Factors Affecting the Perceived Effectiveness of Pair Programming in Higher Education

Edgar Acosta Chaparro, Aybala Yuksel, Pablo Romero and Sallyann Bryant

IDEAS Lab, Dept. Informatics, University of Sussex
Brighton, BN1 9QH, UK
e.a.chaparro@sussex.ac.uk

Abstract. This paper reports the findings of a study conducted on postgraduate students of an Object Oriented Programming (OOP) course in which pair programming was applied as an educational technique. This study addressed the question *Why is pair programming sometimes ineffective?* The focus of the study was on exploring the factors that may affect the success of pair programming.

We employed a combination of data gathering techniques and triangulated them to analyze the data. We observed, recorded and interviewed students who pair programmed. They also completed questionnaires. There was evidence that matching by skill level and the task in hand are the main factors in the success of a pair programming session.

1. Introduction

In recent years, Object Oriented programming is increasingly encountering agreement in academic environments as a more appropriate strategy than the procedural paradigm to approach learning programming [1]. Important characteristics of OOP are its capacity to support well-structured programming, modularization and programming design [1-3]. These characteristics support the position of Meyer [4] who argued that OOP is cognitively more plausible because its characteristics more closely map structures of thought. Nevertheless, OOP is not easy to learn [3, 5, 6]. In this programming paradigm, students have to master both Object Oriented and procedural concepts. Thus, it is worth exploring new tools and methods that could facilitate the learning of OOP [1, 7].

Pair programming, the situation in which two programmers work side by side on the same piece of code, is a well-accredited approach to teaching programming. Students who practice pair programming have shown better results on graded assignments and more satisfaction/less frustration on doing course projects [8]. Flor and Hutchins[9], who analysed pair programming in a working environment, noted that this approach maximizes the space of solutions because it combines two different cognitive systems (each peer). Despite its benefits, studies [10, 11] have shown that sometimes pair

programming is irritating, exhausting and extremely inefficient.

In this paper we present a field study that explored aspects that could affect pair programming in an introductory Object-Oriented programming course. The course was part of a MSc conversion programme in computing.

The research question that guided our field work was:

a) Why is pair programming sometimes ineffective?

As we shall illustrate, observations and data collected during our study provided important findings into this question. For instance, skill level appeared to have a strong influence on pairs' collaboration. The following section talks about collaborative learning and pair programming as a background to our study. The methodology and techniques used in the study will be defined in the section 3. We will then present and discuss our findings.

2. Collaborative Learning

According to Social Constructivism [12, 13], learning is an action that occurs within a social context during the interaction between the learner and its interlocutor(s). Social Constructivism has tended to stress cooperation rather than conflict and emphasise learning as a process triggered by social interaction in the context of a dialogue (i.e. learner-learner). Because of the engagement in collaborative activities, individuals can master something that they could not do before the collaboration [12, 14, 15].

Collaborative learning involves grouping or pairing students to work together. Many conditions may affect the efficiency of collaborative learning. Dillenbourg et al. [16], in a study about efficiency in collaborative situations, listed group composition, group size and individual differences between group members as conditions that could affect collaboration. LeJeune [17] argued in favour of small-groups interactions. She suggested that group size should be between five and seven people and that grouping more than seven people may result in communication problems.

Skill level could also be an important condition for efficient collaboration [16]. Many studies have applied skill level as a criteria for forming pairs [18-20]. Thomas et al. [19] asked students to rate themselves on a scale of 1 to 9 in relation to their programming skill. Then, they paired students with both opposite and the same skill level. The results from this study showed that high skilled students enjoyed pair programming less when they paired with students with lower skill levels. In the same study it was found that high skill level students produce better work when they pair with students of the same skill level.

3. Pair Programming

Pair programming is not a new idea but dates at least from 1970 [21]. However, only recently eXtreme Programming has formalized it as a core practice of its methodology. Lately, pair programming has encountered agreement in academic environments as a promising strategy to approach learning programming. Pair programming is the situation in which two programmers work side by side, designing and coding the same algorithm. It is suggested that there are typically two roles in pair programming: the *driver* who controls both the computer keyboard and the mouse, and the *navigator* who examines the driver's work, offering advice, suggestions and corrections to both design and code [22]. According to Cockburn and Williams [23], who observed the method in academic environments, Pair Programming improves the quality of software design, reduces the deficiencies of the code, enhances technical skills, improves team communication and is considered to be more enjoyable for the participants.

