Investigating Pair-Programming in a 2nd-year Software Development and Design Computer Science Course

Emilia Mendes
Computer Science Department
The University of Auckland, New Zealand
emilia@cs.auckland.ac.nz

Lubna Basil Al-Fakhri
Computer Science Department
The University of Auckland, New Zealand
lubna@ww.co.nz

Andrew Luxton-Reilly
Computer Science Department
The University of Auckland, New Zealand
andrew@cs.auckland.ac.nz

ABSTRACT

This paper presents the results of a pair programming experiment conducted at the University of Auckland (NZ) during the first semester of 2004. It involved 300 second year Computer Science students attending a software design and construction course. We investigated similar issues to those reported in [26] and employed a subset of the questionnaires used by Laurie Williams et al. on the experiments presented in [26]. Our results support the use of pair programming as an effective programming/design learning technique.

Categories & Subject Descriptors

k.3.2 [**Computer and Information Science Education**]: Computer Science Education.

General Terms

Experimentation, measurement.

Keywords

CS2, pair programming, collaboration, software design.

1 INTRODUCTION

Anecdotal evidence from software companies suggest that software developers are often required to work collaboratively throughout their professional life [14], therefore companies expect that future software developers receive such training during their undergraduate education [3].

Pair programming has, over the last five years, been investigated as a promising technique to provide such skills, apart from other cited benefits.

Williams et al. [26] define pair programming (PP) as follows:

"In pair programming, two programmers jointly produce one artifact (design, algorithm, code). The two programmers are like a unified, intelligent organism working with one mind, responsible for every aspect of this artifact. One partner, the driver, controls the pencil, mouse, or keyboard and writes the code. The other partner continuously and actively observes the driver's work, watching for defects, thinking of alternatives, looking up resources, and considering strategic implications. The partners deliberately switch roles periodically. Both are equal, active participants in the process at all times and wholly share the ownership of the work product, whether it is a morning's effort or an entire project."

The benefits of pair programming are reported to include improved quality, teamwork, communication, retention, confidence, comprehension and learning [1],[4],[7],[13],[14], [17],[24],[25], and reduced workloads for the teaching staff (there are half as many projects to grade) [3],[20]. However, disadvantages have also been suggested, including students schedule issues [1],[3], pair incompatibility [1],[12],[13], and unequal participation [18]. A history of pair programming is given in [25].

Williams and Kessler [23] suggest several reasons of why pair programming works, such as the collective ownership of all that is produced, both partners being engaged in the process contributing to a successful outcome, keeping partners much more focused on the task at hand, helping improve each other's skills. Gehringer [6] recommends that pair programming be used even in courses where programming is not taught.

Most studies on pair programming investigate subjects collaboration sharing the same computer/desk/paper. However recently other studies also looked into the impact of distribution to the effectiveness of pair programming [2],[15]. In addition, several pair programming experiments have been employed at introductory courses, and a few at more advanced computer science courses [6],[17]. Only one of the studies in our literature review [4] has used second year students as subjects of a pair programming experiment.

Despite positive results, research findings are at times contradictory [5] (e.g. not all or most studies report statistically significant results in favour of pair programming). Padberg and Muller [16] analysed the cost and benefit of pair programming using an economic model for the business value of a development project. Their findings suggest that "when time to market is the decisive factor and programmer pairs are much faster than single

ITiCSE'05, June 27-29, Lisbon, Portugal.

developers, pair programming can increase the value of a project, but there also are realistic scenarios where the opposite is true."

This paper presents the results of a pair programming experiment conducted at the University of Auckland (NZ) during the first semester of 2004. The experiment's sample population was of 300 second year Computer Science students attending a software design and construction course. Our contribution is therefore to add empirical evidence to the current body of knowledge on pair programming regarding this technique's usefulness or not for improving educational outcomes and enjoyment of students.

