Industry-Inspired Guidelines to Improve Students' Pair Programming Communication

Mark Zarb
School of Computing
University of Dundee
Dundee, Scotland
markzarb@computing.dundee.ac.uk

Submitted to the ACM Student Research Competition Grand Finals 2013

ABSTRACT

Novice pair programmers find communication within their pairs to be one of the greatest difficulties they face when starting to pair program. However, developers cannot pair program without a certain amount of communication. This research describes the development of an analytic coding scheme derived from observation of the communication of expert pairs working in industry. Communication patterns identified from these expert pairs are being used by lecturers to help novices learn to be more effective in their pair communication.

1. INTRODUCTION TO PAIR PROGRAMMING

Pair programming is a method which describes two programmers working together on the same computer, sharing one keyboard. Typically, each member of the pair takes a different role, swapping roles frequently: the driver creates the code whilst the navigator reviews it [1]. Pair programming requires its pairs to communicate frequently, which leads the pair to experience certain benefits over "solo" programming, such as a greater enjoyment, and an increased knowledge distribution [2].

It is one of the key aspects of Extreme Programming, which "favours both informal and immediate communication over the detailed and specific work products required by any number of traditional design methods" [3].

In their paper describing pair programming in an education environment, Srikanth et al. [4] report several advantages to this methodology over more traditional programming methods:

- Students working in pairs reported a higher satisfaction, a quicker problem solving rate, and had improved team communication and effectiveness, when compared to "solo" programmers [3];
- Pair programming students were shown to be more likely to experience more confidence in their submitted work in comparison to "solo" programmers. Their levels of comprehension of new topics and learning were also significantly improved [5, 6].

The research investigated here illustrates how novices find communication to be a barrier when pair programming, and investigates communication trends and patterns for expert pairs. These patterns are cast in the form of guidelines and examples, which are used to assist novice pair programmers in learning to communicate more effectively when working together.

2. BACKGROUND AND RELATED WORK

When working in a pair, programmers are expected to collaborate verbally and non-verbally. This can cause social discomfort for both the driver and the navigator, leading to reduced communication effectiveness and lower productivity [7]. Many programmers approach their first pair experience with a sense of scepticism, having doubts about three areas: (i) their partner's working habits and programming style; (ii) reaching agreement on the implementation process; (iii) the added communication demands that this style of programming requires [3]. In a pilot experiment, roughly 50% of first-time novice pair programmers noted that the various forms of communication difficulties within the pair contributed to "communication" being the main problem with the pair programming process [6]. Furthermore, Stapel et al. [8] hypothesise that there could be a difference in the rate of communication between novice pair programmers and professional ones, in that the novices might be communicating less frequently with their pair partner. Communication is frequently cited as a "vital aspect" of pair programming [9-11]. It is seen that the pair's communication could determine the success of a pair programming experience; ultimately, if the pair does not communicate, then the programmers are effectively only watching each other code [11]. Communication hence is seen as being not only an integral contributor to the success of pair programming [7], but also as one of the main causes of failure [6].

Despite a fair amount of research into pair programming, it is not fully clear what the communication within the pair contributes, or how this is linked to success. If this can be understood, it would lead to improved practices for teaching pair programming to novices, and could help identify obstacles to successful pairing in industrial settings.

3. APPROACH

3.1 Analysing Expert Communication: *pairwith.us*

A series of videos depicting two expert software engineers pairing together was produced independently of this study. The series was made publicly available through vimeo.com, a video-sharing service, with the aim of introducing pair programming to a wider audience. Both members of the pair are agile coaches and avid programmers who have over ten years of industry experience across multiple sectors. At the time of filming, the unscripted videos were broadcast online, and then archived [12] without any post-processing or editing. The files incorporate live code updates, a video of the pair, and an audio feed. The three tracks (code, video and audio) are, for the most part, synchronized. The project, *pairwith.us*, consisted of sixty videos, all authored by the same pair, progressively working on the same code.

The *pairwith.us* team was contacted prior to the study reported here, and gave consent for analysis of these videos for initial research into communication and pair programming. These videos allowed study of the pair's speech, gestures, and actions [13] by using qualitative methods.

