# Exploring the Pair Programming Process: Characteristics of Effective Collaboration

Fernando J. Rodríguez Kimberly Michelle Price Kristy Elizabeth Boyer
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, Florida, USA 32611
{fjrodriguez, kimberlymprice, keboyer}@ufl.edu

## ABSTRACT
Pair programming is a collaboration paradigm that has been increasingly adopted in computer science education. Research has established that pair programming can hold benefits for students' learning and attitudes, but comparatively little is known about the ways in which the collaborative *process* benefits students' CS learning. This paper examines the collaboration process, comparing important outcomes with how students' dialogue and problem-solving approaches unfolded. The results show that the collaboration is more effective when both partners make substantive dialogue contributions, express uncertainty, and resolve it. In particular, driver dialogue expressivity is associated with improved outcomes. The findings provide insight into the ways in which pair programming dialogue benefits student learning during CS problem solving.

## CCS Concepts
**• Applied computing~Collaborative learning**

## Keywords
Collaborative learning, pair programming, block-based programming, student-student dialogue, problem solving

## 1. INTRODUCTION
An important focus in computer science education is to recruit and retain new, diverse students and to train those students as the next generation of computer scientists. With this goal in mind, computer science education researchers and practitioners have looked in part to industry to help identify skills and practices that are essential in computing careers. Collaboration has emerged as a central component of many computationally intense jobs and is now a component of many computer science curricula [1]. Pair programming offers a structured form of collaboration for computer science learning that has been successfully used in a wide variety of K-12 and postsecondary computer science courses [9, 16, 21, 22].

Evidence suggests that pair programming can improve students' programming efficiency. In fact, numerous studies have found that both code quality and efficiency of student pairs is much greater than solo students [3, 10, 15]. Some studies have reported increased retention after including pair programming [12, 13]. On the other hand, research suggests that some pairings do not benefit all students' learning [14] or attitudes [4, 17].


*SIGCSE '17*, March 8-11, 2017, Seattle, WA, USA.

DOI: http://dx.doi.org/10.1145/3017680.3017748

A pressing need in the CS Ed research community is to better understand the collaborative phenomena during pair programming—what is particularly effective versus what should be avoided—in order to benefit learners most effectively. Most pair programming studies, however, do not focus on fine-grained *process* data from pair programming. Instead, they tend to focus on summative outcomes, such as the number of programming exercises completed, course grade, and self-reported attitudes from interviews and post-surveys after the course. Only recently have studies closely examined pair programming behaviors and how they affect code quality and learning [7, 8, 16].

This paper makes a novel contribution to research on pair programming by examining the ways in which students' dialogue moves—such as providing feedback to their partner, asking questions, and conversational grounding—are associated with outcomes. In particular, we examine two complementary outcomes: quality of code produced by the pair during pair programming, and learning gain based on a programming-based pre-post test. The results demonstrate that active, substantial contributions by both partners are a characteristic of effective pair programming, and the expressivity of the student *driver* may be a particularly important factor.

## 2. RELATED WORK
Pair programming has been successfully implemented in many computer science courses, and research shows benefits to learners. Students have reported that pair programming gives them a glimpse into the collaboration that happens in the real world and how the perspective of a partner can help broaden one's own knowledge [4]. Pair programming also gives students a greater sense of responsibility, which translates to increased retention rates [13], even for non-majors [12].

In a study comparing individual students to paired students, individual students tended to postpone critical thinking activities, while student pairs spent the beginning phases of their programming task discussing their plan [8]. Some studies, however, reveal that students can have negative attitudes towards pair programming. Although students may praise pair programming for enabling discussion and planning, some students found it to be time consuming, particularly if partners' opinions differ and tasks are not distributed evenly [9]. Allotting extra time to pair programming activities, as well as providing feedback on collaboration from prior sessions, may help mitigate this concern [17]. Formally introducing pair programming best practices to students prior to a collaborative task has also been shown to improve the quality of the collaboration, encouraging communication and active contribution from both partners [22].

