professors taught cross-listed courses. In one of these, an ITC student said, "I'd rather do the database assignment on a server, having you there to guide us. We can do the reading outside of class, instead of you lecturing about it." Another student added, "We could have class up in the lab, and maybe do half lecture and half lab." The professor replied that she was already giving them too much time to work collaboratively during lecture time, referring to it as "foreign" in engineering. She then reported to students that when she said told other engineering faculty about the students' requests, they had exclaimed, "they want what?"

Knowledge sharing was multi-directional and occurred at any time in ITC courses. Students showed their professors how to do things; professors asked students how to do things; and it was not uncommon for professors to say "well, I'll have to figure that out later" or "I don't know how to do that." These kinds of comments did not undermine the professors' authority; instead, they suggested the reality of using technology: weird things happen that can't be solved immediately and it is okay to not know it all.

The expert/novice divide we observed in CS did not hold in the ITC classes, revealing as myth the alignment of expert/non-expert roles corresponding with professor/student roles.

3.2 Assessment Techniques

3.2.1 Assessment in CS

Assessment of all programming assignments in the CS courses we observed used a computer program. This program, in addition to assessing whether a student's program worked, could tell if a student's program was too much like another program and therefore identify cheaters. Undergraduate teaching assistants briefly reviewed comments to ensure that students had completed the assignment's requirements. Though the grading program had a name and was talked about as if it were a person, it did not take the place of human interaction, as a professor or a tutor or teacher might. Further, while professors frequently talked about the importance of the process of programming as well as of planning and design, in most cases grades depended almost exclusively on whether or not the program generated appropriate output, etc. Even if a student's mistake was in syntax, they got little credit if their program didn't work. This assessment policy reinforces the message that computer science is about programming and, in spite of what we heard CS professors tell students in class about elegant programs, reinforces the notion that it doesn't matter how you get there, the only thing that counts is getting it to work.

Although Computing Curricula 2001 mandates that students learn to critique programs, we did not observe students critiquing their own or others' programs in lecture or in lab. In contrast, ITC used assessment methods that one would typically find in a fine arts or performance arts setting.

3.2.2 Assessment in ITC

In ITC, students would produce something, then be required to demonstrate what they had produced in front of the entire class before submitting it to the professor for individualized assessment. In addition, rather than simply observe and clap their hands, other students were required to make comments aloud about what they liked or didn't like. For example, one student presented an animation showing a woman getting ready to go out, drinking by herself, and partying with friends. After discussing the meaning of the animation, a male student evaluated its aesthetic aspects, saying one part was too long and another too short. The professor suggested that the student do more with music. A female student suggested that the animation be broken up into two different animations. Even the most skilled students were publicly evaluated by their professors and peers.

Clearly, this was a difficult situation for students, who had to become accustomed to public judgment. It is even more difficult for students who avoid public situations. It was not uncommon to observe a student turning red from the neck up to his forehead as more experienced students, either technically, in terms of design, or in terms of particular subject matter, evaluated his work. Yet all students were required to articulate the choices they made in front of each other and all students listened to others making comments on their work. The effect was the leveling of the playing field, since everyone had to suffer the same fate. Most importantly, students gained in two significant ways: 1) they heard each other talk about doing the work rather than hearing only the more theoretical, didactic, and discipline-specific words of a professor; and 2) hearing what other students knew permitted them to make more accurate assessments about their own skills and levels of

understanding, decreasing the likelihood that they will lose confidence without good cause and leave the program.

The language students and professors used often suggested that the works displayed were works in progress and that therefore, students were in the process of learning. In other words, the presentations were not finished pieces, but were a way of practicing new skills and then reflecting publicly on them. Student criticism of each other's work was taken seriously not only by students, but also by professors, implying that student feedback was valuable. Students became accustomed to asking each other for help and critique and were exposed to more examples of similar work.

4 SUMMARY AND IDEAS

In CS, students had few opportunities to share knowledge with each other, either when they were learning (i.e., in lecture or lab) or when they were assessed. In contrast, the fine arts approach to teaching of the ITC puts the burden on students to elicit knowledge from each other and their professors and to put their own knowledge on public display. As a result, students became accustomed to hearing one another talk about their work – in students', rather than only in professors', terms and through different ways of learning by example as well as principle.

