© PHOTODISC

Gender and Race in Predicting Achievement in Computer Science

Sandra Katz, John Aronis, David Allbritton,
Christine Wilson, and Mary Lou Soffa

0278-0079/03/$17.00©2003IEEE

In the study described here, 65 prospective computer or information science majors worked through a tutorial on the basics of Perl. Eighteen students were African American. All actions were recorded and time-stamped, allowing us to investigate the relationship among six factors that we believed would predict performance in an introductory computer science (CS) course (as measured by course grade) and how much students would learn from the tutorial (as measured by the difference between pre-test and post-test performance). These factors are: preparation (SAT score, number of previous CS courses taken, and pre-test score), time spent on the tutorial as a whole and on individual chapters, amount and type of experimentation, programming accuracy and/or proficiency, approach to materials that involve mathematical formalisms, and approach to learning highly unfamiliar material (pattern-matching procedures). Gender and race differences with respect to these factors were also investigated.

This research was supported by a grant from the National Science Foundation (grant number EIA 0089963). The data presented and views expressed are not necessarily endorsed by this agency.

Sandra Katz and Christine Wilson are with the Learning Research and Development Center, University of Pittsburgh, 3939 O'Hara St., Pittsburgh, PA 15260; email: katz+@pitt.edu, clwilson@pitt.edu. John Aronis and Mary Lou Soffa are with the Computer Science Dept., University of Pittsburgh, 6211 Sennett Square, Pittsburgh, PA 15260; email: aronis@cs.pitt.edu; soffa@cs.pitt.edu. David Allbritton is with the Dept. of Psychology, DePaul University, 2219 N. Kenmore Ave., Chicago, IL 60614; email: dallbrit@depaul.edu.

Predictors of grade and gain score included SAT score, pre-test score (negatively correlated with gain), time (negatively correlated with gain and grade), and various measures of programming accuracy and/or proficiency — for example, the total number of program runs that contained errors (negatively correlated with grade and gain). Several measures of experimentation predicted gain score. Experimentation also predicted grade, but only as applied to the least familiar tutorial material. With respect to gender comparisons, experimentation was practiced throughout the tutorial by both sexes. However, male and female students differed with respect to the types of tutorial topics and tasks they experimented with and the degree of experimentation — for example, male students were more likely to invent their own practice exercises. With respect to race comparisons, African-American students, on average, had lower SAT scores, pre-test to post-test gain scores, and scores on various measures of programming accuracy/proficiency than did white and Asian students. This suggests the need for better pre-college preparation in the math, verbal, and programming skills that African-American (and all) students need in order to succeed in undergraduate CS programs.

Women and Minorities Underrepresented in CS

According to the National Science Foundation's 2000 report entitled, Women, Minorities, and Persons with Disabilities in Science and Engineering, the number and percentage of bachelor's degrees in computer science (CS) awarded to women have decreased in the last decade [14]. African-Americans and other minorities (Hispanics and American Indians) have consistently been underrepresented in computer science. Since a bachelor's degree in CS or information science (IS) is required for many challenging jobs in information technology (IT) — in particular, those that involve software design and/or development — women and minorities have, in turn, been underrepresented in these jobs and in the IT workforce in general.

Since satisfactory performance is a requirement for retention in undergraduate CS or IS programs, many studies have been conducted to identify factors that predict achievement. A complex array of experiential, affective, personality, and cognitive factors have been shown to predict achievement in college courses that involve programming — for example, simply owning a computer [16], using a computer in pre-college computing classes [9], prior programming experience [10], confidence, intrinsic motivation, and having clear career goals [2], [6], [12], [17], and various aptitudes, such as math ability, spatial reasoning ability, verbal reasoning ability, and Piagetian formal operations [1], [3], [6], [18].

