Scheduling, Pair Programming, and Student Programming Assignment Performance

Clifford A. Shaffer, *Senior Member, IEEE,* and Stephen H. Edwards

***Abstract*—We seek to address poor performance by undergraduate students on major programming projects caused by procrastination and inadequate time management skills related to scheduling and pacing in project development. During Fall, 2006 we introduced two innovations into our Sophomore-level Data Structures course: Pair programming and a simple scheduling form. Our efforts do not appear to have had a significant difference in overall outcomes. However, the information gathered during this intervention makes a compelling case that students who spread their work over more time will produce a better result without additional expenditure of total effort. Pair programming did not have an identifiable impact on project outcomes or student behavior.**

***Index Terms*—Pair Programming, Scheduling, Extreme Programming, Time Management.**

I. Introduction

A Persistent problem for students in introductory computer classes is managing their time and effort on medium and large (multi-week) programming projects. Many students perform poorly due to poor time management skills. Procrastination is always a concern for students, but computer programs are especially unforgiving to those who tend to delay engaging the work. Last-night debugging marathons are a notorious part of the computer science curriculum. Unfortunately, many students view such desperate debugging sessions as a "rite of passage" that they believe to be inevitable, or at least normal. Unfortunately, this behavior often leads to failure.

While instructors routinely tell students to start programming projects early and spread the workload over time, admonishments alone historically have done little to affect student behavior. In part, this is because telling students what to do is not really teaching. When teaching a topic, teachers do more than simply state content. Instead, we require students to practice the material and/or behavior that we require. But when it comes to time and project management, students are usually allowed to do their programs on their own schedule without making them apply any particular scheduling behavior. Thus, in effect, instructors often do not actually "teach" good time or project management behavior. This is understandable given the inherent difficulties involved. Traditionally, programming assignments are done by students without direct supervision on the part of instructors, and so by its very nature it is difficult for instructors to exercise control over student behavior. Ideally, allowing students to do their projects independently of the instructor trains students to work on their own, which is considered a virtue.

Manuscript received November, 2008.

The authors are with the Department of Computer Science, Virginia Tech, Blacksburg, VA 24061.

This paper presents the results of interventions introduced into a sophomore-level data structures course (CS2606: Data Structures and Object Oriented Development II) in Fall, 2006 at Virginia Tech. We attempted to improve students' time-management behavior by doing two things. From the student's perspective, the bigger intervention was the introduction of pair programming for all projects. This choice was influenced by reports in the literature that indicate various improvements in student behavior when working in pairs (for example, [1]). We tried several variations on the details for how pair programming was implemented, as will be described. The other intervention was to require students to develop schedules with intermediate milestones, and regularly submit reports on their progress in terms of time spent on projects.

By "pair programming" we refer to the style embraced by "Extreme Programming" (XP) [2] advocates, where a pair of programmers work together at a single computer screen during all phases of work. While we did not explicitly monitor how students worked on their projects, we explained the XP pair programming approach and rationale, and made it clear that this is the form of pair programming that we intended for students to adopt. While we know from anecdotal accounts that some students applied "divide and conquer" approaches to their projects, most students report doing a significant fraction of their programming together.

Overall, our results do not show any observable impact from pair programming on attrition within the course, student grades on programming assignments, or student final grades (as compared to earlier course offerings). Students enthusiasm for pair programming was also mixed. These results generally contradict the literature on classroom use of pair programming [3], [4], [5], [1], [6] which indicate positive performance gains and student desire for pair programming. The literature also claims that females especially benefit from pair programming [7], but the female students in our courses showed no greater enthusiasm for it than males (though the numbers were too small to get meaningful statistics).

Students also did not show a desire for keeping schedules. However, students who did turn in schedules regularly scored higher on assignments, and students who reported putting in more time working on an assignment well ahead of the deadline also scored higher on assignments. We cannot tell from the data available if the scheduling exercise helped performance, or if stronger students are simply more likely to turn in scheduling data and work on their projects earlier.

