Enhancing Collaborative Learning Using Pair Programming: Who Benefits?*

Phil Maguire†
Rebecca Maguire‡
Philip Hyland‡
Patrick Marshall†

†Department of Computer Science, NUI Maynooth,
‡School of Business, National College of Ireland.

Abstract

Incoming university students who have not previously studied computer programming often find it a challenging subject, leading to high failure rates. Research has suggested that the lack of a formalised structure for collaborative learning may be one of the factors responsible for students' negative impressions of computer science. In this study we investigated whether the use of pair programming in practical laboratories would facilitate peer learning and enhance students' confidence in their programming ability. Results showed that this intervention was generally well received, although the weaker programmers (as measured by prior exam grades) perceived it to be of more benefit than the stronger ones. Students who reported a lower initial level of enjoyment and confidence in programming were more likely to report learning from the paired intervention, though this did not necessarily lead to enhanced performance. The most frequently reported positive feature of pair programming was that it allowed students to meet more people in the class. Although there was no significant increase in final exam grades for male students, there was a significant increase for female students, suggesting this teaching strategy may have asymmetrical gender benefits.

Keywords: Collaborative learning, computer programming, data structures and algorithms, pair programming, computer science, group work, gender differences.

*URL: http://ojs.aishe.org/index.php/aishe-j/article/view/141

1. Introduction.

1.1 Background.

Following a period of declining enrollment, numbers registering for computer science courses are slowly beginning to increase again (Zweben, 2011). Yet, despite this initial interest, students often find introductory computer programming challenging, resulting in a drop-out rate of approximately 25%, with many other students performing poorly (Williams & Upchurch, 2001). In Ireland, a recent HEA report found that, at 27%, computer science has the highest drop-out rate of all third level courses in the country (Mooney, Patterson, O'Connor & Chantler, 2010). As well as leading to significant attrition within third level, the perception of computer science as a “difficult” subject may discourage students from choosing to study it in the first place (Bennedsen & Caspersen, 2007).

While there are numerous explanations for such high failure rates, one of the key reasons may lie with the way in which programming is typically instructed, with limited opportunities for practice through practical lab based scenarios. For example, in an analysis of students' perceptions of programming, Hawi (2010) found that although they attributed failures in programming to many diverse factors, key reasons cited included inappropriate teaching methods, as well as a lack of opportunity for practice. Furthermore, computing is often viewed as an isolated endeavor, a factor which may impact negatively on students' perceptions and learning strategies, especially for female students (Redmond, Evans & Sahmi, 2013).

In this paper we outline and evaluate a novel technique for programming instruction which makes use of collaborative peer learning. Specifically, we apply a technique whereby, rather than working in isolation, students are required to work in pairs to come up with solutions to programming problems. While we specifically investigate this strategy within the computing domain, the technique has applications for other fields of study which support collaborative problem solving. In the following sections we elaborate on the rationale for this intervention, as well as outlining the findings of prior research which has employed a similar method.

1.2 Individual vs. collaborative learning.

It is well established that, like other skills, programming is best learned through practice and experience (Traynor & Gibson, 2004). Unfortunately, textbooks and lecture material in computer science are often heavy on declarative knowledge (i.e. facts), with particular emphasis on the features of programming languages and how to use them (Robins, Rountree & Rountree, 2003). This mode of instruction is not a sufficient basis for students to learn how to write a program. While most courses in computing now involve a practical component, the best means of implementing this component remains a contentious issue (Maguire & Maguire, 2013; Linn & Dalbey, 1989). Changes to teaching methods, such as the use of clearer textbooks and the introduction of online resources, have done little to improve programming competence (Miliszewska & Tan, 2007).

One issue which may exacerbate students' difficulties with programming is the lack of a formalised environment for collaborative peer learning. In industry, all non-trivial software projects are necessarily collaborative efforts. Professional programmers frequently avail of the expertise of their colleagues to help them solve problems and keep up to date with the latest programming techniques (McDowell, Hanks & Werner, 2003). In stark contrast, a commonly held belief in academia is that students should write programs in isolation. McDowell, Werner, Bullock and Fernald (2006) propose that the overreliance on individual programming may be rooted in the concerns of instructors; specifically, the belief that students working in groups may not learn as much as they would if they completed the assignment alone. As a result, student programmers are conditioned to equate communication and sharing with cheating (Williams & Kessler, 2000).

