Pair Debugging: A Transactive Discourse Analysis

Laurie Murphy
Computer Science and Computer Engineering
Pacific Lutheran University
Tacoma, WA, USA
lmurphy@plu.edu

Sue Fitzgerald
Information and Computer Sciences
Metropolitan State University
St. Paul, MN, USA
sue.fitzgerald@metrostate.edu

Brian Hanks
Computer Science Info Systems
Fort Lewis College
Durango, CO, USA
brianhanks@acm.org

Renée McCauley
Computer Science
College of Charleston
Charleston, SC, USA
mcccauleyr@cofc.edu

ABSTRACT
Previous research on the interaction between pairs suggests there is a positive relationship between transactive discussion and effective problem solving. Debugging is particularly problematic for novice programmers. Some previous studies suggest this difficulty may be lessened by working in pairs. Using transactive analysis, we examined interactions between five pairs of university-level introductory programming students as they debugged Java programs. Transcriptions of their verbal interactions were coded into transactive statement categories which revealed that the nature of participants' discourse varied. Extensions, feedback requests, critiques and completions were the most frequently observed types of transactions. Other transaction types were rarely detected in this context. The amount of discussion for the pairs varied as did the number of transactive statements. Results suggest that pairs who talked more and used completion transactions more often attempted more problems, but those who critiqued more frequently successfully debugged more problems. The teaching implications of this work and recommendations for future research are discussed.

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

General Terms
Human Factors

Keywords
debugging, novice programming, pair programming, transactive discourse analysis

1. INTRODUCTION
Pair programming requires two students to work side-by-side at one computer to code a solution. It is a well-known pedagogical technique for teaching introductory programming. Studies have shown that students who pair program produce higher quality code [11], show greater confidence in their solutions [6, 14], are more likely to pass their programming courses [2, 14, 15], and are more likely to persist in the major [4, 14] than solo programmers. Pair programming is also thought to be particularly beneficial for women students because it dispels perceptions that programming is a competitive, socially isolating activity [22].

This study examines an important facet of pair programming — *pair debugging* — with the aim of gaining a better understanding of how pairs work together to find and fix bugs. Many aspects of debugging are difficult for novices (see [16] for an overview). The study of novice debuggers is particularly challenging because the cognitive burden of debugging and problem solving makes it difficult for novices to simultaneously "think aloud" [7].

The dialog pairs naturally engage in when solving problems presents an ideal context for studying debugging. Furthermore, debugging would seem to be an especially fruitful activity for pair collaboration: it provides two sets of eyes to find and correct errors, a context in which to thoughtfully reason through solutions, and a framework that allows students to share the cognitive complexity of the debugging process.

This preliminary analysis explores the broad question of how novice pair programmers interact with each other while debugging. In particular we ask 1) What is the nature of the discourse between pair programmers as they debug? and 2) What differences, if any, are there in the discourse of successful and unsuccessful pair debuggers?

This work is grounded in an earlier investigation by Teasley [17] that looked more generally at the role of the peer in peer learning as well as earlier studies of pair programming and debugging.

Section 2 of this paper describes Teasley's study and transactive discussion in more depth and gives an overview of other related work; Section 3 presents the study's methodology and protocol; Section 4 describes and discusses the results; Section 5 presents threats to validity and implications for teaching are summarized in Section 6.

2. BACKGROUND

Although pair programming in academic settings has been studied extensively, there is little evidence in the literature that the debugging behavior of pairs has been investigated. One exception is a study conducted by Chaparro et al. [5], in which students found pairing less enjoyable when working on a debugging task than on programming tasks. They also found that paired students were more likely to disagree with the statement, "I think I've learned more this time working in pairs than others that I've worked on my own" when working on a debugging task than on a programming task.

On the other hand, there is some evidence that students who paired were able to debug more successfully than those who worked alone. Hanks found that pairs in closed labs required less assistance from TAs or instructors than soloing students, which may suggest that pairs are more effective debuggers [9]. In a survey of CS1 students given by VanDeGrift [19], students generally agreed with the statement, "I was more efficient in debugging my code while working with a partner on the projects versus working individually on the homework assignments." Success at debugging may also be a key factor in pairs' higher code quality and greater confidence [6, 11, 14]. A more thorough investigation of paired debugging seemed indicated.

