2006

Paired Programming in a Clayton's Capstone Project Course

Steven R. Clark
University of South Australia, steven.clark@unisa.edu.au

Adam Jenkins
University of South Australia, adam.jenkins@unisa.edu.au

Paired Programming in a Clayton's Capstone Project Course

Steven R Clark
School of Computer and Information Science
University of South Australia
Email: steven.clark@unisa.edu.au

Adam Jenkins
School of Computer and Information Science
University of South Australia
Email: adam.jenkins@unisa.edu.au

Abstract

In this paper the authors describe an Information Systems course where paired programming was trialled in the hope of improving both the assessment outcomes and the resourcing of an existing systems design course which incorporated a capstone project. The authors found that it did offer a significant improvement, but the improvements did not match the predicted outcomes, with a number of weaknesses to the model being highlighted. In particular, it was found that paired programming did not reduce teaching load when the group size was reduced.

Keywords

Teaching, assessment, resources, paired programming, systems design, project

INTRODUCTION

Between 2001 and 2006, the University of South Australia has been offering a systems design course as part of the third (and final) year of the Information Systems program. The course featured a difficult and complex student project which served as the major piece of assessment. As such, the course took on the feel of a capstone project, even though this was not the initial intention. Although it was largely successful, especially in terms of learning outcomes, a number of problems were identified with the use of the project for assessment, many of which centred on the use of large(ish) student groups to complete the assignment. In 2006 the course was re-evaluated, and as part of this process it was proposed that a modified paired programming model be employed. This paper explores the application of paired programming to the course, looking at the problems that existed prior to its introduction, the reasons as to why it was trialed in 2006, and how the application of the paired model produced both expected and unexpected results. In short, this paper aims to serve as a war story for other IS educators interested in the application of the paired programming methdology.

There were a number of factors which make the trial of paired programming in 2006 of particular interest. Primarily, this wasn't a situation where the course was being converted from individual assignments to paired/group projects, but rather from large groups to small ones. Additionally, due to a reduction in student numbers, the number of groups remained consistent with previous offerings – thus providing the opportunity to evaluate the model in terms of numbers of groups rather than the overall number of sudents. The course also had a roughly equal mix of both genders and local/international students. And finally, the focus in this course was not on paired programming in terms of improving student learning, but on paired programming in terms of making better use of limited resources and in the improvement of the assessment model - again, not the typical focus of most paired programming research, which often places greater focus on the learnng outcomes.

What is Paired Programming?

Although the paired programming model is normally seen as being closely allied to Extreme Programming and the Agile development methodologies, this does not need to be the case, as the concept was described several years before Extreme Programming was proposed (as argued by both Lui & Chan, 2006, and Williams, Kessler, Cunningham & Jeffries, 2000). The approach itself is quite simple:

> "… two programmers [collaborate] side by side on the design, coding and testing of a piece of software. One, the Driver, controls the keyboard/mouse and actively implements the program. The other, the Navigator/Observer, continuously evaluates the work with a view to identifying tactical defects and providing strategic planning." (Lui & Chan, 2006. page 916)

As is apparent from this description, the main difference between paired programming and more traditional groups is that the two programmers are required to use a single computer. Thus the collaborative effort becomes more intense – rather than each team member working on individual aspects of the project, the paired programming model insists that they work together on all aspects. Significantly, most exponents of paired programming suggest that the roles should be periodically reversed, with the Driver taking on the Navigator's role and vice versa (Williams et al, 2000). Similarly, writers such as Cockburn often recommended that pairs be changed periodically (2002, page 166), although this may not always be practical within a teaching environment.

The use of paired programming in education has been well established. A number of universities have explored the value of such an approach, and many advantages have been identified. The key issue identified in the literature is that paired programming provides a "realistic simulation of the working world" (Todd, Mardis and Wyatt, 2005, page 384), largely because the development of "all non-trivial software projects are collaborative efforts" (McDowell, Hanks & Werner, 2003, page 60). This advantage is further enhanced when the issue of increased retention of female students is taken into account. For example, Todd et al argued that the Computer Science Department at the University of California, Santa Cruz, introduced paired programming, in part, to show women that Computer Science does not need to be a solitary adventure (2005, page 384). Furthermore, by improving the level of social interaction, paired programming produces students who "are more confident, have greater course completion and pass rates in that course, and are more likely to persist in computer-related majors." (Werner, Hanks, McDowell, Bullock, and Fernald 2005, page 2).

