# Pair Programming for Middle School Students: Does Friendship Influence Academic Outcomes?

Linda Werner
University of California
Santa Cruz, CA
831-459-1017
linda@soe.ucsc.edu

Jill Denner, Shannon Campe, Eloy Ortiz
ETR Associates
4 Carbonero Way
Scotts Valley, CA
831-438-4060

Dawn DeLay, Amy C. Hartl, Brett Laursen
Florida Atlantic University
Boca Raton, FL

{jilld, shannonc,eloyo}@etr.org

## ABSTRACT
Research shows the benefits of pair programming for retention and performance in computing, but little is known about how relationship dynamics influence outcomes. We describe results from our study of middle school students programming games using Alice and pair programming. From our analysis using statistical procedures that take into account the interdependence of pair data, we found evidence for partner influence moderated by the role of confidence over improvements in Alice programming knowledge in friend partnerships but not non-friend partnerships. We discuss implications for researchers and educators.

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

## General Terms
Experimentation, Human Factors.

## Keywords
Pair Programming, Game Programming, Middle School, Alice, Friendship, Computational Thinking.

## 1. INTRODUCTION
Educators and researchers are trying to change the 'face of computing'. Those at the K-12 level are hopeful that initial programming environments such as Alice and Scratch [22] and techniques such as collaboration via pair programming and game programming will make computing accessible to all students at an early age. Since computer science (CS) is not yet widely accepted as one of the K-12 core topics [3], we need to provide evidence-based results that show using these programming environments and techniques holds promise for introducing CS concepts and bringing real gains in computational thinking (CT) to K-12 students. We recently presented results supporting the introduction of CS concepts using Alice and game programming for middle school youth [24]. In this paper, we describe results pertaining to the use of friendship in the partnerships, referred to as dyads, with respect to pair programming and gains in aspects of CT.



*SIGCSE'2013*, March 6–9, 2013, Denver, CO, USA.


As a review, "CT is a problem-solving process that includes (but is not limited to) the following characteristics:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- Logically organizing and analyzing data,
- Representing data through abstractions such as models and simulations,
- Automating solutions through algorithmic thinking (a series of ordered steps)
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources,
- Generalizing and transferring this problem solving process to a wide variety of problems." [11]

We ran semester-long game-programming courses called iGame that involved 189 middle school students using Alice and pair programming across four semesters (in and after school). iGame is a study where we address many research questions including the following: Under what conditions does pair programming benefit the learning outcomes of middle school students programming games using Alice? Are the dyadic data interdependent?
We looked closely at pair programming to determine if there were moderating effects of this form of dyadic learning processes. To address this research question, we conducted surveys containing questions about student confidence in computing and Alice programming knowledge. Additionally students rated the degree to which their partner was a friend.

In this paper, we first discuss prior work with pair programming, followed by a description of the iGame course including participant demographics, procedures of the study, measures, and the analysis process. Next, we list the results of the analyses using statistical procedures that take into account the interdependence of dyadic data and discuss these results. The paper concludes with a discussion of future work.

## 2. PRIOR WORK
A long history of research describes the benefits of collaborative learning [4] and specifically pair programming for individual performance and persistence rates at the college level [6][18][26] but there are very few studies of pair programming in K-12 [15][25]. Building on the research of Vygotsky [23], socio constructivists have studied how discussion and reflection on a specific task can help children construct an understanding that

goes beyond what they can achieve on their own. For example, working with a partner encourages students to summarize and explain what they know, respond to immediate feedback, take time to work through what they do not understand, and ask questions--all high level thinking skills that improve performance [20].

In pair programming, two students share one computer and there are suggested roles: one is the driver (working the keyboard and mouse) while the other navigates. Pair programming is a particularly promising means to promote computational thinking because it encourages peer scaffolding, clear roles, and frequent feedback. But simply working with any partner is not enough to promote learning. Students must be involved in the selection of their partner [9], and are more likely to articulate and internalize what they are learning when their partner is on an equal social level and is a friend [2]. This is particularly important for collaboration on an open ended, creative task (unlike tasks in which there is a single 'right' answer) because friends know how to develop a shared space and generate ideas together [19]. In addition, other characteristics of the students, such as previous knowledge or self confidence, may promote or interfere with the development of computational thinking in each of the partners. Few studies examine the association between interdependence of dyads and change in learning outcomes, and none have done this in the context of pair programming.

