A Self-Paced Introductory Programming Course

T. Grandon Gill and Carolyn F. Holton
University of South Florida, Tampa, FL USA

ggill@coba.usf.edu cholton@coba.usf.edu

Executive Summary

In this paper, a required introductory programming course being taught to MIS undergraduates using the C++ programming language is described. Two factors make the objectives of the course—which are to provide students with an exposure to the logical organization of the computer in addition to teaching them basic programming logic—particularly challenging to achieve. First, students enter the course with widely varying backgrounds, with roughly equal numbers having no prior exposure to programming courses, having taken one previous course and having taken two or more previous courses. They also have different work aspirations, with about half believing it is unlikely that they will be employed as programmers within 10 years. This makes choosing an appropriate amount of material to cover problematic. Second, many of the students chose MIS as a major (as opposed to computer science) specifically to avoid the necessity of learning programming. This leads to motivational barriers.

To address these challenges, the course utilizes a self-paced format. Making the self-paced format work required three systems: 1) Content delivery: extensive multimedia aids and web content to support textual materials and to substitute for classroom lectures, 2) Peer support: peer-tutoring and assignment validation, drawing from approaches used in nuclear submarine training that provide flexibility and enhance rigor, 3) Progress monitoring: an administrative information system, used to track student progress and provide students with weekly reports.

Collectively, the three course systems have led to very positive learning outcomes. No pattern of significant differences on grades, course evaluations, satisfaction, self-assessments of learning, or career attractiveness was detected based on gender, ethnicity, employment status or prior programming coursework, among other factors tested, indicating the course effectively accommodates substantial student diversity. The failure rate dropped from 19% to 13% between the first and fifth semesters assessed for the evolving course (with no change in academic standards), and the withdrawal rate dropped from 31% to 19% in the same time frame, an important criterion in a time of dwindling enrollment. Students give high marks to the course's peer assistance, group activities, and its emphasis on non-exam elements, indicating the course design is working as intended. The course also outperforms all other required programming courses in the department on student evaluations.

Material published as part of this journal, either on-line or in print, is copyrighted by the Informing Science Institute. Permission to make digital or paper copy of part or all of these works for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage AND that copies 1) bear this notice in full and 2) give the full citation on the first page. It is permissible to abstract these works so long as credit is given. To copy in all other cases or to republish or to post on a server or to redistribute to lists requires specific permission and payment of a fee. Contact Publisher@InformingScience.org to request redistribution permission.

Keywords: Programming instruction, CS1, self-paced, peer learning, introductory programming, C++.

Introduction

The challenge of teaching programming to students coming in with very different levels of programming expertise has been widely noted in the computer science (CS) literature (Roberts, 2000),

Editor: Chris Cope

with prior programming experience leading to significant differences in student performance (Hagan & Markam, 2000; Wilson & Shrock, 2001). That problem is compounded in the MIS field, where a relatively small percentage of students anticipate working as programmers after graduation, a fact that can seriously impact student motivation.

Confronted with such diversity of background and motivation, there seem to be three generic strategies that an instructor or department might pursue. The first involves teaching the course at a fast pace, and accepting the fact that DWF (D-grades, withdrawals and failures) rates will be high—often as high as 50%. The second strategy involves slowing the course to the point where even ill-prepared or unmotivated students can keep up—typically by reducing the amount of material covered. The third approach is to partition the course into cohorts that proceed at different paces, sometimes accomplished by establishing different tracks within a program.

Where separate tracks are infeasible—precluded, for example, by the number of courses allowed in the major or by a shortage of students or instructors—a variation on the third strategy involves designing a course in such a manner that each student is allowed to proceed at his or her own pace. Doing so allows well-prepared students to forge ahead quickly, while students with little or no background can take the time necessary to master the fundamentals. This strategy is the self-paced approach. Such an approach has occasionally been used to teach programming for at least three decades (e.g., Aikin, 1981; Linder, 1976). Typically, such courses have either emphasized computer-aided instructional tools (e.g., Aikin, 1981; Daly, 1991; Linder, 1976) or independent learning/tutorials (e.g., Cook, 1976; Etlinger, Goodman, & Plummer, 1981; Poindexter, 2003), sometimes conducted in the online environment (e.g., Carrasquel, 1999). There does not seem to be any evidence of self-paced approaches built around more traditional approaches to teaching programming, such as the use of lectures and group projects.