One must consider the differences in pedagogy that are mandated by differences in the underlying notions of mastery and the nature of the material. Computer science is an entire curricular program and cannot possibly be taught using a fine arts approach for every class, especially considering the theoretical nature of much of the subject matter. However, the underlying principles of students sharing knowledge with each other and putting their own knowledge on public display can be integrated into every course. In particular, we believe that CS professors can:

- Ensure opportunities for students to hear each other articulate what they are learning; this does not have to be on graded assignments, but could be practice in constructing algorithms, for example, in groups or pairs in labs. Students could then present their solutions to the rest of the class.
- Begin lectures by telling students a practical application of what they will later describe at the theoretical level. For example, at the beginning of a lecture on finite state machines, a professor might mention that vending machines would fall into the category. That is, what human contexts are made better, more efficient or secure, etc., by using a program or algorithm?
- Require that students display their solutions to the class and require that their classmates give them feedback. In this way, students can learn by example, learn in more than one mode, and hear each other articulate what they are learning. For example, a professor might select a certain number of students to present the design principles of their assignment to the rest of the class.

These techniques allow students to more accurately gauge their progress, overcoming female attrition due to loss of confidence when their performance is equal to that of their peers. Women will be less likely to see themselves as performing below the men around them. In addition, integrating knowledge sharing into lectures and labs can go a long way toward addressing the different ways in which students and faculty work to understand computer science. Also, the interaction among students builds in an environment characterized by sharing and camaraderie that is

preferable to most students, but especially to women. Creating a departmental culture in which students work hard to learn together can be a mechanism for both attracting and retaining female computer science majors.

5 REFERENCES

- [1] Barker, L. J. & Garvin-Doxas, K. (2004). Making visible the behaviors that influence learning environment: A qualitative exploration of computer science classrooms. *Computer Science Education*, 14(2).
- [2] Borg, A. (1998). What draws women to and keeps women in computing? In C. C. Selby (Ed.) *Women in science and engineering: Choices for success*. New York: New York Academy of Sciences.
- [3] Cohoon, J. M. (2001). Toward improving female retention in the computer science major. *Communications of the ACM*, 44(5), 108-114.
- [4] Cohoon, J. M. & Chen, L. (2003). Migrating out of computer science. *Computing Research News*, 15(2), 2-3.
- [5] Farenga, S. J., & Joyce, B. A. (1998). Science-Related Attitudes and Science Course Selection: A Study of High-Ability Boys and Girls. *Roeper Review*, 20(4), 247-251.
- [6] Fisher, A. J., Margolis, J., & Miller, F. (1997). Undergraduate women in computer science: experience, motivation and culture. *Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education*, San Jose, CA.
- [7] Gürer, D. (1999). Testimony to the Committee on Workforce Needs in Information Technology. ACM Committee on Women in Computing. URL: www.acm.org/women/work_shortage.shtml
- [8] Haller, S. M., & Fossum, T. V. (1998). Retaining women in CS with accessible role models. *Proceedings of the ACM SIGCSE Conference*.
- [9] IEEE and ACM Joint Task Force. (2001). *Computing Curricula 2001*. URL: www.computer.org/education/cc2001/final/index.htm
- [10] Information Technology Association of America (1999). Building the 21st century information technology work force: Underrepresented groups in the information technology workforce. Arlington, VA. URL: www.techworkforce.org/recruiting.htm
- [11] Margolis, J. & Fisher, A. (2002). *Unlocking the clubhouse: Women in computing*, MIT Press.
- [12] Margolis, J., Fisher, A. J., & Miller, F. (1998). Living among the "programming gods": The nexus of confidence and interest for undergraduate women in computer science. URL: www-2.cs.cmu.edu/~gendergap/confidence.html
- [13] McHoul, A. W. (1978). The organization of turns at formal talk in the classroom. *Language in Society*, 7, 183-213.
- [14] McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming class, *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, Northern Kentucky, KY.
- [15] O'Leary, D. P. (1999, Version 1: June 1999). *Accessibility of computer science: A reflection for faculty members*. Dianne P. O'Leary. URL: www.cs.umd.edu/users/oleary/faculty/whole.html
- [16] Seymour, E. (1995). The loss of women from science, mathematics, and engineering undergraduate majors: An explanatory account. *Science Education*, 79(4), 437-473.
- [17] Seymour, E. & Hewitt, N. (1997). *Talking about leaving*. Westview Press, Boulder, CO.
- [18] Williams, L. A. (2000). Strengthening the case for pair programming. *IEEE Software*, 17(4), 17-25,
- [19] Wilson, B.G. (1995). Metaphors for instruction: Why we talk about learning environments. *Educational Technology*, 35(5), 25-30.