# First-Year Students' Impressions of Pair Programming in CS1

BETH SIMON
University of California, San Diego
and
BRIAN HANKS
Fort Lewis College

Pair programming, as part of the Agile Development process, has noted benefits in professional software development scenarios. These successes have led to a rise in use of pair programming in educational settings, particularly in Computer Science 1 (CS1). Specifically, McDowell et al. [2006] has shown that students using pair programming in CS1 do better in a CS2 class (with solo programming) than students who don't pair in CS1. This paper seeks to address a similar question, but from a qualitative, student-focused approach. How do students define, experience, and value the pair programming experience? How do they experience and value it compared to solo programming? Does pairing in CS1 impact their confidence in their abilities?

We report on semi-structured interviews with 11 subjects from two institutions where pair programming was used in CS1, and solo programming was used in the CS2. Many of the responses met our expectations; students get stuck less and explore more ideas while pairing, and believe that pair programming helped them in CS1. Other responses were more surprising. Students reported that when solo programming they were more confident and understood their programs better. Many students also said that they started work on their assignments earlier when soloing. Students also continue to use other students as resources even when working "solo."

Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computer and Information Science

General Terms: Human Factors

Additional Key Words and Phrases: Pair programming, CS1, novice, qualitative, debugging, interview

**ACM Reference Format:**
Simon, B. and Hanks, B. 2008. First-year students' impressions of pair programming in CS1. ACM J. Educ. Resour. Comput. 7, 4, Article 5 (January 2008), 28 pages. DOI = 10.1145/1316450.1316455. http://doi.acm.org/10.1145/1316450.1316455.

Authors' addresses: B. Simon, Department of Computer Science and Engineering, The University of California, San Diego, CA 92093; B. Hanks, Department of Computer Science Information Systems, Fort Lewis College, CO 81301.
This paper originally appeared in the Proceedings of the 3rd International Workshop on Computing Education Research, 2007, Atlanta, Georgia, USA.

## 1. INTRODUCTION

Traditionally, learning to program is a solitary activity. Pair programming (PP) transforms it into a collaborative learning process. Using PP in CS1 provides many pedagogical benefits. Compared with students who work alone, students who pair in their introductory programming course are more confident in their work, are more likely to complete the course, and are more likely to remain in or select a computing-related major within one year of completing the course as reported by McDowell et al. [2006].

PP appears to establish a strong foundation for students. Students who pair in CS1 continue to outperform their non-pairing peers, even when they begin working by themselves. For example, students who pair in CS1 are more likely to pass CS2 on their first attempt than students who work by themselves (that is, who solo program (SP)) in CS1 as reported by McDowell et al. [2006].

PP also appears to reduce the gender gap among students in computing-related majors. Female Computer Science (CS) students are typically less confident in their abilities than male students, even when their actual levels of competence are the same, as reported byMargolis and Fisher [2002]. This lack of confidence leads female CS students to doubt their capabilities and question whether they belong, and frequently leads them to select other majors. The gender gap in confidence is significantly reduced when the students PP as reported byWerner et al. [2004]. Women also are less likely than men to continue in computing-related majors. As noted earlier, PP increases the retention rate in computing-related majors for all students; the gap in retention rates is reduced when students pair as reported by McDowell et al. [2003].

Students find PP enjoyable and beneficial to their learning. For example, Melnik and Maurer [2002] found that students agreed with the statements "I personally like pair programming," and "I believe that pair programming improves software quality," while VanDeGrift [2004] reported that students agreed that they "enjoyed working with a partner on the pair programming projects in this course."

This prior research shows strong evidence that PP provides a strong foundation for learning to program, which is satisfying to students. However, why this is so has been largely speculative. In this study, we begin to address the question of why PP is successful by asking students to describe their experience with PP and to compare and contrast it to their SP experiences.

This article discusses PP from the students' perspective. Eleven students were interviewed to gain a better understanding of their experiences, and what they value about PP and SP. Section 2 discusses the methodology, Section 3 summarizes transcript analysis, and Section 4 discusses common themes present in the transcripts. Some concluding remarks are provided in Section 5.

Table I. Subject and Institution Data

| **Institution** | D | F |
|---|---|---|
| **Type** | Large, public research | Medium-sized, public liberal arts |
| **Students Completing CS1** | 122 | 16 |
| **CS1 Completers Enrolled in Next Course** | 108 | 6 |
| **Subjects (Men/Women)** | 6/3 | 2/0 |

## 2. METHODOLOGY

### 2.1 Subjects and Data Collection

We performed semistructured interviews with first-year students during their second programming course to ask about their experiences with, and valuation of, PP and SP.

Thirteen students at two institutions were interviewed. The subjects were nearing completion of the second programming course, and had taken CS1 in the previous term. Two of the interviews were not analyzed, one due to an incomplete audio recording and the second because the student worked alone in CS1. Information about the two institutions studied, the class sizes, and the subjects interviewed is shown in Table I. Both subjects from Institution F ranked in the top quartile of their CS1 class. Of the nine subjects at Institution D, three were in the top quartile, three were in the 2nd quartile, two were in 3rd quartile, and one was in the bottom quartile in CS1. Students completed a language background questionnaire that indicated little programming language exposure other than to Java (which was taught in their classes). Nine students were traditional college freshmen, one student was a sophomore, and another had returned to college for a second degree.

Both institutions used PP in the CS1 courses (which were taught by the authors) and used SP in the second course (not taught by the authors). At Institution D, which was using PP in CS1 for the first time, students completed eight pair assignments in a ten-week term. Institution D also required an individual interview for each assignment worth 10% of the grade. This was used to emphasize communication skills and ameliorate and identify possible dysfunctional pairs. At week 6 of 10, divorces were allowed and students could work solo for rest of term, though very few did. The class was 90% CS majors and 90% college freshmen in their first term, and the majority were Asian.

This was the third year that PP was used in CS1 at Institution F. Students worked with the same partner for the entire term, although there were a few changes due to student attrition. Students worked in pairs on six out-of-class programming assignments and 30 supervised labs in a 14-week term. When they turned in their programming assignments, students filled out short

worksheets to indicate the amount of time they spent driving, navigating, and working alone, and to assess the contribution made by their partner. The class primarily consisted of CS and IS majors and minors with 25% Native American students.

The interviews lasted between 30 and 60 minutes and consisted of five primary questions with a sixth to ask students if there was anything else we should have asked them. The first question asked students what they "learn from doing programming assignments" in contrast to what they learn from lecture, reading the textbook, or studying for exams. This question was used as a calibration and warm-up question and the results are not reported here. Four questions, with sub-questions, were asked about PP:

(1) Describe PP to a hypothetical first-year CS major who doesn't know what it is.
(2) Tell me about valuable and unsatisfactory PP experiences.
(3) Compare the PP and SP experiences.
(4) Describe your valuation of PP versus SP.

The fourth question included four parts: (1) discuss differences in the way you understand your program, (2) identify some aspect of PP that you value, (3) identify an aspect of SP that you value, and (4) describe how your confidence changed when you started SP.

### 2.2 Analysis

Both researchers read the transcribed, audiotaped interviews. After marking statements of interest, the researchers met and discussed the common content observed in the interviews. In addition to marking statements of interest in response to individual questions we developed a set of themes based on overall review of all the transcripts. These themes are also reflective of some of the issues that we, as instructors using PP in CS1 courses, found did and did not meet our expectations.