Most of these prior studies focused on the final outcome of pair programming, but recent research has begun to investigate the process of pair programming. Student behaviors have been video recorded and annotated during pair programming, revealing the

dominant actions such as giving direct commands, asking questions, and nonverbal cues [16]. Individual students have been observed to jump right into programming and address design concerns during testing, while student pairs spent nearly half an hour at the beginning of the task discussing the requirements before starting to program [8]. Another study focused on how inequitable actions, such as less conversation and focus on completing task quickly, can emerge even if curricula actively promote equitable collaboration between partners [7]. Our work builds on the body of prior work by providing further insight into the collaborative process during pair programming.

# 3. PAIR PROGRAMMING STUDY

In this study, 27 pairs of students from an introductory computing undergraduate course used *Snap!*, a block-based programming language. In *Snap!*, *blocks* represent program structures that can be *grabbed* and *snapped* together to create a program. *Snap!* is used in several introductory programming courses, most notably The Beauty and Joy of Computing curriculum [18]. We chose *Snap!* for this study because it offers the benefit of shifting the dialogue focus toward program structure and away from syntactical errors. We examine programming logs and annotated dialogues from paired students in an effort to better understand the collaboration process, and how this process relates to quality of the final code and to students' learning.

## 3.1 Participants

This study was conducted at a large public research university in the southeast. Students from an introductory programming course in Java for majors volunteered to participate in the study. A total of 54 students opted to participate: 14 female and 40 male; 16 Latino, 11 Asian, 1 Pacific Islander, 1 Black, and the remaining 25 White. Participants' ages ranged from 18 to 31, with an average of 19.6 ($\sigma$=2.21). The study took place a few weeks before the end of the semester. Participation in the study counted as partial credit for a homework assignment.

Participants were randomly assigned to pairs based on mutual availability of the partners and no other factors. They collaborated following the pair programming paradigm: one student created and edited the program (the *driver*), and the other student provided guidance and feedback (the *navigator*) [11]. These roles remained constant throughout the one-hour study session. Students were seated in separate rooms for the study. The drivers ran the block-based programming interface in their web browsers and shared their screen with the navigators via Google Hangouts. Navigators viewed the driver's shared screen along with web-based task instructions designed to support their role as navigator. They communicated through Google Chat textual chat. Figure 1 displays a screenshot of the driver's pair programming interface.

## 3.2 Programming Task

Before beginning the programming task, participants received a brief tutorial of the *Snap!* programming language and a "block guide" that included the locations and descriptions of blocks they might need. After this tutorial, we measured students' prior skill at solving problems in *Snap!* by asking them to build a program that generates two random values and displays the larger of the two values. We refer to this problem-based assessment as the *pre-test*. This assessment was done individually, and the students were given three minutes to implement the solution.

After the pre-test, student pairs completed the collaborative task, which was to implement a simple math tutor for kids in *Snap!*. The tutor would generate a mathematical equation, display the operands and the result, and prompt the user to choose the operator that correctly fills in the blank. This problem did present a challenge to the students, in part because they were not previously familiar with block-based programming, and also due to the mathematical nature of the task. Pairs were given one hour to complete the task, and the majority of pairs did not finish. Afterwards, they individually attempted the post-test, which was the same three-minute assessment exercise as used previously.

## 3.3 Programming Actions and Dialogue Tags

We logged students' programming actions within *Snap!* and their chat conversations. The programming actions logged are as follows: CREATE (adding a block to the scripting area); DELETE (removing a block from the scripting area); MOVE (grabbing and dropping a block in the scripting area); SNAP (connecting two blocks); UNSNAP (separating two blocks); PARAM (editing or selecting a block's parameter); CATEGORY (switching the block category in the palette); CAT_R (multiple CATEGORY events in a row); RUN (running the current program); RUN_R (multiple RUN events in a row). CAT_R and RUN_R were treated separately from CATEGORY and RUN events to capture repeated events in problem-solving strategies.

