Computer Science Education

ISSN: 0899-3408 (Print) 1744-5175 (Online) Journal homepage: http://www.tandfonline.com/loi/ncse20

Breaking the communication barrier: guidelines to aid communication within pair programming

Mark Zarb & Janet Hughes

To cite this article: Mark Zarb & Janet Hughes (2015) Breaking the communication barrier: guidelines to aid communication within pair programming, Computer Science Education, 25:2, 120-151, DOI: 10.1080/08993408.2015.1033125

To link to this article: http://dx.doi.org/10.1080/08993408.2015.1033125

Computer Science Education, 2015
Vol. 25, No. 2, 120–151, http://dx.doi.org/10.1080/08993408.2015.1033125

Breaking the communication barrier: guidelines to aid communication within pair programming

Mark Zarb[a]* and Janet Hughes[b]

[a]*School of Computing Science and Digital Media, Robert Gordon University, AB10 7GJ, Aberdeen, Scotland, UK;* [b]*School of Computing, University of Dundee, DD1 4HN, Dundee, Scotland, UK*

(*Received 15 November 2014; accepted 16 January 2015*)

Pair programming is a software development technique with many cited benefits in learning and teaching. However, it is reported that novice programmers find several barriers to pairing up, typically due to the added communication that is required of this approach. This paper will present a literature review discussing the issue of communication, and through a series of observations with industry-based pairs, will derive a set of guidelines which aim to help novice pairs experience better communication within their pairs. An evaluation of the guidelines with undergraduate students is then reported, showing that exposure to these guidelines improved the self-perceived communication experience of novice pairs.

Keywords: pair programming; communication; guidelines; evaluation

1. Introduction

Pair programming is a software development technique where two programmers work together side-by-side on the same machine to achieve their goals. This technique gained popularity in the early 2000s when it was presented as a key practice of the Extreme Programming software development methodology (Beck, 2000). An examination of the literature dates its use back to the early 1980s (Constantine, 1995), with empirical studies discussing the benefits of having two programmers dating back to 1993 (Wilson, Hoskin, & Nosek, 1993).

Many benefits of pair programming are reported for both novices and experts, including the pair experiencing a greater enjoyment of the work at hand, an increased knowledge distribution, and the production of better quality code (Cockburn & Williams, 2001). Further benefits are discussed when considering pair programming specifically in an educational context:

*Corresponding author. Email: m.zarb@rgu.ac.uk

students are more engaged in their collaboration and seem more satisfied with their final work (Williams & Kessler, 2001).

In spite of these benefits, some developers are sceptical of their first pair experience and of its promised collaborative value (Williams & Kessler, 2000), citing doubts about their partner's work habits and the added communication demands that this style of programming requires. *Communication* itself is frequently cited as a common barrier to pair programming by novices (e.g. Begel & Nagappan, 2008; Cockburn & Williams, 2001). However, if a pair does not communicate, they are not pair programming, but effectively they are only reviewing each other's code. Pairs cannot program without exhibiting a certain amount of communication: "Effective pairs chatter; silence is a danger signal" (Williams & Kessler, 2002). Within the literature, it can be seen that communication is not only an integral contributor to the success of pair programming, but also one of the main causes of its failure (Begel & Nagappan, 2008; Murphy, Fitzgerald, Hanks, & McCauley, 2010; Sanders, 2002).

This paper presents research which investigates common communication patterns and trends displayed by expert pairs of programmers. This allows for an understanding of how intra-pair communication is structured. This knowledge is then cast into guidelines and examples which could be used to assist novice pair programmers in learning to communicate more effectively when working together.

2. Background research

2.1. Pair programming: an introduction

Pair programming is widely used in academia (Katira, Williams, & Osborne, 2005), where it is typically introduced in tertiary education. This approach to programming is shown to encourage programmers to talk to each other and to themselves – this "pair pressure" adds benefits such as greater enjoyment and increased knowledge distribution (Bryant, Romero, & du Boulay, 2006; Williams & Kessler, 2001).

Within the classroom, pair programming is seen to be valuable (Begel & Nagappan, 2008; Hanks, 2006; Williams & Kessler, 2002). Its usage has been reported in educational contexts in the United States, the United Kingdom, Germany, New Zealand, India and Thailand (Hanks, Fitzgerald, McCauley, Murphy, & Zander, 2011). Students working in pairs are seen to be more satisfied, solve problems faster than non-paired students, and have improved team effectiveness (McDowell, Hanks, & Werner, 2003; Srikanth, Williams, Wiebe, Miller, & Balik, 2004; Williams, Kessler, Cunningham, & Jeffries, 2000). Pair programming among students is not a deterrent to individual student performance (Johnson & Caristi, 2001): pairing students were shown to be more likely to complete courses related to computer science and achieve a successful grade for their assignments when compared with

their solo counterparts, as well as gaining an improved comprehension of unfamiliar topics (Braught, Eby, & Wahls, 2008; Nagappan et al., 2003; Williams, Wiebe, Yang, Ferzli, & Miller, 2002). Similarly, Porter, Guzdial, McDowell, and Simon (2013) reported that paired students had a higher pass rate than their solo peers and were more likely to continue on the next course. Students who were exposed to pair programming in the classroom reported that having a partner with whom to discuss unfamiliar topics was helpful (Cliburn, 2003) and that this improved their comprehension of unfamiliar topics (Kavitha & Ahmed, 2015). Interestingly, Hanks (2006) shows that paired students experienced the same problems and struggles encountered by solo students, despite the benefits afforded by a pairing approach.

Initial observations with student programmers learning to work in pairs reveal several benefits to this approach (Werner, Hanks, & McDowell, 2004; Williams & Kessler, 2001). Students working in pairs answer each other's questions, rather than considering their instructor as the only source of advice – which contributes to the students' learning process. The use of pair programming, and the subsequent "pair pressure", causes students to work on projects earlier and to budget their time more wisely. Pair programmers took less time to complete set tasks (DeClue, 2003), and programs produced by paired students were seen to be significantly better than programs produced by individual student programmers. Students surveyed by Sanders (2002) following an initial experience of pair programming reported on experiencing a skewed perception of time, in which they felt they worked for less time than they actually did.

Williams et al. (2002) show that student pairs displayed a higher confidence and a more positive attitude in their project work when compared to solo student developers. Furthermore, pair programming has been proven to be useful in a learning environment for solving problems and complex tasks, and finding mistakes in simple code segments (Hulkko & Abrahamsson, 2005; Williams & Kessler, 2002). Programming students agree that they have more confidence in their final solution when it is achieved through pair programming (Williams & Kessler, 2000), and perceived pair programming as being valuable to their learning (VanDeGrift, 2004). The process of pair programming leads to students who are more satisfied with their work regardless of their ability and grade level (Kavitha & Ahmed, 2015; Vanhanen & Lassenius, 2005), and who are more self-sufficient. The student perception of pair programming on various tasks was examined by Chaparro, Yuksel, Romero, and Bryant (2005), who reported that when considering program comprehension, refactoring and debugging, students were effective across all three.

There are several reports which cite issues the students have with this approach. For example, Thomas, Ratcliffe, and Robertson (2003) indicate that their less successful paired students mentioned being frustrated, guilty and feeling like they had wasted their time. This pair incompatibility can be

a great cause of concern for students, with studies showing that amongst groups of students, those with a higher skill level report the least satisfaction when paired with students who are less skilled (Thomas et al., 2003).

Melnik and Maurer (2002) discuss various issues observed in the classroom, noting that some students found the pairing element particularly difficult, and "could not trust other people's code". Furthermore, approximately 50% of first-time student pair programmers reported that the various forms of difficulties within the pair contributed to *communication* being the main problem with the pair programming process (Sanders, 2002).