Gender comparisons of achievement in courses that involve programming have repeatedly found that female students perform as well or better than male students, both at the pre-college and undergraduate level [8], [13], [16], [17]. Largely in response to this finding, researchers have investigated the socio-cultural forces that may contribute to the withdrawal of women from CS programs. Significant factors include women's sense of "not fitting into" male-dominated CS classes — especially when male students appear to enjoy programming for its own sake rather than as a tool for achieving other practical goals, as women commonly view programming [13], a lack of female role models, the presence of which has been shown to promote retention [5], and, worse yet, an "inclement academic climate" in some CS programs, where female students face doubted qualifica-

tions ("you got in because you are a girl"), patronizing behavior, even sexual harassment [4].

Perhaps because so few minority students enroll in CS bachelor's degree programs, there has been no research done, to our knowledge, on the experiences of minorities in these programs, or on racial/ethnic achievement comparisons. However, the National Science Foundation's 2000 report, Women, Minorities, and Persons with Disabilities in Science and Engineering [14], revealed an increasing interest in computer science among African-Americans and American Indians in the past decade: "the number of associate's degrees in computer science increased for blacks and American Indians and decreased for all other racial/ethnic groups in 1996" [14, p. 18]. Furthermore, the percentage of bachelor's degrees in CS awarded to African-Americans in 1996 (11%) was higher than the percentage in most other science fields and in engineering. These findings highlight the need to sustain minority interest and persistence in undergraduate CS programs and to find out why those who drop out do so.

The research discussed in this article investigates a piece of the "achievement and retention puzzle" that has received very little attention to date: how learning strategies and behaviors affect performance in undergraduate computer science courses. Since programming is one of the first skills that computer science students learn, and a stumbling block for many, we focus on students' strategies for learning how to program. Research by Recker and Pirolli [15] suggests that learning strategies can have a strong impact on programming skill. In particular, students who approached learning materials in a reflective manner — e.g., by self-explaining example programs — outperformed less reflective students on laboratory programming tasks. Learning-to-program behaviors may also affect retention. Even students who perform well in CS courses — as women, as a whole, have been shown to do — may nonetheless withdraw because inefficient learning strategies make the process less enjoyable.

> We hypothesized that experimentation as a learning strategy facilitates achievement in computer science.

We hypothesized that experimentation is another learning strategy that facilitates achievement in computer science. We have observed that strong students are typically eager to "try things out" on the computer, while weaker students tend to take a more passive approach (e.g., reading textbooks and lecture notes). To get a first-hand look at the role that experimentation and other strategies play in learning a new programming language, we developed a laboratory study in which students work through a tutorial on the basics of Perl. By logging all student actions, we were able to investigate correlations between six learning factors and two dependent variables representative of achievement: course performance (as measured by final grade), and how much students learned from the tutorial (as measured by gain score from pre-test to post-test). These factors are: preparation (SAT score, number of previous CS courses taken, and pre-test score), time spent on the tutorial as a whole and on individual chapters, amount and type of experimentation, programming accuracy and/or proficiency, approach to programming tasks that involve mathematical formalisms, and approach to learning new material (pattern matching). Gender and race differences with respect to these factors were also investigated.

METHODS

Materials

Five topics are covered in the Perl programming tutorial, with one brief chapter per topic: input/output basics, arithmetic expressions, conditional execution, while loops, and pattern matching (regular expressions). Each chapter consists of explanatory text, sample programs, and recommended practice exercises.

Participants

Sixty-five students (47 males, 18 females) who claimed to have no prior experience with Perl participated in the study. Fifty-six students (43 males, 13 females) were prospective CS majors at the University of Pittsburgh, enrolled in an introductory CS course. Nine students (4 males, 5 females) were prospective information science (IS) majors, recruited from one section of an introductory IS course at the same university. Forty-three students were white (30 males, 13 females). Eighteen were African-American (14 males, 4 females). No other minority groups are represented in this study. Two students (males) were Asian and two were Asian-American (1 male, 1 female). For the purposes of this study, we will refer to these four students as Asians. Given that whites and Asians are well-represented in computer and information science, we classify these groups as the "major-

ity" and African-American students as the "minority" groups in this study. Students were paid a nominal amount for their participation.