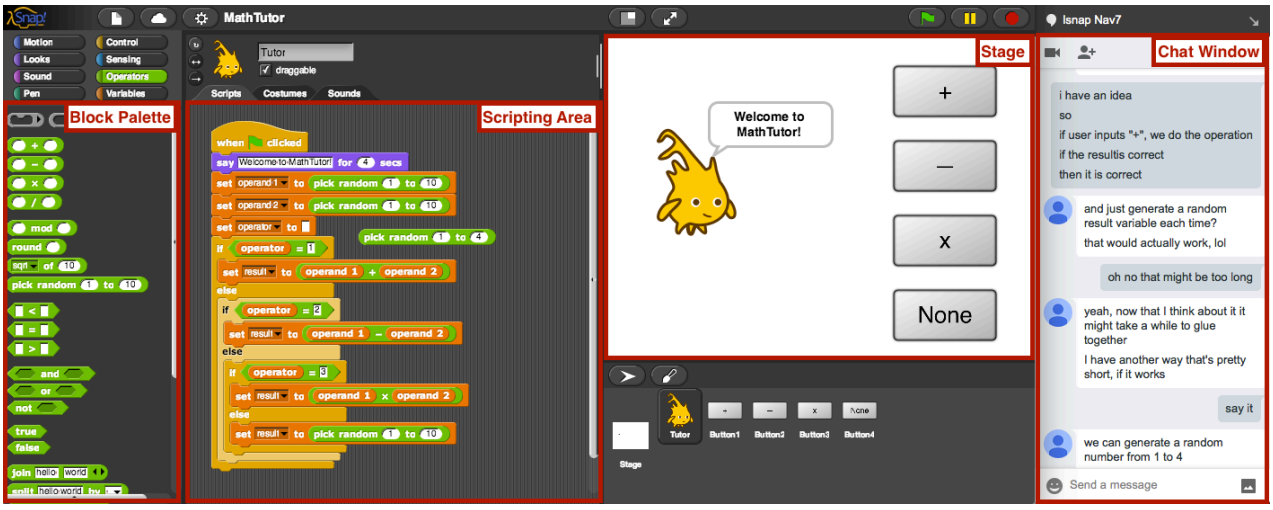


**Figure 1: Driver's pair programming interface.**

## 3.4 Dialogue Analysis

The complete chat logs were extracted from Google Chat. We manually tagged the dialogues with *dialogue act tags*. Our dialogue act classification scheme is shown in Table 1, and was inspired by prior dialogue tagging work for computer science collaborative problem solving, and for general conversational speech [19, 20]. Two researchers each tagged 70% of the data, covering all dialogue moves. The overlapping subset of 20% established the inter-rater reliability of the classification scheme. The Cohen's kappa metric, which controls for agreement by chance, was 0.73, indicating *substantial* agreement [6] and sufficient reliability for further analyses.

**Table 1: Dialogue act tag summary.**

| Tag | Description | Examples |
|---|---|---|
| S | Statement of information or explanation | *We need to create a program for kids to learn math.* |
| U | Opinion or indication of uncertainty | *unsure how to add strings together* |
| D | Explicit instruction | *wait put the if back* |
| SU | Polite or indirect instruction | *maybe we can do if user choice = +* |
| ACK | Acknowledgement | *oh ok gotcha* |
| M | Meta-comment or reflection | *hmmm* |
| QYN | Yes/no question | *can the answer be negative?* |
| QWH | Wh- question (who, what, where, when, why, and how) | *how do I take in their input?* |
| AYN | Answer to yes/no question | *yea* |
| AWH | Answer to wh-question | *the program should be able to generate erroneous questions* |
| FP | Positive task feedback | *oh nice* |
| FNON | Non-positive task feedback | *thats weird* |
| O | Off-task | *wow its sweet in this room* |

# 4. DATA AND LEARNING OUTCOMES

There were a total of 9335 programming events, with an average of 346 per session ($\sigma$=112.3, *max*=654, *min*=111), and 3438 dialogue act tags, with an average of 127 per session ($\sigma$=61.8, *max*=233, *min*=47). Navigators sent more chat messages, on average 87 per session ($\sigma$=53.4, *max*=200, *min*=28), while drivers sent on average 40 messages per session ($\sigma$=18.7, *max*=73, *min*=10), $p$=0.0002.