We investigated similar issues to those reported in [26], and in particular at whether pair programming improves learning and the enjoyment of those students who participate in collaborative activities. We employed a subset of the questionnaires used by Laurie Williams et al. on the experiments presented in [26]. The quality of the code produced through the pair-programming process was not considered important to this study. The hypotheses we investigated were as follows (note, only the alternative hypotheses are presented here for shortage of space):

H1 – An equal or higher percentage of students in paired labs will complete the course with a grade of C- or better compared to solo programmers.

H2 – Students in paired labs will earn exam scores equal to or higher than solo programming students.

H3 – Students in paired labs will earn assignment scores equal to or higher than solo programming students.

H4 – Students in paired labs will earn test scores equal to or higher than solo programming students.

H5 – Students in paired labs enjoy pair programming and will have a positive attitude towards collaborative programming settings.

Our results support all alternative hypotheses, providing additional empirical evidence of the benefits of pair programming for learning.

Section 2 provides a detailed summary of the pair programming literature, followed by Section 3 where our experiment is described. Finally conclusions and comments on future work are given in Section 4.

2 PREVIOUS STUDIES

A summary of the pair programming literature is given in Table 1, where references were obtained from the ACM and IEEE databases, using as search criteria "pair programming" on titles and abstracts. We obtained 29 references however Table 1 only presents those that describe experiments/case studies on pair programming, thus reducing the number to 17. We have organised their findings using the same framework suggested by Gallis et al. [5]. This framework has been proposed to support empirical studies and meta-analysis for developing theories about pair programming. Further support for pair programming empirical studies is also given in [10] where authors propose a cognitive model as an approach to analysing empirical pair programming experiments.

Various methods have been implemented for forming pairs. McDowell, et. al [13], allow students to choose their own partners, similar trend observed in [25],[21]. Nagappan, et. al [14], used a software program to make random partner assignments. Thomas, et al. [18], had students rate their programming abilities and assigned partners based on these ratings. Cliburn [3] assigned pairs by grouping students from different cultural/ethnic backgrounds or upper with lower classmen. Nagappan et al. [14] also discuss forming pairs based on personality profiles such as the Myer-Briggs personality tests and Katira et al. [9] suggest that "pair compatibility in beginning courses may increase if pairs are formed by joining students of dissimilar personality type".

In numerous studies [14],[18],[3],[4] programmers were assigned new partners throughout the semester. However, despite evidence of the benefits of pair rotation [17] many studies have not done so [1],[13],[25],[22],[21].

In terms of experimental design five different choices have been identified: i) comparison between paired and solo students using separate groups and over the same semester/sessions [6],[12],[13],[14]; ii) comparison between paired and solo using same group of students over the same semester/session [4]; iii) alternating between paired and solo students over several terms/sessions [7],[11]; iv) use of only paired students [3],[9],[17],[19].

Some studies have kept the experimental unit fixed [7] while others provided additional or different assignments to paired students (see [26] for a list of such studies).

Table 1 – Literature Review

Authors	Type of study	Sub.	N	Task	Duration	SP?	Independent variable(s)	Main dependent variables (metrics)
Williams [20] [22]	Case study	Stud.	20	Web site development using CSP	Eleven-week semester	Yes	Evaluation of PP to gather experience with the process.	View of programming using CSP (qualitative assessment)
Williams et al. [25] Williams [21]	Exper.	Stud.	41	Four assignments	Six weeks	Yes	PSP (13 solo) versus CSP (14 pairs)	Total work time, productivity and programming quality
Haungs [8]	Case study	Prof.	2	C3 system	Unk.	Yes	Investigate use of PP	Information and Knowledge transfer (qualitative assessment)
Bevan et al. [1]	Case study	Stud.	Unk.	Nine assignments	10 weeks	Yes	Evaluation of PP's effectiveness as a teaching technique and effect on student retention.	Total work time, satisfaction with partner, amount of time worked alone.
McDowell et al. [13]	Exper.	Stud.		Five assignments	Two sessions of academic year	Yes	Solo (141) versus pairs (86 pairs).	Quality – score on programming assignment (functionality and readability), Learning effect – score on final exam.