While attractive in principle, implementing self-paced learning has many pitfalls. First, the notion of weekly lectures must be abandoned, since there is no lecture topic that won't be, all at once, too advanced for one cohort of students and too basic for another cohort. Testing becomes more complicated for the same reason. Even previously straightforward decisions, such as when assignments should be due, become complex. Making the dates too strict penalizes the group of students with the weakest backgrounds—the very group that can least afford to be penalized. Making due dates too relaxed, on the other hand, can encourage widespread procrastination.

Despite these challenges, the payback for a successful self-paced design is high in today's environment. Five years ago, when every computer class was oversubscribed, making introductory programming a "gatekeeper" course for which high attrition was deemed acceptable—even desirable—may have made sense. In today's world of sharply reduced enrollments, such an uncompromising approach could put an entire department's existence at risk. Abandoning rigor to reduce attrition, on the other hand, is a myopic strategy that could easily compromise the long term attractiveness of a program, as graduates discover themselves deficient in skills expected by their employers.

The present paper examines the design and delivery of a successful self-paced programming course. We begin by describing the course and students. We then examine the three systems—content delivery, peer support and progress monitoring—developed to keep the course running smoothly. Finally, we turn to the outcomes experienced during the three years in which the course evolved.

The Course

The course presented herein was a required introductory programming course for undergraduate MIS majors at a large state university, taught using the C++ programming language. It was typically taken during the student's junior year, and was one of the first courses taken in the MIS ma-

jor, with enrollments of 80 to 100 students being typical in the spring and fall semesters during the 2004-2005 timeframe. The purpose of the course was to teach all students the basics of procedural thinking necessary to pursue any career in MIS and to give those students contemplating a career in programming a rigorous foundation in programming fundamentals that they could build upon.

Course content mixed conceptual, algorithmic and architectural elements. Topics included data representation, flowcharting, functions, elementary algorithms, debugging techniques, memory organization, and stream I/O. Advanced students were also introduced to C++ classes and inheritance. The course employed 7 assignments (Gill, 2005b), ranging from a simple "hello world" program to a CGI application performing a mortgage amortization.

The initial motivation for the adoption of a self-paced approach to teaching the course came from the need to accommodate considerable diversity in student backgrounds, experience and aspirations, as illustrated in Table 1. By 2004, however, the goal of reducing attrition had become equally pressing. The university's undergraduate MIS major had been particularly hard-hit by declines in popularity during the three years that followed the "dot-com" bubble, with its enrollment shrinking from a 2001 peak of around 1100 undergraduates (the largest in the state) to roughly 300 students by late 2004. Historically, the programming course had experienced DWF rates of about 50% (with some instructors averaging even higher). Going to a pure self-paced format, it was hoped, would reduce the number of students exiting the MIS major as a result of becoming discouraged by course time demands (students reported the course taking more than twice the time of a typical MIS course) and complex content.

Table 1: Selected diversity-related survey responses
(Spring 2003-Spring 2005, 254 responses)

Area	Item	Result
Gender	Percentage of women	30%
Ethnicity	African Americans	12%
Ethnicity	Hispanic	13%
Ethnicity	Non-US Citizens	13%
Experience	Have taken no programming courses before	35%
Experience	Have taken more than one programming course before	30%
Experience	Working full time	36%
Aspirations	Believe it likely that they will be employed as programmers within 10 years	14%
Aspirations	Believe it unlikely that they will be employed as programmers within 10 years	47%

Self-Paced Design

The self-paced design of the programming course evolved incrementally, rather than being implemented in a dramatic single step. By the time the design reached its final self-paced form, in Fall 2004, it consisted of three systems: a content delivery system, a peer support system and a progress monitoring system These three systems, and the role played by each, are now described.