## 3. RESULTS

### 3.1 What is PP?

After an introductory question about what one learns by doing programming, we asked students to define and describe PP. This was accomplished through the presentation of a hypothetical scenario. We asked each student to imagine that the interviewer was a first-year CS student at a university that does not use PP. They were asked to describe PP to this hypothetical student, imagining that they were meeting after the conclusion of the first term of instruction. At times students were prompted also to address the physical acts or processes associated with PP—how you actually do it. Student answers to this question can be broken down into four categories: (1) general statements about what PP is, (2) statements concerning what one does in PP, (3) statements about the roles played or how switching occurs, and (4) statements about what happens when they get stuck.

3.1.1 *What Is It?* One of the most common explanations of what PP is refers to the increased resources of having two people working on the same problem. Specifically, four students made reference to having "two brains solving the same problem." [F3] These four students all made reference to the resource of the brain or mind of two people being better than one. This definition was also given in a more transactional way as "giving ideas to each other" [D6].

Another common definition focused on the second person specifically as a resource for alternate solutions to problems or processes for doing things (programming or solving problems). Descriptions of this nature included:

—[L]ooking at it from a different angle, and figuring out a different method to solving the problem and vice versa [D3].
—[S]ee how other people think about the code and differently they think compared to how I would do certain things [D5].
—It makes you see things from your point of view and also from the other person's point of view [D8].
—[A]nother person who can sit there and listen to you talk about ideas that you have for this code... could also take that input, decide whether they like it or not, and also add in their own input to it [D3].

Other remarks focused on the pair aspect of the process, calling it "collaborating" [D1] and a "team-building exercise" [D1]. D1 also discussed communication explicitly ("a lot of explaining") while D3 began by stating simply "It's a process."

3.1.2 *How Do You Do It?* Few students gave descriptions of what they actually did, unless prompted by the interviewer. These answers at least confirm that students' actual behaviors mostly match our instructor assumptions. Common answers discuss PP roles and are further described in Section 3.1.3. Descriptions of how to do PP are:

—You're sitting next to them but you're watching what they're typing. You're catching mistakes, whether they're logic errors or you're just getting a code wrong [F1].
—[Y]ou take turns either actually typing in code or you assist the other person.... You could figure out what they're doing and help fix their errors [D5].

Comments on how to do PP addressed how the students get started on an assignment. They said:

—We kinda separately read it (the assignment) and kinda try to figure out what is going on and then we would come together and discuss our ideas of how to approach the assignment [F1].
—[S]it down with your partner before you try and write code, and actually talk about what you're going to do... [it's a] lot easier and smoother to just go right to your code after you've talked about that [D1].

Communication was also a common theme including saying "for the most part we're talking about it and trying to figure it out together" [D10] and mentioning that you have to "set up a schedule" [D4].

One student emphasized the importance of roles and turn-taking, when she said, "Well, you sit on your hands. And you really try not to touch the keyboard when it's not your turn" [D1].

3.1.3 *What Are the Roles?* Several of the descriptions of PP focus specifically on the roles assumed in the process. Notably few students actually use the term driver or navigator and one student asked to be prompted with the term navigator during the interview. General descriptions of the roles and how they function include:

—[It] consists of a driver and a navigator. Of course the driver's the one creating the code and implementing it. And the navigator's checking for mistakes, or perhaps suggesting implementation techniques [D2].

—One person would type the code and the other would guide him, tell him what to do. We'd switch off every 30 minutes or so [D7].

—Two people working on it together and what we did was we took turns. One person would sit at the keyboard and type while the other person would sort of be the navigator telling them what to type. That person was always like the one to correct their mistakes, too, the person that was typing [D10].

There was also some additional focus on the role of the navigator—perhaps reflecting the need to describe the different experience compared to SP. The navigator was described as an "active participant," an "active listener," and "part of the conversation" by F1. F1 goes on to say that, "the active listener will generally have kind of a broader picture of where it's going."

One student specifically mentioned the required compromise needed in this process. In a general discussion of how to do PP this student reflects, "So you have to come up with a happy medium, or an alternate—a kind of compilation of your ideas and their ideas" [D1].

3.1.4 *How Does PP Impact Getting Stuck?* Novice students frequently get stuck while working on their programming assignments. Early in the interviews (while defining PP) the students began to describe the effect that working in a pair has on being stuck. The students discussed the process of PP and what happens when you get stuck:

—Usually the way you would split it up is if one person got stuck on a particular problem, like say it's a logic piece to the code that you're stuck on, the other person might try to jump in and continue coding and then you would switch roles [F1].

—If you get stuck on something, your partner may have a solution to your problem [D1].

### 3.2 What is useful about PP?

Within the same hypothetical scenario (imagine speaking to another novice programmer) we also asked students to describe specifically what was useful about PP. Some students addressed this naturally when describing what PP was, but often they would elaborate at this point. Students identified many ways that PP was useful, giving us general reasons, outlining specific conditions under which it was useful, discussing the value of having a partner as a resource, describing particular utility when they addressed bugs or got stuck, and identifying social or career benefits derived from the PP process.

3.2.1 *General Descriptions.* A number of general descriptions of the usefulness of PP were expressed. These general comments included mention that PP was "really handy" [F3], "easier" [D1, D10], "faster" (in reference to getting stuck) [D1], and "half the work to do" [D7]. Several general comments reflected the students' personal valuation of the process including saying it was "more pleasurable" [F1], "more educational" [D2], "a lot of explaining... which helped me" [D2], and as having "a nice balance to it" [D4].

3.2.2 *Conditions for Utility.* Students were often careful to identify conditions that were required for PP to be useful; most said these conditions existed in their own experiences, but some said they could imagine some teams might not experience these same conditions.

Compatibility between team members was important. Students felt it was important for "both people [to be] enthusiastic" [F1], and wondered "what it would be like to not get along" [D1].

One subject said PP was particularly useful "if you're struggling" [D1], adding "[y]ou have another brain. And hopefully, somebody who's not struggling the way you are". However, D1 also recognized utility for a partner on the other end of the spectrum, saying: "For somebody who is grasping the material, then it's useful, because you're able to make sure you're understanding it. Like I said, teaching someone else."

It is also noted that PP is useful "when a project requires a lot of time.... Because each will probably catch at least one of the other's mistakes throughout the entire project" [D2]. D2 goes on to make an analogy to a real world scenario that he sees as similar: "Given that people can switch, it's just like driving. My parents will switch off when we're going on a long road trip. And usually, I guess the project benefits, ultimately, from two people switching off at different intervals."

3.2.3 *A Partner as a Resource.* Students also noted how a partner served as a resource in various ways. General comments include "we could fill in each other's knowledge gaps" [F1] and that it's "hard to figure things by yourself, so have another person to help you and get ideas from each other, different ways to solve a problem" [D6].

Students discussed how the partner can help in the specific act of designing and coding: they "spot very easily identifiable things, as well as some of the larger things" [D1]. More conceptual issues can be addressed as well: There's

an "off chance that you might be able to help your programming partner with a fundamental concept, or vice versa" [D2].

Finally, comments such as we "worked off of each other's enthusiasm" [F1] also indicate a partner can also act as a non-technical resource.

3.2.4 *Usefulness in Debugging or Being Stuck.* Students recognize the utility in having a partner for "error checking" [F3], that "[t]hey can point out problems, or give you just direction in general" [D1].