Gehringer [6]	Exper.	Stud.	96	Three assignments	Academic semester	No	Solo (19,29,17) versus pairs (41, 23, 28 pairs)	Quality – score on assignment.
Cliburn [3]	Case studies	Stud.	14, 8, 17	Five assignments	Academic semester	Yes	All paired	Learning effect - Final course grade, PP enjoyment (qualitative survey)
McDowell et al. [12]	Expers.	Stud.	95, 19, 102	Unk.	Three months	Yes	Solo (47, 5,44) versus pairs (22, 7, 29 pairs)	Qualitative assessment of program quality, Learning effect – score on final exam.
Nagappan et al. [14]	Expers.	Stud.	199, 502	Three programming projects	Academic semester	No	Solo (69, 102) versus pairs (22, 140 pairs)	Learning effect – course, assignments and exam scores, attitude towards PP (qualitative)
DeClue [4]	Exper.	Stud.	24	Six assignments	Academic semester	No	Solo (22) and same 22 paired for 6 weeks.	Students attitude towards PP and role of PP on code/design quality (qualitative survey)
McDowell et al. [11]	Expers.	Stud.	552	Five or four assignments	Four sections, 3 months each.	Yes	Solo (148) in Spring section pairs (202 pairs) in Fall and Winter sections.	Student success – final course grade and exam attendance. Gender completion rate – final course grade, Course performance - score on final exam and program quality.
VanDeGrift [19]	Exper.	Stud.	546	Three projects and corresponding reports	Academic semester	Yes	All paired	Students processes – subjective assessment of reports, students perceptions on PP and written reports (survey)
Srikanth et al. [17]	Exper.	Stud.	270, 140	Five assignments	Academic semesters	No	All paired	Students and teachers perceptions on pair rotation and students perceptions on peer evaluation (all surveys)
Katira et al. [9]	Exper.	Stud.	564	Up to six assignments	Academic semesters	Yes	All paired	Understand compatibility of pairs - survey
Hanks et al. [7]	Expers.	Stud.	112, 50	Five Assignments	Academic semesters	Unk.	Solo (112) and paired (25 pairs) at different semesters	Program quality – complexity, length, subjective metrics, Students confidence and satisfaction -

Unk. – Unknown Exper. – Experiment Expers. – Experiments Stud. – Students Sub. – Subjects SP? – Same Partner?

3 PAIR PROGRAMMING EXPERIMENT

To evaluate whether pair programming was beneficial for learning and students attitude towards pair programming we designed and conducted an experiment during the first semester of 2004, involving 300 Computer Science students attending a second-year software design and construction course..

3.1 Course Structure

The software design and construction course is structured within three parts[1]. Part I introduces software design using Unified Modeling Language, and testing techniques. Part II furthers Java object-oriented programming skills acquired from the first year courses. Part III introduces basic client-server programming concepts and skills using Java and MySQL database.

Our semester lasts for 12 weeks. Students attended lectures three times a week for 50 minutes, and attended closed lab sessions, led by two teaching assistants, once a week for 11 weeks, and lasting 90 minutes each. Three lecturers, including two authors, led the lectures. There were no specific in-lab assignments and attendance was not mandatory. Course assessment was based on three assignments (20%), one mid-semester test (15%) and one final exam(65%). Lab exercises, assignments, test and final exam were the same for all students.

3.2 Student Population

Of the 300 students enrolled, 74% intended to obtain a Bachelor of Science degree, 10% a Graduate Diploma of Science degree, and 8% a co-joint degree in Science and Commerce. 18% were females and 82% males.

3.3 Assignments

At the start of each part of the course students were given an assignment to do, with a four-week duration. All three assignments contributed the same towards the final grade, and are briefly presented below:

Part I assignment: Students had to draw Use case, Class, Instance and Interaction diagrams describing a small application aimed at keeping a database of its employees. No implementation had to be provided, solely the diagrams.