3.2 Data Analysis

An approach informed by Grounded Theory was chosen, as this method is frequently used as a framework for the analysis of qualitative data [14]. It encourages continual refinement of the generated theory through frequent comparisons between the collected data and its analysis.

Traditionally, defining components of grounded theory include the following stages for the researcher [15]:

- Simultaneous involvement in data collection and analysis through open coding methods;
- Construction of analytic codes from data, not from *a priori* hypotheses;
- Continual comparisons of the data with the codes during each stage of the analysis;
- Iteratively developing and refining a theory during each step of data collection and analysis.

3.2.1 Open Coding

Initially, several videos were eliminated from further study. These included videos that had poor audio quality, lack of video feed, or bad synchronization, which would have caused problems during analysis. Thirty-one videos that had been recorded over a three-month period were ultimately chosen for further analysis, with each video lasting for approximately 30 minutes.

Analysis was made directly from the recordings at this stage rather than via full transcriptions (creating full transcripts is known to be a very time-consuming process that is subject to human error [16, 17]).

Large sections of the video were watched in a continuous, immersive fashion, with pauses only to note keywords and interesting information. Once all videos were watched, a document was created that enumerated and identified the various themes and types of communication exhibited by the two programmers during the course of the recorded project. Various instances of communication and behaviour were observed [18].

3.2.2 Creation and Refinement of Analytic Codes

Following Grounded Theory, the pair's communication topics were continually compared to each other and the originating data in an attempt to condense them into a preliminary set of keywords. These analytic codes represented the pair's various states of communication.

In order to evaluate the validity of the chosen analytic codes, as well as further refine them, a sample of five videos was randomly chosen from the selected set of thirty-one. These videos were fully transcribed, and coded (segmented), using the analytic codes. This allowed identification of analytic codes that were too vague, or that were not able to completely categorise certain parts of the transcript. For example, the code for 'Silence' was eventually divided into two: 'quiet' silence and pair muttering (e.g. whilst the driver is typing out code). The analytic codes were regularly reviewed and refined with each viewing of the selected videos and transcripts, until the list of preliminary analytic codes could completely categorise all the transcripts.

Colleagues (n=3) from within the School of Computing at the University of Dundee were recruited in order to perform an assessment of the codes' inter-rater reliability. All colleagues were recruited from different research groups, with no ties to the study. The raters were provided with a list of nine analytic codes and asked to apply them to various sections of video and transcripts. Inter-rater reliability tests resulted in Kappa = 0.71 ($p < 0.001$), indicating a high agreement with the initial codes for the same video segments. The raters also gave feedback that highlighted the fact that some of the analytic codes were not based on communication, but rather on the pair's behavior (considered a non-verbal interaction). These analytic codes (such as 'Switching Roles') were then removed from the coding scheme as the researcher was specifically interested in verbal interaction. The coding scheme was therefore focused entirely on the pair's verbal communication.

The communication coding scheme thus created consists of the following analytic codes, ordered by frequency of occurrence within the *pairwith.us* videos, as depicted in Table 1:

- *Suggesting*, where a member of the pair starts planning what the next step should be (e.g. "Why don't you try doing it this way?").
- *Thinking*, normally showing the pair not speaking, both focused towards trying to solve a current issue with the code/design.
- *Unfocusing,* where the pair discusses something completely unrelated to their current work (e.g. a movie they saw the previous night or lunch plans).
- *Explaining*, where a member of the pair clarifies a bit of the logic, or the code, to their partner.
- *Reviewing*, where the pair returns to a previous piece of code and attempt to recall how it worked (e.g. "Right, we have this method here – it was used to create the relevant objects which will allow us to eventually modify…").
- *Logic Discussion/Coding*, showing the pair working together, discussing objects/methods not directly tied to the code being written ("I read a similar way of implementing a method like this on a blog discussing the merits of …").
- *Muttering*, when the pair is not speaking using normal conversational forms, but rather is typing at the keyboard or jotting things down on a notepad, muttering out as the work is progressing (e.g. "public" … "set" … "arguments").