## 4.1 Outcome Metrics

We utilized two complementary metrics to quantify the outcomes of the pair programming interaction. First, we measured the quality of the collaborative code with a single task score for the pair of students. Second, we measured individual learning based on gain from pre-test to post-test for each student. We graded both the pre-post assessments and the collaborative programming task quality; this was done with a rubric that considered the functionality of the program and the types of blocks required to complete the given tasks. For example, a point was assigned for including an if/else block, and another point was given for correctly displaying the expected output.

**Collaborative Task Score.** The collaborative code rubric totaled 11 possible points. Across the 27 pairs, the average score on the task was 7.1 ($\sigma$=2.1, *max*=11, *min*=3). For further analysis, we divided the pairs into 3 groups: $high_{code}$ ($score \geq 9$, $n$=8), $medium_{code}$ ($6 \leq score \leq 8$, $n$=11), and $low_{code}$ ($score \leq 5$, $n$=8) performing pairs.

**Individual Learning Assessment.** The individual pre-post assessments were graded using a 10-point rubric. Drivers achieved an average pre-assessment score of 4.1 ($\sigma$=1.5, *max*=7, *min*=1) with average post-assessment score of 8.3 ($\sigma$=1.7, *max*=10, *min*=5). Navigators scored an average of 5.2 ($\sigma$=2.2, *max*=10, *min*=2) on the pre-assessment and 7.9 ($\sigma$=1.7, *max*=10, *min*=4) on the post. A pairwise t-test indicates a significant increase in score from pre- to post-assessment overall and for drivers and navigators separately ($p$<0.0001). Navigators had a higher pre-test score than drivers ($p$=0.0194), but drivers had a higher post-test score than navigators ($p$=0.0254).

**Individual Learning Gain.** In the context of this study, we use learning to refer to the improvement in programming skill, not knowledge of an abstract concept. To control for pre-test score, we calculated normalized learning gain as follows:

$$Normalized\ learning\ gain = \frac{Post\ score - Pre\ score}{Perfect\ score - Pre\ score}$$

Comparing the normalized learning gain from pre to post assessments between roles, drivers had an average learning gain of 0.73 ($\sigma$=0.30, *max*=1, *min*=-0.25), and navigators had an average learning gain of 0.43 ($\sigma$=0.35, *max*=1, *min*=-0.50). Intuitively, drivers closed 73% of the "gap" in their own pre-existing knowledge, while navigators closed only 43% of that gap. A *t*-test shows that the drivers had a significantly higher normalized learning gain than the navigators ($p$=0.0008).

**Pairwise Learning Gain.** In order to treat the pair as a single entity in the following analyses, the combined learning gain for each *pair* was calculated with the following formula:

$$Gain = \frac{(Post_{driver} + Post_{nav}) - (Pre_{driver} + Pre_{nav})}{2 * Perfect - (Pre_{driver} + Pre_{nav})}$$

Groups were divided based on the top, middle, and bottom third percentiles, dividing the pairs into three groups of 9. These are referred to in this paper as the $high_{gain}$, $medium_{gain}$, and $low_{gain}$ learning gain groups, respectively.

## 4.2 Session Metrics

We computed two metrics to capture collaborative events: absolute frequency of each dialogue act or programming action (e.g., "Pair 3 exchanged **four** questions," or "Driver 7 snapped 43 blocks into place.") and relative frequency by student (e.g., "**10%** of **Driver 7's** dialogue moves were *uncertainty* moves."). These metrics inform us in different ways about the process. Pure event counts speak to activity. Relative frequencies control for quantity while capturing how students chose to express themselves.

# 5. RESULTS

We compared the absolute and relative frequencies of each collaborative event across high, medium, and low pairs. All comparisons utilized the Wilcoxon rank sum test. These results

are detailed in Table 2 and Table 3, and summarized in Figure 2, which displays the events that were greater in count or relative frequency for higher or lower performing groups. Each of these findings is discussed in turn in the following subsections.