Spoken data has been an important element of many experiments in computer science education, software engineering, and human-computer interaction research. Frequently, verbal data collection is facilitated by asking participants to "think aloud" or by observing small groups of individuals working together. Hughes and Parkes [12] summarize the use of verbal protocol analysis in software engineering research since the 1980s. A recent study by Werner and Denner [21] evaluated audio tapes of middle school girls engaged in pair programming to characterize the types of interactions that lead to effective problem solving and may therefore help girls persist in CS.

Analysis of spoken data has also been applied to debugging, in particular to reveal debugging strategies [8] and to compare expert and novice debuggers [20]. This research focused on categorizing participants' verbalizations based on their activities (such as hypothesis generation, exploration, or evaluation), instead of on the verbal interactions of the participants.

In an earlier study of student debugging behavior, participants were asked to "think aloud" while debugging small programs [7]. However, many of them found thinking out loud distracting and did not speak much, limiting the conclusions that could be drawn from their actions. In their study of distributed pair programmers, Hanks and Brandt [10] argued that "speak aloud" protocols can interfere with a subject's train of thought, and noted that pair programmers, "naturally vocalize as they discuss problems and describe their thought processes to their partner, with the result that the researcher can more easily understand their problem solving approach" (p. 25). As a result of these earlier observations, we asked our participants to work in pairs on debugging tasks, so that they would naturally speak aloud to convey their thoughts to their partners.

Our approach to analyzing the interactions between the paired students was based on Teasley's work on *transactive discussion* [17]. Teasley's study asked grade school children to solve computer-based logic programs. Their verbal interactions were then coded based on the transactive discussion categories used by Berkowitz and Gibbs [1].

Transactive discussion is a conversational mode in which participants respond to their partner's statements to clarify their own understanding or reasoning. A conversational statement is transactive when it "extends, paraphrases, refines, completes, or critiques the partner's reasoning or the speaker's own reasoning" [17, p. 362]. For example, the participants in the following dialog are engaged in transactive discussion:

> **S2**: *So we took the first one…let's go ahead and make a for loop around the whole section*
> **S1**: *Should we make it outside the while loop?*
> **S2**: *I think it needs to be inside it.*

Berkowitz and Gibbs [1] first used the term *transactive discussion* and defined a classification of transactive statements. In their classification, a transactive statement falls into one of the categories listed in Table 1.

Teasley [17] found that children who engaged in transactive discussion while working on scientific reasoning tasks were more successful learners than children who did not have this type of interaction.

Table 1: Transactive Statements

Feedback request	Do you understand or agree with my position?
Paraphrase	I can understand and paraphrase your position or reasoning.
Justification request	Why do you say that?
Juxtaposition	Your position is X, and my position is Y.
Completion	I can complete or continue your unfinished reasoning.
Clarification	No, what I am trying to say is the following.
Extension	Here is a further thought or elaboration.
Critique	Your reasoning misses an important distinction, or involves a questionable assumption.
Integration	We can combine our positions into a common view.

Based on Teasley's work, we hypothesized that pairs who engage in transactive discussions will be more successful at debugging tasks than those who do not.

3. METHODOLOGY

This study adapted Teasley's study [17] with pairs of university-level introductory programming students debugging a set of simple Java programs. Students were recorded (using audio, video and screen capture) as they worked in pairs to debug logic errors in Java programs.

3.1 Participants

Participants were ten undergraduate student volunteers enrolled in an introductory Java programming course at a small liberal arts university in the United States. Seven participants were female and three were male, and all had completed 10–12 weeks of university Java programming instruction in a traditional "objects later" introductory course.

We obtained IRB approval in accordance with the participating institution's policies. Volunteers were solicited via email and through a sign-up sheet circulated during class. Students were encouraged to sign up with a partner of their choosing, although all but one pair were made up of individuals assigned to an unfamiliar partner for the interview session. All students had previous experience pair programming during in-class exercises assigned 1-3 times per week, although closed labs and homework assignments for the course were completed individually.