2.2. *Communication within pair programming*

Pair programming is a highly communication-intensive process, consisting of both verbal and non-verbal forms of communication (Sharp & Robinson, 2010). Williams and Kessler (2002) write that effective communication within a pair is paramount and that lengthy periods of silence within the pair should be considered a danger signal. Furthermore, several studies, both in industry and in academia, have concluded that apparent successes of pair programming are due to the amount of verbalisation that this style of coding requires (Chong & Hurlbutt, 2007; Freudenberg, Romero, & Du Boulay, 2007; Hannay, Dybå, Arisholm, & Sjøberg, 2009).

An experiment conducted by Bryant et al. (2006) shows that in expert pairs, the communication distribution between the driver and the navigator is 60:40, respectively. After analysing 23 h of dialogue produced by pair programmers, the researchers conclude that "the benefits attributed to pair programming may well be due to the collaborative manner in which tasks are performed" (Bryant et al., 2006).

Watzlawick, Bavelas, and Jackson (1967) consider that within professional relationships there is a necessity for constant communication. This necessity can be seen within pair programming: Flor and Hutchins (1991) observe that the exchange of ideas, feedback and constant debate – thus, communication – between two programmers collaborating on a software maintenance task significantly reduced the probability of ending up with a poor design. Wilson et al. (1993) show that in an academic context, collaborative work benefits problem-solving efforts: teams that were allowed to communicate whilst working on a software development task were seen to have a higher confidence in their solution.

Industrial developers surveyed by Begel and Nagappan (2008) define a prospective partner with good communication as one who embodies the following qualities:

- A good listener;
- Articulate;
- Easy to discuss code with;

- Very verbal, to make the thought process easy to understand;
- Enjoys debating and discussing code;
- Asks questions and provides opinions.

Communication is considered to be a "vital aspect of pair programming" (Lindvall et al., 2002), whilst Beck (2000) writes that it is "the first value of pair programming", and that coding standards should emphasise communication. Aiken (2004) reports that when pair programming, "no more than a minute should pass without verbal communication". The area of communication within pair programming is seen as an important topic of research interest (Stapel, Knauss, Schneider, & Becker, 2010), and it is also considered "one of the most important factors" within software engineering (Gallis, Arisholm, & Dyba, 2003). Furthermore, surveyed developers at Microsoft have rated "good communication skills" as being a top attribute for good pair programming partners (Begel & Nagappan, 2008), and it is seen as an "integral" concept for agile methodologies as it helps people to work better when partnered (Cockburn & Williams, 2001; Nawrocki & Wojciechowski, 2001).

Whilst existing research shows that communication is intrinsic to the pair programming process, different authors are using different measures when investigating the value of communication in pair programming. Two studies serve to illustrate this point, each of which considered levels of communication within pairs. Firstly, Sfetsos, Stamelos, Angelis, and Deligiannis (2006) found that for a group of pair programming students, there was a significant positive correlation between the number of communication transactions within the pair and the pair's productivity. Secondly, Choi, Deek, and Im (2009) found no correlation between a high level of communication and (i) satisfaction, (ii) compatibility between partners or (iii) confidence regarding the finished product. Communication thus is being investigated in terms of the end product and the process, particularly the experience of the social interaction.

Freudenberg et al. (2007) write: "the cognitive aspects of pair programming are seldom investigated and little understood". In one study examining communication per se, Stapel et al. (2010) hypothesise that there could be a difference in the rate of communication between novice pair programmers (defined as "new to pair programming and unfamiliar with each other") and professional ones (not explicitly defined in Stapel et al.'s paper, but used to indicate "experienced" pairs from discussed studies, i.e. industry-based pairs). The authors believe that this is due to the fact that a more experienced and confident pair will probably be more at ease with communicating and sharing ideas, whereas a more novice pair may be concerned about repercussions to sharing the wrong idea. The communication (or lack thereof) within a pair might determine the success of a pair programming exercise: if the pair does not communicate, then the programmers are only reviewing each other's code (Gallis et al., 2003).

Despite the centrality of communication referenced in the literature above, practical issues with pair programming communication remain. As an example, the communicative collaboration required by pair programming can cause discomfort for both the driver and the navigator (Cockburn & Williams, 2001), leading to reduced communication effectiveness and lower productivity (Aiken, 2004). Whilst communication is an important and essential aspect of pair programming, it can also be an issue and a barrier for first-time pair programmers in both industrial (Begel & Nagappan, 2008; Williams et al., 2000) and academic (Sanders, 2002) contexts. The literature posits interesting questions about communication, but ultimately, it can be seen that many authors simply view communication as the essence of paired programming, and as a result, do not investigate how communication happens within pairs and how it is or is not effective (Sharp & Robinson, 2010; Stapel et al., 2010).

2.3. *Existing pair programming guidelines*

Bevan, Werner and McDowell (2002) observe that the structure of the class can fail to encourage a consistent pair programming environment. They therefore present a number of guidelines to be used as a framework by instructors interested in adopting pair programming. Some of these guidelines belong under headings such as: *pair within sections, pair by skill level, institute a coding standard* and *create a pairing-oriented culture*. Similarly, Williams, McCrickard, Layman, and Hussein (2008) draw on over seven years of teaching experience in order to establish eleven guidelines for classroom management when pair programming is being used. These guidelines, like Bevan et al.'s, are aimed towards instructors, providing additional support on the points such as the following: *supervised pairing experience*, *teaching staff pair management*, *balancing individual and collaborative work*, and *pair programming ergonomics*.

With regard to student-based guidelines, several authors make reference to giving students a paper by Williams and Kessler (2000) as "guidelines to introduce the pair programming concepts" (McDowell et al., 2003; Mendes, Al-Fakhri, & Luxton-Reilly, 2005; VanDeGrift, 2004). Williams and Kessler (2000) describe the basics of pair programming under headings such as *share everything*, *play fair*, *don't hit your partner*, *put things back where they belong*, etc. In a separate paper, Williams et al. (2000) refer to these headings as "guidelines for transitioning from solo to pair programming". More recently, a number of amateur and some professional produced videos have been released that give an introduction to and some advice about pair programming (e.g. Williams (2008), or a video produced by Code.org.[1]) Williams' video (2008) presents a valuable set of things to do (including talk, listen, rotate roles, be patient, respect your partner, take breaks, prepare) and things *not* to do (including "be quiet").

The resources cited present sets of guidelines targeted towards solo programmers or instructors, but none mention the process used to create the guidelines, with each paper drawing on the author's observations and experiences to inform and create the guidelines.

To the best of our knowledge, there are no scientifically derived guidelines aimed for novice programmers; in particular, students instructing them on how to cope with various scenarios that may be encountered during a typical pairing session. It is posited that such guidelines may dispel some of the discomfort and scepticism faced by novice programmers needing to try this approach.

3. Observing pair communication

A better understanding of communication within pair programming could lead to improved teaching practices for pair programming novices, which in turn would allow them to communicate more effectively within their pairs. Observing experienced pair programmers working together could provide information to help understand how they communicate with each other.

A framework for the coding and analysis of verbal data has been described by Chi (1997) and was subsequently used in the context of pair programming by Bryant (2004) for the observation of industry-based pairs. This process consists of the following stages:

1. Developing a coding scheme through open coding;
2. Segmenting and coding the sampled transcripts based on the coding scheme;
3. Seeking and interpreting patterns.

This framework can be applied to a grounded theory approach in order to create stages upon which the observation stages of this paper can be built the following: communication data can be collected (via ethnographic observations, videos, etc.) and then used for data analysis purposes. By being immersed in the data, the researcher can develop codes and categories, which can continue iteratively until no new categories or properties emerge from the gathering or analysis of further data (Glaser & Strauss, 1967; Montgomery & Bailey, 2007). The analysis of these codes can then lead to patterns of pair communication being drawn out from the data.

A series of observations were carried out in three separate settings in order to gain an understanding of the various observed "states" of communication, and how the pair transitioned between these different states.

The three settings are as follows:

- *pairwith.us*

  A series of videos has been created by two software engineers with the aim of introducing agile software development to a wider audience.