These comments revisit the benefits associated with having two individuals (or brains) involved—this time in the debugging process. "There's a better chance of someone else catching it, because they are thinking differently than you're thinking" [F3]. But some students see the benefits of two sets of eyes as well—and critically assess the difficulties of debugging one's own code. "Physically just an extra set of eyes going over your stuff, because when you do something yourself, in your mind you engrave it as doing it right and no matter how many times you go over it and proofread it or debug it yourself, you'll never see anything wrong" [F3].

Along these lines another student identifies the possibilities of using a pair's abilities for learning:

> Either he knows what he's doing or is in just the same position as you. If he's in the same position as you, he or she, it's that you guys can grow together. I think that's the best thing about the paired programming [D4].

F1 finds that debugging with a partner makes for a better learning experience:

> The one thing I liked with a partner that I realized is that we could joke about it and then you would never make that problem again—that mistake again because you joked about it and you remembered. You were like, "Oh, I was about to make that mistake again," and you could kind of, you know, it was something that set it aside.

3.2.5 *Social and Career Benefits.* Finally, students also saw a variety of social benefits, including career benefits, to the PP experience. Note that the instructor at Institution D specifically emphasized the importance of career-skills development in the classroom (in lectures and in course design and structure). Socially, students reported that PP "makes it [programming] more of a group effort than a solitary piece" [F1] and that you "learn without having to just work on it by yourself" [D5].

From a career preparation perspective, students identified PP as a "team-building exercise, something you might encounter if you were in the computer science work world" [D1]. Another student said that PP:

> prepares us for the future 'cause in the future you're not gonna [be] programming by yourself. You're gonna be like working on—everyone's working on a different piece and they fit together in the end [D7].

Others discussed, the more general issue of group productivity. D3 says:

> [I]t's useful in terms of learning how to work with other people... here you actually learn how to communicate with another person effectively, how to bounce ideas off of each other, and how to pretty much coexist with each other.

He continues by comparing this to failed group project experiences from high school and argues that PP is a better experience, finishing with:

> So you have to learn how to work with other people and work efficiently with them, or else nothing's going to get done [D3].

One subject in particular mentions the importance of this skill for overly confident students, saying:

> ...the team building experience. That's something, I think especially in like a top university, that's difficult for a lot of people. There are a lot of perfectionists here, and a lot of people that have very much have a do-it-yourself mentality [D1].

### 3.3 Experiences in PP: When is it satisfying?

After the general question of what is satisfying about PP, we also wanted to focus on specific details or experiences of the PP process in action. We asked students to reflect back on some their PP experiences—in most cases reminding them of some of the assignments they had done. We then asked them to tell us about any specific instances in doing these assignments where they thought, "I'm glad I have a partner here right now." We also asked them why it was important to have a partner at that time. We saw several categories of answers, which are discussed below.

3.3.1 *Pair Events that Influenced Learning.* Several students mentioned particular PP "events" and then reflected on how that influenced their learning overall. F1 indicated that a particular problem-solving instance showed him alternate solutions and programming style:

> [Without my partner] I don't think I would have learned as much. I don't think that I would have come up with the same answer. I don't think I would have been exposed to his particular coding style, which is different.

D2 describes a particular instance where both he and his partner learned the importance of taking a break and reflecting. At that time, they also recognized the importance of gauging the time required to complete a task. The rest of the conversation indicates that these "long night" sessions were seen to be problematic, and to be avoided, by this team:

> And we had been switching off—off and on—and when we took a break at around 3:00 a.m.—we realized how valuable, I guess, your perception of time is when you're programming... You need to be

> able to accurately gauge the amount of time that the program's gonna take [D2].
> [You] understand how valuable, I guess, taking breaks is. Because just from that break, we learned something about our programming styles. And that's just from taking a break. And it's ironic how when we stopped working, we also learned something [D2].

D1 stated that she would sometimes get caught up in a problem and "spit out" code verbally. She did this in the middle of trying to discuss the problem solution with a partner in plain English because "at times like if my brain worked faster than my mouth" [D1]. However, she expresses the real benefit of having a partner:

> So the valuable part of it, I think, was not really the spitting out code. But then the going back and explaining what I just said... Like error checking for myself... It solidified the information for me [D1].

3.3.2 *Interacting.* Several of the specific instances reflect a two-way give and take of interaction among the team. Here we get a bit more detail on the process as experienced by the students:

> Basically, we just talked. We looked through the program, like the code. Then we try to go to figure out what area was the part that was giving us the most trouble. Then we just talk it through. Basically, we gave each other ideas like what we wanted to do and what we thought was a problem and then try to figure out ways how to solve that problem [D5].
> We would talk about what we did before, and we would talk about what we still needed to do, and it was just beneficial to have somebody else there who had thought about it himself and could say, "Well, okay. What if we try doing this?" [D3].

Another student expresses not so much the process as the outcome of the interaction:

> We were up doing homework late one night and we ordered calzones and we were discussing the project and there was a difficult piece.... It was like the hinging point for the program and we were able to—I guess it just jelled. We were able to figure it out together and we both had voices in building the answer [F1].

3.3.3 *Leaning on Your Partner.* Other partner interactions were expressed as more unidirectional. These fell into two categories, which sometimes were only evident in the context of the surrounding interview. Many of

the statements made about leaning on one's partner fell into a negative category—reflecting that the partner was used to cover up or hide one partner's weaknesses:

> Pretty much all the time I was thinking, "Wow, I'm really glad this guy's here to help me because I probably couldn't do it on my own.... If we got stuck, that's when I was always glad that we were pair programming because he would usually be the one to figure out what he did wrong [D10].
> I think he helped the most with having like the logical brain and so I don't know like the rational side. A lot of them [assignments] had like little equations I wouldn't have gotten otherwise [D8].
> I was like falling a little behind in my CSE knowledge. I was really thankful that he was able to cover me.... [When the course] went into the class stuff, I didn't really know about methods and return types and stuff, but he knew it [D7].

One student did have descriptions of a more positively-focused interaction during which one partner learned on the other:

> It just clicked for him and it was like, "Oh, that makes sense." And he just—it showed me something that I couldn't figure out.... There's some things that I was able to think through, but like some... it was kind of complicated and I just needed someone's help [D5].

3.3.4 *Social Impacts.* There were also several mentions of the social impact of PP. In one, F1 tells of the motivational factor of having a partner in times of frustration: "The drive, again, when you're working with somebody else where I think it was a late night and I probably would have just gone to bed." D3 expresses the importance of having a partner when you get that sudden list of errors: "We had, like, two errors. Filled those out or fixed those errors, tried to compile it again, and then out of nowhere, this whole list of 30 or something errors pops up. And we sorta just like looked at each other and just went aaayy."

3.3.5 *Coding, Getting Stuck, Getting It Done.* A few references were made to specific types of coding where pairs were useful. In discussing a first assignment with classes where various instance methods (accessors, mutators, etc.) were required, D2 describes the pair interaction:

> We spent a lot of time working out the different methods in each one, and having to redo quite a few of them. Because when one was wrong, a lot of the other ones were also wrong... just sheer rechecking.... That certain task was very detail-oriented.... And there was a lot of intense fact checking. So the fact that we could both spot each other's mistakes [was good] [D2].