These codes were sent back to the *pairwith.us* team, who offered feedback and insight into some of the codes. The pair indicated that 'Muttering', for example, was an important state to them, as it allowed the navigator to understand that the driver was following a line of thought whilst working, and was not simply stuck. This meant that the navigator would not try to intervene during a muttering stage, allowing the driver to complete his thought process.

Table 1. Analytic Code frequency in the videos obtained from *pairwith.us*

Analytic Code	Frequency
Suggestion	28.4%
Thinking	19.0%
Unfocusing	13.3%
Explaining	11.4%
Reviewing	10.4%
Coding	9.1%
Muttering	8.4%

Table 2. Analytic Code frequency in the videos obtained from professional pairs in industry

Analytic Code	Frequency
Suggestion	24.8%
Review	23.5%
Explanation	21.5%
Muttering	10.1%
Coding	8.4%
Unfocusing	7.7%
Thinking	4.0%

3.3 Analysing Expert Communication: Including Other Professional Pairs

Following the creation of the analytic codes, it was important to verify whether the coding scheme created would be valid for use across a range of expert pairs, as the eventual aim is to create a framework elucidating communication dynamics within pair programming in general.

Several companies that practised pair programming were contacted via relevant mailing lists and asked to volunteer to share their pair programming experiences. Two companies gave consent for observation and recording of individual pair sessions in their industry setting.

Various pairs were observed for set periods of time throughout the day within both companies. Both companies specialized in software development – one for networking, and the other for marketing and digital media. The environment was made as natural as possible when videos were recorded for subsequent analysis: the camera was placed outside the developers' line of sight and the researcher was not present during the recording period. All pair members (n=22) gave informed consent, and also completed a survey detailing their experience and history of pair programming. When asked to rate their personal feelings about the benefits of pair programming over solo programming, the mean response was 4.1 (±0.7), on a scale ranging from 1 ("no benefit") to 5 ("very beneficial").

The captured videos were fully transcribed and coded. The frequency of occurrence of each analytic code was extracted, as given in Table 2, and compared with the frequency in the *pairwith.us* videos (Table 1) as discussed below.

There are differences noted between the code frequency within the two sets of videos. Most notably, compared with the *pairwith.us* videos, there is more use of Reviews and Explanations and less use of Thinking (long, silent periods) in the larger set of 11 videos. It is possible that this results from the fact that the sets of videos were captured in quite different settings – the *pairwith.us* videos were recorded with an eye towards their instructional value whereas the industry videos were recorded in a fast-paced work setting. The former were therefore quite contemplative and the latter had less 'quiet' time and more focus on deadlines.

Importantly, the analytic codes derived from *pairwith.us* were successfully used to fully code the transcripts produced from the pairs in industry, showing that they were generalizable beyond the *pairwith.us* setting.

3.4 Creation of Guidelines

Identifying and validating a set of communication states for expert pair programming is the first step towards understanding how expert pairs communicate. By reviewing the co-occurrence relationships between analytic codes, frequently occurring patterns of communication throughout the various industry videos were determined. Transitions between communication states for all pairs were examined, in accordance with grounded theory methodologies. This process led to a better understanding of the communication exhibited within expert pairs. Thereafter, a set of concise instructions were constructed that could be delivered to novice pairs to improve their understanding of successful pair communication.

To understand the most commonly occurring patterns, the codes were analysed to determine which preceded and followed each other by extracting the relevant data from Transana[1], the software tool used to code the videos. The most common analytic code transitions identified were verified against the pair videos. Figures 1-3 demonstrate the three patterns that were found most likely to occur. In this section, each pattern is explained and guidelines arising from this pattern are presented.

3.4.1 Restarting Pattern

At several points, the pairs were observed to completely change the topic of discussion from their work to a more casual topic (e.g. their Father's Day plans, or a review of a recently released film). An informal interview with some of the pairs showed that this action was a conscious one: whenever they were stuck for a period of time, the pair made an effort to break their focus or to stop their current actions to discuss something completely unrelated.

Figure 1 depicts this pattern with the most common next actions. The data shows that unfocusing is most commonly followed by a reviewing action (e.g. "Let's get back to this – what were we doing?"), a suggestion (e.g. "Why don't we try to do it this way?"), or complete silence, whilst the pair pondered their next move.