Pair Programming has been used as a method for teaching programming in higher education [8, 24-26]. McDowell et al [25] for example noted that applying the method to first year students resulted in a greater percentage of students who successfully completed the course. Moreover, quantitative studies [8, 24-27] that compared the performances of pair programming students and solo students showed that the former were more likely to turn in solutions for their assignments, and such solutions were of higher quality.

On the other hand, some studies [10, 11, 20, 28] have suggested that it is not obvious that pair programming is better than solo programming. Tessem [11], for example, showed that some students found the experience irritating, inefficient and exhausting. Gittings et al [10] found very similar results in their study where participants described the experience with pair programming as demanding and sometimes frustrating. Moreover, VanDeGrift [20] showed that the students complained about working among people with different personalities and skill levels. However, as argued in [29] there is evidence to suggest that pair programming in some situations appears to be more engaging, useful and enjoyable.

In order to understand how some factors affect pair compatibility in students learning programming, Katira et al. [18] conducted a study focusing on self-esteem, skill level and personality. These three factors were selected based on the authors' previous experience with pair programming. In our study, different from the work in [18], we decided to conduct an exploratory study which started with as few initial preconceptions as possible.

4. Methodology

In our study, we used four different methods to gather data: participant observation, questionnaires, semi-structured interviews and field notes. The selection of these methods are the result of our analyses of previous studies and related literature [30, 31]. Each of the methods that we used had distinctive weaknesses. For instance, in interviews, the way questions are formulated might bias results. Rather than relying on the results from one method we triangulated them to overcome the weaknesses of a single method. Triangulation is the combination of multiple research methodologies that can be applied in both qualitative and quantitative studies [32]. Triangulation has been used to validate observations by considering the same factors via a number of different techniques. Denzin[32] reported that different methods and observers involve different interpretations to the research, therefore, triangulated analyses are most robust than those employing a single method.

In the autumn of 2004, Pair Programming was employed on an Object Oriented Programming postgraduate course. This was a 10 week course which was structured as one hour lecture and two hours laboratory sessions per week. Laboratory sessions started on week 3. There were in total 80 students enrolled in the course.

There were eight laboratory sessions in the course. In these sessions students were supposed to work on small programming exercises that covered tasks such as coding, debugging, program understanding and refactoring among others. In half of them students were asked to work in pairs, and in one of the remaining sessions students had the option to work in pairs or alone. It was expected that students could complete programming exercises either with a partner (in a pair programming session) or alone (in a solo session). In all sessions, students were paired with no selection criteria, however, throughout the end of the course partners were self-selected. Therefore, partners might be same or different for each session.

During the first laboratory session a brief explanation about pair programming was given. It stressed the characteristics of good collaboration and the different roles played in this relation (driver/navigator). Each pairing session covered a different activity: *program comprehension*, *debugging*, and *re-factoring*.

The findings reported here are based on the data collected from an exploratory study conducted by the lead author. Students were informed about our study and the majority of them (58 out of 80) agreed to participate on the understanding that any reporting of the results would maintain their anonymity.

4.1 Data Gathering Techniques

In our fieldwork, the lead author conducted the role of (volunteer) *demonstrator* in the course. He also sat on lectures, took notes and interacted with students before and after lectures. Occasionally, when the lecturer could not assign students to a pair (odd number of students) he also played the role of partner in a pair. In this case, he worked with his selected peer in solving the exercise.

In this study, we used a combination of data gathering techniques. The following sections describe these techniques.

Participant Observation

The lead author made extensive use of *participant observation*. He sat on lectures and occasionally participated as a member of a pair during the laboratory sessions.

Questionnaires

After each laboratory session where participants where asked to work in pairs (4 sessions), a *questionnaire* was applied in order to obtain their perceptions and experiences with pair programming. The questionnaires consisted of two main parts. The first part included questions about whether students liked or disliked pair programming and why. The second part included questions about their perception on learning when working in pairs. The complete questionnaire can be found in appendix A.

Semi-Structured Interviews

During the last week of the course we audio taped *semi-structured interviews* with 18 of the students. We discussed themes and issues perceived as important in our observations and field notes. We divided the interviews in three parts. First, we asked open questions to obtain students' general opinions about pair programming. Second, we focused on particular issues that we considered relevant after an analysis of the field notes and observations. Finally, we asked students what they liked or disliked about pair programming.