Beyond the domain of computer science, a considerable volume of research extols the virtues of peer learning (e.g. Boud & Lee, 2005; Jackson & Bruegmann, 2009; Hammond et al., 2010; Kepell et al., 2006; Menzies & Nelson, 2012; Staarman, Krol & Meijden, 2005; Topping, 2005; Willey & Gardner, 2010). As a result, there has been a gradual shift in education from an 'instructivist' to a 'social constructivist' approach, whereby the importance of others in learning is emphasised (Tan, Yeo & Lim, 2005). As Boud (2001) points out, in everyday life, nearly all of the information we obtain is provided by peers: we rarely need to consult a teacher or a text source. A peer can be defined as another person in a similar situation to the learner, who is not an expert practitioner. This lack of expertise can be a valuable asset, as it means that peers do not have power over each other by virtue of their position or responsibilities (Boud, 2001). Communication can therefore occur on an equal level, and information is presented in a format which more closely matches the learner’s immediate experience, leading to deeper learning (Assiter, 1995). Peer interactions also allow learners to develop valuable collaborative skills which facilitate future learning (Boud, 2001).

The discouragement of collaborative work at third level may be deterring students from taking up computer science, due to the mistaken impression that software engineering is an isolating and lonely career. In particular, women may become discouraged by the focus on individual, socially isolating work. A 2000 survey of over 400,000 freshmen entering 717 colleges and universities across the US reported the largest gender gap regarding perceived confidence in computer skills in the 35-year history of the survey (Werner, Hanks and McDowell, 2004). Only 23.2% of women (versus 46.4% of men) rated their computer skills as above average, while only 1.8% of women (versus 9.3% of men) intended to pursue computer programming careers. These figures are consistent with recent research carried out by Redmond et al. (2013) who reported that female computing students have significantly lower confidence than their male counterparts, and are also less likely to enjoy studying computer science. The perception of computing as a solitary occupation, and the belief that programming is conducted in a competitive rather than collaborative environment, were identified as two of the key reasons why fewer women are majoring in computer science (Werner et al., 2004). In light of these observations, the current study seeks to investigate whether the implementation of collaborative peer learning has any resulting gender differences. Specifically, we examine whether it is perceived more positively by female students, and whether it results in greater performance differences for one gender over the other.

1.3 Pair programming.

The benefits of collaborative learning are becoming increasingly recognized by educators in computing (Furberg, Kluge & Ludvigsen, 2013; Kaye, 2012; Hwang, Shadiev, Huang et al., 2013; Maguire & Maguire, 2013; O'Donnell, Hmelo-Silver, & Erkens, 2013; Tsai, Li, Elston & Chen, 2011; Yoon & Brice, 2011). For example, Schäfer et al (2013) found that mathematical logic skills were enhanced when students worked together in collaborative games, while Tsai et al. (2011) found collaborative learning enhanced student experience in producing Wiki websites. In addition, Maguire and Maguire (2013) found that working in teams to answer clicker questions had positive effects on student engagement and performance within computer science lectures. As we are interested in investigating how collaborative learning can be used to enhance competence in programming, the current study employs a collaborative programming technique referred to as pair programming.

Pair programming is a novel collaborative paradigm which is growing in popularity in the computer science industry (Braught, Wahls & Eby, 2011; Salleh, Mendes & Grundy, 2011). The idea is that two programmers work collaboratively on the same program at the same workstation. One programmer is designated as the ‘driver’ and has control of the input devices. The other programmer is designated as the ‘navigator’ and has the responsibility of reviewing the code that has been typed to check for deficiencies, such as erroneous syntax and logic, misspellings and design issues (McDowell, Werner, Bullock & Fernald, 2006). The navigator continuously examines the work of the driver, thinking of alternatives and asking questions (Williams & Kessler, 2003). The driver and the navigator change roles frequently and different pairs are formed to facilitate the spread of information through an organisation. Research has shown that programmers working in pairs produce shorter programs with better design and fewer bugs than those working alone (Alistair & Williams, 2001).

This collaborative technique has been successfully applied to the teaching of computer programming (e.g., McDowell, Hanks & Werner, 2003; Salleh et al, 2011; Williams & Upchurch, 2001), and has even been employed in a primary school setting (Denner, Werner, Campe & Ortiz, 2014; Gallardo-Virgen & DeVillar, 2011). A wide range of benefits have been reported, such as improved quality of code, decreased time to complete, improved understanding of the programming process, enhanced communication skills and enhanced learning (Preston, 2005). With pair programming students have the camaraderie of another person for support. The process of analysing and critiquing a program written by another is an excellent way of learning because it requires students to reflect on the code they are writing (Williams & Upchurch, 2001). In addition, students learn more when working with a partner because they derive the satisfaction of producing a quality working program as opposed to a non-working program (McDowell, Hanks & Werner, 2003).

