AUTHOR(S): ZARB, M., HUGHES, J. and RICHARDS, J.

TITLE: Further evaluations of industry-inspired pair programming communication guidelines with undergraduate students.

YEAR: 2015

Publisher citation: ZARB, M., HUGHES, J. and RICHARDS, J. 2015. Further evaluations of industry-inspired pair programming communication guidelines with undergraduate students. In Proceedings of the 46th ACM technical symposium on computer science education (SIGCSE 2015): keep connected, keep committed, keep computing, 4 - 7 March 2015, Kansas City, USA. New York: ACM [online], pages 314-319. Available from: https://doi.org/10.1145/2676723.2677241.

OpenAIR citation: ZARB, M., HUGHES, J. and RICHARDS, J. 2015. Further evaluations of industry-inspired pair programming communication guidelines with undergraduate students. In Proceedings of the 46th ACM technical symposium on computer science education (SIGCSE 2015): keep connected, keep committed, keep computing, 4 - 7 March 2015, Kansas City, USA. New York: ACM, pages 314-319. Held on OpenAIR [online]. Available from: https://openair.rgu.ac.uk.

Further Evaluations of Industry-Inspired Pair Programming Communication Guidelines with Undergraduate Students

Mark Zarb
School of Computing Science
Robert Gordon University
Aberdeen, Scotland
+44 (0)1224 262 768
m.zarb@rgu.ac.uk

Janet Hughes
School of Computing
University of Dundee
Dundee, Scotland
+44 (0)1382 385 195
j.z.hughes@dundee.ac.uk

John Richards
IBM T.J. Watson Research Center
Yorktown Heights
New York, USA
+1 914 945 2632
ajtr@us.ibm.com

ABSTRACT

Pair programming has several benefits when it is successfully used by students and experts alike. However, research shows that novice pairs find the necessary pair communication to be one of the main challenges in adopting this process. A set of industry-inspired pair programming guidelines have been derived and evaluated from qualitative examinations of expert pairs, with the aim of helping novice programmers communicate within their pair. This research describes a further evaluation of these guidelines with a number of student pairs, and demonstrates how novice pairs who were exposed to the guidelines became comfortable communicating with their partners.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer science education.

General Terms

Measurement, Performance, Experimentation, Human Factors, Standardization.

Keywords

Pair programming; communication skills; software engineering; collaboration; students; CS education research study.

1. INTRODUCTION

Pair programming is a method that describes two programmers working together, usually sharing a keyboard and a computer. Typically, each member of the pair takes on a different role, swapping roles frequently: the driver creates the code while the navigator reviews it [11]. Pair programming requires pairs to communicate frequently, which leads the pair to experience certain benefits over "solo" programming, such as greater enjoyment, and increased knowledge distribution [2].

Novices find communication to be a barrier when they are pair programming [9, 11], and industry-inspired guidelines have been presented as a possible solution [17]. These guidelines have been evaluated [16], with initial results showing that pairs who had been exposed to the guidelines reported an improved communication experience whilst debugging existing code.

This research presents a follow-up study that evaluates the guidelines with novice pairs beyond debugging. In the study reported in this paper, pairs were responsible for coding solutions to given problems. Therefore the aim is to find out whether these guidelines can have a positive impact on the communication experience of pairs who are actively creating new code.

2. BACKGROUND

Due to the nature of pair programming, communication - both verbal and non-verbal - occurs nearly continuously. Williams and Kessler [12] write that effective communication is "paramount", and Sharp and Robinson [8] describe pairing as a highly communication-intensive process. Within the classroom, pair programming is seen as being generally valuable [1, 12]. Students working in pairs have been found to be more satisfied with their work output, solve problems faster than non-paired students, and have improved team effectiveness. Paired students are also more likely to complete CS courses when compared to their solo counterparts, gain an improved comprehension of unfamiliar topics, and enjoy increased levels of confidence [3, 4, 9, 13, 14].

Many programmers approach their first pairing experience with scepticism, having doubts about their partner's working habits and programming style, and about the added communication demands that this programming style requires [13]. In a pilot study, approximately 50% of first-time novice pair programmers reported that they perceived communication to be the main problem with the pairing process [7]. Many authors simply state that *communication* is an issue; few studies have investigated which aspects of communication within an agile team are most problematic [8, 10].

2.1 Pair Programming Guidelines

Industry-inspired pair programming guidelines had earlier been created with the aim of delivering concise, industry-derived instructions to novice pairs to improve their understanding of successful pair communication. The creation of these guidelines has been previously reported in [15] and [17], with preliminary qualitative studies suggesting that the guidelines could help improve novice pairs' experienced intra-pair communication. A previous study has shown that when debugging code, pairs who

were exposed to these guidelines had a more positive communication experience [16].