Studies of friendship dyads are flawed, however, when they do not correct for statistical biases inherent in interdependent dyadic data. Previous studies have either involved separate analyses of individuals in dyads, which often produces conflicting results, or treated partners as if their data were independent, violating key assumptions of statistical tests and typically inflating Type II errors [10]. In our study, we distinguished partners based on computer confidence, reasoning that confident partners would be better collaborators, such that they would be both persuasive in, and open to suggestions for, obtaining task mastery [16]. Thus we expected confident partners to be both influential and open to influence.

# 3. METHOD

## 3.1 Participants

The data described in this paper were collected over two years as part of a study of how game creation and pair programming can promote CT. A total of 189 students with parental consent participated in pair programming classes at seven public schools along the central California coast. Participation in all parts of the study was voluntary. We measured students' experience programming, closeness with partner, Alice programming knowledge, and student confidence with computing.

Of these 189 students, all but 18 were members of same-gender dyads. Because most adolescent friendships are same-gender, our analyses focuses on these students. Paired students were either initially paired with a partner or were a replacement for a partner who left the course sometime before the final survey. The current sample was reduced by nine students who dropped out of the study and their partners had to be repaired and two students (one dyad) who were excluded because they were twins (since the focus of the current investigation was not on sibling relationships). Individuals with partners who dropped out were immediately paired with other students in the class who did not have a partner. New partners were assigned after 2 to 12 hours of instruction ($M$=7.38). New partners completed the measure of friendship at the start of their first session of joint instruction. Participants in dyads with reassigned partners did not differ on any study variable from the remainder of the sample. Neither were there differences between students who participated in the after-school program (n=58) and those who participated in a computer class during school hours (n=102).

Of the remaining 160 pair programming students, (60 girls, 100 boys), there were 61 students in grade 6, 49 students in grade 7, and 50 students in grade 8. Students ranged from 10 to 14 ($M$ =12.05, $SD$=1.01) years old. The parental consent form indicated that a total of 64.38% ($n$=103 students) lived with both biological parents, 18.13% ($n$=29 students) lived with one biological parent, 3.13% ($n$=5 students) lived in other types of households, and 14.38% ($n$=23 students) had not reported living arrangements. The final sample included 73 (45.6%) who described themselves as White Caucasian, 22 (13.8%) as Hispanic Latino, 8 (5.0%) as Asian Pacific Islander, 5 (3.1%) as Native American, 3 (1.9%) as African American, 44 (27.5%) as mixed ethnic background, and 5 (3.1%) who did not report their ethnicity.

## 3.2 Procedure

In-school participants were recruited from eight classrooms and from four extended learning programs in seven schools in four cities (or school districts). The schools represented lower and middle socioeconomic status communities. Schools were selected for participation if a technology teacher indicated interest in the project. In some schools ($n$=3), students participated in the project as part of their technology elective class (n= 62 boys and 40 girls). All students in the technology elective classrooms were invited to participate in the study through in-class announcements and a letter sent home to parents. One student declined to participate. In other schools ($n$=4), students participated as part of an after-school enrichment program (n= 38 boys and 20 girls). Recruitment for the after-school program involved announcements made by the extended learning site coordinator and by teachers. Two students in the after-school technology program declined to participate. Written parent consent and student assent were required for participation. Students participated for 20 hours (usually 10 weeks).

Students worked in dyads assigned by their teachers with input from the students themselves. Students worked with various other (typically) same gender classmates during the first two or three class sessions at which time students submitted a short list of classmates' names with whom they would like to partner. Students spent the remainder of the sessions working in the assigned dyads.