The benefits of this technique over individual programming were initially demonstrated by Williams and Upchurch (2001), who observed that students working in pairs found the experience more enjoyable than working alone and repeatedly cited how much they had learned from each other. Team communication and effectiveness also improved. Students enjoyed the camaraderie, and felt more confident. Nagappan et al. (2003) found that students and demonstrators reported labs to be more productive, less frustrating and more conducive to advanced, active learning than traditional labs. McDowell, Werner, Bullock and Fernald (2006) found that paired students were significantly more likely to remain in the course through to the final exam (90.8% versus 80.4%). Paired students were also 18% more likely to attempt the subsequent course in programming were also more likely to register for a computer science major. Building on this work, a systematic review by Salleh et al. (2011) concluded that pair programming consistently leads to improved grades and increased student satisfaction.

1.4 Research Overview and Aims.

Much of the research on pair programming as a pedagogical technique has focused on the teaching of introductory programming to first year students, with fewer studies investigating its applicability for more advanced programming modules (Mendes, Al-Fakhri & Luxton-Reilly, 2006). Also, while there is a growing body of research in the area, few studies have focused on individual differences such as gender (Salleh et al, 2011). In the current study we describe the outcomes of adopting a collaborative pair based paradigm for the teaching of a second year computer science course in data structures and algorithms. We compare the use of individual programming in the first semester with collaborative programming in the second. The continuity of the student cohort through the two semesters permits analysis of various outcomes of the pedagogical intervention, such as perception and performance. The three central aims of the study are as follows:

1. To establish whether students benefit from a peer programming intervention in terms of their academic performance in both continuous assessment and examination results, 2. To identify students perceptions of peer programming (positive and negative) and how these relate to programming confidence,

3. To ascertain whether gender differences exist in both performance and perceptions of the pair programming intervention.

2. Method.

2.1 Participants.

This study was carried out with a second year computer science class in NUI Maynooth taking two consecutive modules in Data Structures and Algorithms. These were spread across two semesters in the same academic year. There were a total number of 121 students enrolled in semester 1 and 99 in semester 2.

2.2 Design.

A quasi-experimental design was employed with programming type (individual or paired) and gender as the key independent variables. Students' performance in assessment components (exam and CA) was the key dependent variable, but measures of attendance, programming confidence and perceptions of the pair programming intervention were also examined.

2.3 Procedure.

2.3.1 Pre-intervention phase (individual programming module).

Computer Science modules in NUI Maynooth are typically assessed with 70% of the marks awarded based on an end of semester written exam and 30% of the marks awarded for continuous assessment (CA). For the two modules in question, CA marks were awarded for each weekly lab session.

For semester 1 (pre-intervention phase) students were required to carry out weekly programming exercises in isolation, as would be standard practice on a course such as Data Structures and Algorithms. The weekly programming exercises involved students sitting at individual computers and working in isolation to code up a solution to a problem given at the start of the lab, which related to that week's lecture topic (e.g. writing a Java program to sort elements in an array). These labs lasted for 2 hours and were assessed at the end by a demonstrator who asked the student to explain their code and then assigned a grade based on how close the program was to delivering the required functionality.

2.3.2 Intervention phase (pair programming).

Pair programming was introduced in the second semester. As in semester 1, 70% of the marks were awarded based on an end of semester exam, with 30% awarded based on performance in pair programming labs. Further detail on the implementation is given below.

Preliminary Questionnaire

Prior to beginning the pair programming intervention, a preliminary questionnaire was handed out in lectures to obtain background information on students' perceived programming ability and confidence. This required students to rate their agreement on a seven-point Likert Scale regarding issues such as enjoyment of computer programming, self-efficacy at programming, likelihood of a programming career, and confidence at writing a program to solve a real world problem. Students were also asked whether they ever requested assistance from others with their programming and, conversely, whether they ever helped others. In addition, further demographic information was collected, such as the age at which students started to program, Leaving Cert Maths grade, the hours spent playing computer games, and the number of friends students had who were also studying computer science.

Pair assignment

An important consideration was deciding how to pair up the students at the beginning of semester 2. Various options were considered including letting students choose their own pairs, assigning students to pairs for the whole semester, or allowing students to change partners throughout the semester. In the interests of increasing information flow in the class, we decided to implement a system where partners would be randomly reassigned a different partner each week. The hope here was that a completely random system would help to smooth out the disparities in programming skills throughout the class and also not unfairly advantage/disadvantage students for the whole semester by having them paired up with students of particularly high or low ability.

Details of intervention

Typically, pair programming involves a setup where two students sit at a single workstation and take turns to type the code (see McDowell, Hanks & Werner, 2003). While this approach may work well for first year labs, where the problems are quite simple and can typically be solved using only a small number of lines of code, it may not be as efficient for more complex assignments. Many of the labs in the second year data structures and algorithms module involved importing large chunks of pre-written code in multiple classes, making the tracking of program structure more challenging as a pair. We were particularly concerned that the considerable disparity in programming skills within the class would mean that weaker students would not be able to meaningfully contribute as ‘navigators’. In light of these concerns, we opted for a less extreme form of collaboration whereby students worked together in pairs, but each at their own workstation. Students initially worked out a structure to the problem together on paper, discussing and agreeing on the design of the program. Once the design was approved by a demonstrator, they each implemented it on their own machine, remaining free to discuss errors and bugs with each other and compare code, though not to type on each others' keyboards.

