It's Never Too Early: Pair Programming in CS1

Krissi Wood
School of ICT
Otago Polytechnic
Dunedin, New Zealand
Krissi.Wood@op.ac.nz

Dale Parsons
School of ICT
Otago Polytechnic
Dunedin, New Zealand
Dale.Parsons@op.ac.nz

Joy Gasson
School of ICT
Otago Polytechnic
Dunedin, New Zealand
Joy.Gasson@op.ac.nz

Patricia Haden
School of ICT
Otago Polytechnic
Dunedin, New Zealand
Patricia.Haden@op.ac.nz

Abstract

This paper describes the use of the Pair Programming software development methodology in the earliest weeks of a first programming course. Based on a broad, subjective assessment of "programming confidence", instructors placed students in level-matched pairs for a portion of their programming exercises. Students who began at the lowest levels of confidence showed significantly better exercise completion rates when paired than when working individually. Student response to the Pair Programming technique was uniformly positive, and teaching staff report pedagogical, mechanical and social benefits from the practice. These data indicate that successful programming pairs can be constructed based on tutors' subjective judgements of student performance very early in CS1, before exam scores or code quality assessments are available. Thus Pair Programming can be an effective classroom intervention even with extreme novices.[·]

Keywords: Programming education, Pair Programming, Novice programmer.

1 Introduction

Failure rates in first computer programming papers (usually called CS1) are alarmingly high, often greater than 40% (Bennedsen and Caspersen, 2007). Recent work (Robins, 2010) has identified student struggles in the first days and weeks of CS1 as a significant contributing factor to this high failure rate. Robins has demonstrated mathematically that students who fail to acquire the core concepts presented in first programming lessons are frequently unable to recover, leading to high drop out and failure rates. He maintains that this is largely due to the scaffolded structure of computer programming, where each skill builds upon, and requires mastery of, a set of simpler skills. Thus it is essential that we find classroom approaches and interventions that can support novice programmers during their earliest teaching sessions. In the current study, we explore the possibility of leveraging a specific programming methodology – Pair Programming – in the very first weeks of CS1. To do this, we introduce a protocol for assigning students to pairs using holistic judgements made by in-class teaching staff. These judgements were made after the second week of CS1 before either exam marks or code quality assessments were available. As detailed below, this pairing protocol resulted in significantly better class performance for those students who initially appeared to be at greatest risk.

Pair Programming is a formal software development protocol where two programmers work synchronously on a single piece of code (Williams and Kessler, 1998). The protocol includes detailed policies for participant roles and procedures. One member of the pair is the Driver, who controls the mouse and keyboard, physically creating the code. The other member of the pair is the Navigator, who oversees the construction process, watches for errors, makes suggestions and locates resources. Partners switch roles at regular intervals, usually every 15 to 20 minutes. Pair Programming originated in industry but has, in the last decade, become increasingly common in the classroom. An active research community is exploring the potential benefits of Pair Programming to students and teachers, while considering mechanical and procedural issues in its use.

Studies have shown that Pair Programming can contribute to an improvement in learning outcomes. In a large longitudinal study involving several thousand students, McDowell, Werner, Bullock, and Fernald (2004) found that students in classes that used Pair Programming were more likely to complete their classes and to continue in a computer science major than were students in comparable classes that used only solo programming. Students from the Pair Programming classes had equivalent exam performance to solo students, addressing the concern of some educators that Pair Programming permits one student to "freeload" on a stronger partner.

Similarly, Mendes, Al-Fakhri and Luxton-Reilly (2005 and 2006) have performed two large-scale studies of Pair Programming at the University of Auckland. In these studies, students in Pair Programming classes performed better on programming exercises, and earned higher exam marks, than solo programming controls.

Williams (2007) describes the lessons learned in seven years of using Pair Programming at a large university in a variety of Computer Science papers at all academic levels, including graduate. Williams details benefits of the protocol for both teachers and students. For students,

Pair Programming supports the building of stronger social relationships (through the need to work together), increases retention, and reduces "waiting time" for teacher feedback as two students working together can often resolve a problem for which a student working alone would require teacher assistance. For teachers, the protocol reduces marking time (by halving the number of submitted assignments), reduces student demand in practical sessions, and improves general work ethic by, they hypothesise, engendering a sense of mutual responsibility between partners.

Braught, Wahls and Eby (2008) performed a tightly-controlled study of Pair Programming. In a large programming paper with multiple sections, they randomly assigned some sections to use Pair Programming and some to use solo programming. Students enrolled in a section without knowing which method would be used in that class, and were not allowed to transfer between sections after the start of the paper. Braught et al. compared code quality on individual assignments, as well as exam marks and time to complete assignments. They found an interaction between programming protocol and scores on the Scholastic Aptitude Test (a test of general academic level administered prior to college or university entrance in the United States) such that greatest benefit of Pair Programming was seen for students with lower SAT scores. This implies that Pair Programming might be especially helpful for those students who would otherwise struggle with a programming paper, which is in accordance with the higher retention and completion rates seen in many Pair Programming studies.

