Learning C++ "Submarine Style": A Case Study

T. Grandon Gill

***Abstract*—This case study describes a successful introductory course in C++ with a design that draws extensively upon techniques used in the training of nuclear submarine personnel. Techniques adopted include emphasis on completion of practical exercises as opposed to concept mastery, self-paced learning based on extensive materials prepared for the course, use of oral examinations to validate student achievement, use of undergraduate teaching assistants to assist and examine students, and a strong peer-learning focus with group collaboration being actively encouraged. Over the two-year period during which the course evolved, substantial increases in completion rates and the amount of material that is covered have been experienced. In addition, certain elements of the course design—particularly the emphasis on group work, use of online support, and use of "state-of-the-art" tools—seem more consistent with current programming practice than the conventional programming course, emphasizing lectures and completion of individual assignments.**

***Index Terms*—Group learning, introductory programming, management information systems (MIS), peer learning, self-paced instruction, undergraduate.**

I. Introduction

THERE is a wide gulf between how programming languages are most commonly taught and how they are most commonly applied in industry [1]. In many programming courses, the emphasis is on individual students writing small stand-alone programs using a generic development environment. The commercial programmer, in contrast, works in collaboration with other programmers, makes extensive use of existing code bases, and employs a complex, feature-rich, integrated development environment (IDE). This disparity is particularly troubling in management information system (MIS) programs, where a student's exposure to programming may be limited to a course or two.

In this paper, an alternative approach to teaching introductory programming is presented in the form of a case study of an introductory C++ programming course taught to undergraduate MIS majors at a large state university. The course evolved over a five-year period and differs from the traditional introductory programming course in a number of ways.

1) Group participation in completing all assignments is actively encouraged.
2) Students build assignments to specification, integrating an existing code base with their own code.
3) All programming is performed using the most current version of the world's most widely used IDE, Microsoft Visual Studio .NET.
4) Traditional testing has been eliminated in favor of highly focused assignment validation examinations.

The transformation of this course was modeled on the process used by the U.S. Navy to qualify its nuclear-trained submarine personnel. The paper first examines the training techniques employed by the nuclear submarine force. The design of the course and how submarine techniques have been incorporated into it are then presented. Finally, some outcome results of teaching the course are examined, and potential implications for other courses are discussed.

II. Nuclear Submarine Training

For decades, the U.S. Navy's nuclear training program has had a reputation for producing individuals—both enlisted and officers—of unparalleled quality in an environment of extraordinary technological complexity. To qualify to stand watch on a nuclear propulsion plant, an individual must demonstrate both a solid understanding of nuclear engineering theory and a mastery of the details of the specific propulsion plant—such as the physical location and function of literally thousands of valves, switches, gauges, and instrumentation displays.

Nuclear training begins with six months of nuclear power school, a classroom-oriented program intended to introduce students to relevant theory in the areas of physics, chemistry, thermodynamics, and engineering. Although more intense in its demands, course content and teaching approaches used during this phase are fairly similar to what might be seen in a typical college science course.

The second phase of nuclear training, used as the model for the course discussed in this paper, is known as the prototype phase. During their six months at prototype, officers and enlisted personnel train side by side on an actual nuclear propulsion plant. The training proceeds in manner listed hereafter.

- After a short period of classroom orientation to the specifics of the plant, students are given a detailed outline of the knowledge they must acquire and a multipage document (known as a qualification card or "qual card") with spaces for hundreds of signatures. Each signature represents either the satisfactory completion of an oral examination on a specific topic or the supervised performance of a practical task, such as standing a watch at a specific workstation.
- To acquire knowledge-based signatures, students use the outline and specially developed training manuals to prepare for their exam. When the student believes he[1] is prepared, he seeks out a qualified watchstander—normally a nuclear-trained sailor or officer assigned to the prototype for a tour of duty—who administers an oral examination.

Manuscript received December 19, 2003; revised May 26, 2004.

The author is with the Information Systems and Decision Sciences Department, University of South Florida, Tampa, FL 33620 USA.

Digital Object Identifier 10.1109/TE.2004.837044

[1]Currently, only men are assigned to nuclear submarines.

If the student passes, he receives a signature. If not, he must further prepare and take the exam again.