## 5.1 Collaboration Events and Code Score

As shown in Table 2, high performing pairs had significantly greater numbers of programming actions (particularly CREATE, SNAP, CATEGORY, RUN, and RUN_R) than low performing pairs, and more CREATE, SNAP, and RUN actions than medium performing pairs. RUN actions were also greater in relative frequency for high performing pairs. In contrast, MOVE actions occurred less often in high performing pairs. High performing pairs differed in dialogue moves as well, with more *driver positive feedback* than both medium and low performing pairs.

Medium performing pairs had more CATEGORY moves and *driver positive feedback* than low performing pairs, and fewer absolute and relative frequency of *driver uncertainty* moves. Interestingly, medium performing pairs had lower proportion of *driver uncertainty* than both high and low performing pairs. At the same time, medium performing pairs had more absolute and relative frequency of *navigator positive feedback* tags than low performing pairs.

**Table 2: Task score results. $^{+}$ indicates significantly greater than high. $^{*}$ indicates significantly greater than medium. $^{\wedge}$ indicates significantly greater than low, $p<0.05$.**

| Event | $High_{code}$ | $Med_{code}$ | $Low_{code}$ |
|---|---|---|---|
| CREATE (Abs. Freq.) | **$\mu$=89.0**$^{*\wedge}$ | $\mu$=70.0 | $\mu$=63.5 |
| MOVE (Rel. Freq.) | $\mu$=0.08 | **$\mu$=0.17**$^{+}$ | $\mu$=0.13 |
| SNAP (Abs. Freq.) | **$\mu$=95.1**$^{*\wedge}$ | $\mu$=74.6 | $\mu$=61.3 |
| CATEGORY (Abs. Freq.) | **$\mu$=47.8**$^{\wedge}$ | **$\mu$=39.7**$^{\wedge}$ | $\mu$=26.9 |
| RUN (Abs. Freq.) | **$\mu$=17.9**$^{*\wedge}$ | $\mu$=10.0 | $\mu$=5.6 |
| RUN (Rel. Freq.) | **$\mu$=0.04**$^{*\wedge}$ | $\mu$=0.03 | $\mu$=0.02 |
| RUN_R (Abs. Freq.) | **$\mu$=11.4**$^{\wedge}$ | $\mu$=4.5 | $\mu$=3.8 |
| Driver Uncertainty (Abs. Freq.) | **$\mu$=2.0**$^{*}$ | $\mu$=0.9 | **$\mu$=2.8**$^{*}$ |
| Driver Uncertainty (Rel. Freq.) | **$\mu$=0.04**$^{*}$ | $\mu$=0.03 | **$\mu$=0.13**$^{*}$ |
| Driver Positive Feedback (Abs. Freq.) | **$\mu$=3.1**$^{*\wedge}$ | **$\mu$=0.7** | $\mu$=0.5 |
| Driver Positive Feedback (Rel. Freq.) | **$\mu$=0.07**$^{*\wedge}$ | $\mu$=0.01 | $\mu$=0.01 |
| Nav. Positive Feedback (Abs. Freq.) | $\mu$=5.1 | **$\mu$=7.2**$^{\wedge}$ | $\mu$=1.9 |
| Nav. Positive Feedback (Rel. Freq.) | $\mu$=0.05 | **$\mu$=0.07**$^{\wedge}$ | $\mu$=0.02 |

## 5.2 Collaboration Events and Learning Gain

There were no significant differences in the number of *Snap!* programming events between any of the groups, but there were differences in dialogue. As shown in Table 3, high gain pairs had more driver's *acknowledgements* and navigator's *statements*, *meta-comments*, and *non-positive feedback* than medium gain pairs. They also had more *driver statements* than low gain pairs. Medium gain pairs had more *navigator off-task* messages than high gain pairs.