In all of the preceding studies, (and in others discussed below) subjective student feedback was gathered, asking students for their views of the Pair Programming experience. Student feedback is nearly universally positive, with students reporting that they enjoy working in pairs, that they feel they can program more quickly with a partner, that they are less likely to "get stuck", and that they appreciate the opportunity to get to know fellow students through working together. Negative feedback (and less positive performance outcomes) occurs primarily in the case of dysfunctional pairings, that is, when partners are unable to work effectively together.

Although some of the reported benefits of Pair Programming can be obtained simply through random pairing (e.g. McDowell et al., 2004 used only random pairing) there is compelling evidence that careful selection of pairs reduces the probability of dysfunctional pairings. Specifically, both educational benefit and student satisfaction appear to be maximised when the two members of a pair have similar levels of programming ability.

In the long-term study described by Williams (2007), teachers experimented with a variety of metrics to determine pairings, including standardised general exam scores, grade point average, the results of personality and self-esteem tests, learning style scores, and work ethic (based on self-report). They paired students in various combinations of these measures, using both similarity and dissimilarity of scores. The most successful pairings were those based on similar mid-term exam score, the most direct measure of a student's programming skill at the time of the pairing. On self-report, Williams' students consistently request to work with a student of equal or greater programming skill. Since it is not possible to give one member a stronger partner without giving the other a weaker partner, Williams recommends attempting to pair students of equal skill levels.

Cliburn (2003) explored directly the effect of partner similarity by constructing highly dissimilar pairs. He originally paired students "from different cultural or ethnic backgrounds [and]…upper with lower classmen". The result was poor collaboration and poor exam performance. He then re-paired students based on their project marks, matching students with similar results. With these pairings he observed better project quality and completion rates, and higher exam scores.

Direct inspection of students' experience of Pair Programming also shows the advantage of pairing students of similar ability. Chaparro, Yuksel, Romero and Bryant (2005) used a variety of metrics to explore students' qualitative views of Pair Programming. Through the use of observation, questionnaires, semi-structured interviews and field notes they determined that students prefer, and find most effective, pairings of similar skill levels. Katira, Williams and Osborne (2005) queried students directly about the "compatibility" of their Pair Programming partner. Students rated as more compatible those partners whom they perceived to be of similar skill. Students' perception of the skill levels of their partners was accurate, as measured by exam scores and grade point average.

More recent studies (e.g. Radermacher and Walia, 2011 and 2012; Braught, Wahls and Eby, 2008) have accepted pairing by skill level as the appropriate default, citing the accumulating evidence in its favour.

While there is a growing consensus that pairing by skill level produces the most successful Pair Programming experience, the measurement of skill remains problematic. As we are interested in the use of Pair Programming very early in a first programming course – ideally in the first weeks – we require a measure of ability to be made before exam or major project scores are available. We have thus used a subjective metric, based on instructor observation of student performance, which can be made in the first weeks of the semester.

Our observational assessment of ability is based on what we call "programming confidence". The term "confidence" in this context is not a personality metric; it does not, in our experience, correlate with self-esteem. It is a description of the way in which students approach programming exercises. The confident student programmer approaches coding exercises boldly, is willing to experiment with the techniques being learned, is relatively unfazed by coding errors and seems to expect to be able to solve the assigned problem. These students may have prior programming experience in school or as a hobby, or they may have a history of success in contexts they perceive to be similar to programming (e.g. games or puzzles), or they may simply feel comfortable with the particular intellectual exercise involved. Student programmers who lack confidence are less able to make independent progress with coding exercises. They frequently become "stuck", and will wait for assistance from the instructor, rather than try an alternative approach

on their own. This slows their work pace, and often makes it difficult for them to complete in-class assignments in the allotted time. Programming confidence, as we define it here, reflects current programming ability, and can change rapidly as the student gains experience. We have observed that students who start out with little confidence can eventually develop considerable programming skill, if they are able to navigate successfully the difficult early stages of learning. In section 2, we discuss further the process used to make our assessments of student programming confidence.

It is interesting to note that Thomas, Ratcliffe and Robertson (2003) attempted to place students on an equivalent continuum of programming confidence by self-report. Each student was asked to rate himself or herself on a 10-point scale from "Code Warrior" to "Code-Phobe". Thomas et al.'s description of these terms is extremely close to our conceptualisation of programming confidence. Based on the students' own rating, Thomas et al. compared the efficacy of same vs. opposite pairings. That is, in one condition they paired two high scoring students or two low scoring students; in the other condition, they paired a high-scoring student with a low-scoring student (middles were always paired with other middles). They report that Same pairings perform better than Opposite pairings on coding exercises, and that students consistently prefer being paired with someone at their own level on the Warrior-Phobe scale.

Another factor that has been explored as a potential determinant of the efficacy of Pair Programming is the time course of the pairing. McDowell et al. (2004) paired students for an entire semester, and pairing was used on both in-class and out-of-class assignments. Radermacher and Walia (2011), in contrast, paired students only for a single 50-minute class session. Based on their lengthy experience with Pair Programming, Williams (2007) and her colleagues (see for example, Nachiappan, Williams, Ferzli, Wiebe, Yang, Miller and Balik, 2003) recommend switching pairs often. They note that this reduces the impact of any dysfunctional pairing and increases the social benefit which many students cite as an advantage of the method. They further advise that Pair Programming be initially used only in-class, until students have mastered the technique. This has the added benefit of eliminating scheduling difficulties, which are noted as problematic by many students in studies using out-of-class exercises (cf. McDowell et al., 2004; Hanks, 2006).