- Certain signatures require that the student demonstrate proficiency at a particular task, such as operating a specific piece of equipment or performing a particular chemical test. For these signatures, the student typically practices with one or more qualified watchstanders and then submits to an examination when he (and the watchstander) believes his proficiency is sufficient.
- Once all required signatures for a given watchstation have been acquired, the student becomes "qualified" to stand that watch. A student graduates from prototype by qualifying on all watchstations required by his rank and specialty (e.g., an officer would have a different set of requirements from those of an enlisted engineering laboratory technician). Those finishing early sometimes qualify on additional watch stations.

Upon completing prototype, the student's next stop is normally a short submarine school (although some are assigned to surface ships as well). Sub school focuses on the nonnuclear requirements of the submarine duty; upon completion, the student is assigned to his first submarine. Upon reporting for duty, he must then repeat the entire qualification process. The submarine qual card—similar to the one used by the student in prototype—includes requirements for both nuclear and nonnuclear signatures. The process for obtaining these signatures is, effectively, identical to the process used in prototype. The final signature on the card typically involves completion of a general oral examination administered by the captain of the ship. Once a crew member has obtained all signatures on the cards, he is "qualified" in submarines and is authorized to wear dolphins—the emblem of the U.S. submarine force—on his uniform. The submarine qualification process normally takes a year to 18 months, measured from the time the crew member first reports to the submarine.

The process through which submariners are trained is unusual in many ways. Most remarkable, perhaps, is that peers (as opposed to formally trained educators or academics) conduct nearly all training. A number of factors enable this process. First, a strong sense of community—driven by common purpose and mutual dependence—permeates the submarine force. Standards are high because every person on a submarine has a job that is critical; a single valve out of place or instrumentation reading ignored could lead to the loss of the ship. Second, there is a history of shared experience. Crew members administering oral examinations understand what these examinations need to accomplish because they have been through the same process themselves. Indeed, every time a submariner transfers to another ship, he must requalify on all relevant watchstations (although an abbreviated version of the qual card is normally used for senior personnel).

Some strong parallels exist between the job of a nuclear submariner and that of a professional programmer. Like today's programmers, nuclear submariners perform their jobs in a highly complex technical environment, must work effectively with their peers if they are to accomplish their jobs, are highly focused on quality, and are constrained by requirements of security and organizational procedures that must be followed. The recognition of these parallels led to the design of the programming course that will now be discussed.

III. Course Design

The course described in this paper is an introductory programming course, taught using C++. It is the first of a seven-course sequence that constitutes an undergraduate MIS major offered within a college of business at a large state university. Being offered as part of an MIS program, rather than a computer science program, impacts its design in some important ways, listed hereafter.

- Because students were unlikely to have subsequent courses in hardware or software architectures, C/C++ was chosen as an initial language, allowing a number of low-level concepts to be introduced (e.g., addressing, memory organization and representation).
- Because students in MIS programs tend to be employment-driven, rather than research-driven, the MS Visual Studio .NET tool—used by over 50% of all commercial developers [2]—was chosen in place of less complex tools that are easier to master.
- Because most MIS majors do not, ultimately, end up in careers as programmers, the ability to interpret and work with complex code was deemed to be more critical than the ability to write such code from scratch.
- Employers of MIS majors repeatedly emphasized the importance of teamwork skills. Thus, activities demanding individual program development, conducted in isolation, were deemphasized.

The challenges associated with incorporating objectives such as these into any programming course have been noted by a number of researchers and educators [1], [3], [4]. As a former submarine officer, however, the course designer felt that a number of the techniques he had encountered during his own submarine training could be applied to help achieve them.

The course was organized around seven assignments. Successful completion of the course depended entirely upon performance on these assignments, making them analogous to watchstation qualifications in the submarine program. Assignments could be prepared individually or in groups, with collaboration being encouraged. To get credit for an assignment, however, each student had to pass a validation examination, administered individually. This step was analogous to getting signatures on a qual card. The nature of the validation exam depended upon the nature of the assignment, as shown in the following examples.

- For pencil and paper assignments (e.g., numbering systems or interpreting data in memory), the instructor developed automated test generation software that created tests in a format that could be uploaded to the university's online course delivery system (Blackboard). These exams were then administered in a computer laboratory, supervised by teaching assistants (TAs). The score required to "validate" the assignment was curved according to the assignment grade. For example, a perfect assignment grade would require at least an 80% score before the assignment was validated, whereas a 75% raw score required only a

60% exam score. Students could take the validation exams again should they fail to achieve the necessary score.