Appendix 1 provides a summary of these guidelines.

3. EVALUATION

Results obtained from a previous study [16] suggest that use of the guidelines led to an improved communication experience when pairs were engaged in a debugging task. The study reported here aims to understand whether the guidelines could also help improve the communication experience in a task where pairs are creating new code.

Quantitative data was to be gathered, with the following hypotheses as per the original study:

- Exposure to the pair programming guidelines positively impacts the pair's success rate.
- Exposure to the pair programming guidelines leads to an improvement in the pair's ease of communication.
- Exposure to the pair programming guidelines positively affects the way partners contribute to the pairing session.

3.1 Methodology

3.1.1 Materials

One of the summer school programmes at the University of Dundee's School of Computing uses a custom programming tool that has been developed to teach programming topics: the Abstract Programming Environment (APE)[1]. The APE tool runs on the NetBeans IDE and provides a graphical front-end (Figure 1) which can be manipulated using Java code. This allows students to 'see' what they are programming. Note that the contrast in Figure 1 has been adjusted to make the image suitable for printing.

Figure 1: The APE graphical front-end

The APE tool consists of several challenges (or 'maps') in which students need to move the character around, eating a number of dots; students must write this movement using Java code. Once all the dots have been eaten, the 'map' is considered complete, and students can move on to the next one.

[1] The APE tool was created by Heron and Belford (see http://monkeys.imaginary-realities.com) and used with permission.

3.1.2 Participants

Participants were recruited from the School of Computing at the University of Dundee. An e-mail was circulated to all students, asking for their participation in exchange for a small compensation in the form of vouchers. A total of 28 participants were recruited (first-year undergraduates: 10 students; third-year undergraduates: 18 students). All had previously used Java as a programming language as part of their courses.

Pairs were arranged so that each pair consisted of students at the same year of study. Within each year, 50% of the pairs were randomly allocated to a group that would be exposed to the guidelines (n = 7 pairs), leaving the rest of the sample (n = 7 pairs) as a control group.

3.1.2.1 Previous Pairing Experience

Each participant was asked to complete a post-test survey immediately following their participation in the study consisting of questions relating to the individual's experience with solo programming, pair programming, and previous pair programming experience with their session's partner. These results were analysed to understand group tendencies and variance, as reported in Table 1 below and analysed using Mann-Whitney U tests.

Table 1: Students' Programming Experience

	Exposed Group		**Non-Exposed Group**	
	M	**SD**	**M**	**SD**
Solo Programming Experience (years)	3.7	2.17	2.7	1.86
Pair Programming Experience (years)	0.3	0.59	0.2	0.41

The data show that the groups had somewhat different levels of experience; on average, more individuals in the "exposed" pairs had solo programming experience. This difference may complicate interpreting further results. Few students reported previous experience with pair programming – furthermore, any reported experience was limited to a number of months, or 'since the start of the semester'. Statistical tests were carried out to establish whether the differences between the two groups were significant and whether they might cause the results to be biased:

No significant differences in 'solo' programming experience were found between the experimental and control groups: $U = 125$, $z = 1.266$, $p = 0.227$ ($p > 0.05$).

Similarly, no significant differences in pair programming experience were found between the experimental and control groups: $U = 106.5$, $z = 0.427$, $p = 0.670$ ($p > 0.05$).

While some caution is warranted, these results suggest that further analysis should not be unduly complicated by what appears to be a small difference in prior experience.

3.1.2.2 Perceived Benefits of Pair Programming

Likert scale data from the post-test surveys were analysed to determine whether there were any significant statistical differences reported between the students who were exposed to the guidelines and those who were not.

As each individual completed their own post-test survey, the population consisted of 28 students, 14 of whom were exposed and 14 students who were not.

Shapiro-Wilk tests were carried out to understand whether the data being analysed were normally distributed. *Ease of Communication* scores for both exposed and unexposed groups were not normally distributed ($p < 0.05$). Similarly, scores for *Perceived Partner Contribution* for both groups were not normally distributed ($p < 0.05$). As the data are not normally distributed for both sets of scores, non-parametric tests were used.

Students were asked to rate the statement '*I feel pair programming is more beneficial than solo programming'* on a 5-point Likert scale.

The exposed group (M=4.5, SD=0.52) and the control group (M=4.1, SD=0.62) report similar scores. As observed in previous studies, there was no significant difference in perceived pair programming benefit between exposed students (*Mdn = 4*.0) and unexposed students (*Mdn* = 4.5), $U = 133$, $z = 1.834$, $p = 0.067$.