Thus, following current best practice for the implementation of Pair Programming in the classroom, we intend to pair students based on programming confidence (as defined above), to include a combination of paired and individual exercises during the semester, and to change pairs for each Pair Programming session. In this way we hope to be able to use Pair Programming in the very earliest stages of programming education, where it is hypothesised that students are at greatest risk of failure (cf. Robins, 2010).

2 Method

The study was conducted during a one semester (16 teaching weeks) offering of a first programming course at Otago Polytechnic in New Zealand. "Programming 1" is a required paper in the first semester of our Bachelor of Information Technology degree. For the majority of students it is their first exposure to formal computer programming, although there are generally a small number of students who have previously taken a programming paper (some who have previously taken Programming 1 but not passed), and occasionally students with hobbyist coding experience. In this offering, 40 students started the paper, including 3 repeaters and 11 with some other prior programming experience.

The focus of Programming 1 is on programming fundamentals, such as variable manipulation and flow of control. The paper is taught in C# using Visual Studio, but is taught exclusively on the console, and contains only minimal Object-Oriented theory (formal OO and GUI work begins in our Programming 2 paper in second semester). Programming 1 comprises two two-hour sessions each week. In a typical session, a new topic is introduced by the lecturer with discussion and code examples. Students are then given a set of practical exercises to perform in class on the discussed topic. Practicals are designed to be completed during class by the majority of students.

In previous offerings of Programming 1, each student worked individually on all practical sessions. In the semester in which this study was conducted, Pair Programming was introduced in selected sessions. Our goal was to begin Pair Programming as early as possible, matching students at comparable levels of ability, as dictated by the current literature. While the common quantitative metrics of ability – exam scores and code quality assessment – are not available in the first weeks of CS1, our experience as programming instructors convinced us that there were observable differences between students even in these early stages. These differences we have summarised as "programming confidence" (see discussion above). We hypothesised that programming confidence could be a criterion for the construction of successful pairs. Further, we believed that judgements of programming confidence could be made simply through observation of student behaviour by experienced programming educators. This hypothesis was based on our conviction, developed over some 40 years of combined CS1 teaching experience, that "we know it when we see it". Thus we determined to assign students subjectively to one of three levels of programming confidence, and to use this assignment to construct programming pairs.

Teaching staff predicted that they needed at least four teaching sessions to identify accurately each student's level of programming confidence. Thus, for the first four sessions (i.e. the first two weeks of the semester), students worked individually while teaching staff carefully observed their behaviour.

The four session topics were:

1. Introduction to the IDE and writing to the screen.
2. Introduction to variables and reading from user input.

3. Introduction to data types and computation.
4. Small interactive program combining reading user input, performing computation using multiple data types, and writing output.

After the fourth session, the two classroom tutors made their initial confidence assignments individually. Each student was assigned to a confidence band, with 1 being lowest confidence, 3 being highest confidence and 2 being intermediate. These assignments were made subjectively, reflecting the tutor's sense of how confidently each student approached the programming exercises. Where there were disagreements between the two tutor judgements, the final banding was made collaboratively through detailed discussion of each student's progress and consideration of the number of in-class exercises the student had been able to complete. Prior to the banding, tutors had anticipated difficulty assigning students who fell at the borders of the banding categories. In practice, while tutors had some uncertainty at the boundary between levels 2 and 3, they had no difficulty identifying those at level 1 and there were no disagreements between the two teaching staff about who belonged in this category.

Although no specific quantitative metrics were used to determine confidence bandings, the in-class tutors identified a number of behaviours which they both used consistently to identify low confidence students. These included:

- Getting stuck: The student simply stops working and either switches to some non-related task or waits passively for tutor assistance.
- Copy-coding: The student begins reproducing code samples verbatim where they are not appropriate.
- Frantic random changes: The student begins inserting and deleting code elements randomly in the hopes that an error will be resolved, without any organised plan.

The consistency of assignment to level 1 by both tutors even in the absence of specific quantitative metrics is notable. The very low confidence student seems almost qualitatively different from his peers, at least in the perception of an experienced programming teacher. In future semesters, we intend to analyse formally the initial judgements of the two classroom staff to obtain a statistical measure of inter-rater reliability.

For the next four weeks of the semester, the two classroom sessions each week were handled differently. In the first session students worked individually; in the second session, students were assigned to pairs and used a formal Pair Programming code development methodology. (The technique was explained prior to the first Paired session.) Students were paired based on banding such that each student worked with a student at the same confidence level. Each student was assigned a different partner for each of the four paired sessions. The pairing assignment was made by the instructors prior to the class session and announced at the beginning of practical work time. In cases where an odd number of students necessitated a cross-banding pairing, this was arranged by the instructors based on their assessment of the students' suitability. Where an odd number of students required one student to work alone, this role was always given to a more experienced Level 3 student. For each session, instructors recorded exercise completions and observed student behaviour.