A couple of statements explicitly addressed getting stuck. D5 says "he caught my errors and figured out what's wrong. Then vice versa." F3 adds the benefit of a pair's fresh viewpoint:

> [we] were there late and stumped, and we got a fresh idea or something, and tried something where the other person came up with it, just something you wouldn't have thought of on your own.

A couple of students expressed their task-focused nature with statements like "it [PP] helped finish the assignments" [D7] and "[PP] saves a lot of time sometimes when you have someone explaining it to you" [D6].

### 3.4 Experiences in PP: When Is It Unsatisfying?

Immediately after the discussion of when PP was valuable, we also asked students to reflect on particular experiences during which working in a pair was unsatisfying. Answers spanned a range from very pragmatic scheduling issues to more reflective concerns about pair interaction.

3.4.1 *Differences in Ability Level.* Although all of the students either had praise for their partners or expressed enjoyment or usefulness in working with their partners, many suggested that differences in partners' ability levels could be a negative issue. Even when they didn't express this dissatisfaction with their own experience, some students postulated that they could imagine it being a source of difficulty.

Many of the self-reported negative experiences identified partners who appeared to have some prior programming knowledge and, hence, either worked too fast without explaining, or just took things into their own hands:

> I think he already had some CSE classes so a lot of the stuff he was doing on his own I didn't really fully understand. So lots of times when it would be his turn he would type something out really quick and I would just sort of go along with it and then it'd be my turn and I'd have to have him helping me out with what I was doing [D10].

Another student (who did leave this partner) had a similar negative experience, but felt that it helped him define what he wanted to get out of a pairing relationship:

> [H]e just took over. He wasn't interested in any input from me.... It wasn't a good experience because I didn't feel like I was learning. I was watching him program. I didn't understand what he was doing.... [B]ut it was also a positive experience because I learned really quickly what I was looking for and also what role I needed to start taking [F1].

Some students let their partner dominate, then tried to figure out later for themselves what they had missed (in preparation for a required solo interview with a TA after the code was due):

> Sometimes I would go home and think, 'Wow. I really wish I tried to do this on my own so I could learn more about it.' Sometimes I'd come out thinking like, 'I don't—I'm not even sure about what we did.' Then I'd have to go home and like study the code before the interview thing that we had to do [D10].

Another student indicated that occasional accidental domination could be a problem when they fell victim to the driver/navigator roles. D6 complained that "we forget to switch and so you end up typing more and then so when I look at the code, I wouldn't really understand a lot of it."

One student very clearly developed a dependent relationship with his partner, which seemed to cause significant hardship. He indicates that at the beginning of the terms his partner's prior knowledge was something on which he depended. Then, as the term progressed, his partner also started to struggle. Here he describes the frustration of becoming dependent on someone, and then having that person be unreliable:

> [B]ut one of the downsides to it was that it kinda like deters you from thinking on your own; you're kinda like depending on the partner. So those added to the frustration [D4].

3.4.2 *Creative Differences.* Another unsatisfactory experience was noted, usually later in the term, when partners had different approaches to solving the problem in question. Though an instructor might imagine the discussion surrounding these experiences to be beneficial, the students definitely experienced them otherwise:

> [We] just both had our different thoughts and we were both really stubborn and like, 'No, this way. Do it this way. Do it the other way' [D5].
>
> If you're definitely not on the same page with that person. And you're trying to both do the same project two different ways, and you end up doing a hell of a lot more work than you need to, because you're trying to not necessarily incorporate each other's ideas but use both of your ideas and mash them together. It doesn't work, because they understand something better than you. You understand some things better than them, and it just doesn't work out very nice. It ends up being a lot more work than it's worth [F3].

3.4.3 *Stuck.* Frustration seemed to mount when both partners in the pair would get stuck. D7 stated, "when we're stuck we didn't really know what to do" [D7]. Co-dependence was raised again, this time as a negative:

> [W]e both didn't know how to do any of it, and we were both depending on each other to figure it out, but we could never figure out

> the solutions.... I couldn't think on my own. I kept depending on him. That was the huge thing. That's my only gripe about the [pair] programming is the dependence [D4].

3.4.4 *Scheduling.* The very mundane issue of scheduling was frequently mentioned. Some pairs seemed to have very explicit schedules about when to meet, but even that was a continued source of difficulty:

> The only thing I can really complain about is I guess are the times we were able to meet [D3].
> And when I called him, he had to reschedule. He said, 'Oh I'm sorry, I can't do it. I have to go see my parents or something.' ... once again, he had to reschedule, I guess the straw that broke the camel's back, that made me sort of frustrated with our pair programming relationship [D2, though he immediately goes on to say they had a good relationship].

## 3.5 Comparing the Experience of PP to SP

Next, we asked subjects to compare the experience of PP with their experience of SP. Here we sought comments on the particular experiences of the programming and homework assignment process and how these differed in pair versus solo scenarios.

3.5.1 *Approach and Procrastination.* Students were asked how their approach to a homework assignment differed. A number of students said they perceived no difference in their approach to assignments, though:

> I don't think that changed at all. I mean both times we just looked over the code (assignment?) and then just start typing. I don't think that changed at all [D10].

Others brought up a number of differences that they perceived to be beneficial to how they now work solo:

> I could just look over the handout myself and I don't have to worry about if my partner knows the stuff or if he understands.... So it's a little faster to get started [D7].
> I'm much more methodical about plotting my approach first now.... When we were pair programming, it's sort of the macho—let's get started. And we'll-figure-out-what-happens-later approach [D2].

But D5 felt that his approach was more organized with a partner than when he was solo, indicating there:

> was more brainstorming before when we did PP. It was like, 'Oh, we want to do x, y, and z.' Well, when I do it solo.... It's in the back of my head [but he doesn't do that as much].... then when it gets stuck, I'm like, 'I don't know what to do.'

Responses varied significantly on whether students tend to start earlier or later on solo assignments than on paired ones. Many responses indicated that scheduling, which may have been pair scheduling in paired assignments or course/life scheduling for solo assignments, determined when they started.

> We [in pairs] usually did it on the weekend. Sometimes we were really lazy about it. We kinda like gauged the difficulty [D4, who procrastinates less by himself].

Others were split. Three said they definitely procrastinate more solo:

> For me, later. I'm a procrastinator, but working in a paired situation, it definitely forces you to. I mean, unless the other person is a really bad procrastinator like myself, then we'll generally start working on it earlier. And I'm not inconsiderate [F3].
> I'm embarrassed. Yes I procrastinate more [F1].

Four students said that they actually start earlier in SP, and they seemed to value that. D2 starts earlier "just because I can." D1 cannot wait to start, saying "because I'm a total nerd, and I love coding—so I would go home the day we got our assignments and finish them that day."

Additionally one of the self-identified procrastinators still reflected on the freedom from pair scheduling with solo saying "when I'm in a mindset to do programming, it's like, 'Oh, I'll start doing programming'" [D5]. When he had a partner, there was a schedule and he had to wait. D10 commented similarly that with solo programming he has a less restrictive schedule that allows him more breaks:

> Whereas now it really depends. Since now I'm on my own schedule I start it whenever I feel like it. I usually plan it out like two or three days before. And work on it every day.