**Table 3: Learning gain results. $^{+}$ indicates significantly greater than high. $^{*}$ indicates significantly greater than medium. $^{\wedge}$ indicates significantly greater than low.**

| Event | $High_{gain}$ | $Med_{gain}$ | $Low_{gain}$ |
|---|---|---|---|
| Driver Statements (Abs. Freq.) | **$\mu$=11.4**$^{\wedge}$ | $\mu$=6.6 | $\mu$=4.4 |
| Driver Statements (Rel. Freq.) | **$\mu$=0.23**$^{\wedge}$ | $\mu$=0.16 | $\mu$=0.11 |
| Driver Acknowledgements (Abs. Freq.) | **$\mu$=8.4**$^{*}$ | $\mu$=4.0 | $\mu$=6.3 |
| Nav. Statements (Abs. Freq.) | **$\mu$=43.3**$^{*}$ | $\mu$=20.2 | $\mu$=24.2 |
| Nav. Meta-comments (Abs. Freq.) | **$\mu$=4.6**$^{*}$ | $\mu$=1.0 | $\mu$=2.3 |
| Nav. Non-positive Feedback (Abs. Freq.) | **$\mu$=3.8**$^{*}$ | $\mu$=1.6 | $\mu$=3.4 |
| Nav. Off-Task (Abs. Freq.) | $\mu$=2.7 | **$\mu$=6.1**$^{+}$ | $\mu$=5.1 |
| Nav. Off-Task (Rel. Freq.) | $\mu$=0.02 | **$\mu$=0.10**$^{+}$ | $\mu$=0.04 |

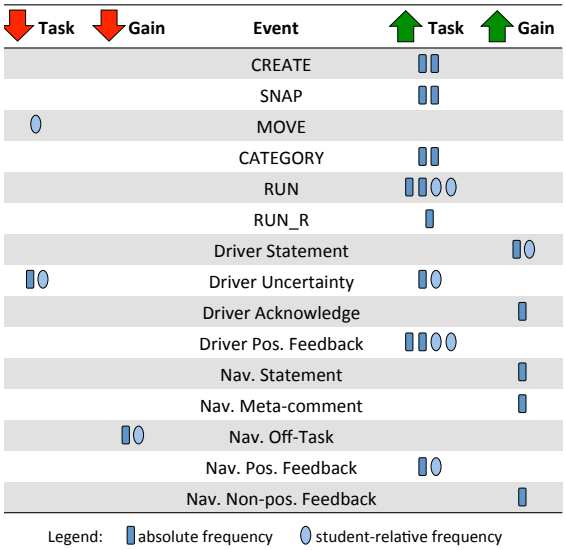


**Figure 2: Summary of event comparisons and relation to task score and learning gain.**

# 6. DISCUSSION

These results highlight several aspects of successful pair programming that may have important relationships with desired outcome.[1] In particular, increased frequency of several dialogue

[1] Note that this exploratory study is correlational in nature, so the discussion will not draw causal relationships between pair programming events and outcomes.

move types was positively associated with problem-solving outcomes.

First, with regard to overall frequency of dialogue moves across both drivers and navigators, *feedback* of any kind (positive and non-positive) was associated with better outcomes. Providing feedback to each other is an important mechanism for collaborators to establish common ground, and it may serve the function of reconciling the collaborators' mental models about the subject matter or about each other's knowledge [14] (Excerpt 1). Another dialogue move that was associated with better outcomes is *meta-comments*, such as acknowledging a previous mistake or thinking out loud. Excerpt 2 shows a $high_{gain}$ pair navigator using a meta-comment to let the driver know that she is thinking about the recently asked question.

**Excerpt 1: Positive feedback from $high_{code}$ pair.**

| Speaker | Tag | Message |
|---|---|---|
| Nav | D | wait put the if back |
| Nav | FP | okay good |
| Nav | SU | i think it should be " if the operand user choice operand2 = result |
| Driver | S | i cant do that |
| Nav | FP | you're on the right track, I think this will work |

**Excerpt 2: Meta-comment from $high_{gain}$ pair.**

| Speaker | Tag | Message |
|---|---|---|
| Driver | QWH | where should that be slid in to? |
| Nav | M | hmm |
| Nav | AWH | it should say operand1 _ operand2 = result |
| Nav | S | the _ is for the operator |

Increased absolute frequency of both driver and navigator *statements* were associated with better outcomes, and these dialogue moves indicate content-related talk about the task at hand. Excerpt 3 shows such an interaction between $high_{code}$ partners. These dialogue moves go hand in hand with increased task activity, which was also positively related to student outcomes. Pairs who engaged in this more substantive talk showed better learning gains, and those who engaged more actively in problem solving attempts (as evidenced by increased frequency of *create, snap*, and *run* events, among others) fared better on programming task score. The benefits of active, on-task programming actions and dialogue are echoed in the finding that navigator off-task moves are negatively associated with outcomes.