At Week 6, after four weeks of using Pair Programming in alternating sessions, student feedback was collected. See below for details. Additionally, students were rebanded at this time. The course instructors had noted that different students were progressing at different rates (as is generally true in Programming 1) and some students who had initially been placed in the same band were now working at different levels of confidence. The rebanding used, as much as possible, the same criteria as the original banding. That is, students who were still obviously struggling were assigned to Level 1, and those who were working independently were assigned to Level 3. The new banding was not based on a student's ranking relative to the rest of the class. Thus it was technically possible that the second banding would have no Level 1 students. In actual fact, the second banding produced 8 Level 1 students (22%), 22 Level 2 students (61%) and 6 Level 3 students (17%). See below for a more detailed discussion of the changes in banding over time.

Weeks 7 to 9 of the paper were spent in revision and preparation for the mid-term exam, so no formal practical sessions were held. After the mid-term exam, students were banded based on their exam score to provide an external comparison for the instructors' subjective bandings. Students scoring 55% or lower were considered Level 1, students scoring 55% to 75% were considered Level 2, and students scoring more than 75% were considered Level 3.

The purpose of the second banding (and the banding based on exam score) was to prepare for pair assignments in the remaining weeks of the semester, where we intended to continue the alternation of individual and paired practical sessions. However, students began to express a preference for working in pairs rather than individually. In view of this attitude, and given the positive impact of Pair Programming that was observed during the first experimental weeks (see below) the instructors decided that educational efficacy took precedence over data collection, and did not require students to perform any practicals individually after week 11. The instructors continued to place students into pairs for the planned Paired sessions if they had not self-paired, but students were also allowed to construct their own pairs. Thus only weeks 3 to 6 (inclusive), 10 and 11 are included in the analysis.

3 Results[1]

3.1 Practical Lab Completions

During the six experimental weeks, there were six individual and six paired practicals. The mean number of individual practicals completed on time per student was 4.58; the mean number of paired practicals completed on time was 4.97 ($F_{1,34} = 2.489$; $p<.05$).

The distribution of the difference between numbers of paired and individual lab completions across students is shown in Figure 1. Of the 14 students who completed equal numbers of individual and paired practicals, 8 (57%) completed all twelve labs. This apparent ceiling effect compromises our ability to sensitively observe the impact of Pair Programming for students at the top end.

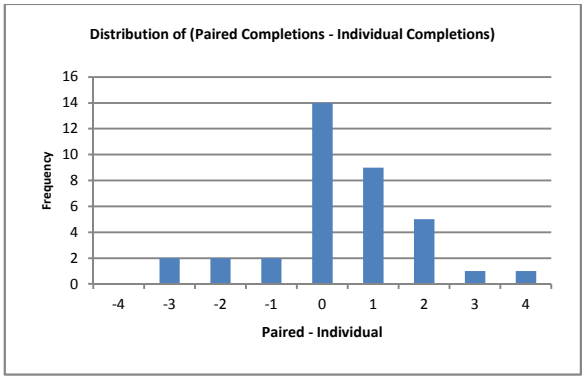

Figure 1: Distribution of Completions

To observe more closely the differential impact of pairing on students of different initial confidence, we can compare completion rates for students based on their first bandings. Due to the low number of students initially banded at Level 3 who did not withdraw from the paper prior to the midterm exam, we combine Levels 2 and 3 for this analysis. Students who had initially been banded at Level 1 completed on average .84 more paired labs than individual; students initially banded in Levels 2 or 3 completed on average .12 fewer paired labs than individual ($F_{1,34} = 4.11$; $p=.05$). This pattern does not seem to be attributable entirely to a ceiling effect, as the total mean labs (out of 12) completed for initial Level 1 students is 9.05, and for initial Level 2/3 students is 10.11. This difference is not significant ($F_{1,34} = 2.53$; $p=.12$). Thus the benefit of Pair Programming as measured by practical lab completion rates appears to be primarily for those students who initially exhibited the greatest difficulty with programming.

3.2 Programming Confidence Bandings

The proportion of students at each Level for each of the bandings is shown in Figure 2. The proportion of students at Level 1 decreased between week 2 and 6, while the proportion at Level 2 increased. Assuming that programming confidence increases with experience, this pattern is as expected. The mid-term banding shows a steep increase in the proportion of students placed at Level 3. Since this banding was based not on instructor judgment (as the Week 2 and Week 6 bandings were) but on exam score, it is not possible to determine whether this shows an actual continuation of the trend of increasing confidence, or is just a reflection of a comparatively easy exam.

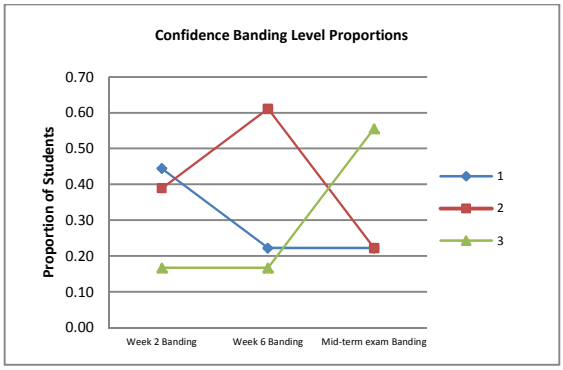

Figure 2: Proportion of students at each level for each banding.