Field Notes

We also made extensive use of *field notes* that had been taken by the lead author during lectures and after discussions with students and with the lecturer. Those notes were taken while he was playing the role of demonstrator. The notes paid attention to the interaction between students-demonstrator and students-lecturer during those sessions.

5. Results

We categorised the findings from questionnaires, interviews and observations under the following categories. Table 1 shows the findings from each technique applied. The following subsections explain this table in detail.

		Questionnaires	Interviews	Observation/ Field Notes
Individual Differences	*Skill Level*	✓	✓	✓
Roles	*Changing roles*		✓	✓
	Playing roles: navigator and driver		✓	✓
Task		✓		✓
General Findings	*Efficiency*		✓	✓
	Enjoyment	✓	✓	✓
	Perception of Learning	✓		✓

Table 1. Results from the study

5.1 Individual Differences

Skill Level

Students who were interviewed suggested that partners in a pair should be of similar skill level. However most of them also mentioned that pairing two novices is not beneficial. For them, if both partners don't have a good idea about what they are supposed to do, they will both struggle and get stuck. Students also reported that there are situations where sessions with two high skilled participants do not seem very efficient. For example, they mentioned that because some exercises during the course were very straightforward, they could not see any benefits in combining efforts to solve something that was not very challenging.

In addition, students who were paired with a higher skilled partner mentioned that pairing was useful when they had good communication and both aimed to help each other. The findings were backed up by the data collected from questionnaires. These data showed a correlation between skill level and students' perception of learning ($x^2(20)=40.417^a$, $p<.05$). The less skilled partners frequently reported that they learned more when working in pairs. Students also suggested in the interviews that there should not be a big skill level gap because this could lead to the more skilled person taking full control of the resources. Results from our observations also support these findings; we could often see episodes where high skilled students took full control of the resources when paired with passive less skilled partners.

Data from questionnaires showed that the correlation between skill level and student's perception of learning was statistically significant ($x^2(20)=40.417^a$, $p<.05$). The less skilled members of the pair reported that they learned more when working in pairs. In contrary, the high skilled members of the pair reported that they learned less when working in pairs apart from one exception. This exception was that students who assigned their skill level considerably higher than their partner reported that they learned more when working in pairs.

5.2 Roles

Roles have been considered a very important topic in pair programming. In industrial setting, Williams and Kessler [22] categorized two roles: *Driver* and *Navigator*.

Changing Roles

Based on the data collected from observation and field notes, we noted that pairs changed roles in a very simple protocol. When one person had an idea about how to solve the exercise, he/she often took control of the keyboard. One student, for example, said:

> "…In my experience when you have some idea you change roles…"

Analysing the data gathered from interviews we noted that students did not agree on the validity of the *driver* and *navigator roles*. Student mentioned that:

> "…I don't think there should be any kind of fixed roles …"

Playing Roles: Navigator and Driver

During the laboratory session the demonstrator was asked to ensure students changed roles frequently (*navigator – driver*). Different from what we were expecting, it was very hard to say what role each peer was playing. We observed that in many situations one student was controlling the mouse while at the same time the other was controlling the keyboard. There were also some situations when the driver was guiding and giving advice to the navigator. These behaviours are different from those reported in the Extreme Programming literature [22] where the *driver* controls both the computer keyboard and the mouse, and the *navigator* examines the driver's work, offering advice, suggestions and corrections to both design and code. However, it must be said that the Extreme Programming literature is focused on industry settings not academic environments, which could explain our difficulty in observing these roles.

5.3 Task

According to the questionnaires and observations, students enjoyed more pair programming in *program comprehension*, *re-factoring* and *coding*. In the case of laboratory sessions where students were asked to work on *debugging* tasks we noted some of them were very tired at the end. Also, questionnaire data showed a significant correlation ($x^2(10)=21.400^a$, $p< .05$) between task and level of enjoyment. Students enjoyed pair programming less when debugging (see table 2).

Task	Enjoyed
Comprehension	84%
Debbuging	58%
Refactoring	78%

Table 2. Students enjoyment classified by task.