However, our data do clearly show that good scheduling behavior is correlated positively with successful outcomes in programming assignments. This is the most significant result of our study.

II. Procedures and Data Collected

Two sections of CS2606 Data Structures and File Processing were offered in Fall, 2006. One was taught by Shaffer, and one by another instructor. There were roughly 30 students in each section. The two sections were given the same lecture topics, the same project assignments and the same homework assignments. Both sections had the same treatment with respect to pair programming and scheduling requirements.

The authors have taught this class numerous times over the years, and generally use a similar format and assignments. Thus we are intimately familiar with how students have performed in the past.

As is typical for this course, we required students to do four major programming assignments, worth a total of 45% of the semester grade. Students were given three to four weeks to complete each programming assignment. A significant fraction of class time was devoted to discussing the assignments, going over design issues, practical implementation issues, and answering student questions. To whatever extent the students have questions, we often allow the questions to drive that portion of the class.

During Fall, 2006, the assignments were as follows:

1) Implement a BST and a k-d tree as indices to a database of city records with names (a string) and two (integer) coordinates.
2) Implement a 2-3$^+$ tree along with the k-d tree as indices to a database of seminar records with many fields.
3) Implement a best-fit sequential memory manager along with a basic buffer pool as intermediary for a disk file.
4) Implement a B$^+$-tree storing a simple seminar record to a file with a buffer pool as intermediary.

We did not design the project list to be harder or different in quality than in previous semesters where we did not use pair programming. However, in retrospect, we believe this project list is a bit harder than what we typically give. Not vastly different, but maybe 10% harder.

In addition to grades on projects, homeworks, and tests, we also collected other information to help us assess what is going on. This included:

- After each of programming assignments 1-3, students wrote a one-page “reflection” document. The purpose was both to get students to think about the new programing and scheduling experience, and also to give us their feedback on it. Students got 1% of their semester grade for each reflection document.
- For the last project, the “reflection” document was a survey of questions answered on a 1-5 point Likert scale. This will be referred to herein as “the survey,” and is discussed further in Section VII.
- During each project we collected mandatory schedule sheets, which are our main source for primary data. These are described in Section VIII.

III. Semester Outcomes

For Shaffer’s section, 34 students began the class, and four dropped the class early in the semester, without penalty. One student withdrew from the course, which is a form of late dropping the class that has a “cost” in that students at Virginia Tech may only withdraw from a limited number of courses. Three students “gave up” the course without officially withdrawing, and so received a grade of “F”. These numbers are typical for this course. The final grade distribution was: 6 A, 2 A-, 3 B+, 2 B, 3 B-, 4 C+, 1 C, 1 C-, 2 D+, 1 D, 1 D-, 4 F. Thus, of 30 students receiving a grade, 21 “passed” (C or better by our system) and 9 “failed” (C- or below). Of the original 34 enrolled in the class, 21/34 or 61.8% successfully completed the course.

How does this compare with previous offerings of the course? Both the drop rate and the pass rate for this semester, while high, are within the expected range reflected in historical data for the course. In Spring 2006, 37 of 52 (71.2%) students completed with C or better (this includes 8 course withdrawals). However, note that this number *does not include* those who dropped. The closest available comparable number for this semester is 21/30 or 70.0% success. What this means is that the gross overall outcome is indistinguishable from the previous semester, and within the historical ranges over the past ten years.

IV. Pair Programming

Instructors have many options on the various details of how students form pairs. For the first assignment, the instructors assigned students to their partners. An effort was made to pair perceived “strong” students with “weak” students, based on knowledge of previous grades. For the second assignment, the instructors again selected the pair partners. Again we tried to make sure that two (perceived) weak students did not work together. It was discovered in retrospect that on both the first and second assignments, the instructors were not always successful in ensuring that two weak people didn’t work together. Despite instructors’ best efforts, it is not always easy to tell “strong” from “weak” students.