Procedure

We first administered a pre-test to measure students' ability to program in Perl. Students then went through a step-by-step lesson on how to use the programming interface. This lesson was done as a group. Students then worked through the tutorial individually, at their own pace, so that we could track the time that they spent on the tutorial as a whole and on each chapter. Since our aim was to observe how students would work naturally, we provided minimal guidance on what students should (or should not) do. The introduction briefly advised students to enter and run each example, but the remainder of the tutorial did not reinforce this advice. Finally, students took a post-test, which was identical to the pre-test.

Data analysis

We coded the following features of each run in students' log files:

- *Identification:* Which tutorial example or practice exercise is the executed program associated with?
- *Correctness:* A program is considered "correct" if it is free of compilation and logical errors. Compilation errors were logged automatically. Logical errors were identified by a human coder.
- *Novelty status:* Is the program a direct replica of an example provided in the tutorial, or does it attempt to modify the example's functionality? Similarly, if the program is associated with a practice exercise, did the student do what was asked for, or attempt to modify the specified functionality? We code the former as a "verbatim run" and the latter as a "modified run." A human coder tagged the novelty status of each program run.

The transcripts were coded by one researcher. To test the reliability of the coding scheme, a second researcher coded a randomly selected sample of four transcripts. Agreement on correctness and novelty status was 94% (kappa = .89). (Agreement on program identification was not measured, because this feature is self-evident.)

The two dependent variables for achievement in this study are pre-test to post-test gain score and course grade. Representative measures for the six factors we considered as possible predictors of achievement are shown in Table I. Additional measures include aggregations and sub-categories of those shown in this table — for example, number of runs of correctly modified examples and practice exercises, number of runs of practice exercises involving pattern matching (example runs excluded), respectively. The behavioral factors are explained further below.

Experimentation

Students could experiment with two types of materials: tutorial-provided examples and/or recommended practice exercises. Since we did not require students to run examples, we consider verbatim runs of examples "experimental," because the student is trying inputs of his or her own choosing to see what the program will output. This is the simplest form of experimentation. Trying out the recommended practice exercises is considered a higher form of experimentation than modifying examples because the student must generate code. Some students "broke out of the tutorial box" altogether and designed their own exercises. We consider this the highest form of experimentation, whether or not the attempted program was completed successfully.

In future analyses, we will determine whether various groups of students (based on gender and race/ethnicity) differed with respect to the types of modifications they made, the number of changes they attempted per program run, etc., and whether these factors predicted achievement.

Programming accuracy and/or proficiency

The measures included in this category (e.g., number of correct runs, ratio of correct runs to total runs) are ambiguous — hence, the slashed rubric. These measures might indicate how much the student had to struggle to get programs to work (coding proficiency). Alternatively or concurrently, they might indicate how careful or accurate the student is. Wherever possible in the analyses that follow, results for these measures are interpreted in light of other findings, such as time.

Ease with formalism

Because computer science is a formal discipline, especially at more advanced stages of study, we were interested in determining whether students' accuracy and degree of experimentation with mathematical materials would predict achievement and differences among groups. Three tutorial tasks (one exercise and two examples) involved mathematical functions — quadratic equations, geometric series, and harmonic series.

Approach to new material

The last tutorial chapter dealt with material that students most likely had not seen before: pattern matching, using regular expressions. (Students were expected to have been exposed to all other tutorial topics, although in a different language than Perl, because a basic programming course was a prerequisite for the course they were taking.) Hence, the pattern-matching chapter gave us an opportunity to see if "successful" students would differ from "unsuccessful" students in their approach to learning highly novel material.

RESULTS

The small number of female and African-American participants (18 per group) limited the exposure of statistically significant differences among groups, hence raising the need for further research along these lines. All reported findings are significant at the .05 level or less, except where otherwise noted.