Part II assignment: Students had to convert a Card game (Solitaire) program written in Java from a procedural design to an object oriented design and document their implementation using Javadoc comments.

Part III assignment: Students had to implement a Java client application to record timing data spent by employees on software projects, where the data was stored on a MySQL database.

3.4 Experimental Design

The experiment's treatment was the pair programming technique and our control was not to use the pair programming technique. Experimental units were the exercises given in the 11 labs and experimental subjects were the students attending lab sessions. The dependent variables were the learning effect (amount learnt), measured using assignments, test and exam scores and course grades and students enjoyment towards pair programming, measured using a qualitative questionnaire. Independent variables were individual programming/task versus paired programming/task.

Due to timetabling issues and students taking different programmes of study we were unable to assign beforehand students to labs. During the first week of the term students had to sign up for one of the eight weekly labs offered unaware of which labs were paired and which were solo. At the end of the first week we randomly chose the labs that would be paired and randomly assigned pairs within each paired lab in order to minimise the effects of uncontrolled variables.

Students attending paired labs were randomly allocated to a different pair every four weeks. During each of the 90 minute lab sessions pairs had to swap roles every 20 minutes, reminded by teaching assistants. Both teaching assistants used the same lab

[1] More information on the course can be found at http://www.cs.auckland.ac.nz/compsci230s1c/

exercises. During the first paired labs students were provided explanations on the pair programming techniques based on guidelines provided by Laurie Williams.

Our sample size is of 300 students, where 114 paired and 186 worked solo.

In order to run this experiment with our second year students we had to obtain approval from the University's Human Ethics Committee. Their condition for approval was that all assignments, test and final exam be done individually in order to be fair to all students. We also share their concern since there are several issues that can affect experimental results that have not been fully investigated. Examples of such issues are as follows:

- When providing different and/or additional assignments do they have to differ only in amount of effort or in complexity as well? By how much?
- How can we be sure they are equivalent and that we are not biasing the experiment?

There were several factors (independent variables) that we were unable to control and that may have had a confounding effect on the results we obtained, e.g. "self-confidence level", gender, ethnic group, communication skills, average grades on first year programming courses. However, recent findings suggest that pairs tend to be highly compatible and successful if paired randomly [9].

3.5 Results

The results presented here are based on data gathered from assignments, test, final exam scores and course grades, and a pair rotation questionnaire which is the same used Laurie Williams et al. at North Carolina State University [26]. This questionnaire had to be answered by each paired student three times, at the end of each four-week block.

All statistical quantitative analysis was carried out using SPSS V10. Confidence limit was set at 0.05. The statistical significance of differences in success rates between paired and solo groups was checked using the Chi-square test, which allows for comparison of categorical data. All remaining statistical analyses based on scores were accomplished using the two-tailed Student's T-test.

Table 2 presents the success rate for paired and solo students. The Chi-square test showed a statistically significant difference in success rate between paired and solo students, favourable to paired students. This result supports the alternative hypothesis H1.

Table 2 – Success Rate between paired and solo students

Programming technique	Succeeded above C-	Failed below C-	Success Percentage	Grade Average
Paired	101	9	91.8%	67.61%
Solo	144	28	83.7%	63.25%

The two-tailed Student's T-test also showed statistically significant differences in scores between paired and solo students (favouring paired students) for all assignments, test and final exam, which is remarkable. These results provided support for alternative hypotheses H2 to H4. We believe that a possible explanation for these results may be that the majority of second-year students come from an Asian ethnic group, where it seems that group work is always favoured as opposed to individualism and competition[2].

Regarding students enjoyment with pair programming (see Figure 1), on a scale from 1 (hated it) to 10 (loved it), our results showed that 74% of answers fell between 6 and 10, i.e. above average, and 40% fell within the last three highest scale points (8 to 10). Therefore our results suggest that the majority liked it.