Content Delivery System

In order for a course to be truly self-paced—from the student's perspective—all course content must be available 24/7. The first step towards achieving this was made in Fall 2001, when the

instructor provided students with a CD containing all course notes and about 6 hours of multimedia content (developed using Techsmith's Camtasia) that demonstrated the Visual Studio environment used in the class. By Fall 2002, the content had been extended to include a draft textbook, later published (Gill, 2004), which incorporated the multimedia content into the text. In Spring 2003, multimedia versions of classroom lectures were made available online, using the institution's Blackboard course management system. By Spring 2004, attendance at live lectures had dwindled to under 10% of all students by the middle of the semester. In Summer 2004, therefore, live lectures were eliminated and the course went to a fully self-paced format.

The content delivery system of the fully self-paced version of the course included the following elements:

- A course web site, organized by assignment, containing multimedia lectures, multimedia content specifically related to the assignments, textbook multimedia segments, sample code, test data sets and executable versions of assigned programming projects.
- A Blackboard site, containing discussion groups for each assignment, practice exams and validation exams for pencil-and-paper assignments, and teaching assistant (TA) generated support materials.
- An Elluminate office—featuring voice chat, whiteboard, and application sharing—used to conduct online conferences and tutorials.

Collectively, these provided the mechanism for transmitting course content to students. These elements were supplemented by a number of weekly assignment walkthroughs, conducted by TAs in computer labs, which students could (optionally) attend.

Peer Support System

The second system instrumental to delivering the self-paced course was that of peer support. The peer support system manifested itself in two ways: through encouraging group work and through the use of undergraduate (peer) TAs.

To encourage group work, students were allowed to submit assignments—which represented 90-100% of their grade—in groups of up to four, and groups were allowed to "cross pollinate" in a variety of ways. Although the benefits of working in teams have been noted in both the CS (Prey, 1995; Wills & Finkel, 1997) and undergraduate education (Chickering & Gamson, 1987) literatures, the obvious drawback of doing so is the potential loss of rigor that can result from the "free rider" effect. To address this problem, a validation system—modeled after nuclear submarine training (Gill, 2005a)—was developed. Under this system, students completing an assignment received no credit for doing so until the assignment was validated. For pencil-and-paper exercises, such validation consisted of completing TA-proctored online tests drawn from the same bank of questions as the assignments. For programming assignments, students had to pass an individual oral examination with a TA or the instructor on the code that they handed in. These exams were designed to ensure that students understood every aspect of the code they submitted, and typically consisted of questions intended to elicit higher order thinking, such as:

- "What would happen if this line of code were removed?"
- "In this block of code, explain the nested loop…" or
- "If you called this function with the following arguments [list arguments], what would the value of this variable be?"

If a student did not successfully validate, opportunities were provided to validate again, after subsequent study. After 2-3 unsuccessful tries, any subsequent attempts had to be made with the in-

structor. A 2005 experiment with 30 student participants found the perceived fairness of oral exams (4.1 on a 1-strong disagree to 5-strong agree scale) to be higher than any other evaluation technique considered, significantly better than both multiple choice (3.17, $p<0.001$) and essay (3.38, $p<0.02$) exams used to assess knowledge of the same assignment content.

The second element of the peer support system involved the use of undergraduate teaching assistants, all of whom had previously completed the class. The benefits of using undergraduate TAs have already been described many times in the literature (Reges, 2003; Roberts, 2000; Roberts, Lilly, & Rollins, 1995; Wills & Finkel, 1997). In the self-paced course, the role of TAs was particularly instrumental for additional reasons that included:

- The complexity of the course organization meant that learning how the course functioned was a non-trivial exercise. By recruiting exclusively from former students, the need to retrain TAs each semester was eliminated. Since course assignments tended to be reused (rigor being maintained through validation), TAs also were intimately familiar with the technical content of the course.
- The variability in the pace at which students completed the course meant that clusters of students interested in the same problem at the same time tended to be small. Undergraduate TAs, being far less expensive than either graduate TAs or hiring more instructors, made it possible to support these small groups effectively. (Normally, about 0.5 hour of TA time per week was allocated for each student.) The larger number of TAs also tended to ensure more eyes were focused on Blackboard discussion groups established for each assignment, leading to typical reply times of a few hours.
- The TAs themselves reported benefiting substantially from the experience of tutoring their peers.
- Student objections relating to the "impossibility" of the course tended to be undercut by the fact that they often found themselves attending other classes with their course TAs—proving that their peers could, in fact, master the material.