Both members of the pair are agile coaches and programmers with over ten years of industry experience at the time of filming. The video output consists of sixty unscripted pair programming videos, all broadcast online between April and July 2009. Following each broadcasting session, the videos were archived without any post-processing or editing. A repository of the videos is made publically available under the name "pairwith.us" (at http://vimeo.com/channels/pairwithus).

- Company 1 (C1)

  C1 is a company which focuses on delivering high quality broadband and telephony around the UK. The team at C1 use agile methodology constantly, implementing practices such as scrum and Extreme Programming. A total of six pairs (all male) were observed at C1, with each session lasting roughly one hour. Individually, the developers ($n = 12$) reported industrial pair programming experience of 4.92 ± 2.30 years.

- Company 2 (C2)

  C2 is one of the leading global technology platforms for social video distribution and analytics. Several teams within C2 use agile practices to continuously test and develop their technology. A total of five pairs (four of which were both male, and one of which was mixed) were observed at C2, with each session lasting roughly one hour. Individually, the developers ($n = 10$) reported industrial pair programming experience of 2.02 ± 1.79 years.

By conducting observation sessions (for C1 and C2) and analysing existing videos (for pairwith.us), the researcher was able to collect over forty hours of pair programming communication. This databank was thoroughly analysed using an approach informed by grounded theory, in order to:

1. Find common communication states across all three scenarios.
2. Understand how the observed pairs transitioned between the different communication states.

A detailed description of how this analysis was performed is given elsewhere (Zarb, Hughes, & Richards, 2012). The analysis showed that the observed pairs were frequently experiencing one of seven possible communication states:

- Review
- Suggestion
- Explanation
- Code discussion

- Muttering
- Unfocusing
- Silence

Further analysis derived a set of common transitions which the pairs were observed to experience (Zarb, Hughes, & Richards, 2013); for example, pairs were frequently observed to make a suggestion, then follow up by explaining it further. Figure 1 gives a summarised version of these results, showcasing the most common transitions between the communication states.

A *guideline* is, by definition, a general rule or a piece of advice, synonymous with a recommendation or a suggestion: an indication of a future course of action. The following section discusses certain conversational patterns within the larger context of the transitions diagram (Figure 1) and extracts guidelines from these patterns.

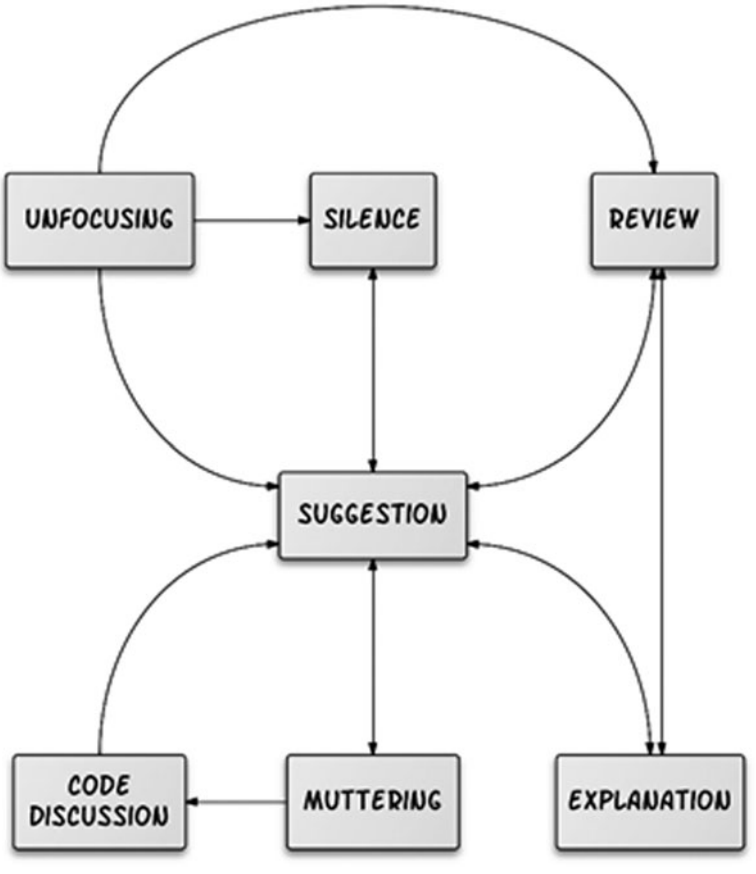

Figure 1. Common states and transitions in pair communication.

3.1. Extracting patterns and generating guidelines

In order to better understand the transitions depicted above, Figure 1 was segmented into several subsections by consulting the frequency with which the transitions happened. Each subsection was verified with the *pairwith.us* team as the depiction of a different stage of the communication process within pair programming. Each subsection is referred to as a "pattern", representing the different communication states a pair can experience, and the various ways of transitioning between these states. Each extracted pattern was used to form the basis for a set of guidelines.

Whereas each code represents a different communication state for a pair, each pattern represents different stages of the pairing process. Patterns can illustrate a whole set of communication states describing, for example, a reviewing cycle, or actions leading to the pair deciding to take a break from their current task.

Figure 1 was segmented into three patterns: one that looks at all possible outcomes from an *Unfocusing* state; and two which consider certain repeated behaviours. The identified patterns are as follows:

1. A pattern linking *Unfocusing*, *Review*, *Silence* and *Suggestion* on the top half of the diagram, explaining actions that follow an *Unfocusing* event. This is called the Restarting Pattern;
2. A pattern linking *Review*, *Explanation* and *Suggestion* on the right-hand side of the diagram, showing repeated communication behaviour. This is called the Planning Pattern; and
3. A final pattern linking *Muttering*, *Code Discussion* and *Suggestion* on the bottom left of the diagram, showing repeated communication behaviour. This is called the Action Pattern.

4. The guidelines

Instances of each pattern were observed in the pair videos and reviewed in order to explore why and how certain patterns were being exhibited. At this stage, observed pairs were consulted about the existence of these patterns. They confirmed that these were behaviours that they recognised within their own pair communication. Discussions with a member of teaching staff (experienced with teaching agile) within the University of Dundee were used to identify guidelines and to structure these in a suitable way for educational purposes.

The pair programming communication guidelines were therefore created to give users more insight into the instructions offered by these patterns. The aim of the research is to investigate whether providing novice pairs with communication patterns from expert pairs will allow them to improve their intra-pair communication. By extracting these communication patterns from the observation sessions, it was possible to present the knowledge

uncovered thus far in a manner that would best benefit novice student pairs. The three patterns are presented next.

4.1. The restarting pattern and guidelines

At several points during the observations, pairs were observed to completely change the topic of discussion from their current work to a more casual topic. For instance, during the *pairwith.us* videos, a member of the pair suddenly interrupts the coding process and starts talking about his Father's Day plans. Similarly, in a separate observation, the pair starts to discuss a recently released film that they had both watched.

Informal discussions with some of the observed industry pairs indicate that these interruptions are usually conscious ones: whenever a pair was stuck for a period of time, they would make an effort to break their focus by stopping their current actions and move onto an unrelated topic of discussion.

This is described here as the Restarting Pattern (Figure 2).

Figure 2 shows that *Unfocusing* is most commonly followed by one of three communication states: reviewing, suggesting and silence. An example of each is given next:

- A reviewing action. The following conversational snippet shows a pair unfocusing (by making jokes about the driver's age), then transitioning into a reviewing state:

D: I've had to turn the font size up. I'm blind.
N: No, you're getting old!
D: I should be wearing glasses, I'm just being stubborn. We had just finished with the casting agent; it was being stubborn.
N: The test is still very much testing the details of the librarian.