The researcher who conducted the interviews was the instructor for one of two class sections and both laboratory sections for the course, and thus was familiar with all of the students in the study. Participants were given a modest monetary compensation for their participation.

3.2 Protocol

Interview sessions of approximately an hour were conducted in the laboratory setting students were accustomed to working in during their weekly closed lab sessions. Interviews involved a pair debugging session followed by a short survey.

3.2.1 Pair debugging session

Once the study was explained to them, students were situated at a designated computer equipped with webcam, microphone, and Morae[1] usability screen capture software. Students were given instructions both in printed form and also as an HTML "read me" file on the machine, which stated:

You and your partner should work together to debug the set of buggy programs below. All the programs are syntactically correct, but each contains a single logic bug (we hope). Work through the programs in numeric order, finding and fixing the current bug before moving on to the next program. If you make a serious attempt to find and fix a bug and yet feel you are completely stuck, you may move on to the next problem. It is possible you will not have enough time to debug all of the programs.

The instructions were followed by a list of Java programs (several adapted from those used in the solo debugging study [7]), which were linked from the electronic version. Students were instructed to edit the files as they debugged and simply save the updated versions under the same names in the current directory.

The list (as shown in Table 2) contained 16 programs[2], although no pair attempted to debug more than 10. We have included our informal assessment of the difficulty of each problem on a scale of 1 to 4, with 1 being easiest.

The participants were permitted to access the Java online API documentation and a copy of the course text was provided to them for reference.

Each pair was given 45-50 minutes to debug as many of the programs as they could. Their interactions were recorded on video and audio and the software captured all actions as picture-in-picture (a small webcam recording embedded within the screen capture).

The researcher remained in the room at a distance to answer questions and to take some notes about obvious student behavior (e.g., writing notes or using the text) that might not be captured via video.

When time was up students were asked to move to different lab machines to complete an online survey about the exercise while the researcher saved the Morae file and the debugged programs.

Table 2: Buggy Programs

Program	Difficulty (1=easy, 4=hard)	Error Type
Average	2	Integer division (forgetting to cast)
Volume	1	Incorrect argument order
Rectangle1	1	Switching rows and columns in a nested loop
Rectangle2	2	Off-by-one loop
Validate	2.5	Using || instead of &&
Rectangle3	2	Incorrect calculation/expression in an if condition
Validate2	2.5	Using && instead of ||
Sort3Integers	2	Operands swapped in a relational expression
TriangleType	3	Incorrect if logic
Raffle1	2	Using != to compare strings
Reverse	4	Incorrect loop condition; processing too much of the array
Raffle2	2	Forgetting curly braces in an if block
Calculator1	3	Failing to reset a variable used inside a loop
Raffle3	3	Forgetting parentheses
Sort3	4	Trying to alter the value of a primitive argument
Car	3	Overshadowing an instance variable with a local variable

A research assistant experienced in working with introductory programming students transcribed the students' verbal interactions. Two of the researchers then independently analyzed these transcripts based on Berkowitz and Gibbs's [1] transactive discussion categories (see Section 2). Both researchers coded each of the five transcripts, and annotated them with the types of transactive utterances detected. For example:

S1: *Here, will that fix it?* [FEEDBACK REQUEST]

Only utterances related to reasoning about the problem of debugging were coded. As they coded, the researchers made note of interesting observations about how students communicated and the sorts of problems that seemed to arise.

The two researchers then worked together to reconcile their coding of three of the transcripts. The researchers were able to come to agreement through discussion and rereading sections of Teasley [17]. Finally, with this experience, each researcher

[1] Available at http://www.techsmith.com/morae

[2] Buggy programs are available from the authors on request.

independently reconciled an additional transcript. Throughout this reconciliation phase, as in the coding phase, the researchers recorded their observations about how students communicated and collaborated.

It is worth noting that our analysis was more finely grained than that presented by Teasley in [17], who gave an example of line-by-line transactive categorizations, but otherwise presented a qualitative discussion of excerpts that were characterized as transactive or non-transactive overall.

A third researcher analyzed the debugged Java programs to determine if they had been debugged correctly and how much time was spent debugging each one.