Students used one of two programming environments in the Alice [1] series developed at Carnegie Mellon. In year 1, students used Storytelling Alice (SA), and in year 2 they used Alice 2.2 because SA was only available on PCs, which were limited at our partner schools. In the rest of this paper, we will refer to both SA and Alice 2.2 as Alice unless it is necessary to distinguish between the two. Alice allows users to control characters in 3D environments using drag-and-drop programming, a language that is closely related to Java and many other modern imperative programming languages. Most code is written in the methods of objects that have properties that store state and functions that return values. Each property, method, and function is attached to an object, with World being the global object. The event system in Alice is primarily used to handle user interactions, such as mouse clicks,

although it can also handle in-World events, such as when the value of a variable changes.

Students engaged with CT in a three-stage progression called Use-Modify-Create[14] over approximately 20 hours during a semester. In the first half of the semester, students played games, completed tutorials, and worked through a series of self-paced instructional exercises built to provide scaffolding, which we call "challenges." During the last half of the course, the students freely designed and developed their own games [8]. Most students completed eight to 10 of 11 required challenges, though some completed up to six additional optional challenges. The student dyad shared one computer, with one driving (controlling the mouse and keyboard) and the other navigating (checking for bugs, consulting resources, and providing input). We asked students to reverse roles approximately every 20 minutes.

Alice programming knowledge assessments were administered as online surveys at the beginning of the first and the end of the last (or second to last) class meeting. Friendship assessments were completed in paper-pencil format at the beginning of the third or fourth class meeting after partners were assigned (or at the start of the first class for students who were re-partnered) and online at the end of the last or second to last class meeting. Students were not permitted to collaborate or share answers. Surveys were administered by research staff.

## 3.3 Measures and Analysis Process

Participants separately completed the same assessments: a) pre-test at the beginning of the project, at the start of the introductory class meeting; and b) post-test at the conclusion of the project, after the last class meeting approximately 10 weeks later.

*Alice programming knowledge* was assessed with an 8-item measure of sample programming code. Students were presented with screenshots of the programming interface (e.g., *What would happen if you were to play the above 3 lines of code?*). Students selected one of five possible response options. Each student received a score that indicated the number of correct responses (range=0 to 8).

*Computer confidence* was assessed with a 3-item scale that measured perceived computing abilities [5]. Items were rated on a scale from 1 (*Strongly disagree)* to 5 (*Strongly agree) (*e.g., *I feel confident about my ability to use computers).* Item scores were averaged; internal reliability was good (Cronbach's α=.84). Each partner in each dyad was classified as either relatively more confident using computers ($M$=4.27, $SD$=0.76) or relatively less confident using computers (pretest $M$=2.99, $SD$=0.89). These classifications did not change from pretest to posttest. In most dyads (n = 50), the computer confidence scores of partners differed by at least 0.50 standard deviations (pretest $M$ =1.03, $SD$=.61).

*The Friendship score* used the Friendship Quality Scale [7]. This scale measures perceptions of closeness and intimacy with the partner. Items were rated on a 1 (*Not true*) to 5 (*Really true*) scale. Dyads were classified as friends or non-friends using responses to a single item (i.e., *My partner is my friend*). To qualify as friends ($n$=48), both partners had to respond to this item with a rating of 3 (*Usually true)* or higher at the pretest. All other dyads were classified as non-friends ($n$=32), meaning that at least one partner responded with a score of 2 (*Might be true)* or lower at the pretest. In six cases, both partners rated the other as 2 or lower on the friendship item.

Our analysis using the Actor-Partner Interdependence Model (APIM), which is designed for correlated data in distinguishable dyads, permits the analysis of non-independent data across multiple time points [13]. It partitions variance shared across partners on the same variable from variance that uniquely describes associations within and between partners. This 4-step analyses process estimates the influence that partners have on changes in each other's computer programming skills as measured by pre and posttest Alice programming knowledge assessment. We distinguished partners on the basis of computer confidence, reasoning that confident partners would be better collaborators, such that they would be both persuasive in, and open to suggestions for, obtaining task mastery [16]. Thus we expected confident partners to be both influential and open to influence [12].