5.4 General Findings

In this section, we talk about general findings that do not go well under the previous categories. General findings include efficiency, enjoyment and learning:

Efficiency

Most of the interviewed students mentioned that they finished the exercises faster working in pairs than working alone. We should mention that this is the student perception of his/her efficiency. There was not any measure of time in our study or any comparison between pair and solo programmers. An example from a student comment is:

> "We finish the program faster than normal time. When we used to do single [sic] takes time, very long…"

Few others mentioned that they finished the program either more slowly or at the same speed when they work in pairs, since both partners can get stuck on the same problem. This is in agreement with a study by Flor and Hutchins [8] which suggested that pairs need to negotiate the manner in which they will solve the problem. However, because they are two different cognitive systems, sometimes with conflict of ideas, they may end up exploring a larger number of alternative solutions. One student commented about this:

> "Sometime you can both get stuck on the same question and it takes longer you to figure out…"

Enjoyment and Learning

Considering the data collected from observations and field notes, it has also been observed that in most cases pairs enjoyed the collaborative experience. They celebrated every achievement when trying to solve the exercise. However, generally when something was going wrong (the pair took a wrong path, or misinterpreted the exercise), higher skilled students were sometimes a little frustrated. For less skilled students doing something wrong was not a problem. Less skilled partners commented on the fact that they were glad to face a problem with someone else rather than on their own.

Questionnaire data also showed that, on average, the majority of students (73%) considered pair programming was an enjoyable task. However, when analyzing only the data from debugging sessions, this figure drops to 58%.

Table 2 shows the students' perception of their learning (question 7 – Appendix A). For the majority of them, they learned more while working in pairs than working on their own. However, it is interesting to note that in Debugging sessions the number of students who do not believe pair programming helped them had increased significantly.

Task	Agree	Neither Agree or Disagree	Disagree
Comprehension	55%	37%	4%
Debbuging	48%	28%	22%
Refactoring	65%	23%	6%

Table 3. Students perception of their learning classified by task.

6. Discussion

As we have seen in previous sections, skill level and task in hand are the most influential factors affecting the perceived effectiveness of pair programming. A difference in skill level between partners strongly affects their collaboration. In addition, students who were working on a debugging task found pair programming tiring and less enjoyable.

Combining both results from interviews and field notes we noticed a relationship between students' opinion about efficiency and their skill level. Students who described themselves as less skilled than their partners mentioned that pair programming is faster than solo programming. This is reasonable, since less skilled students have a strong feeling that working with a more able partner will help them to sort out problems. On the other hand, most of the students who described themselves as having a level of skill higher than or equivalent to their partner mentioned that pair

programming takes about the same time as solo programming.

We also noted that students change roles in a very simple protocol. This is a phenomenon which has also been observed by one of the authors in observational studies of pair programmers in industry. It is not uncommon, for example, during an explanation for one partner to slide the keyboard to the other while asking them to 'show me what you mean'.

Although it was not easy, sometimes we could observe the navigator/driver roles as mentioned in previous studies [8, 24, 26, 28]. However, it seems that there are some other roles in this collaborative situation which might be more relevant than the driver-navigator distinction. Although results were not significant, some other roles were noticed such as teacher-learner and thinker-doer. These roles especially noticed when the partner's skill level gap was bigger.

We found strong evidence that students often have had a pleasant experience because of the support given by a classmate sitting aside, suggesting what to do and answering questions about the problems encountered. It was also observed that pair programming helped less skilled students to expose the limitations in their understanding of the topic without concerns. This reinforces Mercer [12] arguments about the benefits of symmetrical relations (student-student) in the construction of knowledge. Also, as mentioned in Flor and Hutchins [9], when pairs are working together they need to agree on their solution approach to exercises. This is very good because it creates a rich environment of discussion increasing the space of possible solutions.

It is still not clear why the debugging task seems so tiring and not enjoyable, and it will be investigated in more detail in follow up studies. One reason could simply be the nature of the task. A long session of debugging, where it is necessary to try different strategies to figure out errors in the code can be very tiring, because pairs will need to negotiate every single step of the process.

We suggest the practice of pair programming in computer science courses based on our study. As we argue in this document, students have a very positive attitude towards using this approach. Moreover, most of them have the impression that though using pair programming they will learn more. However, we have also shown that it is very important to pair students who are compatible. Therefore, we suggest that students should be matched with a partner who has similar skill level. This should only be avoided when both students are complete novices. Moreover, a novice student should always be paired with a partner with higher skill level.