3.5.2 *Coding, Getting Stuck, Getting Unstuck. Coding:* Many of students' comparisons between paired and solo coding referred to low-level syntax and style comments. F3 notes that "every person does stuff differently. If you put a little closing bracket to a method in a different spot." D3 says, "I pay more attention to format, whereas I'm making sure everything looks as neat as I possibly can."

A few students mention that they tend to outline their work plan more (with code) when soloing:

> Yeah. I try to get like an outline first and then I'll go through and fill everything in [D10].
> [I] get a little bit more formal with it, making sure I don't have like parameter mistakes or return statement mistakes or errors or anything like that. So make sure to write some of those things beforehand [D3].

One student indicated he compiles more frequently solo—and later attributed his reduction in errors partly to that.

A number of students indicated that they seek out other resources more often when SP. Several use Google, with one stating "there's not really code online but there's usually algorithm" [D8]. D4 enjoyed the use of Google referring to it as "self-exploration." Others said they took "more breaks now. I talk[ed] to tutors more often" [D2]. D6 also talked more to tutors, but also reported "working with people a lot."

D1 reflected that with pairing there was "more verbalizing... [so in solo] I would say it to myself, even if I was coding. Because I was used to saying the code when I was typing it."

*Getting Stuck.* A few students report getting stuck more with SP, and having more trouble with it. D1 says "I probably get stuck more frequently. I've just been sitting at the computer too long, so my brain had stopped working through it. And I didn't know how to go any further." D5 reports a "lot more compiling and running the program to figure out where my errors were."

D3 is very clear about the trouble with being stuck without a partner:

> I got stuck. I sat there for hours trying to figure out what was happening, and then somebody noticed some small error that I had, and I fixed it, and everything worked. And I just sort of sat there and cried for a little bit.

F3 also states that bugs are "seen" faster and caught "easier" with a partner. And D1 follows up his preceding statement with the comment "another nice [thing] about pair programming is that when you're just done, and you're at your wit's end, somebody else might not be." D4 takes the opposite viewpoint by reflecting that when your partner is stuck too, things are also bad:

> I think I get stuck more solo programming. But it feels a bit more different when I'm stuck by myself then when I'm stuck with a partner. There's an added frustration if he doesn't help or he can't figure it out.

Others felt that several aspects of PP actually contributed to bugs:

> [You] almost get stuck more in paired programming because you're not going through your own—you're not just working inside your head and blasting out stuff.... You're going slower. You're thinking stuff over more, which, yeah, it could save you from making mistakes, but at the same time, I've seen if you go slower sometimes like that, you tend to trip up more and make more mistakes. If you're trying to work with someone else and you have different ideas about how to do something [F3].

D8 also indicated similar experiences saying "[PP] would give us more errors because he'd like type something and I'd be like you could do this and then that would interrupt his train of thought and so I got a little bit less errors now."

Some students realized that they couldn't accurately gauge the difference, indicating you "probably don't get stuck less often than before, maybe it just feels like it because there's not an open dialogue that's happening" [F1].

Others recognized that it was the development of their debugging skills that had changed [D2].

Many students commented on the frustration and stress of not having a partner:

> If you have a question about something you can't just like turn to someone next to you and ask. Your roommate has no idea about computer science. So you're pretty much on your own figuring everything out for yourself [D10].
> It's a lot more stressful, harder because you don't have that extra person to help you think about what's wrong [D5].

Like others, D5 reports asking others in the class (including his old partner) to help with debugging when he was stuck and D3 says when "you overlook some small detail, somebody else will be able to see it." D4 recognizes the frustration of debugging by oneself, but feels rewarded at the satisfaction of figuring out the fix himself.

> It forces you to look for it yourself, and kind of observe, actually read the code, and figure out what the hell's going on. [Debugging is] always frustrating, but it's not as frustrating because you solve things yourself, and there is that self-satisfaction like, oh, I finally figured the damn thing out [D4].

*Types of Errors.* Most students reported getting the same types of errors when SP as they did PP. F3 says, "Mistakes seem to be a habit." One student reported making more "typos. They're just stupid mistakes" [D6] when SP. Others reported that when soloing you make more "logic errors, you know, like the sequence of steps, say like in a method" [F1], or that errors were now a result of "learning new stuff" [D7] or was "specific to the assignment" [D3]. Interestingly, several couldn't remember what types of errors they got in CS1.

*Getting Unstuck.* Getting unstuck SP was less difficult then might be imagined. Quite simply, SP students reported finding other resources—often their old partners, or simply other students in the class—and asked for help.

> I get people to help me.... I usually just ask people around me. The TAs are the most helpful. The students have their own way of coding and that doesn't help me at all [D4].

D7 comments that getting unstuck in his second course was actually easier because "you have a lot more people."

Others report engaging some other tactics to help them when they are stuck. F3 says in a:

> solo situation, if you're really, truly stuck, then you've got to stop and dig through your notes or dig through your book or go ask a professor or something how to get out of it.... But in paired programming, if one person's stuck and the other not, then you can help each other out.

F1 supports this by noting that the duration of his being stuck is longer because he'll often have to "wait until I come back to class the next day to ask the teacher for help." F1 also uses non-expert resources, such as his domestic partner, noting "I'll walk her through the steps and she knows nothing about coding and sometimes it helps me, having to explain it to somebody else." Still others try tracing their code to "see exactly what's going on and make sure that there's no lines that I don't understand. I'll type a lot of different stuff typing in different things. I pretty much just try everything I can" [D10].

3.5.3 *Time to Complete Assignments.* We asked students to try to gauge whether assignments done using PP took more or less time to complete than if they were done solo. Many of the students indicated that this was difficult to judge, as assignments in their second course were "bigger" and "more complex". Those that did answer generally indicated that they were giving their opinion based on a hypothetical scenario involving assignments of comparable difficulty.

Answers were very mixed. Two students gave reasons why they thought PP took longer and then why they thought it took less time, and couldn't decide which was correct. A number of students gave answers similar to F1, stating that SP takes less time because "it's just me and I don't have to explain an idea or get a consensus or something." D3 mentioned that scheduling issues sometimes "made the assignment take longer."

Others said that programming with a partner took less time:

> [B]ecause [solo] you have to do all the thinking. You can't like partially depend on your partner. You just have to keep on going and pushing yourself. When you can't figure out things, it just—it takes a toll on you [D5].

3.5.4 *Reflection: Social Impacts and Learning.* Socially, students reflected on the "lonely" [D10] nature of SP saying they "spend a lot more time in my room" [D1] and that "it's usually less enjoyable... because you probably don't have someone with whom you can joke about the awful code that you guys are producing... less of a sense of camaraderie" [D2].

Stereotypes were also mentioned: "I started to understand what people meant when they started saying that computer science majors are like the people who are fat and lazy, sit at their computers all day long, staring at the screen" [D1].

A few students reflected on their learning in general during PP and SP assignments (they were not specifically prompted). Responses were mixed. One student said he was learning more "by doing everything by myself" [D10]. F1 disagreed and claimed to have evidence: "I feel like I'm not learning as much and it reflects – it reflected on my last test." He continued by providing this as a possible explanation:

> [Pairing] I was more apt to explore a lot of different solutions than just go with the first one that I came across. I'm not sure why. Maybe it was there is more brainpower actually thinking about different

> solutions but there was also the willpower and the drive to explore different options and now sometimes I just want to get the assignment done [F1].