In addition to grading assignments, holding office hours and answering Blackboard questions/emails, TAs also held weekly walkthrough sessions related to each assignment. For those students who were uncomfortable with the impersonal multimedia aspects of the delivery system, these weekly meetings offered a more traditional classroom experience.

Progress Monitoring System

The final system supporting the self-paced course was the progress monitoring system. One of the major challenges in running a self-paced course is keeping students from procrastinating. The problem becomes particularly acute in a course where later assignments require getting substantial programs to compile and run—an activity often unpredictable in its time demands (even for experienced programmers). The seriousness of the potential problem was most graphically demonstrated the first time the course was run in a fully self-paced format (in Summer 2004), at which time—with 10 days left in a 10 week summer session—less than 10% of the students had accumulated enough points to justify a C grade.

To encourage students to maintain an even pace throughout the semester, a number of modifications to the course were implemented:

- A validation card, modeled after a submarine qualification card (Gill, 2005a), was issued to each student on the first class meeting. This card was used to track student progress throughout the validation process (including any retakes).

- Each student was paired with a mentor, referred to as an "assigned TA", who was charged with monitoring the student's progress and notifying the instructor if further assistance was required.
- A system of participation credit was instituted (up to 10% of the student's grade). To get full credit, a student either had to: a) meet with his or her assigned TA, b) fill out a web-based online progress report form, or c) make an entry to a blog, hosted on LiveJournal. To track these activities, the instructor developed a program that consolidated the data into a weekly report that was then distributed to each student (along with his or her TA) as an HTML-page attachment to an email.
- For the two major programming assignments, two one-time-only tests were developed (and updated each semester). Students who had completed the relevant assignment prior to the date of each test could choose to take the test, and those passing it were exempted from the associated oral exam.

Collectively, these activities served to motivate students to work at a more even pace throughout the semester. For example, in Summer 2005, overall student progress at the half-way point in the term had already exceeded progress at the 90% point for the previous summer—although, paradoxically, the DWF rates at the conclusion of the two sessions were nearly identical.

Results

A number of different sources of information have been used to assess the effectiveness of the self-paced approach to teaching programming. Included among these are enrollment/withdrawal statistics, university course evaluations, and a comprehensive course survey. The last of these was first developed (using NSF-recommended instruments) and administered in Spring 2003. It was then administered every semester thereafter. Students completing the survey, which tracked virtually every aspect of the course, received a + appended to their final letter grade (i.e., C becomes C+, B becomes B+, A becomes A+). Response rates typically ran 60-70% of active students. Students were also allowed an “opt out” provision, whereby they could acquire the same extra credit by completing a short instructor-assigned programming project instead of completing the form (although no student has ever chosen that option).

Accommodating Diversity

As mentioned at the outset of the paper, the principal initial motivation for adopting a self-paced pedagogy was to address the diversity in backgrounds and motivation levels of enrolled students. With respect to this metric, course outcomes have been very promising. Using three techniques—ANOVA, multiple regression and MANOVA (used to look for individual variable effects in the presence of other variables)—a set of 19 student attributes were tested against 86 outcome variables to determine what impact diversity had on these outcomes. As shown in Table 2 (for MANOVA), no pattern of significant impacts could be detected. Results were comparable for the other statistical tests.

These findings are noteworthy in a number of ways. Most unexpectedly, the self-paced approach did not show the sensitivity to previous programming experience that is typically reported in the CS literature (Hagan & Markam, 2000; Holden & Weeden, 2003; Wilson & Shrock, 2001), despite a fairly large sample size (254 students). The most plausible interpretation for this finding is that the self-paced pedagogy enables motivation to dominate experience as a factor in determining individual outcomes. For example, in the self-paced format the student knows precisely what his or her grade would be at any point in time—meaning students who have no interest in programming can cease course-related efforts immediately upon achieving a C, since no uncertainty regarding a yet-to-be-taken final exam is present. The overall motivation of the student popula-

tion may have also have had some impact, since MIS majors frequently cite lack of interest in programming as a reason for choosing the major over CS.

