Experiences with Pair and Tri Programming in a Second Level Course

Maryam Purvis, Martin Purvis, Bastin Tony Roy Savarimuthu, Mark George, and Stephen Cranefield

Department of Information Science, University of Otago
P O Box 56, Dunedin, New Zealand
{tehrany,mpurvis,tonyr,mgeorge,scranefield}
@infoscience.otago.ac.nz

Abstract. In this paper we describe our experiences with multi-programming (pair and tri programming) in a second level course. The course, "Application Software Development" is a second year course, which has a heavy emphasis on java programming in the labs as well as the development of a full-fledged project. The objective of the course is to build an entire project that comprises of various software engineering activities that span across the semester. In general, we observe that multi-programming improves the students' ability in analytical thinking and communicating the conceptual ideas. It also raises certain issues when this approach is adopted in the educational context. In this paper we discuss some of these issues. Overall, multi-programming experience has been a rewarding experience for the students in spite of certain problems that were encountered.

1 Introduction

The concept of collaborative/cooperative learning [3] has been widely researched and advocated across the literature. The term "collaborative learning" refers to an instruction method in which students at various performance levels work together in small groups towards a common goal. The students are responsible for one another's learning as well as their own. Collaborative learning has proven effective in improving critical thinking of the students [5]. The concept of tri programming is inspired by concepts of collaborative learning and the pair programming [1, 8, 9, 10] activity associated with the Extreme Programming [14]. In tri programming the groups will have three people assigned to work together at a single computer to create software. Similar to pair programming, one person types and takes the initiative for the design of the particular code being developed at that time, while the other two catch minor errors and maintain a more strategic view of how the code fits into the rest of the system. The programmers swap roles as often as they like. All the three programmers should be involved simultaneously in all code development.

The group members will also interact with each other in discussing the design and implementation issues related to the programs that they are developing. In this process, they will reinforce each other's knowledge and work towards a common understanding. They also develop the communication skills that are required for their future careers.

R. Khosla et al. (Eds.): KES 2005, LNAI 3682, pp. 701–707, 2005.

It is believed that both pair and tri programming help both the experienced as well as novice programmers. The expert benefits by practicing his or her ability in explaining a particular idea. Often during this process the expert may identify certain errors in his or her own logic. At the same time the novice benefits by getting exposed to various problem-solving techniques and ideas and also by questioning the reasons behind certain design and implementation decisions.

2 Background

2.1 Motivation for Multi-programming

The motivations for tri programming are a) To provide more opportunities for interactions, b) To provide anonymous evaluation of team members and c) To lessen the likelihood of a dominant person taking over the whole work

2.2 Previous Work

Many researchers have been looking into collaborative programming. The term collaborative programming has been mostly used in the context of pair programming. Our work on multi-programming incorporates teams of three as well as teams of two. Some work has been done on teams comprising of more than two members such as tri programming [7]. Most of the work on group programming concentrates on an advanced programming course in which students tend to be more cooperative and mature, and also it focuses on division of tasks among team members as opposed to simultaneous development of a block of code.

The pair programming practice is associated with the Extreme Programming (XP) methodology. XP promotes a set of practices such as planning game, up-front testing, pair programming, code re-factoring and having on-site customer and so on [14]. Due to the success of XP process in certain applications, some researches have attempted to integrate these basic principles into the educational courses. Some have adopted the whole process [6] while others have tried a subset such as applying just pair programming [5, 11, 12].

The research done by Noble et. al [6], in applying some of the XP principles in their software engineering courses were reported to be successful. They were able to shift the focus from a document-centric approach to a more iterative project-planning and communication-centric approach. Others who have tried primarily the pair programming approach in the educational context [11, 12] have reported an improvement in terms of the quality of the code produced, increases in personal satisfaction and decreases in student drop-outs, in particular, in introductory courses. In addition, it has been reported that the students who participated in pair programming performed as well as students who did solo programming in the final exam [11, 12]. In our experience, we believe that, when participating in pair programming, the process of communicating and rationalizing the decisions made during the project development helps the students in mentally formulating the concepts in a better way and generally these students do better in their final exam. There are many factors involved in the success of the pair programming. Thomas et al. [13], report on the selection criteria. According to Thomas, the pairs with similar abilities and similar personalities tend to

work together the best. Other factors are how often the roles of the pair (driver, observer) should be swapped, how often the pairs should be rotated – (new pairs should be formed), how the work done by the pairs should be evaluated, and so on. In this paper we attempt to address these issues.