7. Conclusion

This study suggests that students' skill level and the programming task play a major role in the perceived effectiveness of pair programming as an educational technique. Students seem to enjoy and benefit from pair programming if the skill level gap is not too big. Also, debugging seems to be a difficult task for pair collaboration.

Our findings seem to confirm previous studies of pair programming [8, 24-26] that found a very positive attitude from students towards collaborative settings but they also point out that the pair programming experience of students might be quite different to that of professional programmers. As we explained above, the driver-navigator distinction does not seem to be crucial for students. More research is needed to confirm these findings and possibly to uncover more factors that might be playing an important role in pair programming when applied to teaching.

In the future, we will examine matching pairs according to their programming sub-skills. We will also take a deeper look about the reasons why different task are more enjoyable than others. Later we are planning to design and implement a cognitive tool that will help to match pairs in order to have a more efficient collaboration.

Acknowledgments

This study is partly funded by Teaching and Learning Development Unit (TLDU) of Sussex University. We gratefully acknowledge their support.

References

1. Kooling, M., *The problem of teaching object-oriented programming, Part 2: Environments*. Journal of Object-Oriented Programming, 1999. **11**(9): p. 6-12.
2. Booch, G., *Object-Oriented Analysis and Design with Applications*. 2nd ed. Vol. 1. 1994: Redwood City: Benjamin/Cummings. 185.
3. Kooling, M., *The problem of teaching object-oriented programming, Part 1: Languages.* Journal of Object-Oriented Programming, 1999. **11**(8): p. 8-15.
4. Meyer, B., *Object-Oriented Software Construction*. International Series in Computer Science. 1988, New York: Prentice Hall.
5. Detienne, F., *Assessing the cognitive consequences of the object-oriented approach: A survey of empirical research on object-oriented design by individuals and teams.* Interacting with Computers, 1997. **9**(1): p. 47-72.
6. Thomas, L., M. Ratcliffe, and B. Thomasson. *Scaffolding with Object Diagrams in First Year Programming Classes: Some Unexpected Results*. in *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*. 2004. Norfolk, Virginia, USA: ACM Press.

7. Barnes, D. and M. Kooling, *Objects first with Java: a practical introduction using BlueJ*. 2nd ed. 2004: Harlow: Prentice Hall. 416.
8. Nagappan, N., et al. *Improving the CS1 experience with Pair Programming*. in *SIGCSE*. 2003.
9. Flor, N.V. and E. Hutchins. *Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance*. in *Proceedings of the Fourth Annual Workshop on Empirical Studies of Programmers*. 1991. Norwood, NJ: Ablex Publishing.
10. Gittins, R. and S. Hope. *A study of Human Solutions in eXtreme Programming*. in *13th Workshop of the Psychology of Programming Interest Group*. 2001. Bournemouth UK.
11. Tessem, B. *Experiences in Learning XP Practices: A Qualitative Study*. in *Extreme Programming and Agile Processes in Software Engineering, 4th International Conference*. 2003. Genova, Italy: Springer.
12. Mercer, N., *The guided Construction of Knowledge: Talk among teachers and learners*. 4th ed. 2003, Clevedon: Multilingual Matters. 135.
13. Vygotsky, L.S., *Mind in Society: The development of higher Psychological Processes*. 1978, Cambridge: MIT Press.
14. Bruner, J., *Vygotsky: A historical and conceptual perspective*, in *Culture, Communication and Cognition: Vygotskian Perspectives*, J. Wertsch, Editor. 1985, Cambridge University Press: Cambridge.
15. Lipponen, L. *Exploring foundations for computer-supported collaborative learning*. in *Proceedings of the Computer-supported Collaborative Learning 2002 Conference*. 2002: Mahwah: Erlbaum.
16. Dillenbourg, P., et al., eds. *The evolution of research on collaborative learning*. Learning in Humans and Machines: Towards an Interdisciplinary Learning Science, ed. D.P.R.H. Spada. 1996, Oxford: Pergamon. 189-211.
17. LeJeune, N., *Critical Components for Successful Collaborative Learning in CS1*. J. Comput. Small Coll., 2003. **19**(1): p. 275-285.
18. Katira, N., et al. *On Understanding the Compatibility of Student Pair Programmers*. in *Proceedings of ACM SIGCSE*. 2004. Norfolk.
19. Thomas, L., M. Ratcliffe, and A. Robertson, *Code warriors and code-a-phobes: a study in attitude and pair programming*, in *Proceedings of the 34th SIGCSE technical symposium on Computer science education*. 2003, ACM Press: Reno, Navada, USA.
20. VanDeGrift, T., *Coupling pair programming and writing: learning about students' perceptions and processes*, in *Proceedings of the 35th SIGCSE technical symposium on Computer science education*. 2004, ACM Press: Norfolk, Virginia, USA.
21. Jensen, R. *A pair programming experience*. 2003 March 2003 Issue [cited 2005 12/04/2005].
22. Williams, L. and R. Kessler, *Pair Programming Illuminated*. 2002, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
23. CockBurn, A. and L. Williams, *The cost and benefits of pair programming*, in *Extreme Programming Examined*, I.G.S.a.M. Maresi, Editor. 2001, Addison-Wesley. p. 223-247.
24. Hanks, B., et al. *Program Quality with pair programming in CS1*. in *ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. 2004. Leeds, United Kingdom: ACM Press.
25. McDowell, C., et al. *The impact of pair-programming on student performance, perception and persistence*. in *ICSE '03: In Proceedings of the 25th International Conference of Software Engineering*. 2003. Portland, Oregon: IEEE Computer Society.