4. RESULTS AND DISCUSSION

Here we analyze the nature of the discourse between pair programmers as they debug, we address differences in the discourse of successful and unsuccessful pair debuggers, and we offer some general observations.

4.1 Transactive patterns

We counted the number of transactive statements of each type made by the pairs. For each pair, we then computed the percentage of all transactive statements that were of each type (see Table 3). Of the transactive types detected in the transcripts, four were most common: extension, feedback request, critique, and completion. Justification occurred infrequently, and none of the other transactive statement types (paraphrase, juxtaposition, clarification, or integration) made up more than 3% of all transactive interactions for any pair – these categories are grouped together in the 'Other' type in the table. In fact, juxtaposition was not detected at all.

Table 3: Transactive Statements by Pair

Type	Transactive Statements per Pair by Type					
	P1	P2	P3	P4	P5	All
Extension	17% (17)	26% (19)	30% (9)	21% (20)	37% (40)	30% (121)
Feedback Request	19% (19)	24% (17)	30% (9)	32% (30)	21% (23)	24% (98)
Critique	25% (25)	21% (15)	27% (8)	22% (21)	13% (14)	20% (83)
Completion	17% (17)	18% (13)	7% (2)	16% (15)	20% (21)	17% (68)
Justification	7% (7)	6% (4)	3% (1)	4% (4)	6% (6)	5% (22)
Other	1% (1)	6% (4)	3% (1)	4% (4)	3% (3)	3% (13)
Total	100% (102)	100% (72)	100% (30)	100% (94)	100% (107)	100% (405)

Note: The figures in parentheses are the number of transactive statements of that type.

The most common type of transact detected was ***extension***, which is defined as a further thought or elaboration. Extension can be of one's own reasoning or that of one's partner. The following example, from Pair 5 (P5), includes both:

> **S1:** *It's doing that. OK this goes from zero to less than or equal to. So I think it should be zero less than*
> **S2:** *So that would be the same for that one*
> **S1:** *Oh, yeah. Good call. Oh wait. Oh, yeah, yeah. I think you're right.*
> **S2:** *Yeah, that's plus one too. 5 by 5* [Extension of own]
> **S1:** *1 2 3 4 5*
> **S2:** *If we add… we still have one more in here.*
> **S1:** *Oh, yeah. 1 2 3 4... and it's still 6 rows.* [Extension of other]

All transcripts included repeated sequential occurrences of the *extension* type of transact. This appeared to be a natural way that pairs communicated, each member adding to what the other said or in some cases adding to what they had said previously. The extensions we observed were signs of constructive, cooperative pair interactions.

The second most common type of transact was ***feedback request***, where a participant sought verification of his/her reasoning from the partner. Some examples were obvious as they were phrased as questions, for example the following excerpts from P2:

> *If it does that, it will go like, down every time. Won't it?*
>
> *Will that fix it?*
>
> *I don't know how to fix that?*

There were also more subtle feedback requests such as that expressed here (also by P2):

> *Yeah, well, I was wondering… don't you have to ask 3 times for enter score1, enter score2?*

All pairs exchanged requests for feedback, another sign of productive pairing and trust. However, in all pairs, requests for feedback came from one partner at a disproportional rate (63%, 82%, 78%, 87% and 70% for each pair respectively). There are a number of possible explanations – one partner may be less confident than the other or less knowledgeable; personal style or gender differences may be at play; or the driver/observer roles may have affected the transactional pattern. This phenomenon deserves additional attention.

Critiques, or claims that a partner's reasoning missed an important distinction, took many forms, including questions, statements, exclamations, contradictions and even anthropomorphism:

> *Shouldn't that be a one?* [P5]
>
> *You forgot the one at the end.* [P4]
>
> *What about this part down here? That's going to mess it up.* [P5]
>
> *No that's right.* [P1]
>
> *Only 5 not 6* [P1]
>
> *Whoa!* [P1]
>
> **S1:** *They added the lines; well they don't need the new line character here.*
> **S2:** *Yeah, they do.* [P2]
>
> *It doesn't like that* [P1]

Most critiques were gentle reminders which took place in the natural ebb and flow of the conversation. The five pairs we observed did not engage in negative criticism.