- For proficiency exercises, such as an exercise in using the MS Visual Studio.NET debugger, students were required to recreate the assignment individually and gather relevant screen captures in the computer laboratory, proctored by TAs.
- For programming exercises, each student had to pass a one-on-one oral examination on the code he or she submitted (either individually or as part of a group). Initial oral exams were administered by TAs or the instructor. The course instructor administered all exams that were taken more than once.

Until a student validated an assignment, it was not entered into the course grade book, which was maintained online. Thus, an assignment pending validation was equivalent to an assignment that was never submitted.

The oral exams used to validate assignments were closely modeled after the corresponding exams administered in the submarine program. Their emphasis was on having the student demonstrate complete understanding of the code that he or she submitted, without involving any memorization. During the exam, the instructor/TA and the student both looked at the same block of code, and questions were posed, such as "What would happen if this line of code were removed?" or "What would the value of this variable be at the end of this block given the following inputs to the function…?" Only complete mastery of the code led to a pass; anything less required taking the exam again with the instructor on a subsequent date.

As part of the assignment-centered course design, traditional lectures were deemphasized. In their place, the instructor developed an extensive set of training materials, including a textbook, specifically designed to aid students in understanding and preparing assignments. In addition to the textbook, materials included software developed specifically for the course (e.g., a Windows-based graphic flowcharting tool that generated C++ code from student-created flowcharts), source code to be incorporated in assignments (with executable versions of complete assignments), and over 17 hours of multimedia clips that walked the students through code examples in MS Visual Studio .NET. Students tended to use these materials in a highly focused way, understandably spending the bulk of their time on those segments most directly related to assignment completion. These materials served a role similar to that of the training materials prepared by the Navy. They have also recently been packaged for students and instructors in textbook form [5].

Analogous to the submarine program, students were given considerable flexibility in how they fulfilled course requirements. The 75-minute weekly lectures were effectively optional (since attendance was not taken); and parallel versions, optimized for Web-based delivery, were also provided and could be downloaded 24 hours a day. Students could also choose to attend scheduled laboratory sessions, sprinkled throughout the week. Some of these sessions included step-by-step discussions of material related to the assignments. Others were much more informal, such as a TAs setting up office hours in an open laboratory and students dropping by with questions. Although due dates for each assignment were specified, a lenient policy for late assignments was established (10% of the maximum assignment grade deducted for each week an assignment was late).

The use of TAs in the course also closely paralleled the use of instructors during nuclear prototype training. All TAs were undergraduates, recruited from the pool of students who had recently taken the course. They provided one-on-one or one-on-group assistance to any student(s) who asked for it. They graded assignments and could administer first-try oral exams on all programming assignments. They also "stood watch" in the laboratory and proctored all laboratory-based validation exams.

One of the few aspects of course design that did not have roots in the submarine training model was its heavy reliance on the Blackboard course content delivery system. In addition to its obvious use in disseminating course materials (including online versions of the weekly lectures), the tool was used to keep track of student grades, administer automated validation exams, schedule oral exams, and—most important—serve as the infrastructure for online discussion groups that were set up for each assignment. In these discussion groups, students posted questions, anonymously if desired, relating to assignments or course policies. The instructors and all TAs monitored these boards regularly; therefore, median response times to student posts tended to be short (during 2002, the median response time was about one hour). Students could post questions relating to any aspect of an assignment and were also allowed to post code with which they were having problems. For larger assignments, several hundred postings are commonplace. Students were also encouraged to reply to the posts of their peers, with the active involvement in discussion groups being a principal technique for identifying potential TAs.

IV. Course Results

Over the past three years, a number of quantitative and qualitative indicators have supported the efficacy of the "submarine" approach to teaching introductory programming. The evidence comes in two forms: concrete indicators of achievement and qualitative indicators of process effectiveness.