In step one of the APIM analyses, 30 dyads were omitted from our analyses because partners in these dyads differed in computer confidence by less than 0.50 standard deviations from the mean. In step two, the remaining 50 dyads with partners distinguishable on the basis of computer confidence are analyzed looking at the influence high confidence partners have over changes in Alice programming knowledge of low confidence partners and vice versa. In the third step, we conducted a multiple group APIM to examine whether there were differences between friend and non-friend dyads on the magnitude and direction of influence in Alice programming knowledge. In the fourth step, we conducted a repeated measure ANOVA to determine whether the rate of change in Alice programming knowledge differed for more confident students paired with partners who had greater Alice programming knowledge, as compared to those paired with partners who had less Alice programming knowledge.

# 4. RESULTS

As expected, the data were not independent. There were significant ($p$<.001) concurrent pretest and posttest correlations between Alice programming knowledge and computer confidence (range: $r$=.24 to .35). Autocorrelations indicated that pretest scores were positively ($p$<.001) associated with posttest scores, for both Alice programming knowledge ($r$=.47) and computer confidence ($r$=.66). There was a significant ($p$<.001) over time correlation ($r$=.26) between pretest Alice programming knowledge and posttest computer confidence, but not between pretest computer confidence and posttest Alice programming knowledge ($r$ = .09).

Alice programming knowledge increased over time, $t(153)$= 12.38, $p$<.001 (Pretest: $M$=2.22 $SD$=1.69; Posttest: $M$=4.01 $SD$=1.76), but computer confidence did not (Pretest: $M$=3.69 $SD$=.93; Posttest: $M$=3.71 $SD$=.87). There were no gender differences on either computer programming knowledge or computer confidence.

Figures 1 and 2 show the partner influence on Alice programming knowledge as a function of relative computer confidence in friend ($n$=30) dyads and non-friend ($n$=20) dyads. APIM analysis results for friends revealed statistically significant actor paths for the less confident partner (a2), but not the more confident partner (a1). For less confident partners, pretest Alice programming knowledge scores were positively associated with posttest scores ($\beta$=.57). For more confident partners, there was no statistically significant association between pretest Alice programming knowledge scores and posttest scores ($\beta$=.15). The lack of stability among more confident partners is an indication of change in Alice

programming knowledge. Partner paths revealed influence of the less confident partner on the more confident partner (p2), but not the reverse (p1). The less confident partner's initial Alice programming knowledge was associated with positive changes in the more confident partner's Alice programming knowledge ($\beta$=.47), but the more confident partner's initial Alice programming knowledge did not predict change in the less confident partner's Alice programming knowledge ($\beta$=-.23).

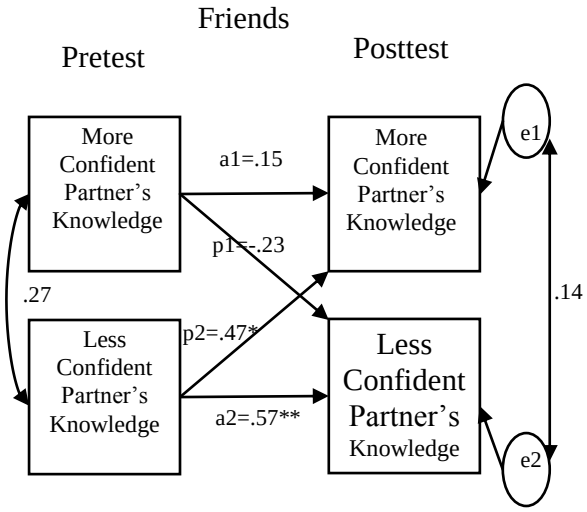


**Figure 1. Friend influence; *p<.05, **p<.01, two-tailed.**

Among non-friends, actor paths revealed stability for both partners (a1 and a2). Pretest Alice programming knowledge scores were positively associated with posttest scores (more confident: $\beta$=.58; less confident: $\beta$=.47). There were no statistically significant partner paths (p1 and p2), indicating that neither partner influenced the other (more confident: $\beta$=.12; less confident: $\beta$=.20).