Table 2: Relationships Tested and Found Non-Significant

Student characteristics (# variables)	Outcomes variables considered (# variables) [1]
• Gender (1) • Ethnicity (4) • Student status (FT/PT) (1) • Employment status (1) • Prior programming courses (6) • Prior C courses (6)	• Final grade letter (1) • University course evaluation items (8) • Satisfaction with assignments (8) • Satisfaction with other course aspects (15) • Student assessment of own learning gains (52) • Attractiveness of IT and programming careers (2)
[1]Tested at α=.05, MANOVA found no significant differences between students across many dependent variables. When relationships were tested separately, the few significant ANOVAs were consistent with random chance given the large number of relationships examined, an interpretation supported by the MANOVA.	

With respect to demographic diversity, not a single dependent variable had a significant relationship with gender, unlike the results of a number of previous studies (Goold & Rimmer, 2000; Sackrowitz & Parelius, 1996). This finding was significant since the design framework was inspired by submarine training techniques that were originally developed for men (Gill, 2005a). Other demographic variables (e.g., African American, Hispanic, Native American, International) were similarly insignificant, as were work status and student status (part-time vs. full-time). These results are consistent with a previous study (Clancy, Titterton, Ryan, Slotta, & Linn, 2003) that found flexible pacing and web support enhanced a course's effectiveness for a diverse group of students.

Retention

Given the previously noted declines in MIS enrollments experienced by the instructor's program, reducing DWF grades was considered to be another key outcome. Even as the course was evolving, relatively low percentages of Ds and Fs were awarded (Gill, 2005a). Withdrawal rates, on the other hand, had tended to be high (in the 30-40% range, contrasted with a typical rate of 15-20% for other MIS undergraduate courses). During the evolution of the course to its final self-paced form, substantial improvement in DWF rates was experienced, as shown in Table 3. As shown in Table 4—which compares the DWF values for pre-self paced sections (Fall 2003 & Spring 2004) with the self-paced sections (Summer 2004, Fall 2004 and Spring 2005)—the likelihood that the values arose from the same distribution is very low ($p<0.01$, using a chi-square test).

In interpreting Tables 3 and 4, it is also useful to point out that the course requirements remained virtually unchanged from Fall 2003 to the end of the period, and that the same validation process was in effect throughout the entire reported time frame. Indeed, when participation credit was introduced in Fall 2004, the course curve—stated in the syllabus at the start of each semester—was adjusted to ensure that the actual assignment requirements for A, B and C grades were nearly unchanged provided full participation scores were realized. Thus, the reduction in attrition observed cannot be attributed to a gradual relaxation of course standards.

Table 3: Retention by semester [1]

Value	Spring '05	Fall '04	Summer '04	Spring '04	Fall '03
Enrollment	71 [2]	79	34	93	116
Passing	65%	63%	68%	54%	50%
D & F	13%	20%	11%	24%	19%
WD	19%	18%	22%	22%	31%
DWF	35%	37%	32%	46%	50%
Completing class (count)	57	66	29	72	80
A grades	38%	33%	0%	17%	26%
B grades	22%	23%	21%	17%	25%
C grades	24%	20%	66%	36%	21%
Passing %	84%	77%	86%	69%	72%
D & F	16%	23%	14%	31%	28%

[1]: A chi square test confirms that the pattern of passing and not passing students observed across semesters is different from what would be expected by chance ($p \leq .05$). These differences are notable in light of similarities in the student body across semesters: MANOVA detects no significant between group differences in age, pre-university programming course work, MIS and software development work experience, or MIS and software development work aspirations.

[2]: For the purpose of the analysis, excused incomplete ("I") grades were omitted from the enrollment figures. In spring 2005, a substantial number (11) of no-show students (i.e., students who had no course contact after the first week of class) were similarly treated as "I" grades.

Table 4: DWF versus passing grades, counts of students

	Self-paced	Not self-paced
Pass	119	108
DWF	65	101

$p<0.01$ likelihood that self-paced and not self-paced came from same distribution, using chi-square test.