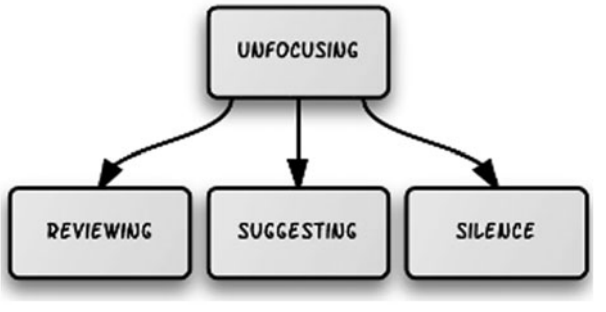

Figure 2. The restarting pattern.

- A suggestion. This conversational snippet shows the pair making jokes, with the navigator choosing to bring back focus by making a suggestion for the next stage in their work plan.

> *D: So what you're saying is "Terror Wrist".*
> *N: Yeah, explain the joke. That makes it so much funnier.*
> *D: All my jokes are bad. (laughs)*
> *N: Look at that. You probably want to implement "help actors get out of character".*
> *D: Good idea.*

- Complete silence. The following conversation shows a pair suddenly unfocusing when the navigator interrupts the coding process. Both programmers have a brief discussion and then engage in a silent period. This typically ends after the navigator makes a suggestion related to the code.

> *N: "Don't chop the dinosaur, daddy!"*
> *D: (laughs) Seek help. What's that from?*
> *N: It's from an Australian advert.*
> *D: Right. OK. You keep saying that.*
> *(A period of silence follows.)*

Three guidelines suggested by this pattern (Figure 2) are as follows:

- 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 (review);
  - Identify a fresh start (suggest);
  - Try to think about your end goal when suggesting next steps, in order to make progress (think/be silent).

- 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.

4.2. The planning pattern and guidelines

Following a *Suggestion*, a pair was sometimes likely to review the existing code to understand how refining it might help them achieve their main goal. As part of this conversation, one of the pair would typically explain the underlying structure or any legacy code that might be unfamiliar to their partner. This is presented in Figure 3.

The following conversation illustrates the driver making a suggestion, the navigator reviewing current procedures and then proceeding on to explaining their reasoning.

D: It still feels like we're missing something. We're getting closer to the general solution, though. I'll stick closer to what I have on screen.
N: We have c*, then* b*... and* a *to* b*... and* b *to* c*... to* d*.*
D: Yep.
N: At the moment we're eagerly calling a *to lock* a *off. If I don't do this, obviously it's going to carry over.*

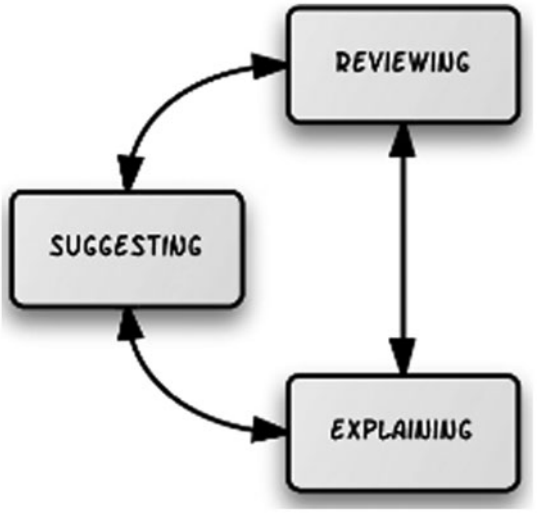

Figure 3. The planning pattern.

A suggestion could also separately lead to an explanation – for example, whilst discussing a method, rather than reviewing the code structure, the pair would explain implications that the method would have with respect to their goal. This concept, as well as that of a member asking for clarification by their partner, is also seen as a way to avoid the pair becoming disengaged (Plonka, Sharp, & van der Linden, 2012). The following shows a navigator making a suggestion and then further explaining how it would impact the written code.

> *N: Could you – double dash. It's one over…*
> *D: Yeah?*
> *N: It's actually a funny thing. If you whip out an agent test right now, it would generate itself. Because you told it to. Do you get it?*

This pattern (Figure 3) occurred most often at the start of the pairing session: the sessions observed typically started with the pair reviewing legacy code and attempting to devise ways to reduce error messages or solve problems.

Three guidelines suggested by this pattern are as follows:

- 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:
  - o Review previous code
  - o Suggest an improvement

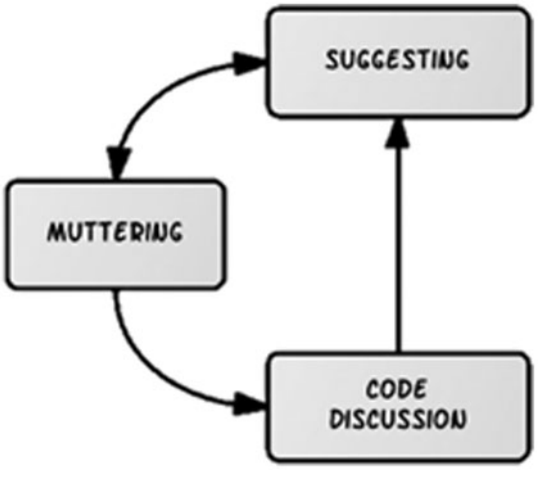

Figure 4. The action pattern.

  - 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.

4.3. The action pattern and guidelines

The Action Pattern (Figure 4) occurred mostly whilst a pair was trying to create code. These instances would typically consist of a member of the pair making a suggestion as to what should be coded, or how certain code should be tackled.

The pair would then either talk about the code, or, alternatively, the driver would start muttering. The muttering frequently led to the navigator making suggestions based on what the driver was saying, which acted as a prompt for discussions.

The following example shows the navigator suggesting next stages (in this case, to code a certain test). The driver starts muttering. After a while, the navigator interjects, discussing the benefits of the current approach.

> *N: Excellent. The method's completed. I guess it's time to go on and do the test now.*
> *D: (muttering whilst typing code and running commands)*
> *N: (reacting to the completed method, and the expected results of the test) I think it's a good example of the level of feedback and the cycle time.*

Writing code is generally handled by the driver, rather than both members of the pair, thus guidelines arising from this pattern are targeted towards individual members of the pair:

- *(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.

4.4. The communication guidelines

Figure 5 summarises the communication guidelines extracted from the patterns depicted in Figure 1.

Restarting	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 (review); - Identify a fresh start (suggest); - Try to think about your end goal when suggesting next steps, in order to make progress (think/be silent).	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.
Planning	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.
Action	*(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 programing, 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.

Figure 5. The communication guidelines.

5. Evaluation of the guidelines

Quantitative results would provide an understanding of the effects that guidelines have on novice students and on their pair programming experience. To that end, a series of studies were designed to understand whether the guidelines can positively impact the students' experience of communication.

In the earlier review of the literature, it was seen that "communication" is often seen as a barrier to successful pair programming for first-time pairs (Begel & Nagappan, 2008; Sanders, 2002; Williams et al., 2000). Furthermore, unequal participation is one of the top perceived problems for students (Srikanth et al., 2004; VanDeGrift, 2004). As the guidelines have been developed to improve this communication, the studies were designed to investigate these two issues, with pairs reporting on their experience of communication, particularly with respect to how easily they were able to communicate with their partner (referred to as "ease of communication") and on their partner's contribution to the pairing session ("perceived partner contribution").

5.1. Study design

Pairs of students were recruited and randomly allocated to one of two groups: a treatment group, which would be exposed to the guidelines prior to the set task, and a control group. Each pair was asked to complete as many tasks as possible from a given set during a 45-min time period. Student success was measured in terms of correct solutions.

This was followed by a five question post-test survey to be completed individually by each participant. Two questions queried the individual on their experience of development as a solo programmer and/or as a pair programmer. These questions were used to determine whether there was any significant difference between the groups that could bias the results. The remaining three questions asked the individual to rate their perception of the benefit of pair programming over traditional programming, the ease of communication during the session, and to rate their partner's contribution. Students were also asked to note which role they had assumed (i.e. driver or navigator) during the recorded session.