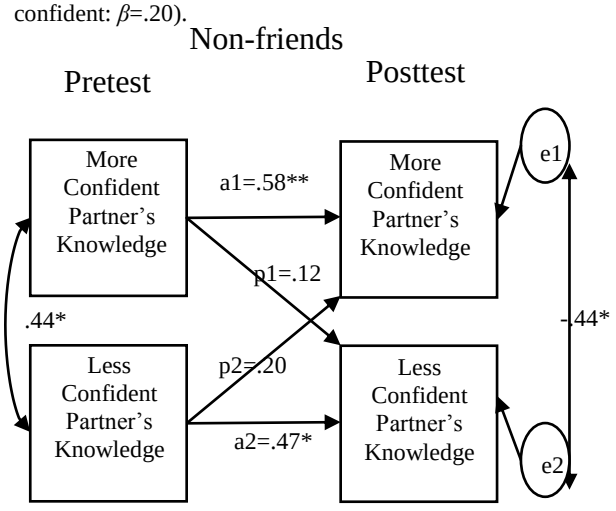


**Figure 2. Non-friend influence; *$p$<.05, **$p$<.01, two-tailed.**

Multiple group contrasts compared friends and non-friends on each actor and partner path. There were no statistically significant differences on partner influence paths (Range: $\chi^2(1, N=50) = 1.51$-$1.67$, $p>.05$). There were statistically significant differences between friends and non-friends on the stability of the more confident partner's Alice knowledge score, $\chi^2(1, N=50) = 4.01$, $p<.05$, but not on the stability of the less confident partner's Alice knowledge score, $\chi^2(1, N=50) = 0.90$, $p<.001$.

Within group contrasts compared the partner influence and actor stability paths of the more confident and the less confident partners, separately within friend dyads and within non-friend dyads. For friends, there were statistically significant differences between the more and less confident partner on partner influence paths, $\chi^2(1, N=30) = 8.86$, $p<.05$, such that the less confident partner had more influence than the more confident partner. Additionally, there were statistically significant differences between the more and less confident partner on actor stability paths, $\chi^2(1, N=30) = 4.23$, $p<.05$, such that the more confident partners' Alice programming knowledge scores were less stable than the less confident partners' scores. For non-friends, there were no statistically significant differences between more and less confident partners on partner influence or actor stability paths (Range: $\chi^2(1, N=20) = 0.19$-$1.13$, $p>.05$).

Step 3 revealed a statistically significant partner influence path in friend dyads, wherein the less confident partner's Alice programming knowledge pretest score predicted change in the more confident partner's Alice programming knowledge posttest score. To better understand the nature of this influence, we divided friend dyads into two groups: (1) Those where the less confident partner had more pretest Alice programming knowledge than the more confident partner did ($n$=10), and (2) those where the less confident partner had less pretest Alice programming knowledge than the more confident partner ($n$=15). Partners who had equivalent levels of pretest Alice programming knowledge were removed ($n$ = 5).

A 2 (Time) X 2 (Alice programming knowledge of partner at pretest) repeated measures ANOVA was conducted. The Alice programming knowledge score of the more confident partner was the dependent variable. There was a statistically significant time by partner Alice programming knowledge interaction, $F(1,23)=24.309$, $p<.001$. The more confident partner experienced the greatest increases in Alice programming knowledge when paired with a friend who had more Alice programming knowledge at the outset. Follow-up paired t-tests revealed a significant increase in Alice programming knowledge for the more confident partner when paired with a more knowledgeable partner, $t(9) = 8.72$, p<.001, but not when paired with a less knowledgeable partner, $t(14) = 1.10$, p>.05.

A similar pattern of results emerged when dyads whose computer confidence differed by less than .50 *SD* ($n$=30) were included. Additional multiple group APIM analyses were conducted using gender and instructional setting (during or after-school) as moderators. There were no statistically significant $\chi^2$ differences, suggesting that the influence of more confident partners and less confident partners did not vary as a function of these variables. Age, grade, length of training (numbers of hours), ethnicity, and household structure were separately entered into the model as control variables. In each case, model fit significantly worsened and the pattern of statistically significant paths did not change.