Closely related to student retention was student achievement, as measured by course grades. From Table 3, it is clear that the initial transition to self-paced organization—in Summer 2004—was accompanied by a sharp reduction in honors grades (Bs and As). With the introduction of the progress monitoring information system in Fall 2004, however, students seemed better able to pace themselves and honors grades increased.

Student Assessments

From the comprehensive course survey, many items suggested that individual elements of the course design were working as intended. For example:

- 74% found TAs to be helpful in their learning. Individual ratings of TAs were high, ranging from 3.6 to 4.8 on a 5 point scale.
- 69% felt that working with peers was much help or very much help in learning.
- Students reported spending an average of 5 hours per week working in groups.
- 69% were satisfied or very satisfied with group activities.
- 67% disagreed or strongly disagreed (median and mode of 1 on a 5 point scale) that more emphasis should be placed on tests (17% agreed).

- Students report spending 15 hours per week on the course, about twice the time reported for a typical MIS courses and about 3 times that for typical business courses.

For the most part, these values were relatively constant before and after the evolutionary change to a pure self-paced format that took place in Summer 2004.

On the university's 1 to 5 overall rating scale (5 being the best), required programming courses typically received the lowest overall ratings of any category of required undergraduate courses within the MIS major (roughly 3.7 compared with 4.2 for the non-programming courses over calendar year 2004). Although falling far short of being a mark of distinction, during Spring and Fall 2004, the evolving format outperformed all other courses in the "required programming" category (e.g., receiving 4.0 vs. 3.9 for traditionally taught programming courses in Spring 2004; 3.6 vs. 3.4 for traditionally taught programming courses in Fall 2004). In Summer 2005 and Fall 2005, the course achieved parity with non-programming courses, scoring 4.2.

Conclusion

This paper has detailed the design of a self-paced introductory programming course and has presented results demonstrating its effectiveness in addressing key issues facing programming instructors—most importantly, the diversity of enrolled students and retention. The approach seems particularly applicable to situations where there is receptivity to distance learning techniques and where student diversity is highest. Interestingly enough, these two characteristics may well go hand in hand in the world of computer science and MIS (Malcolm et al., 2005).

For the self-paced technique described here to be widely applied, there appear to be two areas needing attention: the relatively low student satisfaction with the course (compared with MIS courses in general) and the difficulty faced by new instructors in employing the broad range of technologies used to deliver content and monitor student progress. To address the first weakness, within a year the instructor plans to conduct a trial class where more engaging content (e.g., C#.NET used to build GUI applications) is substituted for console-based C/C++, leaving the self-paced delivery approach unchanged. This experiment should help to determine if having MIS students build more exciting applications can enhance motivation, independent of pedagogy.

References

Aikin, J. O. (1981). A self-paced first course in computer science. *ACM SIGCSE Bulletin, Proceedings of the Twelfth SIGCSE Technical Symposium on Computer Science Education SIGCSE '81, 13*(1), 78-85.

Carrasquel, J. (1999). Teaching CS1 on-line: The good, the bad, and the ugly. *ACM SIGCSE Bulletin, The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education SIGCSE '99, 31*(1), 212-216.

Chickering, A. W. & Gamson, Z. F. (1987). Seven principles for good practice in undergraduate education. *AAHE Bulletin, 39*(7), 3-7.

Clancy, M., Titterton, N., Ryan, C., Slotta, J., & Linn, M. (2003). New roles for students, instructors, and computers in a lab-based introductory programming course. *ACM SIGCSE Bulletin, Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, 35*(1), 132-136.

Cook, C. R. (1976). A self-paced introductory FORTRAN programming course. *ACM SIGCSE Bulletin, Proceedings of the Sixth SIGCSE Technical Symposium on Computer Science Education, SIGCSE '76*, *8*(3), 78-79.

Daly, C. (1999). RoboProf and an introductory computer programming course. *ACM SIGCSE Bulletin, Proceedings of the 4th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education ITiCSE '99, 31*(3), 155-158.