Assessment of pair programming

As mentioned above, 30% of the marks for this module were CA based, relating to student performance in labs. A key concern was how to effectively assess this performance. According to Preston (2005), a critical feature of successful collaborative learning is positive interdependence. Students are considered interdependent when they co-ordinate their efforts with their group mates to successfully complete a task. To enforce positive interdependence, marks were awarded for how effectively the students worked together. Demonstrators assessed how well the students communicated with each other, how well they responded to each other's contributions and how well they co-ordinated their deconstruction of the problem.

Another important feature of collaborative learning is individual accountability, whereby all team members take individual tests and receive individual grades (Preston, 2006). The goal here is to ensure that every student involved in a collaborative learning activity acquires the skills the activity is intended to teach. If the emphasis is placed on program completion rather than acquisition of skills, then there is a higher risk that one member of the pair may develop the project unilaterally (Preston, 2006). In order to enforce individual accountability, students were required to individually respond to specific questions posed by demonstrators at the end of the lab. If they had simply copied code from their partner without understanding it, and were consequently unable to answer the question, then marks were lost for the individual. It was hoped that this system, combining both positive interdependence and individual accountability, would encourage pairs of students to explain the code to each other and foster a collaborative relationship.

2.3.3 Evaluation.

At the end of the intervention phase (i.e. end of semester 2) a final questionnaire was circulated to get feedback from students regarding their experiences of pair programming. For this, students had to rate their agreement with a number of statements on a seven point Likert scale dealing with overall enjoyment ("I *was glad to be working in a pair rather than alone*"), perceived benefits in terms of learning ("*I learned programming skills from being paired*"), social benefits ("*It helped me get to know more people in class*", "*It helped me to develop my communication skills*") as well as potential disadvantages of pair programming ("*I ended up doing most of the work*", "*Working in pairs made me less motivated to get the lab done*").

Along with responses from this questionnaire, information on lecture and lab attendance, as well as CA and examination results, were collated for both the individual and pair programming modules.

3. Results.

3.1 Comparisons in attendance and performance.

Table 1 below outlines the overall mean scores for attendance, assessment and failure rate across the two modules. A series of dependent t-tests revealed no significant differences between the two groups on any of these measures (p > .05). On the surface it would appear that the pair programming intervention had no effect on these outcomes, however further analysis revealed some interesting trends.

Table 1: Comparison Between Two Semesters

	Individual (pre- intervention)	Pair Programming (intervention)
Lecture attendance (mean)	66.7% (SD = 24.7%)	64.4% (SD = 31.7%)
Lab attendance (mean)	79.9% (SD= 24.0%)	76.1% (SD = 30.2%)
Exam result (mean)	52.5% (SD = 18.3%)	58.2% (SD = 20.3%)
Continuous assessment (mean)	69.6% (SD = 24%)	71.1% (SD = 23.6%)
Overall failure rate	18.20%	19.20%

3.1.1 Gender differences in performance.

Out of the full cohort of students who sat both exams, 19 were female. Interestingly these students did significantly better on the second exam (pair programming) than on the first (individual programming), with an average increase of 9.7%. The exam grade for female students increased from 52.9% to 62.6%, $t(18) = 3.072$, $p = .007$. For the male students there was no significant difference in exam scores (57.4% for individual programming, 57.2% for pair programming module; $p > 0.05$).

To further analyse the difference between male and female students we ran a 2 way mixed ANOVA with the exams scores (semester 1 and 2) as the repeated measures factor and gender as the between participants factor. While there was no overall difference between the main effects of males and females, or (as highlighted above) between overall exam scores across the two semesters, there was a significant interaction between these variables, $F(1,84) = 6.323$, $p = .014$, indicating that gender affected the change in grade. Female students were significantly more likely to enhance their exam mark relative to male students following the introduction of pair programming. This effect may be associated with the perceived benefits of the pair programming intervention, which are outlined in more detail below.

3.2 Questionnaire analysis.

In total, 40 students (30 males and 10 females) completed the two questionnaires (pre and post intervention), attended labs, and completed both the semester 1 and semester 2 exams. Thus the remainder of the analyses reported below is based only on these 40 students. One weakness of these data which should be acknowledged is that they are skewed towards students with higher attendance: the average lecture and lab attendance for these 40 students was 81% and 91% respectively, as opposed to 52% and 62% for the class as a whole. All reported Spearman’s rho correlations have significance of $p < .05$.