## 5. DISCUSSION

The investigation of friend influence over learning processes presents an analytic challenge because of the interdependent nature of dyadic data. Taking advantage of recent advances in non-independent statistical analyses, we used a longitudinal

APIM for distinguishable dyads to determine the degree and magnitude of friend influence over change in Alice programming knowledge. We found that computing confidence determines who influences whom. We also found that friendship status moderated the findings such that the less confident partner influenced the more confident partner only if partners were also friends. Follow-up analyses illustrated these influence processes. The greatest increases in Alice programming knowledge occurred among confident partners who were paired with a friend who had relatively more initial Alice programming knowledge.

What are the implications of these results for teachers? Since the benefit of working with more knowledgeable peers is greater when students are confident, it is important for teachers to focus on increasing confidence, as well as knowledge.

This study had several limitations. "Friendship" was assessed by a single question at a single point in time, and we know that middle school friendships tend to fluctuate. This may have introduced some error in the categorization of dyads as friends and non-friends, which suggests that these findings may underestimate the magnitude of results. We had limited statistical power to detect differences between friends and non-friends, suggesting that some non-significant group contrasts may have been a result of a small sample size. Small sample sizes prevented us from definitively determining whether age moderated the pattern of results.

## 6. CONCLUSIONS AND FUTURE WORK

The next set of questions involves looking at intervention efforts to guide positive forms of influence. Also additional analyses will contribute to an understanding of what makes pair programming effective, and how that varies across gender and culture. Longitudinal dyadic data analysis is beginning to show the conditions under which partners learn more, such as when they are paired with friends. The findings have implications for teachers, who often resist pairing children with friends for fear that students will be disruptive and spend time off task. The APIM methodological approach is being used in studies of pathology, but this is the first application of its use in Computer Science education research.

The role of pair versus solo programming is also being investigated by looking at the nature of pair programming. There was great variation in the effectiveness with which partners worked together, and we expect that will impact how much they learn. To this end, we have begun to describe what pair programming looks like, and the conditions under which it is advantageous. This includes a "pair effectiveness" score, based on video recordings of 71 pairs working together for approximately 20 minutes with each partner in both of the two pair programming roles. Our scale includes collaboration categories for inclusiveness, responsiveness, and negative behaviors.

In addition, we are drawing on socio-cultural theory, to focus on the collaborative aspects of the learning process, specifically how students of different cultural backgrounds collaborate during pair programming. We are in the process of clarification of collaboration categories from an earlier study [21] after which we will catagorize all interactions in the recordings. Understanding how partners interact by supporting or undermining each other's goals during pair programming will provide useful information for strengthening classroom practices. Although pair programming has shown benefits for students' learning in general, more research is needed on which collaborative qualities can support girls and Latino-heritage students, who are underrepresented in the computer science and related fields.

## 7. ACKNOWLEDGMENTS

Our thanks go to the teachers and administrators at our seven schools, specifically Anne Guerrero, Shelly Laschkewitsch, Don Jacobs, Sue Seibolt, Karen Snedeker, Susan Rivas, and Katie Ziparo. Thanks also to teaching assistants, Will Park, Chizu Kawamoto, and Joanne Sanchez; and to Pat Rex, for her support with instructional matierials design. Thanks to all of the students who participated. This research is funded by a grant from NSF 0909733 "The Development of Computational Thinking among Middle School Students Creating Computer Games." Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## 8. REFERENCES

[1] Alice website, September 1, 2011. www.alice.org.

[2] Azmitia, M. and Montgomery, R. 1993. Friendship, transactive dialogues, and the development of scientific reasoning. *Social Development*, 2: 202–221. doi: 10.1111/j.1467-9507.1993.tb00014.x

[3] Barr, V. and Stephenson, C., 2011. Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? *ACM Inroads* 2, 1, 48-54.

[4] Barron, B. 2000. Achieving coordination in collaborative problem-solving groups. *Journal of the Learning Sciences*, 9(4), 403-436.

[5] Barron, B., Walter, S.E., Martin, C.K., and Schatz, C. 2010. Predictors of creative computing participation and profiles of experience in two Silicon Valley middle schools. *Computers & Education*, 54, 178-189.