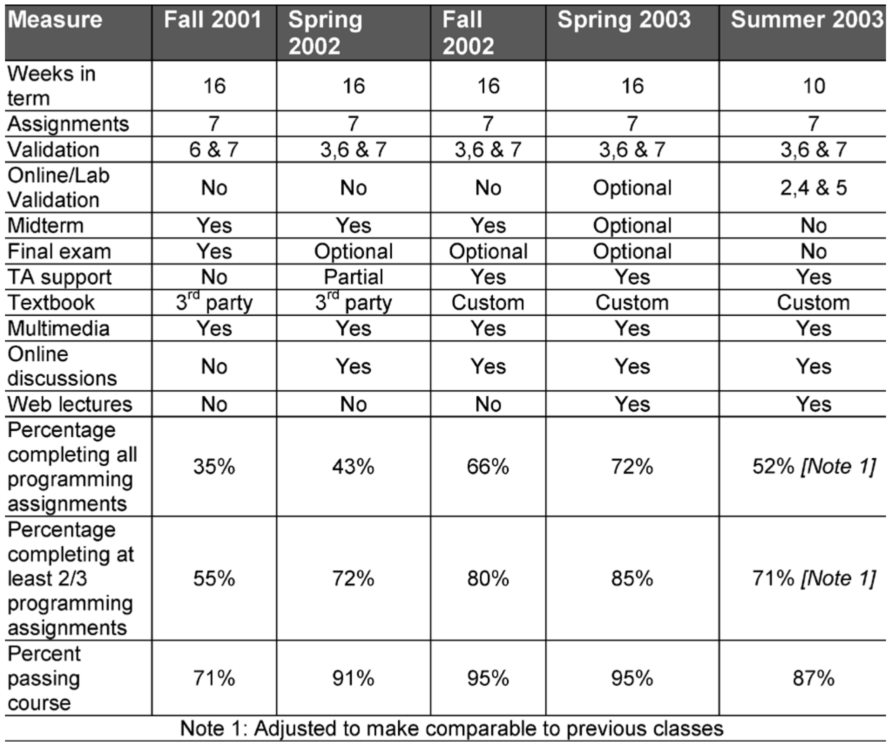
A. *Achievement*

With respect to effectiveness, the most important indicator is the percentage of students completing all assignments. To understand how the approach has impacted this metric, one needs to understand how the course evolved. This evolution is summarized in Table I.

The initial seven-assignment design of the course was established in fall 2001. At that time, oral exams were administered by the instructor to validate the last two programming assignments (6 and 7). No TA support was available, and an off-the-shelf textbook was required, primarily for reference purposes. About ten hours of multimedia content, developed by the instructor, were provided to students on a CD.

In spring 2002, a third oral exam (validating assignment 3) was added, and some TA support was provided for the second half of the course. Online discussion groups for each assignment, modeled after vendor technical support forums, were also

TABLE I
COURSE DESIGN AND PERFORMANCE SUMMARY



Measure	Fall 2001	Spring 2002	Fall 2002	Spring 2003	Summer 2003
Weeks in term	16	16	16	16	10
Assignments	7	7	7	7	7
Validation	6 & 7	3,6 & 7	3,6 & 7	3,6 & 7	3,6 & 7
Online/Lab Validation	No	No	No	Optional	2,4 & 5
Midterm	Yes	Yes	Yes	Optional	No
Final exam	Yes	Optional	Optional	Optional	No
TA support	No	Partial	Yes	Yes	Yes
Textbook	3rd party	3rd party	Custom	Custom	Custom
Multimedia	Yes	Yes	Yes	Yes	Yes
Online discussions	No	Yes	Yes	Yes	Yes
Web lectures	No	No	No	Yes	Yes
Percentage completing all programming assignments	35%	43%	66%	72%	52% *[Note 1]*
Percentage completing at least 2/3 programming assignments	55%	72%	80%	85%	71% *[Note 1]*
Percent passing course	71%	91%	95%	95%	87%

Note 1: Adjusted to make comparable to previous classes

Note 1: Adjusted to make comparable to previous classes

introduced. In fall 2002, the instructor-developed textbook replaced the third-party textbook, and full TA support was provided. In spring 2003, Web-based lectures were made available, and validation exams for nonprogramming assignments were pilot-tested. In summer 2003, the course was offered over a ten-week semester as opposed to a normal 16-week semester. In fall 2003, an object-oriented programming (OOP) module was added over the last four weeks of the course, and the final course design was realized.

Course Results: As the course evolved to its "submarine-style" design, a sharp increase in the percentage of students completing the course with an A or B grade (the 2/3 programming assignments row) occurred, followed by a leveling out as the amount of course content was increased. The only exception to this pattern was during summer 2003, where the time available for the course was substantially compressed. Because validation exams—particularly oral exams on the last two assignments—ensured that consistent standards were maintained from semester to semester, these increases in completion rates represented genuine gains in student performance.