3.2.1 Preliminary questionnaire.

Table 2 provides descriptive statistics for the key questions on the preliminary questionnaire. As can be seen, students tended to rate their programming ability quite favorably, leading to consistently strong correlations between the block of questions relating to programming confidence. Responses to a number of these questions were also positively correlated with exam performance and continuous assessment marks in both modules.

Table 2: Mean Scores For Selection Of Questions On Preliminary Questionnaire On A 7 Point Scale.

Question	Mean	Standard deviation
I enjoy computer programming	5.3	1.42
I think I am good at computer programming	4.45	1.52
I often feel confused when I look at programming code	3.18	1.53
I would consider a career which involves computer programming	4.88	1.73
I enjoy doing computer programming labs	4.9	1.63
I am confident that I can write a computer program to solve a real world problem	4.52	1.69

The question of what age students were when they first started to program had no relationship with any other variable, neither did hours spent playing computer games or the number of friends students had who were also studying computer science. Leaving Certificate Maths grade was correlated with whether students liked programming (r = .449). Interestingly, the number of people a student reported as having helped them with programming was negatively correlated with how much they enjoyed labs in semester 1 (r = -.462). There was also a strong negative correlation between this response and exam score in semester 1 (r = -.525), indicating that those who asked for the most help were less likely to do well in the final exam.

3.2.2 Perceptions of pair programming.

There were a number of individual differences in the extent to which students felt they benefitted from pair programming, with the clearest distinction concerning the perceived effort invested. Specifically, students' appreciation of the pair programming paradigm was negatively correlated with whether they reported doing all of the work (r = -.423). Students who felt they did most of the work were also less likely to report learning from pair programming (r = -.511), although these students were more likely to enjoy programming as measured in the initial questionnaire (r = .492) and were also more likely to perform better in the exams (r = .435).

Conversely, students that indicated they enjoyed learning in pairs were more likely to report having learned from the experience (r = .476), but less likely to enjoy programming in general (r = -.496), less likely to enjoy labs (r = -.502) and less likely to report initial confidence in programming (r = -.415), as established from the preliminary questionnaire data. Taken together with the above results, these findings suggest that students who enjoyed pair programming did not necessarily experience a significant positive increase in their exam scores.

Table 3 displays the key predictors of performance in relation to exam results in both modules. As can be seen, exam results in both modules were correlated with liking programming in general, enjoying labs and reporting confidence in programming ability. In semester 2 scores were negatively correlated with obtaining help and positively correlated with how much effort was expended in the pair programming labs. Those who did better in the semester 2 exam were less likely to report learning from pair programming, but, interestingly, were more likely to report benefits of improved communication skills.

Table 3: Relationship Between Key Measures And Exam Results. Correlations Highlighted Are Significant At The P <.01 Level (**) Or The P <.05 Level (*)

	Like programming	Enjoy labs	Confidence	Maths grade	Number of people asked for help	Do all the work	Learned from PP	Good Communication
Exam result S1	.514**	.513**	.506**	.494**	-.319	.252	-.235	.297
Exam result S2	.488**	.546**	.483**	.298	.525**	.435**	-.347*	.383*

Though not reported in the table above, we observed similar relationships between these measures and CA scores for semester 1 and 2.

3.2.3 Gender differences in perceptions.

In addition to gender differences in overall assessment results (see section 3.1.1), a comparison of male and female responses to questions in the pre and post questionnaires also revealed some interesting trends. Table 4 summarises differences in perceptions, both from the preliminary questionnaire and the final evaluation questionnaire. The data reveal that females were significantly more likely to report poor initial confidence with programming, despite there being no significant difference in mathematical ability and no significant difference in exam grades. Based on a series of independent t-tests, female students were significantly more likely to report that they had learned from pair programming, and significantly less likely to report doing all the work in the labs.

Table 4. Male And Female Responses To Questionnaires. Significant Gender Differences Based On Independent Sample T-tests (p <0.05) Are Highlighted.

Preliminary questionnaire	**Male**	**Female**
Lecture attendance (%)	78.5	88.0
Lab attendance (%)	91.2	90.0
'I enjoy computer programming'	5.3	4.5
'I think I am good at computer programming'	4.5	3.5
'I often feel confused when I look at programming code'	5.1	3.9
'I would consider a career which involves computer programming'	5.0	4.1
'I enjoy doing computer programming labs'	4.9	3.5
'I am confident that I can write a computer program to solve a real world problem'	7.7	3.7
Hours spent playing computer games per week	7.7	3.7
Leaving Certificate Maths grade (points)[1]	53.0	50.4

Final Questionnaire	**Male**	**Female**
'I liked pair programming'	4.3	4.9
'I ended up doing most of the work'	4.3	3.4
'I learned programming skills from being paired'	3.6	4.8
'It helped me get to know more people in the class'	5.6	5.7
'It helped me to develop my communication skills'	4.5	5.1

1 Leaving Cert points are based on the quantised percentage achieved on the exam, with the Higher Level paper permitting points in the range of 100 to 45 and the Lower Level paper permitting points in the range of 60 to 5.

