Investigating the Use of Pair Programming for Teaching Data Structures and Algorithms

Phil Maguire
NUI Maynooth

Rebecca Maguire
Dublin Business School

Patrick Marshall
NUI Maynooth

Abstract

Incoming university students who have not previously studied computer programming often find it a challenging subject, leading to high failure rates (Williams & Upchurch, 2001). As a result, enrolment in computer science courses is declining (Carver et al., 2007), with the participation of female students being particularly affected (Werner, Hanks & McDowell, 2004). Research has suggested that the lack of a formalized structure for collaborative learning may be one of the factors responsible for students' negative impressions of computer science (Werner et al., 2004). In this study we investigated whether the use of pair programming in labs would facilitate peer learning and enhance students' confidence in their programming ability. The hypothesis motivating this intervention was that the more experienced programmers would transmit some of their knowledge to the weaker students and that the class as a whole would benefit from having the support of a partner to identify problem solving strategies and to resolve coding bugs. Results showed that the intervention was generally well received, although the weaker programmers were more positive about it than the stronger ones. Students that reported learning from pair programming were less likely to enjoy programming (r = -.496), less likely to enjoy labs (r = -.502), more likely to struggle with understanding lab material (r = .561) and more likely to report a lack of confidence in programming (r = -.415). Although there was no significant increase in final exam grades for male students, there was a significant 9.7% increase for female students. The most frequently reported positive feature of pair programming was that it allowed students to meet more people in the class.

1. Introduction

Declining enrollment is a common problem faced by Computer Science (CS) departments in recent years (Carver et al., 2007). Students often find introductory computer programming very challenging, so much so that typically a quarter of students drop out and many others perform poorly (Williams & Upchurch, 2001). The high failure rates associated with programming modules may discourage students from choosing to study CS (Bennedsen & Caspersen, 2007).

Research in the software domain makes it clear that studying instructional text is not a sufficient basis for students to learn how to write a program (Williams & Upchurch, 2001). Textbooks and lecture material in CS are often heavy on declarative knowledge (i.e. facts), with particular emphasis on the features of the programming language and how to use them (Robins, Rountree & Rountree, 2003). In contrast, programming requires procedural knowledge and is therefore best learned through practice and experience (Traynor & Gibson, 2004). It has been proposed that the failure to develop appropriate procedural knowledge in programming modules means that the majority of students make very limited progress (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 the situation (Miliszewska & Tan, 2007).

One issue which may be exacerbating 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 et al., 2003). McDowell, Werner, Bullock & Fernald (2006) propose that this 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).

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 and only 1.8% of women versus 9.3% of men intend to pursue computer programming careers. The perception of CS 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 CS (Werner et al., 2004).

2. Collaborative Learning in the Teaching of Programming

A considerable volume of research extols the virtues of peer learning. Boud (2001) points out that, 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).

Pair programming is a novel collaborative paradigm which is growing in popularity in the computer science industry. 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 to review 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; Williams & Upchurch, 2001). 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). Pair programming contextualises the learning activity and allows students to get feedback on their activity that increases their ability to develop monitoring systems for their own learning activities (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 have been 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 (59.5% versus 22.2% for women and 74% versus 47.2% for men).

Much of the research on pair programming as a pedagogical technique has focused on the teaching of introductory programming to first years, with fewer studies investigating its use for more advanced programming (Mendes, Al-Fakhri & Luxton-Reilly, 2006). In this study we describe the outcomes of using a collaborative pair based paradigm for the teaching of a second year computer science course in data structures and algorithms.

3. Method

This study was carried out with a second year computer science class in NUI Maynooth taking two consecutive modules in data structures and algorithms (CS210 and CS211). The continuity of the student cohort through the two modules allowed us to analyse the outcomes of the pedagogical intervention, with individual programming used in the first semester and pair programming used in the second semester.

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 every weekly laboratory session. Demonstrators spent the last few minutes of these weekly two-hour sessions reviewing students' code and assigning marks.

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). However, this approach may work better for first year practicals where the labs are quite simple and typically involve only a small number of lines of code which fit on a single screen. Many of the labs for 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'. Not all studies involving pair programming have been effective. For example, Somervell (2006) found that, not only did students dislike extreme pair programming, but it also served to lower program quality and the marks students achieved. 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.