Predictors of Gain Score

Pre-test score correlated negatively with gain (r = -0.46), whereas post-test score correlated positively with gain (r = 0.76). Thus, students who were initially the least proficient programmers, at least in Perl, benefited the most from the tutorial. Aptitude, according to all three SAT scores, also predicted gain scores (math, r = 0.33; verbal, r = 0.38; total, r = 0.25). Various measures of time correlated nega-

TABLE I
REPRESENTATIVE MEASURES OF POSSIBLE PREDICTORS OF ACHIEVEMENT

Preparation	• Number of previous programming courses taken • SAT score (math, verbal, and total) • Pre-test score
Time	• Total time spent on the tutorial (5 chapters) • Time spent overall on each tutorial chapter • Time spent only on writing code in each chapter (excludes reading time, breaks between chapters, etc.)
Experimentation—type and degree (shown in increasing order)	**Experimenting with Tutorial-provided Examples** • Number of syntactically correct, verbatim runs of examples • Number of attempted modifications of sample programs, whether or not the program ultimately compiled and was free of logical errors • Number of runs of correctly modified examples, using various inputs[1] **Experimenting with Recommended Practice Exercises** • Number of runs attempting recommended exercises, whether or not they compiled and were free of logical errors • Number of correct, verbatim runs of practice exercise programs • Number of attempted modifications of practice programs, whether or not the program ultimately compiled and was free of logical errors • Number of runs of correctly modified practice programs, using various inputs **Free-form Experimentation** • Number of runs attempting student-designed practice exercises—programs not tied to any tutorial example or exercise—whether or not they compiled and were free of logical errors
Programming accuracy and/or proficiency	• Number of correct runs—that is, runs free of syntactic and logical errors • Number of runs with errors • Percent of correct runs (correct runs/total runs)
Approach to examples and exercises involving mathematical functions (ease with formalism)	Experimentation with mathematical examples and exercises, coding accuracy and/or proficiency, according to the measures of experimentation and accuracy/proficiency listed above.
Approach to new material (pattern matching)	Same as for mathematical examples and exercises.

[1]Our logs do not reveal the inputs that students used to test programs. We assume that if a student ran the same, functioning program more than once, they used different inputs for each trial.

tively with gain (e.g., total time spent on the tutorial, r = -0.28).

Accuracy, as measured by the total number of runs with errors, correlated negatively with gain (r = -0.29). When coupled with the negative correlation between pre-test score and gain, these findings suggest that accuracy was indicative of care rather than coding proficiency. It seems as though less skilled coders (in Perl), who were careful and strove for accuracy, learned more than less careful students.

Students who experimented more, and to a higher degree, also got more out of the tutorial. Measures of experimentation that predicted gain include: the total number of runs of modified examples, whether or not correct (r = 0.28); the number of runs of correctly modified examples (r = 0.27); and the number of runs of "free form" experiments (r = 0.25).

Predictors of Course Grade

The nine IS students were excluded from the analysis of grade, since comparisons across the two disciplines would not have been meaningful. Gain score did not correlate with grade, which suggests that the overlap between the tutorial content and the CS course was relatively small. As with gain score, predictors of course grade included pre-test score (r = 0.28), post-test score (r = 0.37), aptitude (SAT total, r = 0.40), and various measures of time (negatively correlated; e.g., total time, r = -0.30).

The same measure of accuracy that predicted gain score also predicted grade — the number of runs with errors (r = -0.38). This finding was supported by several other correlations between accuracy measures and grade — for example, the percent of correct compilations, overall (r = 0.46), and the percent of correct modified runs of examples and exercises (r = 0.31). As with gain score, these correlations between accuracy and grade are difficult to interpret and warrant further investigation: Does accuracy indicate an acquired level of proficiency with coding that is reflected in final course grade, or do students earn high grades in this course partly because they are careful and accurate?