**Excerpt 3: Statements from $high_{code}$ pair.**

| Speaker | Tag | Message |
|---|---|---|
| Nav | S | because then if that erroneous equation ends up being a correct one with their input |
| Driver | S | im thinking an if statement but it may get lengthy |
| Nav | S | we can alter so that it says correct |

Drivers' dialogue moves appear to be particularly important within the pair programming process, with greater absolute frequency of drivers' *uncertainty* moves as well as *statements* associated with better outcomes. Both of these dialogue moves are important elements of a student's expressivity. For example, as students express uncertainty, they often articulate ill-formed misconceptions, which can be productive not only for that student in terms of the widely recognized self-explanation effect [2] but also for the collaborator who can then adapt to the uncertainty [5]. Excerpt 4 illustrates a $high_{code}$ pair navigator helping the driver locate a specific block after the driver expressed confusion. Both the high and low performing groups had higher frequencies of driver *uncertainty* tags, indicating nuance in how this move may affect collaboration.

**Excerpt 4: Uncertainty from $high_{code}$ pair.**

| Speaker | Tag | Message |
|---|---|---|
| Driver | U | unsure how to add strings together |
| Nav | QYN | can you use an operator? |
| Nav | D | go to the ops |

**Implications.** For faculty who use or want to use pair programming in their classes, the findings hold implications for preparing students to pair program and supporting them throughout the process. Many of these best practices are already recommended for pair programming, and are backed up by the empirical evidence presented here. First, faculty should encourage highly interactive, substantive dialogue from both students during pair programming. Pair programming classrooms should be characterized by the lively sound of students talking. Instructors may consider stopping by pairs who are silent, asking the navigator to summarize their most recent achievement, or the driver to articulate their current goal. These self-explanations may be very beneficial to students. Second, drivers should be encouraged to think aloud, and navigators should be encouraged to actively provide feedback. Because of the importance of common ground among the two collaborators, when one student feels uncertain, both partners must recognize that an expression of uncertainty is constructive. However, if a large proportion of dialogue focuses on uncertainty, the pair may need outside help. Finally, navigators should know that it is normal for them to talk more than drivers, and that giving feedback is helpful. On the other hand, if the driver seems not to be taking conversational initiative, asking an open-ended question (rather than a yes/no question) may be productive.

**Limitations.** Although this study considered a larger set of pairs than many previous studies of the pair programming process, this sample size was made possible in part by the textual dialogue mechanism. Rather than transcribing videos of in-person pair programming, we considered computer-mediated collaboration. This modality is common in practice, but is qualitatively different in many ways from in-person pair programming. The results must be interpreted in light of these differences. Additionally, in this study, drivers and navigators did not switch roles during the problem-solving process. This control is desirable in order to separate individuals into roles without overlap in the analysis, but it is not common in pair programming practice, where drivers and navigators should switch roles frequently. Best practice is to switch roles frequently, and the importance of this best practice is also highlighted in our empirical findings. Drivers learned

significantly more than navigators according to performance on individual problem solving after the collaboration.

# 7. CONCLUSION

Effective pair programming holds many benefits for students in computer science courses. Because the driver has the goal of building the program, drivers may not engage in dialogue as easily as navigators; however, the results presented here suggest that more active participation from the driver may be important to improve learning. The findings show that feedback, information statements, and grounding may enhance the quality of the collaboration. CS educators can strive to enhance the effectiveness of classroom pair programming by encouraging active conversational participation from both partners.

Future work should study how promoting conversation in pair programming affects performance in the classroom. Additionally, a deeper look at pair composition along lines such as gender, ethnicity, personality, and prior programming experience will provide insight into forms of collaboration that are most beneficial for many combinations of students. As collaboration continues to play an increasingly prominent role in computer science professional practice, it is crucial to inform our classroom practices with empirical study of the pair programming process.