Likert scale data were analysed to determine whether there were any statistically significant differences between the treatment group students who were exposed to the guidelines and the control group students who were not, in terms of their prior programming experience and in respect of the reported ease of communication and partner contribution.

The following null hypotheses are tested as follows:

1. H_0: The distribution of the pair's ease of communication is equal across the two groups.
   H_A: The distribution of the pair's ease of communication differs by exposure to the guidelines.

2. H_0: The distribution of the pair's perceived partner contribution is equal across the two groups.
   H_A: The distribution of the pair's perceived partner contribution differs by exposure to the guidelines.
3. H_0: The mean number of completed tasks for pairs who were exposed to the guidelines and pairs who were not exposed to the guidelines is equal in the population.
   H_A: The mean number of completed tasks for pairs who were exposed to the guidelines and pairs who were not exposed to the guidelines is not equal in the population.

5.2. Method

5.2.1. Participants

An e-mail was circulated to undergraduate students reading for a Computing degree, inviting them to participate. A total of 28 participants were recruited (Level 1: 10 students; Level 3: 18 students), all of whom had previously used Java as a programming language throughout their courses. Ethical approval was obtained from the School's Ethics Board for all participants and all aspects of the studies.

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

5.2.2. 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).[2] The APE tool runs on the NetBeans IDE and provides a graphical front end (Figure 6) which can be manipulated using Java code. This allows students to "see" what they are programming. The APE tool includes several challenges (or "maps") in which students need to move a yellow 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 (Figure 6; note that the contrast has been adjusted to make the image more suitable for printing).

Each pair was provided with a laptop installed with the APE tool and ten randomly chosen APE maps, each pair being asked to solve the same sequence of maps. Pairs were provided with a list of basic instructions to move the character (Table 1).

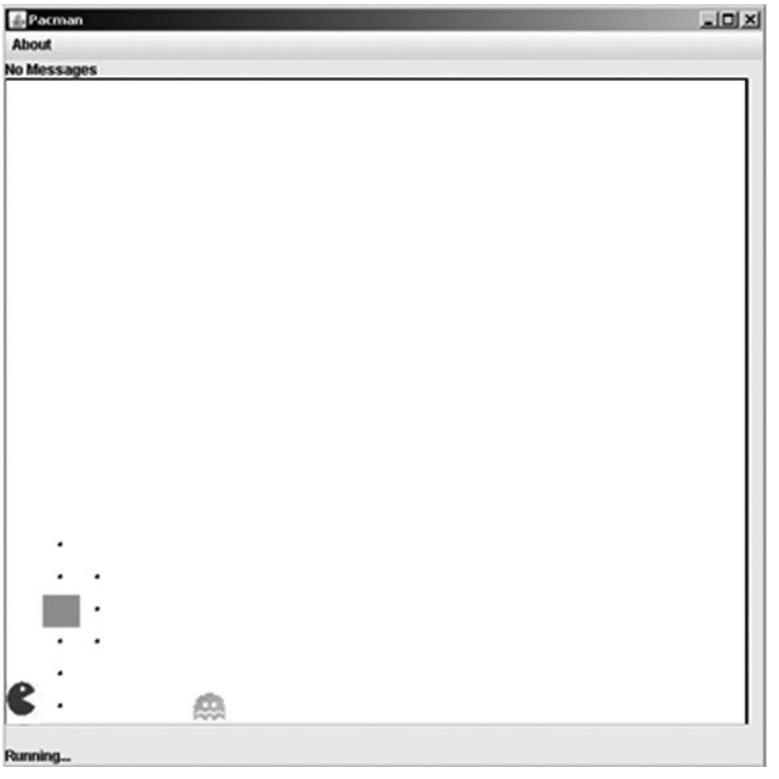

Figure 6. The APE graphical front end.

Table 1. 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 ° to the left.
main.turnRight();	Makes the character turn 90 ° to the right.

5.2.3. *Procedure*

The study was carried out during a four-week period. Pairs were separately invited to a test room with an installed camera and a voice recorder, to allow for data capture, and the laptop with the APE tasks to be solved.

Pairs were given a short welcome presentation and a description of the set-up and tasks to be solved. This introduction made clear that each pair was responsible for the entire programming process (from discussing possible solutions, to attempting to implement the correct code and testing it) and that pairs were free to implement solutions using any programming technique at their disposal (for example, pairs may use *for* loops and *do..while* loops to refactor

the code, or write a parser for a more straightforward manner of telling the character how to move across the map). All pairs were also told to pair program, and switch roles as and when they deemed this to be appropriate. Pairs in the treatment group were given, in addition, a prepared video[3] and the paper copy of the guidelines, and ten minutes to view and read these.

Each pair then was given 45 min to sequentially work their way through as many tasks as they could. The recording devices were switched on and the researcher left the room. Following the test period, the researcher returned, logged the number of programs attempted and distributed post-test surveys (Table 2) which were completed at that point by the individual members of the pair. Finally, all participants were asked to comment on their experience of pair programming. Furthermore, treatment group participants were asked for their impressions of the guidelines and whether or not they consciously made use of the guidelines during the test.

The pair's code was retrieved for later analysis to understand whether the guidelines had any significant impact on the pair's success rate.

5.3. *Results*

The post-test survey data 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 the data used were extracted from Likert scales (and therefore "ordinal"), the Mann–Whitney U-test was applied (Ryu & Agresti, 2008). Furthermore, as a non-parametric test, this is more robust against certain assumptions (e.g. outliers seen in the data) (McElduff, Cortina-Borja, Chan, & Wade, 2010).

5.3.1. *Programming experience*

A preliminary analysis was performed of the participants' reported experience of solo and pair programming to determine whether there were any differences between the two groups which may have affected the data.

Table 2. The list of questions in the post-test survey.

Question	Data gathering
1. I feel pair programming is more beneficial than solo programming.	5-point Likert scale
2. During this session, I found communicating with my partner to be easy.	5-point Likert scale
3. Rate your partner's contribution to today's session.	5-point Likert scale
4. How long have you been a (i) solo programmer; (ii) pair programmer?	Blank space
5. What role do you feel you predominantly assumed during today's session?	Blank space

Table 3. Student programming experience.

	Treatment group		Control group	
	Mean	SD	Mean	SD
Solo programming experience (years)	3.7	2.17	2.7	1.86
Pair programming experience (years)	.3	.59	.2	.41

The data show that the groups had somewhat different levels of experience (Table 3): on average, more individuals in the treatment group had solo programming experience.

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: $U = 125$, $z = 1.266$, $p = .227$ ($p > .05$).
- Similarly, no significant differences in pair programming experience were found between the experimental and control groups: $U = 106.5$, $z = .427$, $p = .670$ ($p > .05$).

The results show that there were no significant differences between the two groups and that further results should not be skewed by any bias resulting from one group having additional previous experience.

5.3.2. Perceived benefits of pair programming

In the post-test survey, students were asked to rate the statement "I feel pair programming is more beneficial than solo programming" on a 5-point Likert scale.

Figure 7 charts student responses for the two groups.

The treatment group ($M = 4.5$, SD = .52) and the control group ($M = 4.1$, SD = .62) report similar scores. There was no significant difference in perceived pair programming benefit between treatment group students ($Mdn = 4.0$) and control group students ($Mdn = 4.5$), $U = 133$, $z = 1.834$, $p = .067$.

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.

Shapiro–Wilk tests were carried out to understand whether the data being analysed were normally distributed. *Ease of communication* scores for both treatment and control groups were not normally distributed ($p < .05$).