Experimentation also predicted grade, but only with respect to learning the least familiar material (as opposed to the tutorial as a whole). For example, the number of correct runs of pattern-matching examples and exercises correlated positively with grade (r = 0.30).

Interactions Among Predictor Variables

Time, experimentation, and accuracy. Getting programs to run takes time, at least for students who have low accuracy rates. The total time that students spent on the tutorial correlated positively with the total number of runs (r = 0.59) and the number of runs with errors (r = 0.57). But experimentation also takes time. The number of correct runs (using multiple inputs) also correlated positively with time (r = 0.39). Experimentation is time-intensive, in part, because it increases the odds of making errors, and error-fixing takes time. For example, the number of correctly modified runs of practice exercises correlated positively with the number of runs with errors (r = 0.27), suggesting that correctly modified programs followed several unsuccessful trials.

These findings suggest that time is a complex measure. On the one hand, time predicts achievement (negatively correlated with gain score and grade). On the other hand, experimentation takes time, but also predicts achievement. Hence, in order to intervene appropriately, instructors should consider what students who take relatively long are doing: Are they spending a lot of time fixing errors? Or are they writing and modifying programs, and running them on various test cases?

Aptitude, time, and experimentation. Time also appears to be related to aptitude. Students with higher math SAT scores finished sooner, overall, than other students (r = -0.36) and spent less time on coding tasks (r = -0.43). This is not surprising, given that several examples and exercises dealt with mathematical functions.

Verbal SAT score correlated positively with time spent on self-designed experiments (r = 0.27). Verbal SAT score also correlated with the number of correct verbatim runs of pattern-matching exercises (r = 0.31). So, it appears that students with higher verbal aptitude experimented more with the most linguistically-oriented material in the tutorial. This finding suggests that students' tendency to experiment is related to their comfort level with, and/or interest in the material presented to them. Further research is needed to identify the factors that govern when, how, and how much students experiment.

Degree of experimentation. Students who experiment tend to do so in various ways. In particular, free-form experimenters also had higher numbers of runs of correctly modified examples and practice exercises (r = 0.42).

Aptitude, background, accuracy, and experimentation. We saw that aptitude correlated positively with gain score and course grade. However, there could be a hidden variable contributing to this effect — namely, programming background. Total SAT score correlated positively with the number of previous courses taken that involved programming (r = 0.38).

As one would expect, the number of prior programming courses that students took predicted how well they would do on the tutorial pre-test (r = 0.38) and post-test (r =

0.31). Background also predicted programming accuracy and/or proficiency. For example, the number of prior programming courses predicted the percent of correctly modified examples and exercises (r = 0.26).

Male and female students differed as to the types of topics and tasks they experimented with, and on the degree of experimentation.

Gender Comparisons

Male students had more prior programming experience than female students (t(59) = 2.32). Both men and women showed evidence of experimenting, though on different types of material. Men spent more time working on self-designed practice exercises (t(46) = 1.97, p = 0.06). Whereas men had more correct runs of modified examples (t(62) = 2.94), women had more correct runs of modified practice programs (t(63) = 2.18). Whereas men had more correct runs of modified pattern-matching examples (t(61) = 2.49), women had more correct runs of modified examples and exercises involving mathematical functions (t(63) = 2.04). Thus, both male and female students experimented on the most challenging tutorial material, though on different types of challenging material.

These findings are partially consistent with prior research that reveals gender differences in software use and development. For example, Kafai and her colleagues found significant gender differences in elementary students' design of video games [8]. Our observation that male students experimented to a higher degree than did female students (by inventing their own exercises) concurs with Margolis and Fisher's [13] finding that male undergraduate CS majors claimed that they "played" with computers and programming more and at an earlier age than did female students. However, our observation that women experimented more than men in some ways also suggests that women may "play" more than they think they do — at least when learning a new programming language in a laboratory setting, as in the current study. Do male and female students benefit differently from different forms of experimentation, or would both sexes benefit most from a diverse repertoire of "programming play" techniques? This is an interesting question for further research.