In the third assignment, all students were required to work with a partner. Students were given the option to select their own partner (provided they did not repeat a partner from a previous assignment), and the instructors picked partners for students who did not pick their own. In Shaffer’s section, only a few students picked their own partners for the third assignment, with roughly half of all students in the section asking the instructor to pick a partner for them. In the other section, students were more willing to pick their own partners.

On the fourth assignment, we allowed students to pick partners, or work alone, or put their name into a pool for the instructors to pick partners for them. 12 out of 49 students elected to work alone, and two students split their partnership in the middle of the project. One pair was allowed to repeat working together, but the rest were all new partnerships. In Shaffer’s section, 14 students asked the instructor to pick a partner for them, while five picked their own partners (one with a member of the other section). While the distribution was similar in both sections in terms of solo vs. pairs, we note that most of the students in the other section who desired a partner picked their own.

V. Class Forum

For many years, an online class discussion forum played a key role within our data structures course (see [8] for a detailed analysis of the forum discussion in an earlier year of this course). Since the projects are fairly difficult, and a major focus for the students, the class forum has traditionally provided a mechanism for students to discuss design and implementation issues. This opened up many opportunities for peer teaching and a generally high level of student interaction. It gave students an opportunity to receive more input regarding design issues than we would have time for in regular classroom discussions.

This semester saw a sudden decrease in class forum use. Unfortunately, this is a subjective observation by the instructors, since we do not routinely keep statistics on forum message counts and other objective information. But the decrease was substantial, perhaps dropping to only one half or one third of the previous level of student posts. In past semesters, the post rates were so high that students would complain about the sheer mass. This semester was completely different.

We propose two possible explanations for the great in student posts. (1) Fewer students in the course. Collectively, there were about 60 students in the two sections, where in the past we typically had over 100 students taking the course in a given semester. However, we doubt that this is a primary cause, because in the past we had much duplicate posting. This semester, the posts did not span typical discussion possibilities.

(2) Pair programming might be reducing the perceived need for discussion about the projects. Since students have somebody to bounce ideas off of and to verify spec conformance, they might feel less need to ask questions of the instructor or the class as a whole. Unfortunately, actual student understanding of the projects appears to be less than they think. More often than in the past, students misinterpreted the assignment requirements, and they were less likely than in the past to come up with good designs. We hypothesize that this is because they are not seeing as much input about possible design choices due to less student-directed discussion of design issues on the forums.

VI. Informal Analysis of Reflection Assignments

After completing each programming assignment, each student was required to turn in an assignment (each worth 1% of the semester grade) that encouraged them to "reflect" on the programming project just completed. Students were guided by a series of suggested issues to address in their discussion, but the documents were free-form prose.

An informal analysis of the reflection documents reveals the following. The level of acceptance for pair programming ran through great variation of opinion, from highly positive to highly negative. While the majority of students reported that they were in favor of pair programming, many expressed concerns about the relationship they had with their own partner. The number one complaint about pair programming is getting a bad partner. This did not change over the course of the semester.

The majority of students reported that they did not like to do the (mandatory) scheduling sheets. They did not perceive the self-scheduling process to be of much value. Partly, this seems to be because they focused on the time-estimation aspects of the scheduling process. Since they believed (correctly) that they are poor at estimating required time, their conclusion generally was that there was nothing else of value in the scheduling process. This is in sharp contrast to the instructors' primary rationale for conducting the scheduling process, which was to encourage (and monitor) starting work early on projects and spreading the workload throughout the three to four week period. There was a significant minority of students who said that scheduling was valuable to their success, and were greatly in favor of doing some form of scheduling.

During the first two assignments, we required students to spend at least one two-hour session in the departmental computer lab each week. For the first assignment, we required this time to be during a limited set of hours when the class TAs were available to monitor them. A significant majority of students didn't like having to go to lab, especially during constrained hours. For the second project we allowed students to go to the lab anytime it was open. A few students thought being forced to go to lab was really important, particularly as a mechanism to help them exercise control over a procrastinating partner. We abandoned requiring students to spend time in the lab for the third and fourth programming assignments. Perhaps coincidentally, student score averages were lowest on the last two assignments.