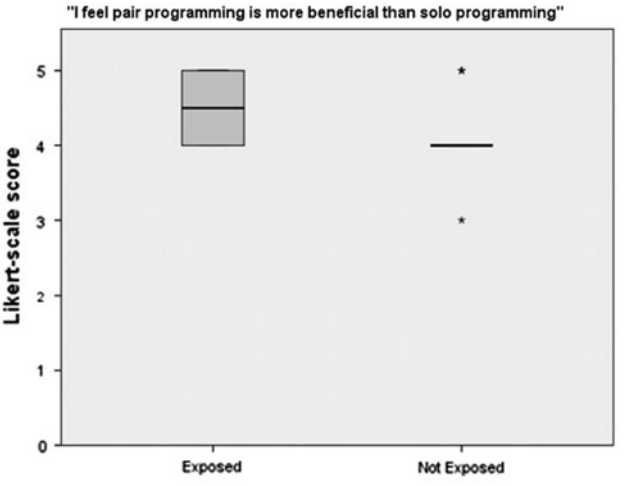

Figure 7. Reported scores for "I feel pair programming is more beneficial than solo programming".

Similarly, scores for *perceived partner contribution* for both groups were not normally distributed ($p < .05$). As the data are not normally distributed for both sets of scores, non-parametric tests were used.

5.3.3. *Ease of communication*

The post-test survey results relating to *ease of communication* were analysed, and descriptive statistics were used to gain an overview of detail (Table 4).

Figure 8 depicts the distribution of scores reported for *ease of communication* between the two groups, ranging from 1 ("strongly disagree") to 5 ("strongly agree"). The asterisk indicates outliers in the data.

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.

A Mann–Whitney U-test was run to determine whether there were differences in *ease of communication* between the treatment and control groups. There was a statistically significant difference in ease of communication scores between students to the guidelines ($Mdn = 5.0$) and those who were not ($Mdn = 4.0$), $U = 169$, $z = 3.721$, $p = .001$.

Table 4. Descriptive statistics for ease of communication (part 2).

	Treatment group		Control group	
	Mean	SD	Mean	SD
Ease of communication	4.9	.27	4.0	.78

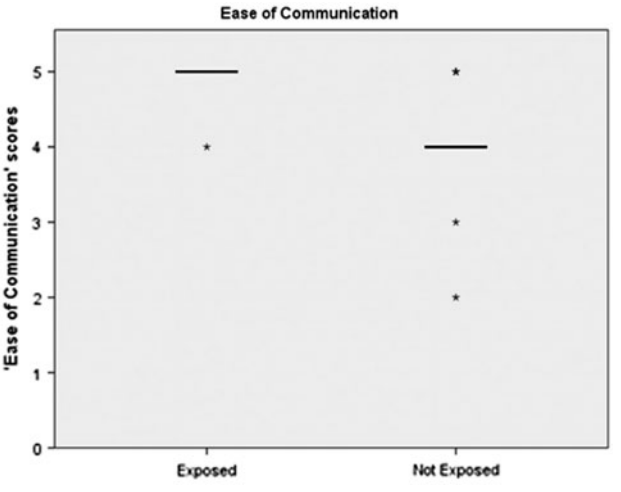

Figure 8. Reported scores for ease of communication.

In this case, $p < .05$, therefore, the null hypothesis (*the distribution of the pair's ease of communication is equal across the two groups*) was rejected.

5.3.4. *Perceived partner contribution*

As before, the post-test survey results relating to *perceived partner contribution* were analysed, and descriptive statistics were used to gain an overview of detail (Table 5).

Figure 9 shows the distribution of Likert scale scores for students' *perceived partner contribution* between the two groups, ranging from 1 ("no participation") to 5 ("excellent"). The asterisk indicates outliers in the data.

It can be seen that generally 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 whether there were differences in *perceived partner contribution* between the treatment and control groups. A statistically significant difference in perceived partner contribution scores was evident: treatment group students ($Mdn = 5.0$); control group students ($Mdn = 4.0$), $U = 146$, $z = 2.587$, $p = .027$.

Table 5. Descriptive statistics for perceived partner contribution (part 2).

	Treatment group		Control group	
	Mean	SD	Mean	SD
Perceived partner contribution	4.9	.36	3.9	1.07

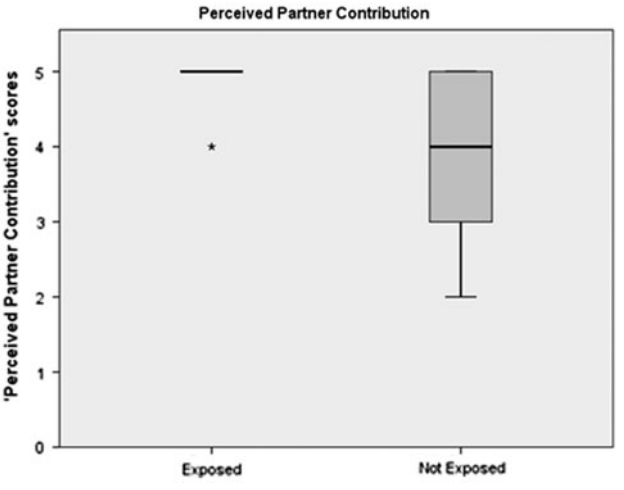

Figure 9. Reported scores for perceived partner contribution.

In this case, $p < .05$, therefore, the null hypothesis (the distribution of the pair's perceived partner contribution is equal across the two groups) was rejected.

5.3.5. *Successfully completed programs*

The number of tasks attempted was noted and scored. Each attempt was scored by the researcher and also compiled to see correct result was produced (i.e. if each map was solved successfully). The total number of successfully completed tasks was noted for each pair. It should be noted that none of the participants skipped a task, i.e. all tasks attempted by pairs were completed successfully (apart from tasks the pair had been working on where the time ran out).

An independent samples *t*-test was run to determine whether there were differences in completion scores between pairs who were exposed to the pair programming guidelines ($n = 7$), and those who were not ($n = 7$).

There were no outliers in the data, as assessed by inspection of a boxplot (Figure 10). The tasks completed for each level of exposure were normally distributed, as assessed by Shapiro–Wilks test ($p > .05$), and there was homogeneity of variances, as assessed by Levene's test for equality of variances ($p = .903$).

The treatment group pairs successfully completed slightly more tasks (4.0 ± 1.00) than the control group pairs ($3.3 \pm .76$). The difference is not statistically significant: $t(12) = -1.508$, $p = .158$.

This result shows that exposing pairs to the guidelines did not improve the number of tasks successfully completed.

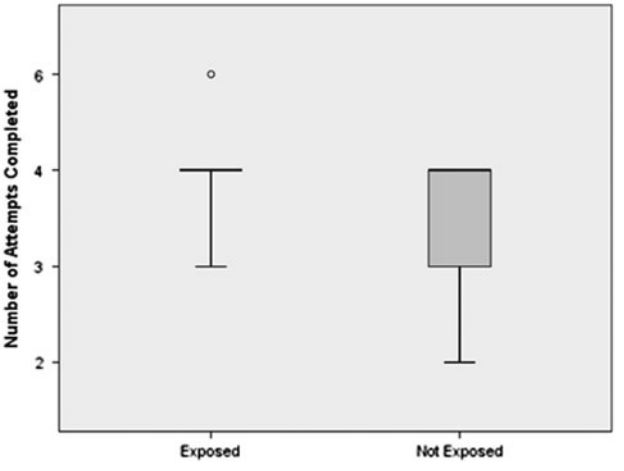

Figure 10. Number of tasks completed.

5.4. Feedback from participants

At the conclusion of the test, all participants were asked to comment on their experience of pair programming.

Asked whether their expectations of pair programming matched the actual experience, all students agreed that they could understand "where and why it [is] useful", admitting that they felt that the pair programming process was more natural than it had seemed at the start of the semester. None of the participants indicated that communication was an issue during this phase.

All treatment group participants indicated that they found the pair programming guidelines to be beneficial, as 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, and noticing the driver muttering, was useful.

> The [restarting guideline] would be the most useful one, whereas [planning and action] would come more naturally. They are definitely good if you don't know your partner well.