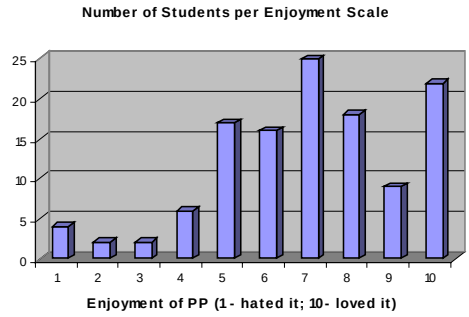

Figure 1 – Students enjoyment with pair programming

Finally, 54% of those students who paired would like to have pair programming as part of future Computer Science courses and 57% liked to swap pair every 4 weeks. Most students did not mind to pair with another student of different gender or ethnical group, and language skills also did not seem to be an issue.

4 CONCLUSIONS AND FUTURE WORK

We have reported on an experiment which compares the performance of students who engaged in pair-programming during voluntary laboratory sessions with those who worked solo during voluntary laboratory sessions. Although the laboratory sessions did not contribute directly to the final grade, the effects of being involved in a pair programming experience appear to have improved the quality of independent assignment work, examination scores, and percentage passing overall. Furthermore, the majority of students enjoyed the experience and would like to have pair-programming used in future courses.

We have shown that experience with pair programming has positive outcomes, even when it does not form part of the formal requirements of a course. These results provide support for the use of pair programming in the computer science curriculum.

Future work will verify these results by repeating the experiment in 2005 during the same course. We are interested in finding out more about the students who did not enjoy the pair programming experience. We also intend to investigate the long-term impact of pair programming by looking at the performance of students in subsequent courses, and compare the effects found in this experiment with those obtained from experiments in introductory courses.

ACKNOWLEDGMENTS

We would like to thank Laurie Williams for providing us with the questionnaires used in our experiment and for her encouragement, Jonathan Teutenberg and Santokh Singh for their help running the

[2] this data was obtained from a students statistics booklet published by the University of Auckland

labs and finally those students who volunteered their time and grades for our research.

REFERENCES

[1] J. Bevan, L. Werner, and C. McDowell, Guidelines for the use of pair programming in a freshman programming class, Proceedings 15th Conference on Software Engineering Education and Training, 25-27 Feb. 2002, pages 100 – 107, 2002.

[2] G. Canfora, A. Cimitile, and C.A. Visaggio, Lessons learned about distributed pair programming: what are the knowledge needs to address?, Proceedings Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 9-11 June 2003, pages 314 – 319, 2003.

[3] D.C. Cliburn, Experiences with pair programming at a small college, Journal of Computing Sciences in Colleges, October 2003, 19(1), 2003.

[4] T.H. DeClue, Pair programming and pair trading: effects on learning and motivation in a CS2 course, Journal of Computing Sciences in Colleges, May 2003, 18(5), 2003.

[5] H. Gallis, E. Arisholm, and T. Dyba, An initial framework for research on pair programming, Proceedings International Symposium on Empirical Software Engineering, 30 Sept.-1 Oct. 2003, pages 132 – 142, 2003.

[6] E.F. Gehringer, A pair-programming experiment in a non-programming course, Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, October 2003, 2003.

[7] B. Hanks, C. McDowell, D. Draper, and M. Krnjajic, Program quality with pair programming in CS1, Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education, June 2004, 2004.

[8] J. Haungs, Pair programming on the C3 project, Computer , 34(2), Feb 2001, pages 118 – 119, 2001.

[9] N. Katira, L. Williams, E. Wiebe, C. Miller, S. Balik, and E. Gehringer, Paired programming/ collaborative learning: On understanding compatibility of student pair programmers, Proceedings of the 35th SIGCSE technical symposium on Computer science education, March 2004, 2004.

[10] K.M. Lui, and K.C.C. Chan, A cognitive model for solo programming and pair programming, Proceedings of the Third IEEE International Conference on Cognitive Informatics, Aug. 16-17, 2004, pages 94 – 102, 2004.