As discussed above, students initially placed in confidence Level 1 turned in significantly more practical labs from paired than from individual sessions, while students initially placed in confidence Levels 2 and 3 did not. This indicates that the instructors' subjective ratings of programming confidence do correspond to some student quality relevant to performance in Programming 1. To interpret this pattern fully, it will help to explore precisely what is being measured in the instructors' confidence judgements. The difficulty of accurately predicting, or even measuring, programming skill has been discussed widely (see, for example, McCracken et al., 2001) and complicates all research into programming education. It would be useful to discover that something as simple as tutor observation could be used to make such a prediction.

If early programming confidence is a useful predictor of eventual programming performance, we would expect to see a correlation between initial banding judgement and final course mark. This was not observed (Spearman-r = -0.17; ns). However, it is interesting to consider student performance not just as a function of initial confidence, but as a function of the *change* in confidence seen between Week 2 and Week 6. Since confidence banding judgements were absolute, not relative, we would have hoped to see all students' confidence scores improving with experience, and this pattern was seen generally in the summary of proportions shown in Figure 2, where many students moved from Level 1 at Week 2 to Level 2 at Week 6. However, not all students' banding scores did increase. In fact, of the 36 students who earned final marks in the paper, 20 (56%) actually maintained the same confidence banding from Week 2 to Week 6 (33% went up; 11% went down). Perhaps when predicting eventual programming skill it is useful to look not only at where the student starts, but how rapidly he or she gains programming confidence. To assess this, we can look at the relationship between students' change in confidence in the early weeks of the paper, and their eventual final course mark. For this analysis, we omit the 6 students originally at Level 3 since it was not possible for them to increase their confidence band. Of the

[1] To allow comparisons between analyses, four students who withdrew from the paper prior to the midterm exam have been omitted from all results summaries.

remaining students, one student's banding dropped from Week 2 to Week 6, 14 stayed the same, and 11 improved. The mean final course marks for the three groups are shown in Figure 3. Those who improved their confidence rankings from Week 2 to Week 6 earned significantly higher final course marks, on average ($F_{2,23} = 3.4$; p=.05).

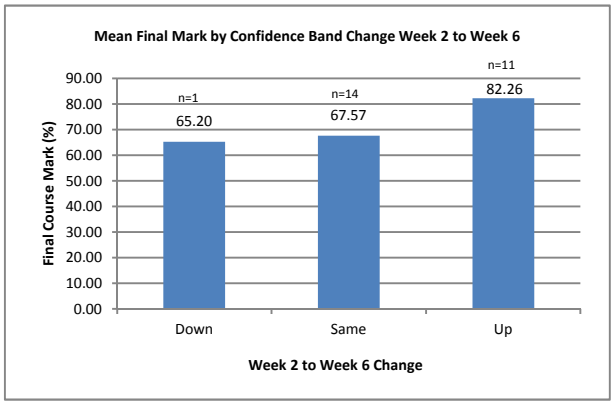

Figure 3: Mean final mark by confidence band change.

3.3 Student Feedback

After the first four weeks of alternating individual and Pair Programming sessions (at Week 6), students completed a brief questionnaire covering their attitudes toward the Pair Programming techniques. The questionnaires were submitted anonymously, and were administered by a non-teaching member of the research team. The questions asked are shown in Table 1.

1.	Which do you enjoy more: pair programming or working alone? Why?
2.	Do you feel you program better in a pair or on your own? Why?
3.	What did you ***like*** about the pair programming sessions?
4.	What did you ***dislike*** about the pair programming sessions?
5.	Would you like to continue to use pair programming during the remainder of the semester?
6.	Which best describes your programming education experience prior to this paper? 1) No prior experience 2) Hobbyist or self-taught 3) Have taken one or more previous programming papers.
7.	Any other comments?

Table 1: Feedback questionnaire Week 6

After Week 12 of the paper, feedback was again collected. Since prior experience was not expected to be as relevant, given that even complete novices had been through 12 weeks of programming education, Question 6 was replaced with a question designed to elicit students' opinions about how best to construct a pair: *"Think about the most effective pairings that you have been in this term. What do you think makes a Pair Programming partnership successful?"*

Figures 4 to 6 show summaries of responses to the three binary questions (numbers 1, 2, and 5 in Table 1) comparing Week 6 and Week 12.