[1] http://www.transana.org/

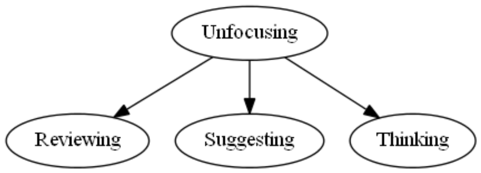

Figure 1. Restarting Pattern.

Guidelines suggested by this pattern are:

- If you and your partner are stuck in a thinking/silent period and cannot seem to progress, actively break your focus by discussing something completely off-topic and unrelated to the issues at hand. This will allow you to tackle the problem with a fresh outlook.
- Following this stage, attempt to:
    - Look back on your last couple of steps and review your previous work.
    - Identify a fresh start.
    - Try to suggest next steps related to your end-goal in order to make progress.
- If your partner is attempting to break focus, don't dismiss this. Breaking one's focus using jokes, private conversations, etc. can lead to a fresh perspective, which you and your partner may need.

3.4.2 Planning Pattern

Following a suggestion, the pair was most likely to review the existing code to understand how refining it might help them achieve their overall goal. As part of this conversation, one of the pair would normally explain the underlying structure.

A suggestion could also separately lead to an explanation – for example, whilst discussing a method, rather than reviewing the structure, the pair would explain implications that the method would have with respect to their goal (Figure 2).

This pattern occurred most often at the start of the pairing session: sessions observed typically started with the pair reviewing legacy code and then attempting to devise ways to reduce error messages or solve problems.

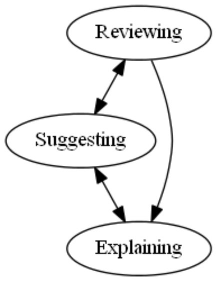

Figure 2. Planning Pattern.

Guidelines arising from this pattern are:

- Suggestions and reviews are useful states that will allow you to drive your work forward. When in these states, feel free to communicate about a range of things (e.g. review previous code, suggest an improvement, review methods to be changed, suggest potential impact).
- At any stage, do not hesitate to ask your partner for clarification about any suggestions they make or actions they are carrying out that you might not understand.
- Think about what your partner is saying and doing. Offering an interpretation of the current state can help move the work forward.

3.4.3 Action Pattern

The action pattern occurred mostly whilst a pair was trying to create code. These instances would typically start with a member of the pair making a suggestion as to what should be coded, or how a certain error should be tackled. The pair would then either talk about the code, or, alternatively, the driver would start typing and muttering. The muttering frequently led to the navigator making suggestions based on what the driver was saying, which acted as a prompt for discussions (Figure 3).

Guidelines arising from this pattern:

- NAVIGATOR: Whilst the driver is coding, actively look to make suggestions that contribute to the code.
- NAVIGATOR: If the driver is muttering, use this opportunity to make sure your suggestions have been properly understood.
- DRIVER: Whilst you are programming, or thinking about your code, voice your thoughts (even if it is just mumbling and muttering while you're typing). This helps the navigator know that you are actively working, have a clearer sense of how you are approaching the task, and will allow for them to make useful suggestions based on your current actions.

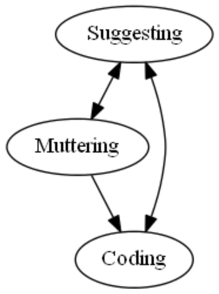

Figure 3. Action Pattern.

3.5 Involving Novice Students

The guidelines were introduced to a set of students, to investigate the experiences resulting from their application.

3.5.1 Observations

One of the modules within the School of Computing at the University of Dundee is “Agile Software Engineering”, in which students learn various agile methods and are asked to adopt an agile approach for their assignments over the course of a semester. In 2012 the class was randomly split into seven teams of 3-5 people. Whilst working on assignments for this module, teams were asked to practise pair programming rather than program “solo”. Whilst coding, each team would decide which pairs of students would tackle certain tasks. The class gave informed consent for their pairing sessions to be observed for the duration of the semester.

During a pre-test period of four weeks, students were observed using pair programming techniques on a weekly basis during their normal lab time. Following the end of the pre-test, each team was invited to a semi-structured interview to give their thoughts on pair programming thus far.