Taken in conjunction with the improvement in exam performance in semester 2, these results suggest that pair programming was more beneficial for the female students in the class than for the male students, perhaps because they had lower confidence to start with.

4. General Discussion.

This study aimed to establish the effect that collaborative learning, operationalised through the technique of pair programming, would have on performance, attendance and perceptions of programming among a group of second year computer science students. Results revealed some interesting findings, which gives a critical insight into the relative advantages and disadvantages of collaborative programming. While there appeared to be no overall increases in attendance, assessment and exam performance when pair programming was employed (compared to individual programming), a more detailed analysis revealed a variety of individual differences which are important to consider when evaluating this technique. The most significant of these are discussed in more detail below.

4.1 Gender Differences.

In line with predictions, we observed a number of gender differences in both the perceived benefits and outcomes of the pair programming intervention. Most notably, the exam results of female students significantly increased following the pair programming labs, an effect which was not observed within the male sample. This observation might be explained by differing perceptions of pair programming between males and females. Although there were no gender differences in how much students enjoyed pair programming, female students reported that they learned significantly more from this technique. This finding is consistent with previous research that has shown significant benefits of pair programming for female students. For example, McDowell et al. (2006) found that the use of this paradigm resulted in a significant decrease in the gender gap in programming confidence, narrowing from an average of 11.6% to only 3.5%. In addition, McDowell et al. (2003) found that significantly more female students who worked in pairs went on to declare a computer science-related major than those who programmed on their own.

Beyond the differences revealed following the intervention, the fact that the females in our study reported lower levels of confidence in their programming ability than males raises some concerns. This gender gap in confidence is consistent with earlier studies (e.g. Redmond et al, 2013; Werner et al, 2004) and is a worrying trend, given that this difference was not associated with lower ability (as measured both by prior results in the first programming module and Leaving Cert maths grade). Lower feelings of self-efficacy might explain why females report greater learning from pair programming. For example, Carver et al. (2007) speculated that pair programming changes the atmosphere of the class, making it more supportive and less competitive, which may be more conducive for learning by female students. This has important implications for educators considering methods and techniques for enticing females into careers such as computing and other currently male dominated scientific subjects.

4.2 Differences in ability.

Beyond gender differences, analysis of student feedback revealed that, although weaker programmers appreciated pair programming, stronger programmers expressed lower levels of satisfaction by having to 'carry' weaker students (i.e. by "doing most of the work") and were less likely to report learning from their partners. This may be symptomatic of the way in which pairs were assigned in labs, raising the question of whether an alternative pairing method may have been more effective.

While we implemented a random pairing system in class, a better alternative may be to pair students by ability so as to encourage more reciprocal relationships. This view is supported by Braught, MacCormick and Wahls (2010) who found that students have poorer learning outcomes when paired randomly, rather than by ability. In particular, they found that lowest-quartile students who were paired by ability performed better than those who were paired randomly and those who worked alone. This observation, while being somewhat counterintuitive, refutes the idea that weaker programmers can learn more from stronger programmers: rather, weak students perform better when paired with other weak students. Braught et al. (2010) speculate that the reason for this effect is that students paired by ability are more likely to be compatible, and more likely to interact collaboratively, leading to deeper learning. For mismatched pairs, stronger students are likely to take over and simply give directions, meaning that weaker students are reduced to the role of observers, and do not experience the process of resolving programming problems.

One strategy for pairing students in this way would be to ask them to rate their own programming ability and assign partners based on these ratings (Thomas, Ratcliffe, & Robertson, 2003). For example, Carver et al. (2007) split the class into three divisions of ability, and randomly paired students within each division. The system worked well, insofar as students agreed that their skills were well matched and they got along with their partner (Carver et al., 2007).

4.3 Overall evaluation.

Caution should be exercised in interpreting the results of this study, given both the small sample size (under 50% of the class completed all assessments and both feedback questionnaires) and also the lack of a control group. It cannot be known for certain that the observed differences were related to the introduction of pair programming per se. For example, in relation to gender differences, it might simply be the case that female students preferred the material covered in the second semester than that from the first semester. Further study is required to resolve these issues.

It is also important to note that not all studies involving pair programming have been effective. For example, Somervell (2006) found that, not only did students dislike extreme pair programming, it also served to lower program quality and the marks students achieved. In another controlled study Lewis (2011) observed that, while there were no differences in performance between students working individually or in pairs, those using pair-programming actually took longer to complete the problems. These fundamental limitations of the pair programming technique may explain why our intervention had a negligible overall impact on student performance.