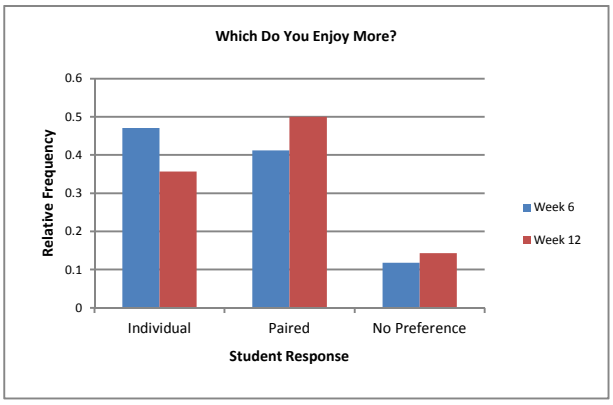

Figure 4: Student preference

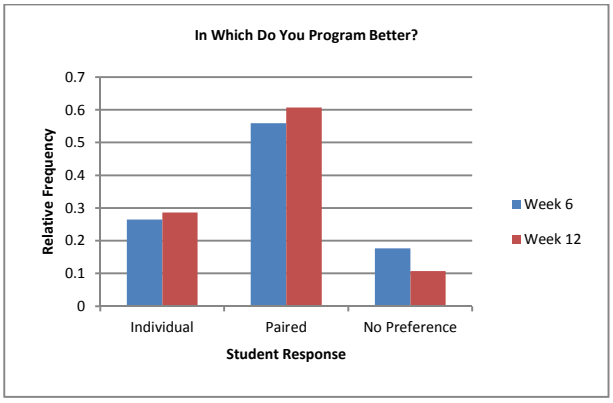

Figure 5: Student judgement of quality

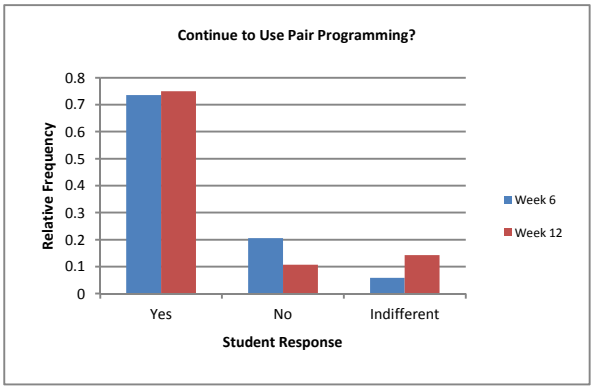

Figure 6: Student willingness to continue

There are no significant differences between the patterns of responses to these questions at Weeks 6 and 12 (by $\chi 2$). The main effect of response collapsed across Weeks is significant for all questions (by $\chi 2$; p<.002).

In Week 6, a greater proportion of students preferred working individually to working in pairs than at Week 12 (47% to 41% at Week 6; 35% to 50% at Week 12). Although this effect is not statistically significant, the trend corresponds to the classroom instructors' observation that students became more comfortable with the protocol over time. Based on students' free comments (see below) this appears to be due both to a reduction in social awkwardness as students get to know each other, and increased value of the protocol as the programming tasks become more challenging.

In both Week 6 and Week 12, a greater proportion of students felt they "programmed better" in a pair (58% to

27% collapsed across weeks). Student free comments identify a number of possible rationales for this, including the sharing of ideas, greater opportunity for code checking and increased motivation to do well. In both weeks the majority of students stated that they wished to continue to use Pair Programming during the remainder of the semester (74% to 16% collapsed across weeks). Student free comments show a number of caveats, however, primarily an unwillingness to work with partners who were perceived as weaker programmers.

The reluctance of students to work with a weaker partner can also be seen by looking at the pattern of responses in Week 6 to the three binary questions as a function of self-reported experience level (Question 6 in the Week 6 survey). Figures 7 to 9 show these results.

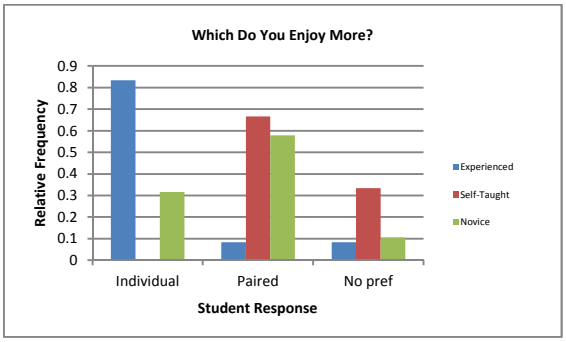

Figure 7: Student preference by previous experience

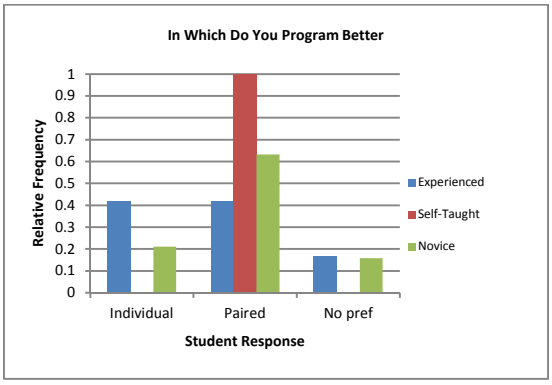

Figure 8: Student judgement of quality by previous experience

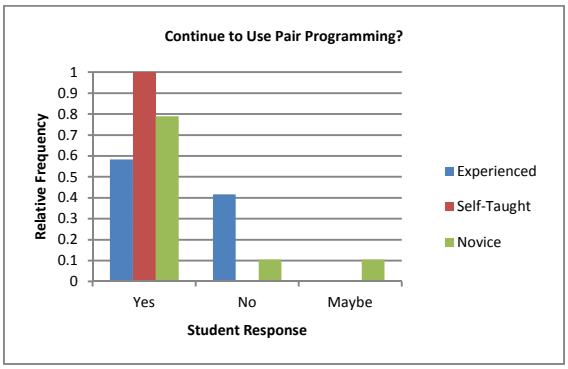

Figure 9: Student willingness to continue by previous experience

Students identified themselves as Experienced (n=12), Self-taught (n=3) or Novice (n=19). Students who classified themselves as Experienced were significantly more likely to prefer working alone than were their less experienced classmates (by $\chi 2$; $p<.02$). A similar trend of reluctance of the Experienced students to work in pairs was seen in the questions about programming quality and desire to continue using Pair Programming, but these effects were not statistically significant (by $\chi 2$).