The class was then randomly split into Group A and Group B, for the delivery of the guidelines, with the latter acting as a control group. This split was presented to the class as an opportunity to practise either "advanced pair programming" or "advanced scrumming and team management". Group A (consisting of 16 students) was introduced to the guidelines above and asked to adhere to them during future pair programming sessions. Introduction was carried out by explaining each pair programming guideline individually, supported by examples from the *pairwith.us* footage. Group B (consisting of 12 students) was given control guidelines related to team meetings and scrumming, created by the class lecturer. All introductory sessions (pair programming and control) had approximately the same format and duration.

Following another four-week period of observation, each group was asked to attend a final semi-structured interview to once again give their thoughts on pair programming. Each team within Group A was asked about their use of the guidelines, whereas Group B was exposed to the pair programming guidelines, which prompted discussions. Both groups reacted positively to the guidelines, stating that they seemed natural. It was noted that the Group B students related the guidelines to their own experiences of pair programming and stated that they felt the guidelines should be introduced earlier during the module to aid their pair experience. Perceptions of the value of the guidelines are evidenced by the following quotes:

"I found that the restarting pattern came in useful when I was thinking about other modules as well."

"The action pattern, noticing the driver was muttering… that was useful."

"We definitely use the restarting pattern […] – we went to the shop, getting away from the computer was helpful."

"I can see the benefit – it gives a structured way to keep things going".

"It may have been a useful thing to know at the start."

"There's a definite benefit in introducing this – it's *deeper* than being taught formal steps."

4. FINAL RESULTS

A final study was carried out to determine the effect that the guidelines had on student communication. Undergraduate students from within the department were recruited, and randomly sorted into pairs. For the purpose of this study, each pair consisted of two students at the same level of. Seven pairs were randomly chosen to be 'exposed' to the guidelines, which were supplemented with video clips from the *pairwith.us* project. The unexposed pairs (n=6) would act as a control group.

Each pair was invited to attend a task-based study at various times over a period of two weeks, adapted from the study reported in a paper by Murphy et al [19]. The study was structured as follows: the pair would enter the task room, and be presented with 19 files of compiled code, with logical errors. If the pair was selected to be part of the experimental group, they would watch the guidelines video and be presented with a list of guidelines prior to starting the task. As per the original study, students were asked to work their way through the list of buggy programs, and solve as many logical errors as possible within the 45-minute limit.

Following the study, students were individually asked to fill in a survey where they rated several statements on a Likert scale ranging from 1 (disagree) to 5 (agree), expanding on their thoughts about pair programming, and the communication they experienced with their partner during that particular pairing session. The analysis of these results is reported below.

Independent-samples t-tests were used to (separately) compare ease of communication and perceived partner contribution for novices who were exposed to the guidelines, and those who were not.

There was a significant difference in the scores reported for "ease of communication" for exposed (M=4.57, SD=0.514) and unexposed (M=3.92, SD=0.900) novices; $t(24) = 2.32$, $p = 0.029$.

There was also a significant difference in the scores reported for "partner communication" for exposed (M=4.79, SD=0.426) and unexposed (M=4.17, SD=0.835) novices; $t(24) = 2.44$, $p = 0.023$.

These results statistically show that when exposed to the guidelines, individuals experienced an improved communication with their partners within their pair. This indicates that the patterns and guidelines presented in this paper can aid novice pairs to communicate better.

5. CONTRIBUTIONS

Experienced, confident pairs communicate in a small number of ways, and transitions between these identifiable states are evident. During this study, we have identified a set of analytic codes that accurately describe the communication states exhibited by expert pairs, as well as common transitions between these states. These communication patterns have informed the development of guidelines, which have been well received as providing useful help to novices learning pair programming.

Investigations indicate that students' experience of pair programming has been enhanced by knowledge of the guidelines. Furthermore, during post-study interviews, students who did not have access to the guidelines throughout the semester expressed a desire to be exposed to them as part of an introduction into pair programming.

Feedback from students exposed to the guidelines is positive and suggests they can improve their pairing experience. Furthermore, exposure to the patterns and associated guidelines has a significantly positive effect on the observed communication within novice pairs.