> They seem like really good tips if you get stuck; a lot is self-explanatory, which is good.

> I think we definitely used the restarting pattern. [You] definitely pick up on when people are getting frustrated, so we went out to the shop; getting away from the computer was helpful.

When the treatment group students were asked their opinion regarding introducing the guidelines as a taught component that complemented an

introduction to pair programming, there was not unanimity. Some argued that pair programming should be fully understood prior to the introduction of the guidelines: "it was good to get to grips with pair programming [by themselves], and learn from [their] mistakes before being taught [the guidelines]", and "it might have been too much information at the start". Conversely, a number of participants felt that the concepts could be introduced earlier:

> The concept is very straightforward: you are in a pair, and programming. Being given these guidelines would have shown the more advanced side at the start, I think.

Treatment group participants agreed that following the initial introduction, the guidelines were not something they needed to actively think about or study in order to implement, but that an awareness of the guidelines was often enough:

> We did a lot of it without thinking about it.
> We followed them because after a while, they occurred naturally.

These comments are encouraging, indicating that the guidelines were adopted quite naturally by the participants. Participants found them to be potentially useful in different situations and scenarios than those initially envisioned by the researcher. For example, one pair asked about using the planning guidelines in their own time to learn and understand how to write Android code from scratch.

6. Discussion

The data gathered from this study supports the following hypotheses:

1. The distribution of the pair's ease of communication scores differs by 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 by 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 and the accepted hypotheses are preliminary, but they show that the guidelines may help improve students' experienced communication within their pair. It is posited that this stronger "partner contribution" was due to the fact that individual members of the pair are more confident communicating their ideas (possibly due to the additional advice provided by the guidelines); in turn, to their partner, it seems as if they are making more contributions during the pairing session. 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.

Communication is often reported as one of the main barriers to pair programming for novice pairs (Begel & Nagappan, 2008; Sanders, 2002; Williams et al., 2000) by introducing the guidelines seen in this paper, it is seen that pairs experience better communication, which may lead to more successful adoption of pair programming.

6.1. Threats to validity

All participants who applied for the study were recruited. Following the recruitment, when students were randomised into pairs, the pairs were also alternated into the test or control group. Threats with regard to previous pairing experience were controlled through the pre-test survey, Section 5.3.1 of this paper reports this in further detail.

All participants had previous contact with the researcher as a laboratory tutor; to counteract this, the researcher followed set instructions and procedures with all pairs to ensure standardisation across all pairs. Follow-up studies would benefit from a larger number of potential participants, where these threats to validity can be kept to a minimum.

6.2. Limitations

These findings are limited by the subject sample being 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.

Furthermore, whilst participants were asked to note down which role they assumed during the study, there was insufficient data to perform a thorough analysis of how the roles were assumed.

7. Conclusions

This paper has investigated the issue of *communication* for inexperienced pair programmers. It has reported on a series of observations of industry expert pair programmers. This work identified communication states frequently experienced by the industry pairs, leading to an understanding of how expert pairs transitioned between various communication states. This knowledge was used to establish communication guidelines for novice pair programmers. Novice pairs reacted positively to the guidelines, indicating that the guidelines were beneficial and useful. Further evaluations indicate that exposure to the guidelines resulted in a positive impact on the students' intra-pair communication, and on their perception of their partner's contribution.

7.1. Future work

There are several aspects of this research that could be investigated further. Whilst this study reported on participants solving simple coding exercises, it would be valuable to run this study to test the guidelines against a wider range of programming abilities (for example, code analysis, debugging and testing). A follow-up study would be more focussed on studying participants' role preference and behaviour, whilst targeting a wider range of participants from multiple institutions.

7.2. Conclusions

At the time of writing, several educators have expressed an interest in adopting the guidelines for the teaching of pair programming within their institutes.

To conclude, this work presents initial evidence showing that it may be possible to improve communication levels between novice students who are pairing together by presenting them with industry-inspired guidelines. Novice pairs who had been exposed to the guidelines reported significant improvements in their perceived communication and partner contribution than students who had not.

The guidelines developed in this paper can be used to aid pairs that are sceptical or anxious about communicating with a new partner. Novice pairs can use these guidelines to explore different ways of dealing with issues that typically arise during pair programming.

This is captured in the following statement, made by a student participant during the evaluation stages:

> There's a definite benefit in introducing this. In pair programming, we're told to "work in pairs: go!", and there weren't formal steps, apart from the fundamentals. There wasn't a lot of what to do if you became stuck.

Acknowledgements

The authors wish to thank Professor John Richards for insightful discussions and support with this work. Thanks to the *pairwith.us* team, the pairs at C1 and C2, and to all the students at the University of Dundee's School of Computing who agreed to participate in the various studies.

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes

1. Code.org video on pair programming: https://www.youtube.com/watch?v=vgkahOzFH2Q
2. The APE tool was created by Heron and Belford (see http://monkeys.imaginary-realities.com) and used with permission.
3. A copy of this video is available at the following URL: https://www.youtube.com/watch?v=ONnYCT_LJio

References

Aiken, J. (2004). Technical and human perspectives on pair programming. *ACM SIGSOFT Software Engineering Notes, 29*(5), 1–14. doi:10.1145/1022494.1022512

Beck, K. (2000). *Extreme programming explained: Embrace change*. Reading, MA: Addison-Wesley Professional.

Begel, A., & Nagappan, N. (2008). Pair programming: What's in it for me? In *Proceedings of the 2nd ACM-IEEE international symposium on Empirical software engineering and measurement*, Madrid, Spain.

Bevan, J., Werner, L., & McDowell, C. (2002). Guidelines for the use of pair programming in a freshman programming class. In *Proceedings of the 15th conference on software engineering education and training*. Kentucky: IEEE.

Braught, G., Eby, L. M., & Wahls, T. (2008). The effects of pair-programming on individual programming skill. In *Proceedings of the 3rd technical symposium on computer science education*. Portland, OR: ACM.

Bryant, S. (2004). Double trouble: Mixing qualitative and quantitative methods in the study of extreme programmers. In *IEEE symposium on visual languages and human centric computing*. Rome: IEEE.

Bryant, S., Romero, P., & du Boulay, B. (2006). The collaborative nature of pair programming. In D. Hutchison, T. Kanade, & J. Kittler (Eds.), *Extreme programming and agile processes in software engineering*, (Vol. 4044, pp. 53–64). Oulu: Springer Berlin/Heidelberg.

Chaparro, E. A., Yuksel, A., Romero, P., & 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*. Brighton: University of Sussex.

Chi, M. T. (1997). Quantifying qualitative analyses of verbal data: A practical guide. *Journal of the Learning Sciences, 6*, 271–315.

Choi, K. S., Deek, F. P., & Im, I. (2009). Pair dynamics in team collaboration. *Computers in Human Behavior, 25*, 844–852. doi:10.1016/j.chb.2008.09.005

Chong, J., & Hurlbutt, T. (2007). The social dynamics of pair programming. In *Proceedings of the 29th international conference on software engineering*, Minneapolis, MN.

Cliburn, D. C. (2003). Experiences with pair programming at a small college. *Journal of Computing Sciences in Colleges, 19*, 20–29.

Cockburn, A., & Williams, L. (2001). The costs and benefits of pair programming. In G. Suxxi & M. Marchesi (Eds.), *Extreme programming examined* (pp. 223–243). Reading, MA: Addison-Wesley.