26. Williams, L., et al., *Strengthening the case for pair-programming.* IEEE Software, 2000. **17**(4): p. 19-25.
27. Hanks, B. *Empirical Studies of Pair Programming*. in *Position paper for EEAP '03 - 2nd International Workshop on Empirical Evaluation of Agile Processes*. 2003. New Orleans, Lousiana, USA.
28. Melnik, G. and F. Maurer. *Perceptions of Agile Practices: A Student Survey*. in *Extreme Programming/Agile Universe*. 2002. Chicago, IL.
29. Bryant, S. *XP: Taking the psychology of programming to the eXtreme*. in *16th Workshop of Psychology of Programming Interest Group*. 2004. Institute of Technology, Carlow, Ireland.
30. Hundhausen, C.D., *Integrating algorithm visualization technology into an undergraduate algorithm course: ethnographic studies of a social constructivist approach.* Computer and Education, 2002. **39**: p. 237-260.
31. Cohen, L., L. Manion, and K. Morrison, *Research Methods in Education*. 5th ed. 2003, London: RoutledgeFalmer. 446.
32. Denzin, N., *Triangulation in educational research.* 2nd ed. Educational Research, Methodology and Measurement: An International Handbook, ed. J.P. Keeves. 1997, Oxford, UK: Elsevier, Science Ltda. 318-322.

Appendix A: Pair Programming Questionnaire

The aim of this questionnaire is to evaluate the collaborative learning technique we have applied today. Your answers are anonymous. Please fill in the questionnaire on your own.

1. Gender

a) female b) male

2. My level of programming skill compared to that of my partner for today's session is

a) Considerably higher b) Higher c) Equivalent d) Lower e) Considerably lower

3. I enjoyed doing pair programming today

a) Strongly agree b) Generally agree c) Neither agree nor disagree d) Generally disagree e) Strongly disagree

4. The thing I liked the most about working in pairs was that (please mark one option only)

a) Talking to someone about the exercise makes things clear for me
b) The other person can verify that what I'm doing is correct
c) My partner can help me if I get stuck

d) Working with someone makes it easier to be aware of what I'm doing and why I'm doing it
e) Other reason (please explain)

5. The thing I liked the least about working in pairs was that (please mark one option only)

a) Talking while trying to understand something makes things difficult for me.
b) The other person just wants to do things his/her own way
c) I have spent too much time explaining things to my partner
d) There are too many things to be aware of and that is too distracting for me
e) Other reason (please explain)

6. It was easy to share resources such as the keyboard and mouse

a) Strongly agree b) Generally agree c) Neither agree nor disagree d) Generally disagree e) Strongly disagree

7. I think I've learned more this time working in pairs than others that I've worked on my own

a) Strongly agree b) Generally agree c) Neither agree nor disagree d) Generally disagree e) Strongly disagree

8. Other comments?