VII. End-of-Semester Survey

After the final programming project of the semester, students were given a survey to complete. 45 student responses were received. In this section, we present the questions and a summary of the students' responses.

The first question asked: *Over the course of doing your four programming assignments in this course, what percentage of the time that you spent working on the assignments were you actually with a partner?* The average of responses was 73.8%.

For the remaining questions, all answers were marked on a scale of 1-5, where 1 means "strongly disagree," 2 means "disagree," 3 is neutral, 4 means "agree," and 5 means "strongly agree." The questions and a summary of responses appears in Figure 1.

The results support our interpretation of the student reflections reported in Section VI. Both the survey and the reflections show that students were mildly positive about pair programming (neutral to agreeing that pair programming should be used and that it improved their performance). Both the survey and the reflections show that students were generally negative about keeping schedules (neutral to disagreeing that scheduling improved their performance or had other benefits).

Anecdotal evidence acquired in the semester since the course ended indicates that many students have concluded in retrospect that scheduling is beneficial to them. In particular,

Question	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Avg
I turned in better programming assignments and got better scores on them because I worked with a partner.	2	7	9	12	15	3.69
Overall, my partners did their fair share of work on the programming assignments.	0	3	7	25	10	3.93
Overall, I did my fair share of the work on the programming assignments.	0	0	2	24	19	4.38
I prefer to pick my own partner when doing pair programming.	1	6	13	9	16	3.73
Overall, I prefer pair programming to doing programming assignments alone.	5	7	9	11	13	3.44
I think that CS2606 should continue to use pair programming in the future.	6	5	11	8	15	3.47
I think there should be a requirement to spend some amount of time in the lab during programming assignments.	17	11	8	7	2	2.24
Being required to make an estimate of total time required to complete the project benefi ted me.	14	10	9	10	2	2.47
Being required to make an estimate of time required for the various parts of the project benefi ted me.	14	8	7	15	1	2.58
Being required to make a schedule of due dates for intermediate project milestones benefi ted me.	11	10	8	13	3	2.71
Being required to keep track of the hours I spent on the project benefi ted me.	10	10	11	12	2	2.69

Fig. 1. Table of survey results

students who had to repeat the course in the following semester report that they think the scheduling process is beneficial. Note that in Spring 2007, the instructor (not involved in the study reported here) agreed to require students to turn in schedule sheets.

VIII. Schedule Sheet Data

Aside from pair programming, the other major intervention used on this class was a requirement for students to complete and turn in scheduling sheets. This was done roughly once per week during the three to four week period for each project. Thus, students typically turned in four schedule sheets during a typical project cycle.

The schedule sheets were derived from [9]. They looked as follows. In the first column, students entered their own list of tasks required to complete the project. Students were given a lot of leeway in deciding how to break down the projects into tasks. The instructors provided a rough outline of tasks, but the students were expected to refine these further into subtasks, which were then listed in the second column of the schedule sheet. The third, fourth, and fifth columns required students to indicate which member of the pair was the coder, debugger, or tester for each subtask. Each member of the partnership was required to have primary responsibility for at least one of these for each substask. The sixth column indicated the self-imposed deadline that the students placed on that subtask. The seventh column indicated the initial time estimate (in hours) to complete the subtask. The eighth column indicated the current (revised) estimate for time needed to complete the subtask, which could be updated as desired by the students. The ninth column indicated the elapsed time (in hours) so far spent on the subtask, with the final column indicating the estimated time remaining for the subtask. The original time estimate was not meant to change, but students were expected to update the current estimate and elapsed time columns each week.