Constantine, L. L. (1995). *Constantine on peopleware*. Englewood Cliffs, NJ: Yourdon Press.
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*, 49–56.
Flor, N. V., & Hutchins, E. L. (1991). A Case Study of Team Programming During Perfective Software Maintenance. In *Proceedings of the fourth annual workshop on empirical studies of programmers*. Norwood, NJ: Ablex Publishing.
Freudenberg, S., Romero, P., & Du Boulay, B. (2007). *"Talking the talk": Is intermediate-level conversation the key to the pair programming success story?* Paper presented at the Agile Conference (AGILE), 2007. Washington, DC: IEEE.
Gallis, H., Arisholm, E., & Dyba, T. (2003). An initial framework for research on pair programming. In *International symposium on empirical software engineering*, Rome, Italy.
Glaser, B. G., & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. New York, NY: Aldine de Gruyter.
Hanks, B. (2006). Student attitudes toward pair programming. In *Proceedings of the 11th annual conference on innovation and technology in computer science education*. Bologna: ACM.
Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: A literature review. *Computer Science Education, 21*, 135–173.
Hannay, J. E., Dybå, T., Arisholm, E., & Sjøberg, D. I. (2009). The effectiveness of pair programming: A meta-analysis. *Information and Software Technology, 51*, 1110–1122.
Hulkko, H., & Abrahamsson, P. (2005). A multiple case study on the impact of pair programming on product quality. In *Proceedings of the 27th international conference on software engineering*. St. Louis, MO: ACM.
Johnson, D. H., & Caristi, J. (2001). Extreme programming the software design course. In M. Marchesi, G. Succi, D. Wells, & L. Williams (Eds.), *Extreme programming perspectives*. Reading, MA: Addison Wesley.
Katira, N., Williams, L., & Osborne, J. (2005). Towards increasing the compatibility of student pair programers. In *Proceedings of the 27th international conference on software engineering*, St. Louis, MO.
Kavitha, R., & Ahmed, M. I. (2015). Knowledge sharing through pair programming in learning environments: An empirical study. *Education and Information Technologies, 20*, 319–333. doi:10.1007/s10639-013-9285-5
Lindvall, M., Basili, V. R., Boehm, B. W., Costa, P., Dangle, K., Shull, F., … Zelkowitz, M. V. (2002). Empirical findings in agile methods. In *Proceedings of the second XP universe and first agile universe conference on extreme programming and agile methods*. London: Springer-Verlag.
McDowell, C., Hanks, B., & Werner, L. (2003). Experimenting with pair programming in the classroom. *ACM SIGCSE Bulletin, 35*, 60–64. doi:10.1145/961290.961531
McElduff, F., Cortina-Borja, M., Chan, S.-K., & Wade, A. (2010). When t-tests or Wilcoxon–Mann–Whitney tests won't do. *AJP: Advances in Physiology Education, 34*, 128–133.
Melnik, G., & Maurer, F. (2002). Perceptions of agile practices: A student survey. In *Proceedings of the second XP universe and first agile universe conference on extreme programming and agile methods*. London: Springer-Verlag.
Mendes, E., Al-Fakhri, L. B., & Luxton-Reilly, A. (2005). Investigating pair-programming in a 2nd-year software development and design computer science course. In *Proceedings of the 10th annual conference on innovation and technology in computer science education*. Monte de Caparica: ACM.
Montgomery, P., & Bailey, P. H. (2007). Field notes and theoretical memos in grounded theory. *Western Journal of Nursing Research, 29*, 65–79. doi:10.1177/0193945906292557
Murphy, L., Fitzgerald, S., Hanks, B., & McCauley, R. (2010). Pair debugging: A transactive discourse analysis. In *Proceedings of the sixth international workshop on computing education research.* Aarhus: ACM.
Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. (2003). Improving the CS1 experience with pair programming. In *Proceedings of the 24th technical symposium on computer science education.* New York, NY: ACM.

Nawrocki, J., & Wojciechowski, A. (2001). Experimental evaluation of pair programming. In *Proceedings of European software control and metrics*, London, UK.

Plonka, L., Sharp, H., & van der Linden, J. (2012). Disengagement in pair programming: Does it matter? In *Proceedings of the 34th international conference on software engineering*. Piscataway, NJ: IEEE.

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming *Communications of the ACM, 56*, 34–36. doi:10.1145/2492007.2492020

Ryu, E. & Agresti, A. (2008). Modeling and inference for an ordinal effect size measure. *Statistics in Medicine, 27*, 1703–1717. doi:10.1002/sim.3079

Sanders, D. (2002). Student perceptions of the suitability of extreme and pair programming. In M. Marchesi, G. Succi, D. Wells, & L. Williams (Eds.), *Extreme programming perspectives* (pp. 168–174). Reading, MA: Addison-Wesley Professional.

Sfetsos, P., Stamelos, I., Angelis, L., & Deligiannis, I. (2006). Investigating the impact of personality types on communication and collaboration-viability in pair programming – An empirical study. In D. Hutchinson, T. Kanade, & J. Kittler (Eds.), *Extreme programming and agile processes in software engineering* (Vol. 4044, pp. 43–52). Oulu: Springer Berlin/Heidelberg.

Sharp, H., & Robinson, H. (2010). Three 'C's of agile practice: Collaboration, co-ordination and communication. In T. Dingsøyr, T. Dybå, & N. B. Moe (Eds.), *Agile Software Development* (pp. 61–85). Berlin: Springer.

Srikanth, H., Williams, L., Wiebe, E., Miller, C., & Balik, S. (2004). On pair rotation in the computer science course. In *Proceedings of the 17th conference on software engineering education and training*, Norfolk, VA.

Stapel, K., Knauss, E., Schneider, K., & Becker, M. (2010). Towards understanding communication structure in pair programming. In A. Sillitti, A. Martin, X. Wang, & E. Whitworth (Eds.), *Agile processes in software engineering and extreme programming* (Vol. 48, pp. 117–131). Trondheim: Springer.

Thomas, L., Ratcliffe, M., & Robertson, A. (2003). Code warriors and code-a-phobes. *ACM SIGCSE Bulletin, 35*, 363–367. doi:10.1145/792548.612007

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, VA.

Vanhanen, J., & Lassenius, C. (2005). Effects of pair programming at the development team level: An experiment. In *Proceedings of the international symposium on empirical software engineering*. Noosa Heads: IEEE.

Watzlawick, P., Bavelas, J. B., & Jackson, D. D. (1967). *Pragmatics of human communication: A study of interactional patterns, pathologies, and paradoxes*. New York, NY: WW Norton & Company.

Werner, L. L., Hanks, B., & McDowell, C. (2004). Pair-programming helps female computer science students. *Journal on Educational Resources in Computing, 4*(1), Article 3.

Williams, L. (2008). Introduction to Pair Programming (Version 2) [Lecture notes]. Retrieved from https://www.youtube.com/watch?v=rG_U12uqRhE

Williams, L., & Kessler, R. (2000). All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM, 43*, 108–114. doi:10.1145/332833.332848

Williams, L., & Kessler, R. (2001). Experiments with Industry's? Pair-programming? Model in the computer science classroom. *Computer Science Education, 11*, 7–20. doi:10.1076/csed.11.1.7.3846

Williams, L., & Kessler, R. (2002). *Pair programming illuminated*. Boston, MA: Addison-Wesley Longman Publishing.

Williams, L., Kessler, R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software, 17*, 19–25. doi:10.1109/52.854064

Williams, L., McCrickard, D. S., Layman, L., & Hussein, K. (2008). Eleven guidelines for implementing pair programming in the classroom. In *Proceedings of the 2008 AGILE conference.* Toronto: IEEE.

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education, 12*, 197–212. doi:10.1076/csed.12.3.197.8618

Wilson, J. D., Hoskin, N., & Nosek, J. T. (1993). The benefits of collaboration for student programmers. *ACM SIGCSE Bulletin, 25*, 160–164. doi:10.1145/169073.169383

Zarb, M., Hughes, J., & Richards, J. (2012). Analysing communication trends in pair programming using grounded theory. In *Proceedings of the 26th BCS conference on human-computer interaction*, Birmingham, UK.

Zarb, M., Hughes, J., & Richards, J. (2013). Industry-inspired guidelines improve students' pair programming communication. In *Proceedings of the 18th ACM conference on innovation and technology in computer science education*, Canterbury, England, UK.