These results show that following the session, the student perception was that pair programming was more beneficial than solo programming, regardless of whether they were exposed to the guidelines or not.

3.1.3 Procedure

The study was carried out during a 4-week period during the students' second semester of study. Pairs were invited to the test room separately, and on different days. The test room was equipped with one laptop, and consisted of a camera and a voice recorder. Ten APE maps were chosen at random for the students to solve. All pairs were given a maximum time-limit of 45 minutes to solve as many maps as they could in a sequential order. During this time, the recording equipment was switched on, and the researcher left the room.

Pairs were provided with a list of basic instructions to move the character (Table 2), but were free to implement solutions using any programming technique at their disposal (e.g. in this study, students have used *for* loops and *do..while* loops to move the character. Some of the pairs were also observed to write a parser, which allowed for a more domain-specific way of telling the character how to move across the map).

Table 2: Basic instructions for the APE tool

Instruction	What it does
main.move();	Makes the yellow character move one space forward in whatever direction is being faced.
main.turnLeft();	Makes the character turn 90 degrees to the left.
main.turnRight();	Makes the character turn 90 degrees to the right.

Each pair was responsible for the whole programming process: from discussing possible solutions, to attempting to implement the correct code and testing it.

Both the control group and the test group followed the process described above; prior to the task, pairs within the test group were also exposed to the pair programming guidelines by watching a short 3-minute video[2] which showed an experienced pair applying the guidelines in three separate scenarios. This video was supplemented by a printed copy of the guidelines (Appendix 1), which was left with the pair for reference.

Following the test period, the researcher would return, log the number of programs completed by the pair, and distribute the post-test surveys, which were completed individually by the members of the pair. The five-question survey was based on the survey used in [16], and was used to collect data from the individual developers immediately after their debugging session. This data was used to determine if there was any significant difference between the groups that could bias the results. Each survey consisted of Likert-scale questions relating to their experience with communication and partner contribution during the test, as well as questions on the student's experience with programming. These were used to measure central tendencies and variance within the groups, in order to ascertain that there were no significant differences between the groups that would threaten the validity of the work.

3.2 Results

Three measures were taken for each pair, based on the measures taken in [16]: *success* was measured by the number of programs completed successfully (when compared to the number of programs attempted); *ease of communication* and *perceived partner contribution* were measured using the post-test Likert scales as discussed above.

3.2.1 Successfully Completed Programs

Following the test period, the number of tasks attempted was noted by the researcher, and scored at a later date. Each attempt was scored by the researcher, and also compiled, to see if the correct result was produced (i.e. if each map was solved successfully). The total number of successfully completed tasks was then noted for each pair.

An independent-samples t-test was run to determine if there were differences in completion scores between pairs who were exposed to the pair programming guidelines (n = 7), and those who were not (n = 7). The tasks completed for each level of exposure were normally distributed, as assessed by Shapiro-Wilks test ($p > 0.05$), and there was homogeneity of variances, as assessed by Levene's Test for Equality of Variances ($p = 0.903$).

The exposed pairs completed a slightly greater number of tasks (4.0 ± 1.00) than the unexposed pairs (3.3 ± 0.76). The difference is not statistically significant: $t(12) = -1.508$, $p = 0.158$.

This result shows that exposing pairs to the guidelines does not increase their chances of successfully completing their tasks: exposure does not improve success rate at least for this short programming task.

3.2.2 Ease of Communication

'Ease of Communication' was reported as a Likert scale on the post-test survey in response to the following statement: "*During this session, I found communicating with my partner to be easy*". The scale ranged from 1 (strongly disagree) to 5 (strongly agree). The post-test survey results relating to *ease of communication*

[2] A copy of this video is available at the following URL: http://youtu.be/ONnYCT_LJio. Should this link be broken, please contact the lead author.

were analysed, and descriptive statistics were used to further understand the results (Table 3).

Table 3: Descriptive Statistics for Ease of Communication

	Exposed		Not Exposed	
	M	**SD**	**M**	**SD**
Ease of Communication	4.9	0.27	4.0	0.78

It can be seen that the students who were exposed to the guidelines reported a higher score than students who were not, with a lower variance.

As the data used is extracted from Likert scales, a Mann-Whitney U test was used for its analysis [6]. This test was run to determine any differences in *ease of communication* between the exposed group, and the control group.

There was a statistically significant difference in ease of communication scores between exposed students ($Mdn = 5.0$) and unexposed students ($Mdn = 4.0$), $U = 169$, $z = 3.721$, $p = 0.001$.