2.3 Class Statistics in 2003 and 2004

This paper concerns with the student data and feedback provided by the students for the years 2003 and 2004. The sample sizes were 144 and 110 respectively. In 2003 multi-programming (both pair and tri programming) was done and in 2004 pair programming was undertaken. The students had the option to go to solo programming if a problem such as scheduling was negatively impacting their work. In 2003 there were 29 groups of three, 16 groups of two and 25 solos. In 2004 there were 35 pairs and 40 solo programmers.

2.4 Evaluation of the Group Work

Each group submitted one copy of the project work. The project work contributes to 10% of the total mark for the paper. 80% of the project mark comes from marking the quality of the project submitted (this part of the mark is the same for all the group members); the remaining 20% of the project mark was determined from peer assessment.

2.5 Group Assessment

Assume that the three members in a team are A, B and C. While evaluating a student (say A) if two team members (B & C) are happy with the performance of the other team member (A), then A gets 2 points. If B is happy and C is not happy, then A gets 1 point and if both B and C are not happy about A's performance, then A gets 0 points. The pair programmer's work was assessed only based on what they handed in. This is because the lack of anonymity would put the team members on a difficult position in order to be objective.

3 Issues Related to Multi-programming

In this section, we point out some of the issues that came up while conducting pair and tri programming practices. Most of the results that have been reported are based on our observations and casual interactions with the students during the laboratory work and feedback that we have received from the students throughout our experiment.

3.1 Performance of Multi vs. Solo Programmers

Overall, multi programmers performed at least as well in the projects as the solo programmers. This has been reported in the literature widely [5, 11, 12]. Table 1 shows the average marks scored by multi programmers and solo programmers in two different phases of the project in 2003. It is observed that multi programmers perform better than the solo programmers.

Table 1. Average marks scored by multi and solo programmers in two different phases of the project

	Phase I	**Phase II**
Multi programmers	88.16	85.94
Solo programmers	74.82	57.5

3.2 Scheduling

Most students suggested that they had scheduling problems when the group size was 3. It is natural that when the size of the team increases it creates problems in reaching a common agreement on problem solving methodologies and also in agreeing to a common schedule. This has also been reported in the XP community [7].

3.3 Formation of Groups Based on Skills

In 2003, the groups were based on the previous performance of the students. The team was chosen in such a way that it comprised of a good, average and below average students. In 2004 the groups were based on voluntary pairing. It has been observed that, though the grouping was arranged based on the skill levels of the students, the students are not enthusiastic to work with team members that they do not know and expressed their reservations in the feedback. However, most of them realised the need for teamwork in real life projects with unknown team members. Still, most students preferred voluntary team formation over the assignment of team members according to their skill levels or previous performances.

3.4 Impact of Peer Pressure

Researchers have reported that students develop a positive peer pressure while programming with a partner [2, 5]. We also believe that peer pressure plays a constructive role in adhering to the deadlines and fulfilling commitments. Some students have mentioned that peer pressure forced them to learn better and prepare well ahead of labs and project meetings. Multi-programming imparts the sense of joint responsibility among the team members. This results in timely presence in project/lab meetings and during collaborative work sessions.