In the remaining survey questions students were asked to identify specific things that they liked and disliked about Pair Programming, and to provide any further comments they wished to make. There was good consistency among student comments, and we were able to identify a small number of comment categories. The complete comment coding for Week 2 and Week 6 is given in Table 2. For each general class of comment, Table 2 shows the proportion of students who made the comment in each week, and the change in proportion from Week 6 to Week 12.

		Week 6 n=33	Week 12 n=27	
Type	**Comment**	**Pr (Wk 6)**	**Pr(Wk 12)**	**Change**
Adv. Indiv.	Can work at own pace	0.12	0.11	-0.01
	More effective learning	0.30	0.11	-0.19
	Can use own methods	0.09	0.15	0.06
Adv. Pair	Allows discussion	0.06	0.04	-0.02
	Builds sense of community	0.39	0.26	-0.13
	Faster	0.42	0.33	-0.09
	More fun	0.03	0.04	0.01
	Can get help when stuck	0.52	0.44	-0.07
	Can learn from explaining	0.06	0.07	0.01
	More code checking	0.15	0.15	0.00
	Motivating	0.03	0.04	0.01
	Can get other viewpoints	0.27	0.59	0.32
Disadv. Pair	Boring for navigator	0.06	0.07	0.01
	Enforced social interaction	0.24	0.15	-0.09
	Evaluation apprehension	0.06	0.15	0.09
	Partners can be incompatible	0.06	0.15	0.09
	Don't like working with stronger partner	0.06	0.04	-0.02
	Classroom is too noisy	0.03	0.00	-0.03
	Don't like working with weaker partner	0.21	0.07	-0.14

Table 2: Summary of student comments

In Week 6, the most commonly mentioned advantage of Pair Programming was that one could get help from the partner when stuck (mentioned by 52% of respondents). Often the note "instead of having to wait for the lecturer" was added. Novice programmers traditionally need a great deal of assistance, and in large classes it can be difficult for an instructor to respond to all requests in a

short time. While one might expect that two novices working together would not be able to provide useful support to each other, this does not seem to be the case based on this feedback.

In Week 12, the "able to get help" comment was still often made (mentioned by 44% of respondents) but at this point the most frequently cited advantage of Pair Programming was the ability to get another person's viewpoint and suggestions (this was often phrased as "two heads are better than one"). This shift seems to reflect students' increasing independence from the lecturer between Weeks 6 and 12.

In both weeks, students often noted that working as a pair was faster (42% and 33% in Weeks 6 and 12 respectively), and that it built a sense of community among the students (39% and 26%) as it required them to meet and get to know their classmates. This impact on the social dynamics of the classroom was among the features that the instructors found most salient during this semester (see further discussion below).

The most frequently cited disadvantage of Pair Programming in Week 6 was the difficulty of working with a weaker partner (mentioned by 21% of respondents). By Week 12, this had fallen to only 7%, perhaps indicating that some of the novice programmers had "caught up" quickly to the more experienced members of the class.

A commonly cited advantage of individual programming, especially early in the semester, was that students felt they learned more effectively when they had to work everything out on their own (30% in Week 6; 11% in Week 12). This illustrates the value of including both individual and Pair Programming sessions.

4 General Discussion

In the interest of finding teaching interventions that can be used successfully in the earliest weeks of a first programming course, we introduced the Pair Programming methodology into our CS1 paper. Based on previous explorations of the pedagogical use of Pair Programming, we intended to construct pairs on ability level, but wished to do so before any exam or significant project marks would be available. We thus used a holistic, subjective judgement made by classroom instructors based on task performance and work style that reflects an attribute we call "programming confidence". Results of the first semester show that Pair Programming increases practical lab completion rate significantly for those students who were initially judged as having the lowest confidence.

Initial confidence judgements were not correlated with final course mark. Some (but not all) students who had started with low confidence performed very well in the paper; some (but not all) students who started with high confidence levels failed to achieve a high final mark. Thus low initial confidence in isolation is not an indication of future poor performance. However, inspection of change in confidence during the early weeks does seem to give a better insight into eventual outcome. Specifically, students whose confidence level improved between Weeks 2 and 6 of the paper earned higher final marks, on average, than those whose confidence remained at the same absolute level. Thus, it is apparently difficult to catch up if you fall behind in the first six weeks of CS1. This finding is in concert with the mathematical model of Robins (2010) which demonstrates that failure to thrive in the earliest weeks can be a significant contributor to low pass rates in CS1. To identify students at risk, perhaps with an eye to providing additional support, it seems productive to watch carefully for students who do not gain confidence with programming even very early in their first course. This identification can possibly be made by careful instructor observation – no elaborate assessment metric is required.