One side effect of enhanced completion rates was the awarding of an unusually high number of "A" grades by the spring of 2003 since completing all assignments virtually guaranteed an "A." A natural concern resulting from such a distribution is that some students might not be sufficiently challenged. Educators have remarked that introductory programming courses often have a bimodal (or barbell) distribution, with a mix of very strong and very weak students [6]. Designing effective courses in the presence of such a distribution is very challenging [7], with solutions ranging from limiting access to the course to achieve greater uniformity [7], to redesigning courses based around Bloom's taxonomy of learning [6], [8].

In order to address the barbell issue, the instructor added an additional module to the course in fall 2003, a four-week introduction to OOP, compressing the original content accordingly. An additional assignment was added, and two of the original nonprogramming assignments were combined into a single assignment. Under this redesign, the requirements for a "C" in the course did not change significantly. To achieve an "A," however, a student needed to complete the final OOP assignment (or have nearly perfect scores on the preceding six structured programming assignments). This change increased course content by nearly 30%.

These measures provide compelling evidence of increasing student achievement as the course evolved. Such an increase does not, however, prove that the "submarine-style" model employed was necessarily the cause. As previously illustrated in Table I, many changes were introduced concurrently as the course design evolved. These changes make impossible the isolation of the effects of individual design modifications in a rigorous way. To address this issue, additional metrics measuring student perceptions of the learning process were gathered to assess the source of achievement changes.

B. Process Effectiveness

To assess the learning process, an end-of-semester survey was administered in spring 2003, and at the end of subsequent semesters. Students were offered extra credit for participating. The instrument drew a substantial number of its questions from

TABLE II
PROCESS-RELATED MEASURES FROM END-OF-SEMESTER SURVEYS

Technique	Process-related Survey Items and Outcomes
Emphasis on assignments	84% found assignments, in general, to be moderate to a great deal of help to their learning 65% disagreed or strongly disagreed that more emphasis should be placed on tests (19% agreed) 84% felt grading system was much help or very much help to their learning Students reporting spending 14.1 hours/week on the course, nearly twice as much time as reported spent on typical MIS courses, and nearly three times what they reported for typical business courses
Reliability of oral examinations	75% of student found oral exams to helpful in their learning Asked to rate "*The oral exam I took on Assignment X provided a fair assessment of my knowledge at the time*" on a Strongly disagree (1) to Strongly agree (5) scale, response means were 4.33 (Assignment 3), 4.26 (Assignment 6) and 4.14 (Assignment 7), all suggesting that students perceived the oral to provide an unbiased assessment of achievement
The effectiveness of undergraduate teaching assistants	75% found TAs to be helpful in their learning. Individual ratings of TAs were high, ranging from 3.5 to 4.6. Students estimated that they spent 4.7 hours per week interacting with TAs
The degree to which peer collaboration was encouraged.	90% were satisfied or very satisfied with group activities 71% felt that working with peers was much help or very much help in learning 75% felt their ability to work in teams was helped "a lot" or "a great deal" by the course 93% were either satisfied or very satisfied (64%) with online discussions
The quality of course materials	58% were satisfied or very satisfied with course text prepared by instructor 77% were satisfied or very satisfied with course handouts 84% were satisfied or very satisfied with course multi-media materials
The self-directed approach to the learning process	The was considerable evidence that a wide range of learning approaches was supported by the course design: 29% of students found having weekly lectures online little or no help, whereas 31% found them very helpful 33% found Blackboard to be of little or no help, while 28% found it very helpful 14% found class discussions no help, while 19% found them very much help No significant correlations between course achievement and learning approach used were observed.

previously validated course surveys.[2] Specific items were also added to gauge student perceptions of a number of submarine-inspired techniques, including the following:

- the course's heavy focus on assignments;
- the reliability of oral examinations;
- the effectiveness of undergraduate teaching assistants;
- the degree to which peer collaboration was encouraged;
- the quality of course materials;
- the self-directed approach to the learning process.

These results are summarized in Table II. The responses confirmed that students were aware of key design elements. They also suggest that, overall, students perceived these elements to be supportive of their learning process.

The results of the survey appear to confirm a number of findings in the existing literature. The effectiveness of undergraduate TAs has been documented in many studies [9]–[12]. Studies have also identified benefits, from allowing students to proceed at their own pace with assignment extensions readily given [7], to providing alternative paths through a course to support diversity of learning styles [13], [14] to emphasizing collaborative work [1], [15], [16].