3.5 Performance of International Students on Multi-programming

It has been observed [4] that it is always ambiguous to measure the performance of international students as multi-cultural issues played a major role in their performance and also their written and communication skills. We have observed that, international students, especially Asian students had problems while communicating with their partners in the multi-programming. In a group consisting of only international students, the students often discussed their issues in English and the accent played an important role in the level of understanding between the students. If the group was composed of students belonging to the same nationality, they tended to discuss their ideas in their native language. In a group consisting of mixed nationalities (interna-

tional as well as native English speakers) the international students as well as the native English speakers had difficulty in explaining their views clearly to each other. Some international students felt that this approach resulted in better understanding of the concepts when their group mates explained it in English and also they considered this as a chance to improve their communication skills. On the other hand, most of the native English speakers found it hard to work with the international students, as they had to explain the students over and again till they understood. They considered this to be a waste of time.

3.6 Policies for Swapping the Roles in Multi-programming

Most of the students complained about one person being the driver all the time. It is normal to have totally different personalities in a group. Some students had quoted that some of their teammates had dominating personalities and their own voices were never heard within the team. So policies concerning swapping of roles must be developed.

4 Recommendations to Improve the Process

4.1 Forming Well Balanced Groups/Pairs

To encourage better interaction and cooperation between the team members, the students may select their own partners if they choose to, otherwise an effort should be made to form groups of people with similar skills as well as similar interests (this can be done by asking students to fill a brief set of questionnaire that identifies the students like or dislikes). Similar thoughts have also been expressed by McDowell et.al [11, 12] and Thomas et.al [13].

4.2 Student Assessment

Firstly, in order to improve the team sprit, the evaluation mechanism should be based on the quality of the project as opposed to the performance of an individual. This shifts the focus of students in creating a good quality project and may resolve some of the personal conflicts for the sake of creating good quality software. Secondly, the assessment criteria should be as closely related to the training provided in the course as possible in order to avoid confusion with regard to what is learnt and what is expected in the course. For example, when pair programming is adopted for the practical aspect of the course, the practical test should be also based on the pair programming. There would be other opportunities to evaluate the student understanding of the concepts covered in other course assessment forms, such as final exam and individual lab work that might be designed for learning the basic techniques and principles associated with the course. Thirdly, to apply pair programming in the classrooms, some modification might be needed to the normal process. For example, in the educational setting, we need to ensure that the students have had adequate opportunities to learn the required concepts and techniques - in particular, in a course that is taught at the first or the second year level. These students are still struggling to learn the basic ideas and need to practice on all aspects of the programming including both coding

and code reviewing. In the industrial setting, one assumes that the participants know the basic principles. So, to address this issue and ensure that the students get adequate practice in learning the basic programming technique, we propose the following process.

For a group of 3 people (assume A, B, C denote students with ascending order of capability), choose three problem sets (assume X, Y, Z denotes the problems). The problems have ascending difficulty levels. For the simple problem X, A is the driver and the rest of the team are observers. For the problem Y with medium difficulty, B will be the driver and the other team members act as observers. To solve the difficult problem Z, C plays the role of the driver while the other team members are observers. In the scenario described above, all the students are engaged, and each get a chance to play the roles associated with pair programming (drivers and observers). The problem sets are similar in the basic concepts; they can include some extensions to keep the more capable students interested while providing enough repetitions for the less able student. By this process we have introduced enough redundancy and repetition to ensure that the students have had adequate chance to learn the material. Also, there is a clear structure for the students to swap roles in order to experience the benefits of different roles associated with the pair programming.

4.3 Improving Communication Skills Through Swapping Team Members

In order to improve the communications skills and provide better chances for interacting with different types of students, the project can be partitioned into several subtasks. At the end of each task, the students may have the opportunity to swap teams. This way not only the students get to work with new partners, but they get to experience how to understand a piece of code written by another group and how to interface it and maintain it in order to add additional functionalities.

4.4 Motivating Students on Pair Programming

McDowell et. al [11, 12] have discussed the need for motivating students for pair programming. We agree with their viewpoint that students must be motivated and enough background information must be provided so that pair programming will be effective. To prepare and motivate the students for the process associated with the pair programming and the benefits gained, the students should be initially exposed to the appropriate literature in this topic and perhaps be given a chance to write a short document on their understanding of the process. As a part of this process, the students can get engaged in a simple task and outline some of the ground rules to be followed as well as reach an agreement on when and how the roles should be swapped, so that all participants benefit from different aspects of pair and tri programming.