### 3.6 Valuation of PP in Contrast To SP

In addition to asking for a comparison between the experiences of PP and SP, we asked the students a series of questions to try and elicit their valuation of various aspects of the two techniques. We tried to get them to focus on the subject of learning by asking about their understanding of code produced, specifics of what they valued about PP, what they valued about SP, the impact of SP on their confidence, and whether they thought PP helped prepare them for SP.

3.6.1 *Understanding of Resulting Code.* Almost universally, students said that they understood the resulting code of their homework assignments better or more thoroughly with SP. This was because they were "writing it" [D6] or "doing it" [D4] themselves. D10 says "some of the stuff that he wrote maybe I wasn't paying attention or I didn't understand fully." F3 says even if you were doing most of the typing "someone could be feeding you all of the ideas and concepts and if they're feeding you really fast and you type the entire time, then you burn through the program. The program could be done, and you might not have understood any of it." D7 explains that it was even possible for both partners to "get lazy and just like, 'Oh, we really don't know what we did, but it worked, so we're done'."

On the other hand, D8 claims to have received "a little bit broader understanding [with PP] because you'd see how he'd solve a problem and we'd run into more errors, I think." D3 felt that his partner taught him things: "He wrote it out for me and showed it to me, and I was like, 'Oh, okay. Now I understand how it works'." D1 didn't understand programs any differently when SP because "I felt like I did a lot of the algorithms [when pairing]."

3.6.2 *Benefits of PP.* When asked specifically about the benefits of PP over SP, three main categories of answers emerged: The power of the partner's brain, pairing as a good way to begin learning programming, and social benefits. Students reiterated that pairing means "potentially doing less work and it could be easier because you have two minds working on it" [F3]. They also appreciated that "when I can't figure something out there's someone to help me" [D6]. D3 goes on to contrast PP with SP, in which "it's just going on in your head or is stuff you're writing down, so you really don't have anybody to just sort of bounce ideas off of and see if they like it or if they don't."

Three students stated that PP is specifically valuable for beginning programming students. They say "it helps you get started" [D4] and that "you're both like new to it and you're both working together to figure things out" [D10]. F1 explains in more detail:

> [W]here coming in as a like a first time student doing programming for the—really, the first time ever, it's made my entry into other classes easier that don't necessarily utilize pair programming, but

> it's helped give me some skills. It's help build a group of people that I do know, of other students that I can go to for help or feel comfortable asking questions to.

Six other students also commented on the importance of various social impacts. They indicated that you "meet more people" [D7] and "you're not alone sitting at your computer" [D5]. D4 valued the fact "that you could build a relationship with a person, that you could grow together." D2 was happy that "you have a comrade who's willing to suffer with you. So you have a good time with you on the program. And they're pretty big things, in terms of morale." D1 also refers to morale when she said "I like to teach people. So with pair programming, I feel like I'm already contributing."

3.6.3 *Benefits of SP.* Three main benefits of SP were expressed: 1) scheduling, 2) creative freedom when producing or working on code, and 3) increased satisfaction or learning due to the individual effort.

Five students felt one of the benefits of solo was "creating your own schedule" [D4], which was expanded upon as allowing you to be "more methodical" [D4]. Others "needed to work at my own pace" [F3] or "procrastinate" [F3].

Four students reflected on the freedom of SP. F1 valued the "uninterrupted thought process. Like when you're working alone you're working through a problem and you're following different tracks on how to solve that problem. I like that discovery process." Others said they "[could do] whatever I want instead of asking a partner if he wants to work on it" [D7], "be more creative with your code" [D3], and "solve problems any way you want to" [D2]. D2 also commented on "freedom of style" with reference to formatting.

Eight students indicated that a benefit of SP was "the high, I guess, when you solve something" [F1] or that "you just have a better understanding of the code and Java overall" [D10]. Students felt more "accomplished" [D5] and "autonomous" [D1]. D8 indicated that he "learn[s] more because I have to do it myself and I can't like lean on him if I don't understand it."

3.6.4 *Impact on Confidence.* All but one student had a definite response to how the switch to SP impacted their confidence. Results were split, with more students indicating an increase in their confidence and the importance of solo programming in CS2.

Two students thought SP lowered their confidence in their abilities saying "when you don't have someone else there to work with you... you second guess yourself a lot more... so it's slowing me down" [F3]. Still, F3 ends up saying "but I guess that's okay." D5 states specifically that "struggling [solo made him less confident]."

Five students made statements about the increased confidence they have as a result of SP. D10 expressed a common opinion: "Well, I think it gave me more confidence in my programming ability because now I was actually doing all of it. I knew how to solve all the problems. I knew what I was doing more." D1 said that SP "reinforced" her confidence. D4 actually thinks "I've gotten like overly-confident, actually... it boosts your self-esteem a lot if you can do them fast and efficiently or whatever."

3.6.5 *PP as Preparation for SP.* Most students felt that PP helped prepare them for SP, either by learning how others solved problems or simply because as beginning programmers it was necessary to have someone else for support. F1 said, "It's helped me begin to look at problems in a lot of different ways because I worked with other people who attack problems differently."

Others indicated that it "was necessary" [D8] to have PP first, that "you learn [together]" [D4] and that it helped "get a feel for programming" [D3]. D1 (a very high achiever) reflected specifically back to the beginning of CS1.

> I think it was a good way to start. I didn't feel like I was totally alone in the lab. The first assignment we ever got—like, I don't remember exactly what we had to do. I just remember a whole bunch of jargon being thrown at me. And I felt like the whole rest of the class understood what was being instructed, and I was the only one in there that did not. And it turned out that everybody was kind of just as lost as I was [D1].

F1 appreciated that PP prepared him for his educational career in computing at his institution saying "it's built up like a base of students or a peer group that I can go to" and "it's helped me see how fun it can be, which is ultimately why I'm going to school for a career in something that's related to this."

One person felt he would have learned better by starting off solo and two people felt more help was needed in making the transition. F3 stated "My mind is the other way around. You've got to learn how to program alone before you can program with somebody else."

## 4. DISCUSSION

In this section we present a series of overarching themes identified from overall immersion in the data. Additionally, we discuss our results in the context of related prior work in PP and SP in educational settings.

### 4.1 Overarching Themes

4.1.1 *Comparisons between PP and SP. Theme: SP is more frustrating.* Students clearly state that they find some aspects of programming to be more frustrating when working alone compared to programming in pairs. This is especially obvious in discussions of debugging. Students in their first term of programming solo already recognize what experts know—that sometimes you just need to take a break or get someone else to look at your code. It's clear that deadlines for assignments often provide additional stress regarding debugging and that (as noted in Section 4.1.4) students will seek others out, including former partners, when they are stuck. At times of frustration, they describe SP as lonely and stressful and indicate that they are more likely to give up.

These very honest and emotional comments on the frustration of SP, experienced after an entire term of programming experience (in CS1), is a sobering reflection on the types of frustration that must be felt by students programming solo in CS1. These students have already seen many compiler errors and

have tasted the victory of completing an assignment, but still, they find SP very, very frustrating when things are going wrong.

*Sub-Theme: I get stuck more or find it harder to get unstuck when SP.*

In addition to being stressed when programming solo, students also believe they get stuck more often, more deeply, or have more trouble getting unstuck when SP. Again, students reflect on the value of having another brain, or even someone to remind them to take a break. Several students report long and fruitless solo debugging sessions where little was accomplished until something finally just worked (although they often didn't understand why at the time), someone else intervened, they sought help, or they simply quit for the time being.