Assignment	Est. Hrs.	Early Hrs.	Final Hrs.	Score	Late
1	29.2	13.2	27.6	78.6%	27.3%
2	32.4	17.1	35.4	86.6%	20.7%
3	27.7	11.3	33.1	69.9%	10.3%
4	35.7	16.4	41.2	69.2%	23.1%

Fig. 2. Summary of schedule and performance data

Figure 2 shows average results from schedule sheets to provide a better picture of student behavior on the four programming assignments in the semester. The column labeled “Est. Hrs.” shows the average of the initial estimates of time to complete the project provided by students before they started work. The “Early Hrs.” column shows the average hours that students reported having already worked approximately one week before the assignment was turned in. “Final Hrs.” shows the average total time students reported working on each assignment once the assignment was turned in. “Score” shows the average scores achieved on each assignment, not including any early bonuses or late penalties. Finally, “Late” shows the percentage of students who submitted each assignment late (past the due date, incurring a score penalty).

To assess the impact of both pair programming and schedule use, schedule data were analyzed along with information about assignment scores, test scores, course final grades, and partner pairings.

A. Results for Pair Programming

The primary motivation for introducing pair programming in this course was to improve student success. Student success can take many forms, but the primary indicators of interest in this course were the rate at which students drop the course, the rate at which students successfully pass the course, and student scores overall. As discussed in Section III, these outcomes fall within the historical data for this course over the past ten years, so pair programming had no discernible effect.

We were able to compare student scores when working in pairs against student scores when working alone, since 29% of the students elected to complete the final assignment without a partner. Students working in pairs achieved a mean raw score of 65.7% on this assignment, while students working alone achieved a mean score of 66.5%. Thus, there was no significant difference in outcome observable from the decision of whether to work in a pair or solo.

We next attempt to measure the impact that a partner's ability has on programming assignment scores. If we use course test scores as an independent indicator of student ability, we can examine the extent to which a student's ability and/or their partner's ability contribute to assignment performance. An analysis of variance shows that both a student's own test scores, his/her partner's test scores, and the interaction between the two all have a significant effect on assignment scores (df = 88, F = 10.4, $\alpha < 0.05$). Higher test scores point to better programming performance, for both partners.

If we look at pairs where one of the partners drops out of the course, we see a negative effect. Partnering with a student who (eventually) does not complete the class reduces one's programming scores (df = 98, F = 6.89, $\alpha < 0.5$).

Overall, however, statistical tests reveal no significant relationships between participation in pairs and student test scores or student final course grades. Perhaps this is because students paired with strong partners had increased scores, while students paired with weak partners had decreased scores, to the extent that there was no net effect on scores from working in pairs.

B. Results for Schedules

Results from the scheduling activities were more positive. The motivation for requiring students to turn in schedules was to encourage students to pay more attention to their time management and to start working on assignments earlier.

First, we investigated whether paying more attention to scheduling was related to better scores on assignments. We compared students who turned in all their schedule information against students who turned in an incomplete set of schedule forms. (Students were penalized for not turning in schedule forms, which certainly affected the turn-in rate). Students who turned in all schedule data achieved a mean score of 83.4% while those who omitted some or all schedule data achieved a mean of 55.8%, a significant difference (df = 116, F = 45.32, $\alpha < 0.05$).

For corroboration, we also examined the schedule sheets turned in approximately one half to one week before each assignment was due, looking at the hours students reported having completed so far–the "Early Hrs." in Figure 2. Students who turned in this schedule sheet and reported some number for "Early Hrs." achieved an average of 80.8%, compared to 51.9% for those omitting this data or not turning in the sheet (df = 116, F = 30.63, $\alpha < 0.05$). From this, we can conclude there is a clear performance difference between the group of students who turned in scheduling information and those who did not.

Using data reported on the schedules, we also examined the amount of time students put in "early." Because of the wide variability in programmer abilities and the amount of time it takes them to write programs, we looked at the percent of total time students completed early, where any remaining hours were completed during the final week before the project was turned in. An analysis of variance shows that there is a significant positive relationship between the percentage of total hours that are worked early and assignment scores (df = 86, F = 25.89, $\alpha < 0.05$).