V. POTENTIAL CONCERNS

In considering the adoption of the submarine approach to teaching C++, potential concerns need to be addressed. The first is that submarine training has historically been limited to men. Since some researchers have raised questions about teaching programming to men and women using the same techniques [17], adopting a strategy developed for training an exclusively male population must be initiated cautiously.

The experience of the course presented offers some comfort in this regard. Specifically, virtually no significant differences in outcomes between men and women could be detected in the survey data. For example, the survey data for all 2003 sections contained roughly 112 usable responses (varying by question), with a breakdown of 30% female and 70% male. Of 16 items relating to time spent on various tasks, assignments, and types

[2]See "Student Opinion Survey" (http://oerl.sri.com/instruments/cd/studcourse/instr16.html accessed on 4/14/2003), "Computer Programming Survey" (http://oerl.sri.com/instruments/cd/studcourse/instr11.html accessed on 4/14/2003), and "Student Assessment of Learning Gains (SALG)" (http://www.wcer.wisc.edu/salgains/instructor/ accessed on 4/14/2003). Copies of the survey instrument used to measure the course may be obtained from the author by email (ggill@coba.usf.edu).

of courses, not a single significant difference between male and female means was encountered. Of eight measures adapted from the Computer Programming Survey, one $p < 0.05$ measure was encountered, with women being more likely to agree with the statement that "The assignments contributed to my understanding of the course materials" (4.2 versus 3.7 on a disagree–agree 1-to-5-point scale). Out of 16 satisfaction measures derived from the student opinion survey, one $p < 0.05$ measure was encountered, with women reporting a slightly higher satisfaction with laboratory exercises (3.6 versus 3.1 on a dissatisfaction–satisfaction 1-to-5-point scale). Finally, out of 51 items derived from the Student Assessment of Learning Gains instrument, one significant item was found, with women reporting the assignment 4 debugging exercise to be more helpful than men (4.2 versus 3.8 on an unhelpful–helpful 5-point scale). Even the observed differences can probably be discounted; 75 items would be expected to yield about four items with $p < 0.05$ through pure random chance. Even the number of women who went on to become TAs in 2002–2003 (three out of ten assistants) proved to be representative of the course population as a whole.

Likely to be of greater concern is a second issue: Which skills are actually being taught? Submarine training, particularly nuclear training, has always emphasized comprehension and adherence to procedures versus creativity. Given the dangers inherent in running an underwater nuclear reactor, such an emphasis makes considerable sense—on a submarine! Could exclusive reliance on oral examinations to validate assignments lead to the same outcome in students? If so, the result might be students who are good at interpreting existing code but are less creative in their problem-solving abilities.

There is some evidence that such concern is warranted. Students completing the "submarine-style" introductory course appeared to do neither better nor worse in subsequent OOP courses (taught in a more conventional manner, with emphasis on lectures and tests) than students who took a conventional introductory course. For example, a regression analysis of performance on subsequent OOP courses found a high significance ($p < 0.001$) for the first course grade in predicting the second course grade, but whether or not the course was taught "submarine-style" proved to be insignificant. Unfortunately, too many issues are in play here to make a rigorous determination one way or the other. For example, the non-"submarine style" instructor was highly talented and had many years of experience. In addition, pedagogy sensitivity may exist, with some students thriving under one approach but not the other—independent of their programming skills. Furthermore, activities, such as instructors' curving the grade, could easily cause grades in subsequent courses to be dubious measures of "success" in the first course.

Because entry-level MIS graduates are more likely to be responsible for maintaining existing code than writing brand new code, the seriousness of such an outcome may be limited as far as MIS employers are concerned. For computer science programs, however, the nature of the skills being developed by the pedagogy is an issue that warrants careful future study.

VI. Conclusion

In teaching a difficult subject—such as computer programming—pedagogies that have proven successful from other fields can be worth considering. In identifying such potential teaching techniques, one must ensure that the objectives of the approach being used as a model are consistent with those of the material being taught. Because the objectives of nuclear submarine training—communicating a complex body of material with a particular emphasis on understanding existing systems, achieving high quality, promoting teamwork, and adhering to specifications—are similar to those desired of MIS undergraduates, testing how submarine training techniques could be applied in the MIS classroom was reasonable.