In this case, $p < 0.05$, therefore the null hypothesis (*the distribution of the pair's ease of communication is equal across the two groups*) was rejected.

3.2.3 Perceived Partner Contribution

'Perceived Partner Contribution' was reported as a Likert scale on the post-test survey in response to the following statement: "*Rate your partner's contribution to today's session*". The scale ranged from 1 (no participation) to 5 (excellent). Descriptive statistics were used to gain an overview of detail (Table 4).

Table 4: Descriptive Statistics for Perceived Partner Contribution

	Exposed		Not Exposed	
	M	**SD**	**M**	**SD**
Perceived Partner Contribution	4.9	0.36	3.9	1.07

It can be seen that typically, students who were exposed to the guidelines rate their partner's contribution to be quite high, with low variance.

A Mann-Whitney U test was run to determine if there were differences in Perceived Partner Contribution between the exposed and unexposed groups. There was a statistically significant difference in perceived partner contribution scores between exposed students ($Mdn = 5.0$) and unexposed students ($Mdn = 4.0$), $U = 146$, $z = 2.587$, $p = 0.027$.

In this case, $p < 0.05$, therefore the null hypothesis (*the distribution of the pair's perceived partner contribution is equal across the two groups*) was rejected.

3.3 Driver-Navigator Role Preference

As part of the post-test surveys for this study, students were asked to indicate which role they had experienced for the duration of the session of the study.

Results were as follows:

- 9 students indicated that they were drivers;
- 11 students indicated they were navigators;
- 8 students ticked both boxes, indicating that they experienced both roles during the session.

This data shows that more students indicated they performed as the navigator over the driver role, with an approximate ratio of 45:55.

The data was then explored on a 'pair-by-pair' basis, giving the following results:

- 9 pairs consisted of a driver and a navigator;
- 2 pairs consisted of a navigator and an individual who indicated they had experienced both roles;
- 3 pairs consisted of both members within the pair indicating they experienced both roles.

The first and last responses are consistent with the typical role relationships in pair programming, and with what students are taught: a pair consists of a driver and a navigator, and these roles should be switched often (although switching often in a short 45-minute coding session is not highly likely).

The second statement does not fit this pattern, showing that whilst one member of the pair was a permanent navigator, the second member of the pair found it necessary to switch between the two roles. A review of the audio files was performed. It revealed that in both cases, the driver would sometimes stop typing, and brainstorm possible solutions and next steps with the navigator. Following this, he or she would go back to driving the session. It is possible that during these brainstorming sessions, the driver felt that he or she was also navigating, and thus felt they had experienced both roles during the session. It is unclear as to why the driver felt the need to switch back-and-forth between the roles, or why their navigator did not take over the driver role, but this hints at possible pair programming dynamics that may exist outside of the traditional 'driver-navigator' claim.

3.4 Discussion

The data gathered from this study supports the following hypotheses:

1. The distribution of the pair's ease of communication scores differs with exposure to the guidelines; i.e. pairs who were exposed to the guidelines reported significantly higher scores for ease of communication than the control group.

2. The distribution of the pair's perceived partner contribution scores differs with exposure to the guidelines; i.e. pairs who were exposed to the guidelines reported significantly higher scores for perceived partner contribution than the control group.

3. The mean number of completed tasks for pairs who were exposed to the guidelines and pairs who were not exposed is equal in the population; i.e. there was no significant difference in the number of completed programs between pairs who were exposed to the guidelines, and the control group.

These results show that the guidelines may help improve students' experience of communication within their pair. It is posited that this stronger 'partner contribution' was due to the fact that individual members of the pair were more confident

communicating their ideas (possibly due to the additional advice provided by the guidelines) and contributed more successfully as a result.

Furthermore, the use of the guidelines may support students in dealing with issues and barriers that typically arise during pair programming sessions in a structured way. However, whilst these guidelines can be seen to aid the pairs' perceived communication, there is no evidence to suggest that the guidelines have any impact on student success, at least in such a short programming session.

3.5 Limitations and Further Work

These findings are limited by the subject sample (from a single institution), and a relatively small sample group. A sample size of 28 participants gives a margin of error of 18.51% (CI: 95%).

The margin of error could be reduced by running this study with more participants (e.g. with 50 participants, the margin of error drops to 13.84%). Increasing the sample size could give evidence to further support these conclusions, and allow these results to be further generalised beyond the scope of this study.