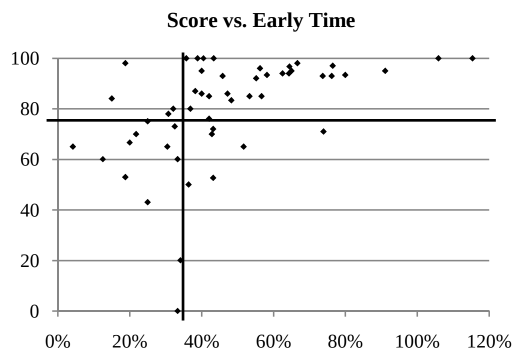

Fig. 3. Scatterplot showing student performance on projects versus fraction of original time estimate completed approximately one week prior to turning in the project. Project maximums (vertical axis) are normalized to 100 to be comparable. Time percentages (horizontal axis) can be greater than 100% since students might spend more time than originally estimated.

Unfortunately, this result is difficult for students to make use of, since they do not know how long it will eventually take them to complete the project. However, students are aware of their initial estimates (however flawed they may be). From these, we computed the percentage of the initially estimated time that had been completed "early." This measure is driven completely by information available to students in advance. The same positive relationship exists between the percent of the initial estimate that was completed early and the assignment scores (df = 98, F = 33.05, $\alpha < 0.05$).

Figure 3 shows a scatterplot of score versus percent of the original estimate completed approximately one week before the project was turned in. The horizontal line is the mean across all four projects (where project maximum scores are normalized to 100 for comparison). The vertical line is the optimal partition (the one with the least error after an exhaustive exploration of all possible partitions), which is 36.8% of initially estimated time completed before the last week. These two lines group most of the submission into the upper right (good) and lower left (bad) quadrants, with the upper left/lower right as sparse as possible. The scatter plot contains about 100 points, but only about 50 or so are visible because both schedule sheets and project scores went to pairs of students.

We see that about two thirds of the students scoring below the mean completed less than 36.8% of the estimated time in advance. The vast majority of submissions that scored below the overall class mean for programming assignments involved students who put in fewer than 50% of their initially estimated hours before the final week. Conversely, the vast majority of students who scored above the mean completed more than 36.8% of the estimated time in advance. In fact, this scatterplot greatly understates the effect of doing a significant percentage of the project early, since we are displaying raw scores. We gave a bonus for turning projects in early and a substantial penalty for turning projects in late. When these bonuses and penalties are included, the effect of doing work early becomes even greater.

We also compared both student initial estimates of total time needed, and final records of total time used, against project scores. There was no significant relationship found against assignment scores. Thus, there is no reason to believe that students who did well simply put in more time.

However, while there is clearly strong evidence that students who turn in and hold to schedules will then score well on the corresponding assignments, this does not necessarily mean that the act of scheduling causes better performance. To investigate this issue, we also examined the relationship between maintaining a schedule and overall student ability, using exam scores in the course as a separate indicator of student ability. An analysis of variance indicates that students who turn in schedules are also more likely to have scored higher on tests (df = 108, F = 7.64, $\alpha < 0.05$).

As a result, it is possible that stronger students, who are more likely to score well on programming assignments, are also more likely to complete schedule sheets and/or put in more time earlier when working an assignment. However, these results are also consistent with the idea that students who start work earlier will perform better. It might also explain why some students, in retrospect, have come to see schedules as beneficial, even if they resisted them in the beginning.

It is important to remind the reader at this point that our interventions had no significant impact on overall class performance. We believe that we are observing a very real and very important correlation between good time management practice and project performance. However, exhorting students to use good time management, and even requiring them to keep schedules, did not appear to be enough to change behavior (or, at least it didn't change gross outcomes).