According to Preston (2005), a critical feature of successful collaborative learning is positive interdependence. Students are considered interdependent when they must 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. 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. 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.

Another important consideration was how to pair up the students. 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.

Prior to beginning the pair programming intervention, a preliminary questionnaire was handed out in lectures to obtain information on students' programming ability and confidence (see examples in Table 1). At the end of the semester a final questionnaire was circulated to get feedback from students regarding their experience of pair programming. Students put their names on the questionnaire so that these data could be correlated with lecture and lab attendance and CA and exam results for both CS210 and CS211.

4. Results

A total of 99 students registered for the final CS211 exam, and 90 of them sat the exam. Of the full cohort, the average lecture attendance was 64%, while the average lab attendance was 76%. In the previous CS210 module, which used individual programming, the average lecture attendance was 67% while the average lab attendance was 79%. There was no correlation between attendance and exam performance. For the CS211 module, the module mean was 56.9% with a CA mean of 71.1%. The overall failure rate was 19.2%. For the CS210 module, the module mean was 54.9% while the CA mean was 69.6%. The overall failure rate was 18.2%. There were no significant differences in results between the two modules ($p > .05$).

There were a total of 40 students that completed the two questionnaires, attended labs and completed both the CS210 and CS211 exams. The correlation analyses reported below are based on these 40 students. It is worth noting that because the questionnaires were handed out in lectures, the response data is skewed towards students with higher attendance: the average lecture attendance for the 40 students was 81% while the average lab attendance was 91%. Because of the high number of correlations carried out, only correlations with significance of $p < .01$ are reported.

There were consistently strong correlations between responses to a block of questions from the preliminary questionnaire relating to programming confidence: for example, do you enjoy computer programming, do you think you are good at programming, would you consider a career involving programming, are you confident that you could write a program to solve a real world problem. These responses were also consistently correlated with exam performance and continuous assessment marks.

The question of what age student were when they first started to program had no correlation 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) and their exam scores (r = .494). It was also correlated with reported improvement in communication skills (r = .427), indicating that more mathematically minded students may have benefitted to a greater degree from the social interaction provided by pair programming. Interestingly, the number of people a student reported as having helped them with programming was negatively correlated with how much they enjoyed labs (r = -.462). There was also a strong negative correlation with exam score (r = -.525), indicating that those who asked for the most help were less likely to do well in the final exam.

Students appreciation of the pair programming paradigm was negatively correlated with whether they reported doing all of the work (r = -.423) and positively correlated with whether they felt they had learned from the experience (r = .476). Students that reported doing all the work were more likely to enjoy programming (r = .492), less likely to learn from pair programming (r = -.511) and more likely to perform better in the exams (r = .435). Students that reported learning from pair programming were less likely to enjoy programming in general (r = -.496), less likely to enjoy labs (r = -.502), less likely to report a lack of confidence in programming (r = -.415), more likely to enjoy pair programming (r = .476) and less likely to report doing all of the work (r = -.511).

Exam results in both modules were correlated with liking programming in general (r = .488), enjoying labs (r = .546), reporting confidence in programming ability (r = .483) and negatively correlated with obtaining help (r = -.525). CA scores were correlated with hours spent programming (r = .541) and strongly correlated with exam performance (r = .621). The highest correlation of all was between students' overall CS210 result and their CS211 result (r = .754).

The difference between students' CS210 and CS211 exam result was uncorrelated with any other measure. However, the difference between the CA marks in CS210 and CS211 was correlated with students' response s to whether they felt confused by the lab exercises (r = .564). That is, students initially reporting feeling confused by the lab material were more likely to increase their CA mark by programming in pairs.

4.1 Analysis of Effects by Gender

Previous research on pair programming has shown significant benefits to 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 CS-related major than those who programmed on their own. Carver et al. (2007) speculated that a reason for this might be because paired programming changes the atmosphere of the class, making it more supportive and less competitive.