Completions were also prevalent. *Completions,* where partners completed or continued unfinished reasoning, were difficult to distinguish from *extensions*. Examples include:

> **S1&S2:** *Side1 equals side2 and side2 has to equals side3.*
> **S1:** *Which means side3 equals side2* [P1]

S2: *We want it, ok so those are all doing their thing, else and then it'll go to the next line so we need it to be after…*
S1: *Cause we don't want it to print the answer every time so after this guy, right?* [P4]

When communicating effectively, pairs often finished one another's thoughts, sometimes even speaking in unison, a sign that their reasoning was also in sync.

Pairs used ***justification requests*** to ask for clarification. *Justification requests* were sometimes posed as questions and sometimes not:

Why'd you choose 6 6 6? [P1]

What do you mean it doesn't? [P1]

Huh? [P5]

You totally lost me with what you're doing [P1]

Requests for justification indicated that one partner was not following the reasoning of the other partner. The tone of such requests was often a bit irritable. *Justification requests* occurred relatively infrequently in the pairs we observed.

Clarification was used rarely. Through the use of clarification a speaker corrects a misunderstanding about what he or she was trying to say.

S1: *You didn't put rectangle3.*
S2: *Yes, I know I'm not exactly the best speller, but I'm pretty sure I spelled rectangle right considering it's right up there.*
S1: *No, I mean there should be a 3.* [P3]

The pairs we observed rarely corrected misunderstandings verbally.

Paraphrase also did not occur often:

S1: *Oh, wait, here we have to remember, is our num3 bigger than our num2, yes. is our num2 bigger than our num1, yes. Now is our num2 smaller or is num3*
S2: *That's the same thing as num2 greater than num3.* [P1]

We speculate that our participants have little practice at paraphrasing the words of another, particularly in a programming context. Increased use of this skill could be helpful during debugging.

There were only two instances of ***integration*** (combining two positions into a common view) and none of ***juxtaposition*** (simply re-stating the position of each party). These transactions seem less applicable to debugging, when a correct answer exists and must be found, than they might be to other phases of programming, such as designing or writing code.

4.2 Transactive discussion vs. success

The number of debugging problems attempted and fixed by each pair is shown in Table 4. Pairs 1 and 5 ran out of time while working on their last problem before they were able to find and fix the bug. Pair 4 did not follow the experiment instructions correctly, and attempted to debug the problems in alphabetical order instead of the order specified in the list (Table 2). As a result, they spent 28 minutes working on one of the harder problems (Calculator1) that was not attempted by any of the other pairs.

Table 4: Problem Counts by Pair

	Number of Problems	
Pair	Attempted	Fixed
1	10	9
2	10	9
3	6	5
4	4	4
5	10	7

Of the pairs who attempted to debug the programs in the correct order, Pair 3 worked on the fewest problems, and had the lowest percentage of *completion* transactions. Only 7% of their transactive statements were completions, while the percentage of completions for the other four pairs ranged from 16 to 20%. In a completion transactive statement, one student is able to complete or continue the thought of another. This type of reasoning may be important for successful debugging. Completions may also be a natural communication style for partners with comparable coding ability, a factor that has been found to contribute to pair success [3].

Pair 3 also engaged in the least amount of discussion. There were only 171 utterances made by the two students in this pair, which was less than half as many as that made by the other pairs (which ranged from 351 to 389). In line with Teasley's earlier findings [17], we too speculate that the mere act of talking with a partner leads to increased debugging success.

Pair 5 had the lowest percentage of correctly debugged programs, and they also had the lowest percentage of *critiques*. Recall that a critique is a transactive statement in which one partner suggests that the other partner's reasoning misses an important distinction, or involves a questionable assumption. Only 13% of the transactive statements made by this pair were critiques, while the percentage for the other pairs ranged from 21 to 27%. This suggests that students who understand their partner's actions well enough and have the confidence to critique them may be more successful at debugging. Along these lines, Thomas et al. found that pairs in which both students had similar levels of confidence in their own coding skills received the highest scores on assignments [18].

Conversations between pairs included statements which were not necessarily transactive reasoning or about problem solving. The following excerpt is not transactive, but did lead to a solution.