Student feedback regarding the use of Pair Programming was generally positive, with respondents identifying advantages mechanical (not having to wait so long for instructor attention), intellectual (the value of a second viewpoint) and social (an effective way to get to know other class members). Students expressed concern about uneven or incompatible pairings, and the classroom instructors report that it is necessary to watch closely for dysfunctional pairings (for example, where one member of the pair is being too dominant) and intervene when required.

In addition to the observed advantages accruing to students, the classroom teaching staff reported a number of positive consequences of using Pair Programming. These included:

- *Shorter waiting times:* Our Programming 1 paper is taught in groups of up to 23 students at a time. During practical work time, classroom instructors move about the room answering questions or offering assistance when students are not progressing. In the first weeks of CS1 when most students have very little idea of how to program, this can be a taxing process for instructors. At our institution we have recently begun assigning two instructors to Programming 1 simply to reduce student wait times. This unfortunately imposes a staffing burden that can be very difficult to manage. With the introduction of Pair Programming, instructors notice a significant reduction in "students waiting with their hands up". Partially, this is because each instructor intervention now covers two students, but more positively, even novice students, when working with a partner, seem to be able to progress more consistently. As the students frequently observed, two heads are indeed better than one.
- *Increased Engagement:* An historical problem for more experienced students in Programming 1 has been lack of engagement. This is a particular issue for those students who have previously failed the paper, and are repeating it. For these students, the earliest weeks can seem rather pointless. Instructors noted, however, that when working with another more experienced student, repeaters and students with some other prior experience were much more engaged than in previous years. The opportunity to discuss the work with a student of similar level and to perhaps share interesting approaches or possible extensions of the exercises, made the early weeks much more rewarding for students at the top end.
- *Increased Motivation and Performance:* Each set of practical tasks contains one or more "challenge problems", optional exercises of greater difficulty. The instructors note that students are more likely to attempt

the optional challenge exercises during the Pair Programming practicals than during the individual practicals. This may be a reflection of the confidence obtained from knowing one has a partner to help out on a difficult problem, and/or the desire to perform well when working with another student. Interestingly, the same "striving for excellence" was observed in the major individual project assignment where an unusually high number of students attempted extra credit work, in contrast to previous years.

- *Social Dynamic:* The change which the instructors find the most compelling argument for continuing to use Pair Programming is not directly related to programming performance, but is a generally increased sense of community among the students. Compared to previous years, students are more likely to offer help to each other even in individual labs. Students are more likely to discuss individual assignments and ask for feedback. The general sense of camaraderie and inclusion is higher.

It should be noted that our department has recently introduced a number of other policies that might have contributed to this increased sense of community. In 2012 we have appointed a dedicated first year coordinator responsible for pastoral care of new students, we have established a student common room, built a school Facebook page and increased orientation activities for first year students. All of these probably contribute to the social cohesion seen in Programming 1. However, classroom instructors note that in the specific context of their classroom, they saw social relationships develop during Pair Programming which then grew to include other classroom activities.

In summary, we have found Pair Programming to be a valuable technique from the earliest days of CS1 when students at the same level of programming confidence, as judged by in-class teaching staff, work together. In coming semesters we will continue to introduce Pair Programming early in CS1, and also to incorporate it into our more senior programming papers. With wide-ranging benefits to both students and teaching staff, we see Pair Programming as an essential tool in successful programming education.

5 References

Bennedsen, J. and Caspersen, M.E. (2007): Failure rates in introductory programming. *ACM SIGSCE Bulletin*, **39**(2):32-36.

Cliburn, D. (2003): Experiences with pair programming at a small college. *Journal of Computing Sciences in Colleges*, **19**(1):20-29.

Hanks, B. (2005): Student performance in CS1 with distributed pair programming. *ACM SIGSCE Bulletin*, **37**(3):316-320.

Katira, N., Williams, L. and Osborne, J. (2005): Towards increasing the compatibility of student pair programmers, *Proceedings of the 27th International Conference on Software Engineering*, St. Louis, MO, USA, pp. 625-626.

McDowell, C., Hanks, B., and Werner, L. (2003) Experimenting with pair programming in the classroom. *Proceedings of the 8th annual conference on Innovation and technology in computer science education*, Thessaloniki, Greece, pp. 60-64.

Mendes, E., Al-Fakhri, L., and Luxton-Reilly, A. (2005): Investigating pair-programming in a 2nd-year software development and design computer science course. *Proceedings of the 8th annual conference on Innovation and technology in computer science education*, Thessaloniki, Greece, pp. 296-300.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001): A multi-national, multi-institutional study of assessment of programming skills of first-year CS students, *ACM SIGCSE Bulletin*, **33**(4):125-180.

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

Radermacher, A. and Walia, G. (2011): Investigating the effective implementation of pair programming: An empirical investigation, *Proceedings of the 42nd ACM technical symposium on Computer science education*, Dallas, Texas, USA, pp. 655-660.

Robins, A. (2010): Learning edge momentum: a new account of outcomes in CS1. *Computer Science Education*, **20**(1): 37-71.

Thomas, L., Ratcliffe, M., and Robertson, A. (2003): Code warriors and code-aphobes: a study in attitude and pair programming, *ACM SIGCSE Bulletin*, **35**(1):363-367.

Williams, L. (2007): Lessons learned from seven years of pair programming at North Carolina State University. *SIGSCE Bulletin*, **39**(4):79-83.