*Theme: I explore more ideas or solutions with PP.*

Students clearly felt that the exploratory process and contemplation of alternate solutions was greater when PP than when SP. Students expressed this as a benefit of PP, saying that they could see how other people think and learn new or different ways to solve a problem. They expressed frustration that, when you work alone, the "gene pool" [D2] of ideas can get small. Some indicated that even though they may have had time to explore other solutions on their own, the temptation to just get their homework done meant that they rarely did so.

Many of the central concepts in CS1 are introduced via a range of programmatic structures. Conditional execution is taught using *if*, *if-else*, *if-else if-else*, and switch statements. Looping is taught using *while*, *do while*, and *for*. Part of what we hope that students come to understand is the grace, or lack thereof, of utilizing certain programmatic structures in certain scenarios. We want them to come to appreciate why it might be better to design a solution one way rather than another. We want to discourage the attitude of, "If it works, that's good enough, right?" These understandings are hard to develop. As such, PP seems to contribute greatly to this endeavor.

*Theme: SP may or may not take less time.*

Students reported varying beliefs on whether it took less time to accomplish a programming task solo versus in a pair. A couple of students said it took less time, stating that they didn't have to explain their thoughts or processes to anyone. They talk about "spitting out code" [D1] and just having an idea of how to solve a problem and running with it. But those who spoke most clearly about debugging frustration said that it took more time to solo program.

It seems likely that those who have the strongest grasp of the course concepts would, in fact, find SP to take less time. They are more likely to have a good first attempt at a problem solution and, given the size of beginning programming assignments, may find it faster to iterate using a code and debug cycle. Having a more recent and complete mental state of the intended program behavior may make it easier to debug. But as we know, a single bug can quickly suck away inordinate amounts of time—part of the rationale of increased efficiency of PP teams in industry.

4.1.2 *Difficulties. Theme: A big problem with PP is scheduling.* Students reiterated one of the most common findings regarding PP in education—that

successfully making time to work with a partner is a big problem. They told stories of both fixed schedules gone awry, and of more-flexible schedules (based on perceived difficulty of the assignment) turning out to be inadequate. These difficulties were often cited when we asked students if they ever worked alone on pair assignments. They said they just looked at the problem, or would set up a program "shell." When debugging dragged on longer than expected, one partner sometimes finished the assignment.

Even when most students live on campus and when labs are available for use 24/7, beginning students struggle to schedule their lives. Within the model of the American higher-education system, there is probably little to be done to alleviate this problem other than adding a fixed, required block of time (such as a lab or in addition to a lab) to guarantee that students will have time to get together to PP. An alternate solution would be to support non-collocated PP via technology.

*Sub-Theme: When to get started on SP assignments differs.*

We were not surprised to hear some students admit that they procrastinate more on SP assignments. Reduced procrastination is one of the unspoken benefits that PP in education should provide. So we were very surprised when almost all students at Institution D stated that they started homework assignments earlier when SP, due to not having to wait for a scheduled time to meet with a partner. Some students expressed their appreciation at being able to get an early start coding when the mood struck them. It should be noted that in the second course at Institution D, bonus points were available for submitting the homework two days early. Students did not uniformly acknowledge earning these points, but this policy may have influenced their attitude on the importance of starting early.

4.1.3 *Learning. Theme: PP is a good way to learn programming in CS1.* We found it encouraging that many of the students made statements that indicated that PP helped them learn to program, and that it gave them a sound foundation for subsequent SP. D3 captured the essence of this theme when he said, "Programming in pairs helped me a lot, so when it came time to program by myself, I was ready."

Although not explicitly stated, there was an underlying sense that PP is natural in CS1. It was as if the students were saying, "Of course you PP in CS1. How could you possibly learn to program any other way?" This sense was supported by the students during the interviews when they discussed the value of PP, how it prepared them for SP, and how it affected their learning.

None of the subjects stated that they felt PP was a waste of time, unhelpful, or interfered with their learning. From their point of view, PP was an excellent way to learn programming.

*Theme: Students feel that they understand their programs better when they work by themselves.*

It was clear that the students believe that they understand their programs better when they work by themselves. This belief is based on the fact that they have to complete the entire program on their own. Because they have written every line, they feel that their understanding is greater. (Note that this does

not mean that they are working entirely by themselves, as discussed in the next section).

Some of the comments suggest that the students relied too much on their partners in CS1. They would get the program to work as a pair, but individually they felt that their understanding of the solution was weak. We found this theme of particular interest because it appears to contradict evidence that shows that students do better in CS1 when they pair. It is difficult to reconcile this viewpoint with the empirical results shown in the literature. One would expect that many of the benefits of PP (such as greater success) would be associated positively with greater understanding of their work—this theme seems to conflict with existing PP literature. There were a small number of comments that contradicted this theme; for example, D1 pointed out that, "If you can explain the material to somebody else, then you really know it." This statement captures what we believe is an important aspect of PP: that it is a collaborative learning process. It is conceivable that, when pairing, students actually gain a good understanding of their programs without fully realizing it.

4.1.4 *Social Aspects. Theme: Students get a feeling of pride from completing programs by themselves.* There were several comments that highlighted a benefit of SP from the students' perspective: The pride and sense of accomplishment associated with completing a program by themselves. We were surprised that students did not express a similar sense of pride when PP, particularly in light of their statements indicating that it was less frustrating and helped them succeed when they might otherwise have given up. Perhaps being part of a pair "dilutes" the sense of pride. Or, it may be that this sense of accomplishment is greatest when students overcome particularly challenging problems, which are more frequent in SP.

*Theme: PP is a good way to meet my fellow CS students.*

Pair programming is clearly more social than SP. Several students identified this as a positive aspect of PP, especially in their first programming course. PP allowed them to establish relationships with peers who will be their classmates throughout the remainder of their college experience.

We conjecture that these connections are part of the reason that PP leads to increased student retention in the CS major. Through these connections, students see that there are others in the same situation—taking the same courses, suffering the same frustrations, and struggling with the same concepts. This reduces the sense of aloneness and not belonging that leads students to drop out of computing majors.

*Theme: Solo students get lots of help from other students.*

Traditionally, faculty expect students to work by themselves on programming assignments. If needed, they are expected to ask the instructor or teaching assistants for help. Requesting help from other students is implicitly or explicitly forbidden. However, students in this study were quite frank about their use of other students when SP. D6 confessed that, "I ask my friend to look over the code before asking the TA." D7 noted the prevalence of this behavior:

> Well, even though it's called solo programming, I don't much [think?] people really did it themselves. There was a lot of asking other people for help and stuff [D7].

It is natural to ask for help when faced with a difficult problem. For students, the most convenient resource is probably other students. They are familiar with the programming assignments, may be less intimidating than the instructor, and are more immediately available in the computing labs, particularly at off hours. It would be naïve to think that the students are not discussing the assignments with each other. In fact, some students gain a reputation for being particularly helpful. These students become a useful resource for the others, as noted by D4:

> [T]here's always one or two students who finish the code a lot faster than everyone else so you just ask them, 'Hey, can you help me out?'

PP encompasses the idea that students are effective teaching resources for each other by formalizing this relationship. When faced with solo programming, students still turn to their colleagues for assistance, regardless of any edicts against the practice.