S1: *Oh, you're right. So this one up here is just. Oh, will you go up to the example one? I want to see what it… Ok, for example, for 5 and 9. Oh.*
S2: *Do you want to try 5 by 9?*
S1: *1 2 3 4 5 6 7 8 9. Oh, OK. Gotcha. Yeah, let's try 5 and 9 so that we...*
S2: *Enter the number of rows.*
S1: *5 rows*
S2: *This is 5.*
S1: *9. Let's see if it looks like theirs. Perfect. OK.*

The pair is testing whether their output matches the example given in the problem statement, which is an important debugging strategy, especially for novices. Even so, none of their utterances quite fit into one of the prescribed transactive categories used in our analysis.

In contrast, the next excerpt is neither transactive, nor about the problem to be solved:

S1: *Rectangle3. Okie-dokie. So it's supposed to look like that.*
S2: *See what it looks like now.*
S1: *OK, javac rectangle3 dot java.*
S2: *Got somewhere to be?*
S1: *No, it's, she emailed the thing, the sheet about… OK.*
S2: *This is cool. we should put that.*
S1: *Oh, the. It's… phone is not happy with me. It's smashed.*
S2: *Oh, no. What'd you do?*
S1: *Smashed it in my car door. On accident. Uh. OK, let's see what it's doing… So it's just doing a normal rectangle.*

While this sort of chitchat is obviously irrelevant to the problem at hand, it may help partners become more comfortable working with each other.

The proportion of the conversation that was transactive was smaller for pairs 2 and 3 than it was for the other pairs. Only 18 or 19% of their utterances were transactive, while the percentage for the other pairs ranged from 26 to 28%. However, they exhibited different levels of success at debugging the programs, as pair 2 was able to debug nine problems while pair 3 only debugged five. It is also worth noting that pair 3 spoke approximately half as much (as measured by changes in speaker turns) as all the other pairs. Although sample size prevents any definitive conclusions, it suggests that while transactive dialog may be an important element of successful pair debugging, other elements are also influential and deserve additional investigation.

4.3 Other Pair Interactions

In this section we offer additional observations about pairs which are not based in transactive discourse analysis yet still seem relevant to the debugging process.

4.3.1 Praise and humor

Some pair members were very positive in their praise of each other or of their pair. Others offered words of encouragement.

I did it! Yes, we did it! Alright [P3]

We are fantastic [P5]

Ah, perfect. I don't know how you did that, but good job. [P5]

Nicely done. [P2]

That is amazing. How did you figure that out? [P5]

Good call, I said good call [P1]

Humor was sometimes used, indicating partners were comfortable with each other.

I see why it does scan dot next. So while it doesn't equal stop, so we're assuming that no one has named their child stop. That's a pretty hefty assumption [P5]

Let's say, Leah sold 90 tickets cause she's awesome. OK, good job Leah. And then Peter sold one. He got his mom to buy one. [P4]

Although humor cannot be taught, positive reinforcement can and should be.

4.3.2 Getting stuck and unstuck

Pairs, like any novice debuggers, sometimes found themselves stuck. Some were distracted by cosmetic differences in programs which sometimes sidetracked them from finding logic errors. However, students had a number of strategies for getting unstuck including 'just trying' things,

S1: *I don't know. I'm just sitting here. I feel like we should do something.*
S2: *Go ahead*
S1: *Umm, I don't know what though.*
S2: *Just try, see.* [P4]

rolling back to a previous version of the code,

We can always go back to what we have. [P4]

and seeking additional information from outside sources or previous examples.

Will we get in trouble if we use Google? [P2]

Do we have the little like Java happy sheet that was from the tests? [P4]

There's one of the ones that we did earlier that had the quit. I'm pretty sure. [P2]

From time to time, the participants would express frustration or anger. They found ways to control their emotions as a way of getting unstuck or preventing themselves from getting stuck:

We won't just get mad if it … if we go all funky [P2]

In addition, we detected frustration, mental fatigue, confusion and other emotions which are typical during problem solving, especially for novices. Being part of a pair gave them an opportunity to air, and perhaps work through, any negative feelings.