# 8. Acknowledgements

Thanks to the members of the LearnDialogue Group for their helpful input. This material is based upon work supported by the National Science Foundation under grants CNS-1622438, DUE-1625908, and a Graduate Research Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

# 9. REFERENCES

[1] Astrachan, O. et al. 2011. CS Principles: Piloting a New Course at National Scale. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2011), 397–398.

[2] Chi, M.T.H. et al. 1994. Eliciting Self-Explanations Improves Understanding. *Cognitive Science*. 18, 3 (1994), 439–477.

[3] Cockburn, A. and Williams, L. 2001. The Costs and Benefits of Pair Programming. *Extreme Programming Examined*. 223–243.

[4] Falkner, K. et al. 2013. Collaborative Learning and Anxiety: A phenomenographic study of collaborative learning activities. *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2013), 227–232.

[5] Forbes-Riley, K. and Litman, D. 2009. Adapting to Student Uncertainty Improves Tutoring Dialogues. *Proceedings of the 14th International Conference on Artificial Intelligence in Education (AIED)* (2009), 33–40.

[6] Landis, J.R. and Koch, G.G. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics*. 33, 1 (1977), 159–174.

[7] Lewis, C.M. and Shah, N. 2015. How Equity and Inequity Can Emerge in Pair Programming. *Proceedings of the 11th International Computing Education Research Conference (ICER)*. (2015), 41–50.

[8] Li, Z. and Kraemer, E. 2014. Social Effects of Pair Programming Mitigate Impact of Bounded Rationality. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2014), 385–390.

[9] McChesney, I. 2016. Three Years of Student Pair Programming – Action Research Insights and Outcomes. *Proceedings of the 47th ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2016), 84–89.

[10] McDowell, C. et al. 2002. The Effects of Pair-Programming on Performance in an Introductory Programming Course. *Proceedings of the 33rd ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2002), 38–42.

[11] Nagappan, N. et al. 2003. Improving the CS1 Experience with Pair Programming. *Special Interest Group on Computer Science Education (SIGCSE). Conference* (2003), 359–362.

[12] O'Donnell, C. et al. 2015. Evaluating Pair-Programming for Non-Computer Science Major Students. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2015), 569–574.

[13] Porter, L. and Simon, B. 2013. Retaining Nearly One-Third more Majors with a Trio of Instructional Best Practices in CS1. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2013), 165–170.

[14] Radermacher, A. et al. 2012. Assigning Student Programming Pairs Based on their Mental Model Consistency: An Initial Investigation. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2012), 325–330.

[15] Radermacher, A. and Walia, G.S. 2011. Investigating the Effective Implementation of Pair Programming: An Empirical Investigation. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2011), 655–660.

[16] Ruvalcaba, O. et al. 2016. Observations of Pair Programming: Variations in Collaboration Across Demographic Groups. *Proceedings of the 47th ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2016), 90–95.

[17] Seyam, M. and McCrickard, D.S. 2016. Teaching Mobile Development with Pair Programming. *Proceedings of the 47th ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2016), 96–101.

[18] Snyder, L. et al. 2012. The First Five Computer Science Principles Pilots: Summary and Comparisons. *ACM Inroads*. 3, 2 (2012), 54–85.

[19] Stolcke, A. et al. 2000. Dialogue Act Modeling for Automatic Tagging and Recognition of Conversational Speech. *Computational Linguistics*. 26, 3 (Sep. 2000), 339–373.

[20] Vail, A.K. and Boyer, K.E. 2014. Identifying Effective Moves in Tutoring: On the Refinement of Dialogue Act Annotation Schemes. *Proceedings of the 12th International Conference on Intelligent Tutoring Systems (ITS)*. (2014), 199–209.

[21] Werner, L. et al. 2013. Pair Programming for Middle School Students: Does Friendship Influence Academic Outcomes? *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2013), 421–426.

[22] Zarb, M. et al. 2015. Further Evaluations of Industry-Inspired Pair Programming Communication Guidelines with Undergraduate Students. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE)*. (2015), 314–319.