Etlinger, H. A., Goodman, G. I., & Plummer, C. (1981). FORTRAN: A self-paced, mastery-based course. *ACM SIGCSE Bulletin, Proceedings of the Twelfth SIGCSE Technical Symposium on Computer Science Education SIGCSE '81, 13*(1), 62-73.

Gill, T. G. (2004). *Introduction to programming using Visual C++.NET*. Hoboken, NJ: Wiley.

Gill, T. G. (2005a). Learning C++ 'Submarine Style': A case study. *IEEE Transactions on Education*, *48*(1), 150-156.

Gill, T. G. (2005b). Assignment-centric design: Testing the assignments not the lectures. *Decision Sciences Journal of Innovative Education*, *3*(2), 329-336.

Goold, A. & Rimmer, R. (2000). Factors affecting performance in first-year computing. *ACM SIGCSE Bulletin,* June, *32*(2), 39-43.

Hagan, D. & Markam, S. (2000). Does it help to have some programming experience before beginning a computing degree program? *ITiCSE Helsinki, Finland,* July, 25-28.

Holden, E. & Weeden, E. (2003). Software development: The impact of prior experience in an information technology programming course sequence. *Proceeding of the 4th Conference on Information Technology Education. Lafayette, Indiana,* October 16-18, 41-46.

Linder, W. H. (1976). COMPUTOR-TUTOR: From a student project to a self-paced CMI/CAI course". *Proceedings of the Sixth SIGCSE Technical Symposium on Computer Science Education SIGCSE '76, 8*(3), 57-60.

Malcolm, S., Teich, A., Jesse, J., Campbell, L., Babco, E., & Bell, N. (2005). *Preparing women and minorities for the IT workforce: The role of nontraditional educational pathways.* New York: AAAS.

Poindexter, S. (2003). Assessing active alternatives for teaching programming. *Journal of Information Technology Education*, *2*, 257-265. **http://www.jite.org/documents/Vol2/v2p257-265-25.pdf**

Prey, J. C. (1995). Cooperative learning in an undergraduate computer science curriculum, *IEEE Frontiers in Education Conference, 3c3*, 11-14.

Reges, S. (2003). Using undergraduate teaching assistants at a state university. *SIGCSE '03. Reno, Nevada.* February 19-23, 103-107.

Roberts, E. (2000). Strategies for encouraging individual achievement in introductory computer science courses. *SIGCSE '00. Austin, TX,* March, 295-299.

Roberts, E., Lilly, J. & Rollins, B. (1995). Using undergraduates as teaching assistants in introductory programming courses: An update on the Stanford experience. *SIGCSE '95, Nashville, TN,* March, 48-52.

Sackrowitz, M. & Parelius, A. (1996). An unlevel playing field: Women in the introductory computer science courses. *ACM SIGCSE Bulletin, Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education,* March, *28*(1), 37-41.

Wills, C. E. & Finkel, D. (1997). Study of a group project model in computer science. *IEEE Frontiers in Education Conference, T3C,* 299-303.

Wilson, B. & Shrock, S. (2001). Contributing to success in an introductory computer science course: A study of 12 factors. *SIGCSE. Charlotte, NC,* February, 184-188.

Biographies

T. Grandon Gill received his M.B.A. degree (with high distinction) and D.B.A. degree in the management of information systems from Harvard Business School. He is currently an Associate Professor with the University of South Florida, Tampa. His teaching areas have included programming, management of information systems, database design, the Internet, and case method research. His research interests include expert systems, organizational learning, and management of information systems (MIS) education and include numerous publications in prestigious journals, such as the *MIS Quarterly.* He has also done extensive programming in a variety of languages, and has designed and programmed a number of commercial software applications.

Carolyn F. Holton is a doctoral candidate in Information Systems and Design Science at the University of South Florida. She holds a B.B.A. in International Business from The George Washington University, and an M.B.A. also emphasizing international issues from Duke University. Prior to returning to academia, her professional life led her to several novel applications of IT, some of which were featured in Fast Company and other business magazines in several countries. In addition to MIS education, her research interests include the communications content impacts of monitoring computer-mediated communications, text mining of computer-mediated communications, and IS leadership.