5. THREATS TO VALIDITY

The small number of participants in this study prevents us from drawing broad conclusions from these results. The disproportionate number of female volunteers adds an interesting element to the study but prevents firm conclusions about male/female or male/male patterns of interaction (the class as a whole was closer to 70% male).

The task of removing individual bugs from a set of small programs is also different from the more iterative code-test-debug cycle that novices typically engage in during introductory programming courses. Katz and Anderson [13] found that debugging strategies used by novices differ depending on whether they are debugging their own program or one written by someone else. Having not seen these programs before may have had an influence on the debugging strategies and success of participants in this study.

The choppy nature of participants' back-and-forth discussion made coding of the transcripts quite challenging. Additionally, some utterances did not quite fit into any of the transactive categories, yet seemed to express the general spirit of transactive discourse.

6. IMPLICATIONS FOR TEACHING AND FUTURE RESEARCH

This study addressed the questions of 1) What is the nature of the discourse between pair programmers as they debug? and 2) What differences, if any, are there in the discourse of successful and unsuccessful pair debuggers?

With regard to the nature of pair discourse, the extension, feedback request, critique, and completion transactive types were most commonly used. Justification occurred infrequently, and paraphrasing, clarification, and integration were used rarely. Juxtaposition was not detected at all. We also observed non-transactive discussions that led to solutions.

We found that pairs that talked more and used completion transactions more often attempted to solve more problems. Those that used critique transactions more often successfully debugged a higher percentage of the problems.

While this study's results should not be overstated, its methodology suggests some avenues for future exploration: pair programming appears to be a reasonable means of drawing out students' thinking about debugging. Transactive discussion categories also offer a fresh lens through which to understand pair interactions, and may be helpful in explaining other success factors such as comparable ability [3] and self-confidence [18].

Both this research and the work of Teasley [17] suggest that pairs who engage in transactive discussion will be more successful than those who do not. From a pedagogical perspective, it may be helpful to give students examples of the different types of transactive interactions situated within a programming/debugging context as a way of demonstrating how to pair program effectively and, in general, become better collaborators. Examples include asking for affirmation before making a change: "Does it make sense to swap the loops around?"; paraphrasing their partner's reasoning: "So, what you're doing is stopping before the loop goes out-of-bounds?"; and critiquing or questioning a partner's actions: "Why did you move that line?" In addition to teaching students how to play specific roles such as that of the traditional driver and navigator, perhaps we should also teach them ways of effectively communicating in pairs.

This preliminary analysis of the data has brought to light several new questions that are worthy of further study including:

- Is use of completion typical in successful pair efforts?
- How important is the ability to critique the partner's work?
- Are feedback requests stylistic, gender-related, or indicators of deeper problems?
- Is the key to pair success not so much following prescribed patterns, but more about being given the opportunity to reason out loud, share thoughts and bounce ideas off of each other?

Future work with this dataset will involve looking more specifically at pairs' debugging strategies in combination with the screen capture and survey data, possibly also in comparison to the transactive categorizations.

7. ACKNOWLEDGMENTS

We are grateful to Josh Tenenberg and Tina Ostrander for their helpful contributions to early phases of this work, to the anonymous reviewers for their thoughtful comments and suggestions, and to Crystal Hartman for her careful transcription. We also greatly appreciate Vicki Plano-Clark's useful insights and suggestions regarding qualitative research. This study was supported by a Regency Advancement Award from Pacific Lutheran University. Data analysis was facilitated through a workshop grant from the National Science Foundation (DUE 0923592). Any opinions, findings, and conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] Berkowitz, M. W. and Gibbs, J. C. 1983. Measuring the development of features of moral discussion. *Merrill-Palmer Quarterly* 29, 399–410. Referenced in Teasley [17].

[2] Braught, G., Eby, L.M., and Wahls, T. 2008. The effects of pair-programming on individual programming skill. In *Proceedings of the 39th Technical Symposium on Computer Science Education* (Portland, OR, USA, March 12-15, 2008). SIGCSE '08. ACM Press, New York, NY, USA, 200-204.

[3] Braught, G., MacCormick, J., and Wahls, T. 2010. The benefits of pairing by ability. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (Milwaukee, Wisconsin, USA, March 10-13, 2010). SIGCSE '10. ACM, New York, NY, 249-253.