IX. Conclusions

We present a number of conclusions related to pair programming and scheduling. With respect to pair programming, we have found no evidence to support any claims regarding it effectiveness. Pair programming did not appear, for us, to provide any of the benefits that we have seen presented in the literature. Pair programming did not appear to make students perform better. Partners did not appear to work harder or otherwise influence their behavior so as not to "let down" their partner (in contrast to the positive behavioral results discussed in [1]). *Taken as a whole*, the class seems to perform about the same if they work in pairs or do not work in pairs. We see no harmful effects from allowing students to work in pairs, including no evidence that one student will receive a better grade than deserved due to the work of their partner. We are willing to speculate that some students work better in pairs while others work better solo. Thus, it could be that giving students the option on whether to partner or not will provide some benefit.

Starting a project early and working steadily over time does not result in increased total time to complete the project, but it does correlate to better scores on projects. So it is clearly in a student's best interest to do this. Unfortunately, getting students to do so seems difficult. Students do not perceive much value in tracking their own time spent on projects, or in attempting to plan their time through scheduling. The simple act of requiring students to turn in schedules does not seem to influence their behavior much.

The scheduling process might work better if there were required intermediate deliverables, and feedback to the students about whether they are on track. It may also work better if students are explicitly instructed in where the value lies, and what the motivation for the activity is, so that they are less focused on the difficulty of estimating time requirements for software projects. In the future, we plan to improve such intermediate feedback.

An effective strategy might be to meet with individual students at some point during the assignment period (say a week before the due date). The purpose of the meeting would be to check in with them on their progress and make sure that they are on track. This could avoid misunderstandings of the assignment, since it gives them an explicit opportunity to discuss the assignment with the instructor. It also should further motivate them to get significant work done prior to the meeting.

One management option is to break larger projects into a series of smaller ones, thus requiring students to spread the work out. This approach does nothing to train students to use better time management. Instead, it merely puts off giving students experience with managing larger projects.

References

[1] L. Williams and R. Kessler, "The effects of "pair-pressure" and "pair-learning" on software engineering education," in *Proc. 13th Conf. on Softw. Eng. Educ. and Training*. IEEE Computer Society, 2000, pp. 59–65.

[2] K. Beck, *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley, 2000.

[3] B. Hanks, "Student attitudes toward pair programming," in *ITiCSE '06: Proc. 10th annual SIGCSE conf. on Innovation and tech. in comp. sci. educ.*, June 2006, pp. 113–117.

[4] E. Mendes, L. Al-Fakhri, and A. Luxton-Reilly, "A replicated experiment of pair-programming in a 2nd-year software development and design computer course," in *ITiCSE '06: Proc. 10th annual SIGCSE conf. on Innovation and tech. in comp. sci. educ.*, June 2006, pp. 108–112.

[5] A. Cockburn and L. Williams, *The costs and benefits of pair programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 223–243.

[6] L. Williams and R. Kessler, *Pair Programming Illuminated*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[7] C. McDowell, L. Werner, H. Bullock, and J. Fernald, "Pair programming improves student retention, confi dence, and program quality," *Comm. of the ACM*, vol. 49, no. 8, pp. 90–95, August 2006.

[8] S. Edwards and C. Shaffer, "An analysis of a course-oriented electronic mailing list," *Comp. Sci. Educ.*, vol. 9, no. 1, pp. 8–22, April 1999.

[9] J. Spolsky, "Painless software schedules," http://www.joelonsoftware.com/articles/fog0000000245.html, March 2000.

Cliff Shaffer is an associate professor in the Department of Computer Science at Virginia Tech since 1987. He received his PhD from University of Maryland in 1986. His current research interests include problem solving environments, bioinformatics, component architectures, visualization, algorithm design and analysis, and data structures.

Steve Edwards is an associate professor in the Department of Computer Science at Virginia Tech. He received his PhD from Ohio State University. His current research interests include software reuse, component-based software, automated testing, automated grading, and computer science education.