[11] C. McDowell, L. Werner, H.F. Bullock, J. Fernald, The impact of pair programming on student performance, perception and persistence, Proceedings 25th International Conference on Software Engineering, 3-10 May 2003, pages 602 – 607, 2003.

[12] C. McDowell, B. Hanks, L. Werner, Experimenting with pair programming in the classroom, Proceedings of the 8th annual conference on Innovation and technology in computer science education, June 2003, ACM SIGCSE Bulletin, 35(3).

[13] C. McDowell, L. Werner, H. Bullock, and J. Fernald, The effects of pair-programming on performance in an introductory programming course, Proceedings of the 33rd SIGCSE technical symposium on Computer science education, February 2002, ACM SIGCSE Bulletin, 34(1), 2002.

[14] N. Nagappan, L. Williams, M. Ferzli, E. Wiebe, K. Yang, C. Miller, S. Balik, Improving the CS1 experience with pair programming, Proceedings of the 34th SIGCSE technical symposium on Computer science education, ACM SIGCSE Bulletin, January 2003, 35(1), 2003.

[15] H. Natsu, J. Favela, A.L. Moran, D. Decouchant, and A.M. Martinez-Enriquez, Distributed pair programming on the Web, Proceedings of the Fourth Mexican International Conference on Computer Science, 8-12 Sept. 2003, pages 81 – 88, 2003.

[16] F. Padberg, and M.M. Muller, Analyzing the cost and benefit of pair programming, Proceedings Ninth International Software Metrics Symposium, 3-5 Sept. 2003, pages 166 – 177, 2003.

[17] H. Srikanth, L. Williams, E. Wiebe, C. Miller, and S. Balik, On pair rotation in the computer science course, Proceedings 17th Conference on Software Engineering Education and Training, 1-3 March 2004, pages 144 – 149, 2004.

[18] L. Thomas, M. Ratcliffe, and A. Robertson, Code warriors and code-a-phobes: a study in attitude and pair programming, Proceedings of the 34th SIGCSE technical symposium on Computer science education, January 2003, ACM SIGCSE Bulletin, 35(1), 2003.

[19] T. VanDeGrift, Coupling pair programming and writing: learning about students' perceptions and processes, Proceedings of the 35th SIGCSE technical symposium on Computer science education, March 2004.

[20] L. Williams, But, isn't that cheating? [collaborative programming], Proceedings 29th Annual Frontiers in Education Conference, 2, 10-13 Nov. , pages 12B9/26 - 12B9/27 vol.2, 1999.

[21] L. Williams, Integrating pair programming into a software development process, Proceedings 14th Conference on Software Engineering Education and Training, 19-21 Feb 2001, pages 27 – 36, 2001.

[22] L. Williams, and R.R. Kessler, The effects of "pair-pressure" and "pair-learning" on software engineering education, Proceedings 13th Conference on Software Engineering Education & Training, 6-8 March 2000, pages 59 – 65, 2000.

[23] L.A. Williams, and R. Kessler, All I really need to know about pair programming I learned in kindergarten, Communications of the ACM, May 2000, 43(5), 2000.

[24] L. Williams, and R.L. Upchurch, In support of student pair-programming, Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, February 2001, ACM SIGCSE Bulletin, 33(1), 2001.

[25] L. Williams, R.R. Kessler, W. Cunningham, and R. Jeffries, Strengthening the case for pair programming, IEEE Software, 17(4), July-Aug. 2000, pages 19 – 25, 2000.

[26] L. Williams, C. McDowell, N. Nagappan, J. Fernald, and J. Werner, Building pair programming knowledge through a family of experiments, Proceedings International Symposium Empirical Software Engineering, 30 Sept.-1 Oct. 2003, pages 143 – 152, 2003.

Note. The data set can be made available to the reviewers for independent assessment of the statistical analyses presented in this paper but cannot be published for confidentiality reasons. Please contact Emilia Mendes at emilia@cs.auckland.ac.nz.