[4] Carver, J.C., Henderson, L., He, L., Hodges, J., and Reese, D. 2007. Increased retention of early computer science and software engineering students using pair programming. In *Proceedings of the 20th Conference on Software Engineering Education and Training* (Dublin, Ireland, July 3-5, 2007). CSEET '07. IEEE Computer Society, Washington, DC, 115-122.

[5] Chaparro, E.A., Yuksel, A., Romero, P., and Bryant, S. 2005. Factors affecting the perceived effectiveness of pair programming in higher education. In *Proceedings of the 17th Workshop of the Psychology of Programming Interest Group* (University of Sussex, Brighton, UK, June 26-July 1, 2005), 5-18.

[6] DeClue, T.H. 2003. Pair programming and pair trading: Effects on learning and motivation in a CS2 course. *Journal of Computing Sciences in Colleges* 18(5), 49-56.

[7] Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., and Zander, C. 2008. Debugging: Finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education* 18(2), 93-116.

[8] Hale, J.E., Sharpe, S. and Hale, D.P. 1999. An evaluation of the cognitive processes of programmers engaged in software debugging. *Journal of Software Maintenance: Research and Practice* 11, 73-91.

[9] Hanks, B. 2007. Problems encountered by novice pair programmers. In *Proceedings of the Third International Workshop on Computing Education Research* (Atlanta, Georgia, USA, September 15-16, 2007). ICER '07. ACM Press, New York, NY, USA, 159-164.

[10] Hanks, B. and Brandt, M. 2009. Successful and unsuccessful problem solving approaches of novice programmers. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (Chattanooga, TN, USA, March 4-7, 2009), 24-28.

[11] Hanks, B., McDowell, C., Draper, D., and Krnjajic, M. 2004. Program quality with pair programming in CS1. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Leeds, UK, June 28-30, 2004). ITiCSE '04. ACM Press, New York, NY, 176-180.

[12] Hughes, J. and Parkes, S. 2003. Trends in the use of verbal protocol analysis in software engineering research. *Behaviour & Information Technology* 22(2), 127-140.

[13] Katz, I. and Anderson, J. 1987. Debugging: An analysis of bug location strategies. *Human-Computer Interaction* 3(4), 351-399.

[14] McDowell, C., Werner, L., Bullock, H. E., and Fernald, J. 2006. Pair programming improves student retention, confidence, and program quality. *Communications of the ACM* 49(8), 90-95.

[15] Mendes, E., Al-Fakhri, L.B., and Luxton-Reilly, A. 2005. Investigating pair-programming in a 2nd-year software development and design computer science course. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Caparica, Portugal, June 27-29, 2005). ITiCSE '05. ACM Press, New York, NY, USA, 296–300.

[16] McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L. and Zander, C. 2008. Debugging: A Review of the Literature from an Educational Perspective, *Computer Science Education: Special Issue on Debugging* 18(2), 67-92.

[17] Teasley, S. D. 1997. Talking about reasoning: How important is the peer in peer collaboration? In Resnick, L. B., Saljo, R., Pontecorvo, C., and Burge, B. (Eds.) *Discourse, Tools, and Reasoning*. Springer-Verlag, Heidelberg, Germany, 361-384.

[18] 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 34th SIGCSE Technical Symposium on Computer Science Education* (Reno, Nevada, USA, February 19-23, 2003). SIGCSE '03. ACM, New York, NY, 363-367.

[19] VanDeGrift, T. 2004. Coupling pair programming and writing: learning about students' perceptions and processes. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (Norfolk, Virginia, USA, March 03 - 07, 2004). SIGCSE '04. ACM, New York, NY, 2-6.

[20] Vessey, I. 1985. Expertise in debugging computer programs: A process analysis. *International Journal of Man-Machine Studies* 23, 459-494.

[21] Werner, L., and Denner, J. 2009. Pair Programming in Middle School: What Does It Look Like?. *Journal of Research on Technology in Education*, *42*(1), 29-49.

[22] Werner, L. L., Hanks, B., and McDowell, C. 2004. Pair-programming helps female computer science students. *Journal on Educational Resources in Computing (JERIC)* 4(1).