[6] Braught, G., Eby, M., and Wahls. T., 2008. The effects of pair-programming on individual programming skill. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education* (SIGCSE '08). ACM, New York, NY, USA, 200-204.

[7] Bukowski, W. M., Hoza, B., and Boivin, M. 1994. Measuring friendship quality during pre and early adolescence: The development and psychometric properties of the Friendship Qualities Scale. *Journal of Social and Personal Relationships*, 11, 471–484.

[8] Campe, S., Werner, L., and Denner, J. 2012. Game programming with Alice. In P. Phillips, S. Cooper, and C. Stephenson, *CSTA Voice Special Issue Computer Science K-8: Building a Strong Foundation*, 13-14.

[9] Ciani, K., Summers, J., Easter, M., and Sheldon, K. 2008. Collaborative learning and positive experiences: Does letting students choose their own groups matter? *Educational Psychology*, 28(6).

[10] Cook, W. L., and Kenny, D. A. 2005. The actor-partner interdependence model: A model of bidirectional effects in developmental studies. *International Journal of Behavioral Development, 29,* 101-109.

[11] CSTA, 2011. Operational Definition of Computational Thinking for K-12 Education. Accessed on August 10, 2012 at http://csta.acm.org/Curriculum/sub/CompThinking.html.

[12] Delay, D., Hartl, A., Laursen, B., Denner, J., Werner, L., Campe, S., and Ortiz, E. Submitted for review. Learning from friends: Measuring influence in a dyadic computer instructional setting. Submitted to special issue Methodological Issues with the Measurement of Change in Education: The Use of Longitudinal Studies in *International Journal of Research and Method in Education.*

[13] Kenny, D. A., Kashy, D. A., and Cook, W. L. 2006. *The Analysis of Dyadic Data*. New York: Guilford Press.

[14] Lee, L., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., and Werner, L., 2011. Computational thinking for youth in practice. *ACM Inroads* 2(1), 32-37.

[15] Lewis, C., 2011. Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education*, 21(2), 105-134.

[16] Liem, A. D., Lau, S., and Nie, Y. 2008. The role of self-efficacy, task value, and achievement goals in predicting learning strategies, task disengagement, peer relationship, and achievement outcome. *Contemporary Educational Psychology*, 33, 486-512.

[17] Linn, M. C., 1985. The Cognitive Consequences of Programming Instruction in Classrooms. *Educational Researcher*, 14(5), pp. 14-16+25-29 at http://www.jstor.org/stable/1174202.

[18] McDowell, C, Werner, L., Bullock, H., and Fernald, J. 2006. Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, 48(8), 90-95.

[19] Miell, D. and MacDonald, R. 2000. Children's creative collaborations: The importance of friendship when working together on a musical composition. *Social Development*, 9: 348–369. doi: 10.1111/1467-9507.00130

[20] Palincsar, A.S., and Brown, A.L. 1984. Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and Instruction*, 1(2), 117-175.

[21] Ruvalcaba, O., Werner, L., and Campe, S. 2012. Computer-based collaboration in middle school: A sociocultural perspective on pair programming with Mexican-heritage and European-heritage students. *Proceedings of the 2012 Conference of American Educational Research Association*, Vancouver, British Columbia, Canada.

[22] Utting, I., Cooper, S., Kölling, M., Maloney, J., and Resnick, M. 2010. Alice, Greenfoot, and Scratch -- A Discussion. *ACM Transactions on Computing Education* 10(4), Article 17 (November 2010), 11 pages. DOI=10.1145/1868358.1868364 http://doi.acm.org/10.1145/1868358.1868364

[23] Vygotsky, L.S. 1978. *Mind in Society*. Cambridge, MA: Harvard University Press.

[24] Werner, L., Campe, S., Denner, J. 2012. Children learning computer science concepts via Alice game-programming, *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE, Raleigh, N. Carolina, USA.

[25] Werner, L., Denner, J. 2009. Pair programming in middle school: What does it look like? *Journal of Research on Technology in Education*, 42(1), 29-49.

[26] Werner, L., Hanks, B., & McDowell, C. 2004. Pair programming helps female computer science students. *ACM Journal of Educational Resources in Computing*, 4(1).