Another finding of our study was that students who received the most help were in fact less likely to enjoy programming and less likely to do well in the exam. It may be the case that the manner in which help is dispensed in class serves to undermine students' confidence in programming. Accordingly, we recommend that, if pair programming is used, clearer instruction should be given to students and demonstrators regarding how best to advise and assist other students. Preston (2005) notes that pair programming practitioners rarely provide feedback to students on co-operative skills. He recommends that students should be shown both positive and negative examples of co-operative behaviour and that lecturers should observe and provide feedback on how students interact in labs (Preston, 2006). Salleh et al. (2011) also note that studies should focus more on issues such as pair compatibility and how this interacts with pair programming as an effective pedagogical tool.

4.4 Conclusion.

While there is much research to suggest benefits of pair programming (e.g., McDowell et al, 2003; 2006; Preston, 2005; Shalleh et al, 2011; Williams & Upchurch, 2001) the current study has not provided firm evidence that this technique is universally beneficial. We found no clear advantage for paired, as opposed to individual, programming in terms of performance or attendance across the group has a whole. The results do, however, suggest that this technique may be of greater benefit to female students, as well as increasing perceived gains in learning for weaker students.

Beyond academic gains one clear positive aspect on which students agreed was that pair programming helped them to meet more people in the class. Even if pair programming does not significantly enhance exam grades for all students, the social aspect of meeting others may be an important factor when it comes to subsequent subject choice. If pair programming is to be used for this aim, our findings support the use of randomised pairings, but within ability-matched pools. Furthermore, in order for the full benefits of the paradigm to be realised, pair programming should be supplemented with instruction on how to collaborate most effectively.

5. References.

Alistair, C. & Williams, L. A. (2001). The costs and benefits of pair programming. In G. Succi & M. Marchesi (Eds.), *Extreme Programming Examined*, 223-247.

Assiter, A. (Ed.) (1995). *Transferable Skills in Higher Education.* London: Kogan Page.

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

Boud, D. (2001). I*ntroduction: Making the move to peer learning. In D. Boud, R. Cohen & J. Sampson (Eds.), Peer learning in higher education: Learning from and with each other*, 1-17. London: Kogan Page.

Boud, D., & Lee, A. (2005). 'Peer learning'as pedagogic discourse for research education 1. *Studies in Higher Education,* 30(5), 501-516.

Braught, G., Wahls, T. & Eby, M. (2010). The benefits of pairing by ability. In *SIGCSE'10*, 249-253.

Braught, G., Wahls, T., & Eby, L. M. (2011). The case for pair programming in the computer science classroom. *ACM Transactions on Computing Education (TOCE)*, 11(1), 2.

Carver, J.C., Henderson, L., He, L., Hodges, J.E. & Reese, D.S. (2007). Increased retention of early computer science and software engineering students using pair programming. *Conference on Software Engineering Education and Training,* 115-122.

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair Programming: Under What Conditions Is It Advantageous for Middle School Students? J*ournal of Research on Technology in Education*, 46(3), 277-296.

Furberg, A., Kluge, A., & Ludvigsen, S. (2013). Student sense making with science diagrams in a computer-based setting. *International Journal of Computer-Supported Collaborative Learning*, 8(1), 41-64.

Gallardo-Virgen, J. A., & DeVillar, R. A. (2011). Sharing, talking, and learning in the elementary school science classroom: Benefits of innovative design and collaborative learning in computer-integrated settings. *Computers in the Schools,* 28(4), 278-290.

Keppell, M., Au, E., Ma, A., & Chan, C. (2006). Peer learning and learning-oriented assessment in technology-enhanced environments. *Assessment & Evaluation in Higher Education*, 31(4), 453-464.

Hammond, J. A., Bithell, C. P., Jones, L., & Bidgood, P. (2010). A first year experience of student-directed peer-assisted learning. *Active Learning in Higher Education,* 11(3), 201-212.

Hawi, N. (2010). Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. *Computers & Education,* 54(4), 1127-1136.

Hwang, W. Y., Shadiev, R., Huang, Y. M., Cai, Y. T., Yang, Y. S., & Su, J. H. (2013). Effects of drag-and-response interaction mechanism of multi-touch operated tabletop technology on users' awareness and collaborative performance*. Computers & Education*, 67, 130-141.

Jackson, C. K., & Bruegmann, E. (2009). Teaching students and teaching each other: The importance of peer learning for teachers. A*merican Economic Journal: Applied Economics, 1(4), 85-108.*

Kaye, A. R. (2012). *Collaborative learning through computer conferencing: the Najaden papers.* Springer Publishing Company, Incorporated.

Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education,* 21(2), 105-134.

Linn, M.C. & Dalbey, J. (1989). Cognitive consequences of programming instruction. In E. Soloway & J.C. Sphorer (Eds.), *Studying the novice programmer,* 57-81. Hillsdale, NJ:Lawrence Erlbaum.

Maguire, P., & Maguire, R. (2013). Can Clickers Enhance Team Based Learning? Findings from a Computer Science Module. *AISHE-J: The All Ireland Journal of Teaching & Learning in Higher Education*, 5(3).

McDowell, C., Hanks, B., Werner, L. (2003). Experimenting with pair programming in the classroom. S*IGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '03)*, 60-64.

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

Mendes, E., Fakhri, L., and Luxton-Reilly, A. (2006) A replicated experiment of pair-programming in a 2nd-year software development and design computer science course. In *Proceedings of the 11th annual SIGCSE conference on Innovation and t echnology in computer science education (ITICSE '06),* 38(3), 108-112.

Menzies, V. J., & Nelson, K. J. (2012). Enhancing student success and retention: an institution-wide strategy for peer programs. In 1*5th International First Year in Higher Education Conference: New Horizons*, 26-29.

Miliszewska, I., & Tan, G. (2007). Befriending computer programming: A proposed approach to teaching introductory programming. *The Journal of Issues in Informing Science and Information Technology*, 4, 277-289.

Mooney, O., Patterson, V., O'Connor, M. & Chantler, A. (2010). *A study of progression in Irish higher education*. Higher Education Authority.

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 Thirty-Fourth Technical Symposium on Computer Science Education (SIGCSE 2003),* 259-362.

O'Donnell, A. M., Hmelo-Silver, C. E., & Erkens, G. (Eds.). (2013).*Collaborative learning, reasoning, and technolog*y. Routledge.

Preston, D. (2005). Pair programming as a model of collaborative learning: A review of the research. *Consortium for Computing Sciences in Colleges*, 39-45.

Preston, D. (2006). Using collaborative learning research to enhance pair programming pedagogy. *ACM SIGITE Newsletter,* 3(1), 16-21.

Redmond, K., Evans, S., & Sahami, M. (2013). A large-scale quantitative study of women in computer science at stanford university. In *Proceedings of the 44th ACM technical symposium on Computer Science Education* (pp. 439-444). ACM.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education* 13(2), 137-172.

Salleh, N., Mendes, E., & Grundy, J. (2011). Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review. *Software Engineering, IEEE Transactions on,* 37(4), 509-525.

Somervell, J. (2006) Pair programming: Not for everyone? In*ternational Conference on Frontiers in Education: Computer Science and Computer Engineering,* 303-307.

Staarman, J. K., Krol, K., & Meijden, H. V. D. (2005). Peer Interaction in Three Collaborative Learning Environments. *Journal of Classroom Interaction,* 40(1), 29-39.

Tan, S. C., Yeo, A. C. J., & Lim, W. Y. (2005). Changing epistemology of science learning through inquiry with computer-supported collaborative learning. *Journal of Computers in Mathematics and Science Teaching,* 24(4), 367-386.

Thomas, L., Ratcliffe, M., & Robertson, A. (2003). Code warriors and code-a-phobes: A study in attitude and pair programming. P*roceedings of the Thirty-Fourth Technical Symposium on Computer Science Education (SIGCSE 2003)*, 363-367.

Topping, K. J. (2005). Trends in peer learning. *Educational Psychology,* 25(6), 631-645.

Traynor, D., & Gibson, P. (2004). Towards the development of a cognitive model of programming: a software engineering approach. In *Proceedings of the 16th Workshop of Psychology of Programming Interest Group*.

Tsai, W. T., Li, W., Elston, J., & Chen, Y. (2011). Collaborative learning using wiki web sites for computer science undergraduate education: A case study. *Education, IEEE Transactions on,* 54(1), 114-124.

Werner, L., Hanks, B. & McDowell, C. (2004). Pair programming helps female computer science students persist. *ACM Journal of Educational Resources in Computing,* 4(1).

Willey, K., & Gardner, A. (2010). Investigating the capacity of self and peer assessment activities to engage students and promote learning. *European Journal of Engineering Education*, 35(4), 429-443.

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

Williams, L. A. & Kessler, R. (2003). *Pair Programming Illuminated*. Addison-Wesley.

Williams, L. & Upchurch, R. (2001) In support of student pair programming, *SIGCSE Conference on Computer Science Education*, 327-331.

Yoon, J., & Brice, L. (2011). Water Project: Computer-Supported Collaborative E-Learning Model for Integrating Science and Social Studies. C*ontemporary Educational Technology*, 2(3).

Zweben, S. (2011). *Computing degree and enrollment trends.* Computing Research Association.