6. ACKNOWLEDGMENTS

The author would like to thank his supervisors, Dr Janet Hughes and Prof John Richards, for their attention and support throughout this project.

The author would also like to thank the programmers behind the *pairwith.us* project for allowing him the use of their videos, as well as the developers and team leaders at Sky Network Services and Unruly Media, who allowed access to observe and record their pair interactions.

The research work disclosed in this paper is funded by the Strategic Educational Pathways Scholarship (Malta). The scholarship is part-financed by the European Union – European Social Fund (ESF) under Operational Programme II – Cohesion Policy 2007-2013, "Empowering People for More Jobs and a Better Quality of Life".

7. REFERENCES

[1] Williams, L.A. and R.R. Kessler, *All I really need to know about pair programming I learned in kindergarten.* Communications of the ACM, 2000. **43**(5): p. 108-114.

[2] Bryant, S., P. Romero, and B. du Boulay, *The Collaborative Nature of Pair Programming*, in *Extreme Programming and Agile Processes in Software Engineering*, P. Abrahamsson, M. Marchesi, and G. Succi, Editors. 2006, Springer Berlin/Heidelberg. p. 53-64.

[3] Williams, L., et al., *Strengthening the Case for Pair Programming.* IEEE Software, 2000. **17**(4): p. 19-25.

[4] Srikanth, H., et al., *On Pair Rotation in the Computer Science Course*, in *Proceedings of the 17th Conference on Software Engineering Education and Training*. 2004, IEEE Computer Society. p. 144-149.

[5] Williams, L., et al., *In Support of Pair Programming in the Introductory Computer Science Course.* Computer Science Education, 2002. **12**(3): p. 197-212.

[6] Sanders, D., *Student Perceptions of the Suitability of Extreme and Pair Programming*, in *Extreme Programming Perspectives*, M. Marchesi, et al., Editors. 2002, Addison-Wesley Professional. p. 168-174.

[7] Cockburn, A. and L. Williams, *The costs and benefits of pair programming*, in *Extreme programming examined*. 2001, Addison-Wesley Longman Publishing Co., Inc. p. 223-243.

[8] Stapel, K., et al., *Towards Understanding Communication Structure in Pair Programming*, in *Agile Processes in Software Engineering and Extreme Programming*, A. Sillitti, et al., Editors. 2010, Springer Berlin Heidelberg. p. 117-131.

[9] Beck, K., *Extreme programming explained: embrace change*. 2000: Addison-Wesley Professional.

[10] Lindvall, M., et al., *Empirical Findings in Agile Methods*, in *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2002*. 2002, Springer-Verlag. p. 197-207.

[11] Gallis, H., E. Arisholm, and T. Dyba. *An initial framework for research on pair programming*. in *International Symposium on Empirical Software Engineering*. 2003.

[12] Marcano, A. and A. Palmer. *pairwith.us*. 2009 [cited 2012 July 31]; Available from: http://vimeo.com/channels/pairwithus.

[13] Bryman, A., *Social Research Methods*. 2012: Oxford University Press.

[14] Lazar, J., J.H. Feng, and H. Hochheiser, *Research methods in human-computer interaction*. 2009: Wiley.

[15] Glaser, B.G. and A.L. Strauss, *The discovery of grounded theory: Strategies for qualitative research*. 1967: Aldine de Gruyter.

[16] Chong, J., et al. *Pair programming: When and why it works*. in *17th Annual Workshop of the Psychology of Programming Interest Group*. 2005. Brighton, UK.

[17] Wetherell, M., S. Taylor, and S. Yates, *Discourse as data: A guide to analysis*. 2001: Sage Publications Ltd.

[18] Zarb, M., J. Hughes, and J. Richards, *Analysing Communication Trends in Pair Programming Using Grounded Theory*, in *Proceedings of the 26th BCS Conference on Human-Computer Interaction*. 2012, British Computer Society: Birmingham, United Kingdom.

[19] Murphy, L., et al. *Pair debugging: a transactive discourse analysis*. in *Proceedings of the Sixth international workshop on Computing education research*. 2010: ACM.