Race Comparisons

White and Asian students had higher SAT scores (t(59) = 3.8 for SAT math; t(59) = 3.5 for SAT verbal; t(62) = 2.0 for SAT total) and gain scores than African-American students (t(62) = 2.2). White and Asian students also scored higher on various measures of accuracy/proficiency — for example, the percent of correct compilations (t(62) = 2.0) and the number of correct runs of modified practice exercises (t(62) = 2.2). Correspondingly, African-American students took longer, on average, to work through the tutorial as a whole (t(62) = 4.1) and through individual chapters, most likely because they were trying to fix errors.

African-American students scored higher than White and Asian students on one experimentation measure: the number of runs of correctly modified practice exercises (t(62) = 2.2). As we have seen, experimentation correlated positively with gain score and, to a lesser extent, with course grade. Hence, experimentation appears to be one learning behavior that should be encouraged with all students who are having difficulty learning to program, regardless of race.

As Jennifer Light [11] has argued, complex problems such as social inequality can not be solved by simple solutions. The so-called "digital divide" — that is, inferior access to computers and the Internet by African-Americans and other minorities — and the under-representation of minorities in the IT workforce are technological manifestations of social inequality. Light claims that simply providing minorities with more access to technology will not solve these problems. Our research supports Light's position. Most of the African-American students who participated in our study reported on a background survey that they had ample access to computers currently and during their pre-college years. So their relatively poor performance on the tutorial and in the course can not be attributed to insufficient computer access. The significantly lower SAT scores of these students, coupled with lower accuracy/proficiency scores, suggest that they entered the course underprepared — in math skills, verbal skills, and basic programming skills, all of which predict achievement. Hence, better preparation in these skills before minority students enter baccalaureate CS students appears to be critical for success.

Race and Gender Comparisons

Comparisons among African-American males, white and Asian (white/Asian) males, and white and Asian females were made using independent t-tests. Because there were too few African-American

females (4), no tests of significance were conducted comparing African-American females to other groups.

White/Asian males differed significantly from White/Asian females in the number of runs of correctly modified examples, with males scoring higher ($t(43) = 2.5$). This is consistent with the finding reported previously — that men overall had more runs of correctly modified examples than did women ($t(62) = 2.94$). Hence, men apparently tried out their modified example programs on more test cases than did women.

White/Asian males differed significantly from African-American males in all 3 measures of SAT (higher math, $t(45) = 3.2$; verbal, $t(45) = 4.6$; and total score $t(45) = 4.6$, for white/Asian males), and in several measures of time, with African-American males spending more time overall on the tutorial ($t(45) = 3.9$) and on most tutorial chapters.

Comparing white/Asian females with African-American males, we note similar differences with respect to aptitude and time as with white/Asian males and African-American males. White/Asian females had higher SAT verbal scores ($t(23) = 3.7$) and spent less time than African-American males on several measures — e.g., total time spent on the tutorial ($t(19.4) = 2.8$), time spent working on the examples and practice exercises only (excludes reading time; $t(18) = 3.0$), and time spent on several chapters. White/Asian females also scored higher than African-American males on several accuracy/proficiency measures. For example, white/Asian females had fewer runs with errors ($t(15.6) = 2.4$) and a higher percentage of correct compilations ($t(24.6) = 2.1$). Hence, it appears as though the race difference with respect to accuracy/proficiency reported previously — in which white/Asian students, overall, outperformed African-American students — was mainly due to the high accuracy/proficiency scores of white/Asian women, since white/Asian and African-American men did not differ significantly on this factor.