There were 21 female students in a class of 99, of which 19 sat the final exam. Female students did significantly better on the CS211 exam than the CS210 exam, with an average increase of 9.7%.The exam grade for female students increased from 52.9% to 62.6%, $t(1,18) = 3.072$; $p = .007$. Of the 78 male students, 71 attended labs, and of these 67 sat the final exam. For male students, there was no significant difference in exam scores (57.4% for CS210, 57.2% for CS211; $p > 0.05$). To analyse the difference between male and female students we ran a repeated measures ANOVA with CS210 and CS211 exams scores as the repeated measure and gender as a between participants measure. There was a significant interaction, $F(1,84) = 6.323$, $p = .014$, indicating that gender affected the change in grade. Specifically, female students were significantly more likely to enhance their exam mark relative to male students following the introduction of pair programming.

Caution should be exercised in interpreting these results, due to the lack of a control group. It is not certain that these differences are related to the introduction of pair programming per se. For example, it might simply be the case that female students preferred the material covered in the second semester over that from the first semester. Further study is required to resolve these issues.

A comparison of male and female responses to the questionnaires is given below in Table 1.

Significant differences are highlighted. Responses are on a Likert scale from 1 to 7, with 7 the most positive response.

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 am good at computer programming"	4.5	3.5
"I would consider a career which involves computer programming"	5.1	3.9
"I enjoy computer programming labs"	5.0	4.1
Hours spent playing computer games per week	7.7	3.7
Number of friends studying computer science	5	5.1
Leaving Cert Maths grade (points)	53	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 to get to know more people in the class"	5.6	5.7
"It helped me to develop my communication skills"	4.5	5.1

Table 1. Male and female responses to questionnaires

Women in the class were significantly more likely to report poor confidence with programming, despite there being no significant difference in mathematical ability and no significant difference in exam grades. 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. These results suggest that pair programming was more beneficial for the female students in the class than for the male students, perhaps because the women had lower confidence to start with.

In sum, the findings of this study highlight advantages and disadvantages of collaborative programming. Experienced programmers express lower levels of satisfaction by having to 'carry' weaker students and are less likely to report learning from their partners. Weaker programmers on the other hand appear to appreciate pair programming and improve their CA scores. This

study has illustrated that females have lower confidence in programming and are more likely to report learning from pair programming. One positive aspect on which students agree is that pair programming helps them to meet more people in the class, with 80% of students responding positively to this question (i.e. giving a rating of 5 or more). Even if pair programming does not significantly enhance exam grades for all students, the social aspect of meeting people in the class may be an important factor when it comes to subsequent subject choice.

5. General Discussion

The findings of this study were mixed. Although weaker programmers appreciated pair programming, the stronger programmers were less favourable and did not appear to benefit. We recommend that students should be paired by ability so that pairs are more likely to involve reciprocal relationships. Braught, MacCormick and Wahls (2010) found that students have poorer learning outcomes when paired randomly as opposed to 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 refutes the idea that weaker programmers can learn more from stronger programmers: weak students perform better when paired with other weak students. Braught et al. (2010) speculate that the reason for this 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 appropriately is to ask them to rate their own programming ability and assign partners based on these ratings (Thomas, Ratcliffe & Robertson, 2003). In light of our finding that meeting new people in the class is one of the most positive aspects of pair programming, we recommend continuing to randomise pairings within ability-matched pools. For example, Carver et al. (2007) split the class into three divisions of ability, and randomly pairing students within each division. The system worked well in that students agreed that skills were well matched and they got along with their partner (Carver et al., 2007).

Another finding of our study was that students who received the most help were less likely to enjoy programming and less likely to do well in the exam. This indicates that the

manner in which help is dispensed in the class may serve to undermine students' confidence in programming, rather than enhance it. Accordingly, we recommend that, if pair programming is used, some 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).

In conclusion, our study has shown that pair programming helps weaker programmers, particularly female students, and that it facilitates social interaction. We recommend that in order for the benefits of this paradigm to be more evenly distributed among the class, students should be paired by ability and given some feedback on how to collaborate more effectively.

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). Introduction: 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.

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

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.

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.

McDowell, C., Hanks, B., Werner, L. (2003). Experimenting with pair programming in the classroom. *SIGCSE 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 technology in computer science education (ITICSE '06), 38*(3), 108-112.

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.

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.

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.

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

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

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

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*.

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

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.