Finally, data from the post-test surveys on the distribution of driver-navigator roles reported in section 3.3 shows that on some occasions, pairs did not work in pairs consisting of one driver and one navigator, and hints that these roles may be more fluid based on the situation currently being tackled. Similar work can be seen in [5]. A study considering possible pair dynamics outside the traditional driver-navigator roles would allow for further understanding of these pair dynamics, and how certain combinations may impact successful collaboration.

4. CONCLUSION

Previous research indicates that the pair programming guidelines help novice pairs communicate more effectively whilst working on debugging tasks. The research presented in this paper shows that the guidelines contribute to greater communication effectiveness when students are creating new code; significant differences were identified between the students who had been exposed to the guidelines and the control group when considering the individual members' perceptions of (i) their experienced communication and (ii) their partner's contribution to the session.

5. ACKNOWLEDGMENTS

The authors would like to thank all the student pairs who have contributed time, effort and invaluable feedback towards evaluating these guidelines.

6. REFERENCES

[1] Begel, A. and N. Nagappan. *Pair programming: what's in it for me?* in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. 2008: ACM.

[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] McDowell, C., B. Hanks, and L. Werner, *Experimenting with pair programming in the classroom.* SIGCSE Bull., 2003. **35**(3): p. 60-64.

[4] Nagappan, N., et al. *Improving the CS1 experience with pair programming*. in *ACM SIGCSE Bulletin*. 2003: ACM.

[5] Plonka, L., et al., *Collaboration in pair programming: driving and switching*, in *Agile Processes in Software Engineering and Extreme Programming*. 2011, Springer. p. 43-59.

[6] Ryu, E. and A. Agresti, *Modeling and inference for an ordinal effect size measure.* Statistics in Medicine, 2008. **27**(10): p. 1703-1717.

[7] 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.

[8] Sharp, H. and H. Robinson, *Three 'C's of agile practice: collaboration, co-ordination and communication*, in *Agile Software Development*. 2010, Springer. p. 61-85.

[9] 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.

[10] 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.

[11] Williams, L. and 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.

[12] Williams, L. and R. Kessler, *Pair Programming Illuminated*. 2002: Addison-Wesley Longman Publishing Co., Inc. 288.

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

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

[15] Zarb, M., *Developing a coding scheme for the analysis of expert pair programming sessions*, in *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. 2012, ACM: Tucson, Arizona, USA. p. 237-238.

[16] Zarb, M., J. Hughes, and J. Richards, *Evaluating Industry-Inspired Pair Programming Communication Guidelines with Undergraduate Students* in *Proceedings of the 45th ACM technical symposium on Computer Science Education*. 2014, ACM: Atlanta, GA, USA.

[17] Zarb, M., J. Hughes, and J. Richards, *Industry-inspired guidelines improve students' pair programming communication*, in *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. 2013, ACM: Canterbury, England, UK. p. 135-140.

Restarting Guidelines	If you and your partner are stuck in a 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 think about your end goal when suggesting next steps in order to make progress.	If your partner is attempting to break focus, do not dismiss this. Breaking one's focus using jokes and private conversations can lead to a fresh perspective, which you and your partner may need.	If you are in disagreement with your partner, you may find it helpful to break for lunch/coffee/etc. – during which you should physically walk away from your desk.	Give your partner space to read the code before suggesting next steps.
Planning Guidelines	Suggestions and reviews are both useful states that will allow you to drive your work forward. When in these states, feel free to communicate about a range of things; a potential cycle could be as follows: - 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 that they make, or actions they are working on that you do not necessarily understand.	Think about what your partner is saying and doing. Offering an interpretation of your own understanding of the current state can help move the work forward.	Learning to say I *don't know* or *I don't understand* is critical. Always explain things immediately – try to avoid replying to a question with *you'll see in a while*, as this will distract your partner.	Make a note of previously discussed suggestions and reviews so that similar discussions are not unnecessarily repeated over and over.
Action Guidelines	(*for the driver):* Whilst you are programming or thinking about how to structure your code, try to be more verbal – for example, by muttering whilst you are typing. This tends to help the navigator to know that you are actively working, and have a clear sense of how you are approaching the task at hand. If you verbalise your thoughts, this will help the navigator make informed suggestions based on your current actions.	*(for the navigator):* Whilst the driver is programming, actively look to make suggestions that contribute to the code.	*(for the navigator):* If the driver is muttering, use this opportunity to make sure your suggestions have been properly understood.	*(for the driver):* When silent, it can look as if you are clicking randomly on the screen, which risks your navigator becoming bored and distracted. Voicing your thoughts can help counter this.	*(for the navigator):* Think ahead, since you'll be driving in a short while: what is the current course of action not covering? Is there anything worth verifying that might have been left out?