Early Identification

A tutorial such as the one used in this study could serve as an instrument to identify students early on who are likely to succeed (or not) in introductory courses that focus on programming concepts and skills. This information could be used by instructors to determine which students are likely to need extra help. In conjunction with prior research, this study also suggests what types of interventions are likely to be effective: exercises that promote coding accuracy and proficiency, and encourage experimentation and reflection on the results of experiments [15]. In future studies, we will develop and assess interventions that have these features, and attempt to specify more precisely what types of experimentation promote achievement in undergraduate programming courses.

Acknowledgment

The authors appreciate the helpful comments of the anonymous reviewers.

References

[1] R. Cafolla, "Piagetian formal operations and other cognitive correlates of achievement in computer programming," *J. Educational Technology Systems*, vol. 16, no. 1, pp. 45-55, 1987.

[2] J.P. Charlton and P.E. Birkett, "Psychological characteristics of students taking programming-oriented and applications-oriented computing courses," *J. Educational Computing Res.*, vol. 18, no. 2, pp. 163-182, 1998.

[3] C.A. Clement, D.M. Kurland, R. Mawby, R., and R.D. Pea, "Analogical reasoning and computer programming," *J. Educational Computing Res.*, vol. 2, no. 4, pp. 473-86, 1986.

[4] K.A. Frenkel, "Women and computing," *Commun. ACM*, vol. 33, no. 11, pp. 34-45, 1990.

[5] L.S. Hornig, "Women in science and engineering: Why so few?," *Technology Rev.*, vol. 87, no. 8, pp. 31-41, 1984.

[6] C.M. Jagacinski, W.K. LeBold, and G. Salvendy, "Gender differences in persistence in computer-related fields," *J. Educational Computing Res.*, vol. 4, no. 2, pp. 185-202, 1988.

[7] Y.B. Kafai, *Minds in Play: Computer Game Designs as a Context for Children's Learning*. Hillsdale, N.J.: Lawrence Erlbaum, 1995.

[8] Y.B. Kafai, "Video game designs by children: Consistency and variability of gender differences," in *From Barbie to Mortal Kombat: Gender and Computer Games*, J. Cassell and H. Jenkins, Eds. Boston, MA: M.I.T. Press, 1998, pp. 90-114.

[9] D. Kagan, "Learning how to program or use computers: A review of six applied studies," *Educational Technology*, vol. 28, no. 3, pp. 49-51, 1988.

[10] A.A. Koohang and D.M. Byrd, "A study of selected variables and future study," *Library and Information Science Res.*, vol. 9, no. 1, pp. 214-288, 1987.

[11] J. Light, "Rethinking the digital divide," *Harvard Educational Rev.*, vol. 71, no. 4, pp. 709-733, 2001.

[12] G.A. Marcoulides, "The relationship between computer anxiety and computer achievement," *J. Educational Computing Res.*, vol. 4, no. 2, pp. 151-158, 1988.

[13] J. Margolis and A. Fisher, *Unlocking the Clubhouse: Women in Computing*. M.I.T. Press, 2001.

[14] National Science Foundation, *Women, Minorities, and Persons with Disabilities in Science and Engineering: 2000*. Arlington, VA: NSF, NSF 00-327, 2000.

[15] M.M. Recker and P. Pirolli, "Student strategies for learning from a computational environment," in *Intelligent Tutoring Systems*, C. Frasson, G. Gauthier, and G.I. McCalla, Eds. Berlin, Germany: Springer-Verlag, 1992, pp. 382-394.

[16] H.G. Taylor and L.C. Mounfield, "Exploring the relationship between prior computing experience and gender on success in college computer science," *J. Educational Computing Res.*, vol. 11, no. 4, pp. 291-306, 1994.

[17] S.E. Volet and I.M Styles, "Predictors of study management and performance on a first-year computer course: The significance of students' study goals and perceptions," *J. Educational Computing Res.*, vol. 8, no. 4, pp. 423-449, 1992.

[18] N.M. Webb, "Microcomputer learning in small groups: Cognitive requirements and group processes," *J. Educational Psychology*, vol. 76, pp. 1076-1088, 1984.