5 Conclusions

As long as there is active participation by all the members of the group, multi-programming will be fruitful activity. Though it has been observed that most three-member groups faced scheduling problems, three people can generate a better discussion on rationalizing some of the programming decisions that are made. Most students

also felt that voluntary pairing should be preferred over non-voluntary pairing. We have also presented certain recommendations to improve the multi-programming method of programming. We believe most students benefit from pair and tri programming. Similar to the benefits of code review in the traditional software development process, pair programming subjects the project to constant code and design review. For this reason it has been reported that the quality of the code is generally improved. In addition, we believe that multi-programming tends to make students reflect more on the concepts and improves the analytical thinking ability of the students. In a way, it helps students to move toward a deeper learning style. It also facilitates knowledge sharing among students and improves the communication and interpersonal skills that are required to work effectively in a team environment. We are planning to formalize and adopt other XP practices in our courses. We have already incorporated some aspects of XP testing practice and in the future we will introduce other practices such as game planning, project iteration, and re-factoring.

References

1. Williams, L., R. R. Kessler, W. Cunningham, and R. Jeffries. "Strengthening the Case for Pair Programming." *IEEE Software* 17.4 (July/Aug. 2000): 19-25.
2. Williams, L., Wiebe, E., Yang, K., Ferzli, M., Miller, C., *In Support of Pair Programming in the Introductory Computer Science Course* Computer Science Education, 2002.
3. Antil, L., J. Jenkins, S. Wayne, and P. Vadasy. "Cooperative Learning: Prevalence, Conceptualizations, and the Relationship between Research and Practice." American educational research journal 35, no.3 (1997): 419-454.
4. George, P. G. (1994). The Effectiveness of Cooperative Learning Strategies in Multicultural University Classrooms. *J. of Excellence in College Teaching*, 5(1), 21-30.
5. Gokhale, A. (1995). Collaborative learning enhances critical thinking. Journal of Technology Education, Vol 7, no 1, Fall 1995
6. Noble, J, Marshall, Stuart, Marshall, Stephen, Biddle, R. Less Extreme Programming. ACE 2004: 217-226
7. TriProgramming: www.c2.com/cgi/wiki?TriProgramming, Accessed on 20th Feb, 2005.
8. Williams, Laurie, Robert R. Kessler, Ward Cunningham, and Ron Jeffries, Strengthening the Case for Pair-Programming, IEEE Software, July/Aug 2000.
9. Williams, Laurie and Kessler, Robert R., *All I Really Need to Know about Pair Programming I Learned In Kindergarten*, Communications of the ACM, May 2000.
10. Williams, Laurie and Kessler, Robert R. The Effects of Pair-Pressure and Pair-Learning on Software Engineering Education. Conference of Software Engg. Edu. and Training, 2000.
11. McDowell, Charlie, Linda Werner, Heather Bullock, and Julian Fernald, The Impact of Pair Programming on Student Performance, Perception, and Persistance, In Proc. of the 25th International Conference on Software Engineering (ICSE 2003), pp. 602 - 607, 2003.
12. Charlie McDowell, Brian Hanks, and Linda Werner, Experimenting with Pair Programming in the Classroom, Proceedings of the 8th Annual Conference on Innovation and Technology in Comp. Sci. Education (ITiCSE 2003), 2003, Thessaloniki, Greece.
13. Thomas, L., M. Ratcliffe, and A. Robertson, Code Warriors and Code-a-Phobes: A Study in Attitude and Pair Programming", Proceedings of SIGCSE 2003, pages 363-367, 2003.
14. Beck, K. & Fowler, M. (2000), Planning Extreme Programming, Addison-Wesley. Beck, K. *Extreme Programming Explained: Embracing Change*. Addison-Wesley, 2000.