### 4.2 Relation to Prior Work

In this study, students claimed to have a better understanding of their work with SP, largely because they have to write every line of code by themselves. While this seems intuitive, it appears to contradict much of the PP literature. For example, McDowell et al. [2006] found that students who paired in CS1 were more successful in CS2 than students who had worked alone. These same students were also more likely to persist in computing-related majors. Hanks et al. [2004] found that pairing students were more likely than solo students to turn in solutions to their programming assignments, and that these solutions were more likely to compile. VanDeGrift [2004] reported that students agreed that they "gained more understanding of concepts in the course by explaining them to my project partners." These findings would seem to be more consistent with greater understanding.

On the other hand, the students in this study felt that PP prepared them for SP, and that PP was a good way to learn programming. These impressions are much more consistent with the literature cited above.

The students in this study felt that scheduling problems were one of the most significant negatives associated with PP. This is consistent with results reported by VanDeGrift [2004]. On the other hand, the students in a study by Hanks [2005] reported few scheduling problems. These contradictory findings suggest that scheduling issues may be more related to environmental conditions (such as whether students live on or off campus or if scheduled lab time exists) than to PP specifically.

Students in this study had few problems with their partners. However, they envisioned situations in which they believed that PP would not work due to incompatible partners. Katira et al. [2004] reported that the great majority of

students are satisfied with their partners. The most frequent cause of incompatibility appears to be widely divergent programming skills as reported by Katira et al. [2004] and Thomas et al. [2003].

### 4.3 Threats to Validity

Although this study has provided some valuable insights, care should be taken when generalizing these results. This study focused on the students' subjective impressions of the PP to SP transition, with no objective measures of learning, motivation, or cooperation. The small sample size also limits generalizability.

The students may not have been entirely truthful with the researchers, who were the instructors of the CS1 courses taken by the students. The students may have censored themselves in an attempt to please the researchers. This may be particularly true at Institution F, where the CS department has only 3 tenure track faculty and students are very likely to take courses from the instructor again. However, as some of the results did not match our preconceived notions, we believe that the students gave honest appraisals of their experiences.

The subjects of this study may not be representative of the student population. No attempt was made to randomly select participants. At Institution D, subjects were solicited by a general email to all students enrolled in the second course and the volunteers included students from every quartile of CS1. Volunteers were compensated with $20 for their time. All six potential subjects were solicited at Institution F, although only three volunteered. We interviewed a small number of students from 2 institutions. There may be environmental and cultural differences between the two institutions that influenced our results. For example, differences in teaching approaches or student preparedness may lead to different views of PP and SP that are not apparent in this study. We did note one difference between institutions: The students at Institution D indicated that they procrastinated less when SP than when PP, while the Institution F students did not express similar behavior.

### 4.4 Considerations for Institutions Considering Pair Programming in CS1

Based on these interviews and their relationship with previous studies of PP, we have several suggestions for faculty and institutions that are considering the use of PP.

The transition from PP to SP is challenging for students. The challenge is greater when this transition occurs at the beginning of the second programming course, with a new instructor who has different expectations and evaluation criteria. One way to address this would be to have a transition assignment at the end of CS1, in which students would work solo. This would allow them to initially experience SP in a familiar and secure environment.

Students use each other as resources even when they are supposed to be working alone. Another suggestion is for faculty to identify those students who are the best mentors, and formalize their role. This would help ease the transition from pair to solo programming, while acknowledging that students ask for help from each other. Also, it would clarify the kinds of assistance that are approved and appropriate.

The students in this study felt that they understood their programs better when they worked alone. To encourage better understanding in PP students, we suggest that instructors ask students to individually explain their code, either orally or in a journal as done by VanDeGrift [2004].

Scheduling conflicts are a significant problem for paired students. This would be alleviated by required lab periods that students must register for as part of their schedule. These lab periods would be reserved for students to work on their programming assignments. Additionally, the student mentors should be available during these times.

Students who SP express a strong sense of pride when they overcome challenges and complete their programs. We believe that this sense of pride is a strong motivator for students. We have a challenge as educators: Can we instill the same sense of pride in group accomplishment as in solitary success?

## 5. CONCLUDING REMARKS

Pair programming has been empirically shown to provide learning benefits to students in their first course. The students in this study have qualitatively reinforced these quantitative results. They stated that it was beneficial to their learning and helped them be successful in CS1. Although they could imagine situations where PP would not be effective due to partner incompatibility, consistent with other results of Katira et al. [2004] and Thomas et al. [2003], these students did not experience many such problems.

Prior research has shown that PP is an effective way to learn to program. This study provides insights into why PP is beneficial from the students' viewpoint. Students believe that when pairing they explore more ideas and potential solutions and get stuck less often. They also felt that PP was a good way for them to meet other CS students, and believe that PP is a good way to learn to program. Students also identify some drawbacks associated with PP: They feel that it is more difficult to schedule times to work on their programming assignments, believe that they don't understand their programs as well, and have a lesser sense of accomplishment.

It is reassuring to us that the students in this study expressed viewpoints that were consistent with prior research. We believe that this provides further motivation for using pair programming in CS1. We also believe that this study provides insights that can guide further investigations into PP, perhaps integrating it with theories of learning and motivation.

## REFERENCES

HANKS, B. 2005. Empirical Studies of Distributed Pair Programming. PhD thesis, University of California, Santa Cruz.

HANKS, B., MCDOWELL C., DRAPER, D., AND KRNJAJIC M. 2004. Program quality with pair programming in CS1. In *Proceedings of the International Conference on Innovation and Technology in Computer Science Education*. Leeds, England, 176–180.

KATIRA, N., WILLIAM, L., WIEBE, E., MILLER, C., BALIK, S., AND GEHRINGER, E. 2004. On understanding compatibility of student pair programmers. In *Proceedings of the Technical Symposium on Computer Science Education*. Norfolk, VA, 7–11.

MARGOLIS, J. AND FISHER, A. 2002. *Unlocking the Clubhouse: Women in Computing*. MIT Press, Cambridge, MA.

MCDOWELL, C., WERNER, L., BULLOCK, H., AND FERNALD, J. 2003. The impact of pair programming on student performance, perception and persistence. In *Proceedings of the International Conference on Software Engineering*. Portland, OR, 602–607.

MCDOWELL, C., WERNER, L., BULLOCK, H. E., AND FERNALD, J. 2006. Pair programming improves student retention, confidence, and program quality. *Comm. ACM 49*, 8, 90–95.

MELNIK, G. AND MAURER, F. 2002. Perceptions of agile practices: a student survey. In Extreme Programming and Agile Methods. In *Proceedings of XP-Agile Universe.* vol. 2418, 241–250.

THOMAS, L., RATCLIFFE, M., AND ROBERTSON, A. 2003. Code warriors and code-a-phobes: a study in attitude and pair programming. In *Proceedings of the Technical Symposium on Computer Science Education*. Reno, NV, 363–367.

VANDEGRIFT, T. 2004. Coupling pair programming and writing: Learning about students' perceptions and processes. In *Proceedings of the Technical Symposium on Computer Science Education*. Norfolk, VA, 2–6.

WERNER, L., HANKS, B., AND MCDOWELL, C. 2004. Pair programming helps female computer science students persist. *ACM J. Educ. Resour. Comput. 4,* 1.

Received October 2007; accepted October 2007