The results of the case study presented in this paper are promising. Use of the techniques has coincided with a major increase in the amount of material that has been covered. Student survey results support the conclusion that the increase in content can, to a large extent, be attributed to the adoption of the "submarine-style" pedagogy. The reader is cautioned, however, not to view the techniques described as a panacea for all the challenges of a teaching programming. Whether or not these techniques would prove effective in teaching creative programming and design skills has yet to be tested, and submarine training objectives (which place little or no emphasis on achieving these outcomes) provide us with little basis for assuming that the techniques will be effective. Thus, careful pilot testing is warranted before applying the pedagogy outside of introductory MIS programming courses. Given the benefits observed in the case study presented here, however, the cost and effort associated with initiating such pilot testing are likely to be justified.

References

[1] J. C. Prey, "Cooperative learning in an undergraduate computer science curriculum," in *Proc. IEEE Frontiers in Educ. Conf.*, 1995, 3c3, pp. 11–14.

[2] A. MacCormack and K. Herman, "Microsoft .Net," Harvard Business School Publishing, Case 9-602-086, 2002.

[3] E. Roberts, "Computing education and the information technology workforce," *SIGCSE Bull.*, vol. 32, no. 2, pp. 83–90, Jun. 2000.

[4] A. B. Tucker *et al.*, "Strategic directions in computer science education," *ACM Comput. Surv.*, vol. 28, no. 4, pp. 836–845, Dec. 1996.

[5] T. G. Gill, *Introduction to Programming Using Visual C++ .NET*. Hoboken, NJ: Wiley, 2005.

[6] R. Lister and J. Leaney, "Introductory programming, criterion-referencing, and bloom," in *Proc. SIGCSE 2003*, Reno, NV, Feb. 19–23, 2003, pp. 143–147.

[7] E. Roberts, "Strategies for encouraging individual achievement in introductory computer science courses," in *Proc. SIGCSE 2000*, Austin, TX, Mar. 2000, pp. 295–299.

[8] M. V. Doran and D. D. Langan, "A cognitive-based approach to introductory computer science courses: Lessons learned," in *Proc. SIGCSE 1995*, Nashville, TN, Mar. 1995, pp. 218–222.

[9] C. A. Twigg, "Improving quality and reducing cost: Designs for effective learning," *Change*, pp. 23–39, Jul./Aug. 2003.

[10] S. Reges, "Using undergraduate teaching assistants at a state university," in *Proc. SIGCSE 2003*, Reno, Nevada, Feb. 19–23, 2003, pp. 103–107.

[11] C. E. Wills and D. Finkel, "Experience with peer learning in an introductory computer science course," *Comput. Sci. Educ.*, vol. 5, no. 2, pp. 165–187, 1995.

[12] E. Roberts, J. Lilly, and B. Rollins, "Using undergraduates as teaching assistants in introductory programming courses: An update on the stanford experience," in *Proc. SIGCSE 1995*, Nashville, TN, Mar. 1995, pp. 48–52.

[13] L. Thomas, M. Ratcliffe, J. Woodbury, and E. Jarman, “Learning styles and performance in the introductory programming sequence,” in *Proc. SIGCSE 2002*, Covington, KY, Feb. 27–Mar. 3, 2002, pp. 33–37.

[14] A. Hirumi, “Student-Centered, technology-rich learning environments (SCenTRLE): Operationalizing constructivist approaches to teaching and learning,” *J. Technol. Teacher Educ.*, vol. 10, no. 4, pp. 497–537, 2002.

[15] N. Herrmann, J. Popyack, B. Char, P. Zoski, C. Cera, R. Lass, and A. Nanjappa, “Redesigning introductory computer programming using multi-level onlie modules for a mixed audience,” in *Proc. SIGCSE 2003*, Reno, NV, Feb. 19–23, 2003, pp. 196–200.

[16] C. E. Wills and D. Finkel, “Study of a group project model in computer science,” in *Proc. IEEE 1997 Frontiers in Educ. Conf.*, 1997, T3C, pp. 299–303.

[17] P. De Palma, “Why women avoid computer science,” *Commun. ACM*, vol. 44, no. 6, pp. 27–29, Jun. 2001.

T. Grandon Gill received the A.B. degree in applied mathematics (*cum laude*) from Harvard College, Cambridge, MA, in 1975 and the M.B.A. degree (with high distinction) and the D.B.A. degree in the management of information systems from Harvard Business School, Harvard University, Cambridge, MA, in 1982 and 1991, respectively.

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.

Dr. Gill has received numerous teaching awards, including the Florida Atlantic University award for excellence in undergraduate teaching.