THE OLD SYSTEM: BSDI/BSD 2001-2005

From 2001 to 2005, students were offered what amounted to a Clayton's[1] capstone project course. The course, Business Systems Design and Implementation (BSDI), later renamed Business Systems Design (BSD), tended to examine both design and implementation issues (as the title might suggest), with a focus on database development. In practice, however, the focus was clearly on the major project – a complex project involving the development of a database system in Oracle, using the Oracle Designer and Developer tools. The students were expected to design their database, build it in Oracle, develop forms to interact with the data (using Forms Builder), generate reports (Report Builder), design and build a navigation scheme, and to write user documentation, system documentation and training documentation. All students were given the same project.

Due to the scale and nature of the project, students were expected to work in groups of about four to five people. Some students chose to work individually (most notably the off-campus students), and they were given a project with less scope, but the vast majority of students worked as part of a group. Given that there were (on average) 80 students, this equated to approximately 15 groups per semester.

Assumed Knowledge and Student Background

By the time students entered into BSDI/BSD, it was assumed that they would have completed at least one semester of Database Design (covering SQL and PL/SQL, both of which were regarded as required skills for the project), and one semester of Systems Analysis (which wasn't considered to be absolutely essential, but which was nevertheless seen as desirable). In preparation for the major project, students were expected to complete practical work which covered most of the basics of working with Oracle, from designing the Entity Relationship Diagram (ERD), through generating the database, and on to building basic forms and reports which incorporated some PL/SQL-based programming. The lectures tended to focus on theoretical issues which were not directly relevant to the project.

Although the majority of the students between 2001 and 2005 were from either a dedicated Information Systems degree or from combined programmes where Information Systems was a major part of their study, a sizeable proportion (20%) of those enrolled were from non-IS backgrounds (Table 1). This meant that, on average, one out of five students lacked (to varying degrees) the assumed prior knowledge.

Resources

Because of the size and complexity of the Oracle installation, the Oracle developer software was only installed in a single lab, called, unsurprisingly, "The Oracle Lab". This was a dedicated lab that was used exclusively for the Oracle courses (Database Design, Systems Analysis and BSD/BSDI). Although there was occasionally some overlap, course scheduling meant that it was never used for more than two courses in any given semester. The Oracle Lab consisted of 20 computers, of which 18 were assumed to be working at any given point in time. To ensure students didn't use the Oracle Lab for other courses, the only software installed on the computers were those packages essential for Oracle development. Some students chose to work from home (typically 2-3 groups

[1] The capstone project course you have when you don't have a capstone project course.

per semester) and a small number of students (typically no more than 5 students) were given access to the Oracle software at a different location on campus.

Year	IS Students	Business	Marketing	Commerce	Comp Sci	Total
2001	73	3	4	0	0	80
2002	85	13	0	8	0	106
2003	59	9	1	6	2	77
2004	66	16	0	5	0	87
2005	44	8	0	2	1	55
Subtotal	327	49	5	21	3	405
2006	28	1	0	1	4	34
Total	355	50	5	22	7	439

Table 1. Comparison of student backgrounds and numbers, 2001-2006

Although the lab only held 20 computers, the room itself was quite large, giving students plenty of room around each computer. This was not of particular significance in the 2001-2005 version of the course, but became significant when paired programming was introduced in 2006.

Teaching resources were limited to 2-3 lab assistants/tutorial staff. Due to the breakdown of responsibilities, it was typical that only a single lab assistant/tutor was sufficiently versed in the Oracle tools to provide assistance in the major project. The other staff tended to assist primarily with the theoretical components of the course. While this was less than ideal, it was difficult to find people with sufficient knowledge of the Oracle software who were also willing to work as tutors for this course.

PROBLEMS WITH THE COURSE, 2001-2005

During this period a number of problems were identified by the teaching staff. The most significant of these were:

- Student specialisation
- Unfair distribution of workload.
- Plagiarism
- Teaching load
- Computer access
- Stress

An initial attempt to address these was made in 2004 when BSDI was split into Business Systems Design (BSD) and Business Systems Implementation (BSI). However, as the project remained as part of Business Systems Design, the overall effect was minimal.

Student Specialisation

Students undertaking this subject tended to specialise in different areas of the project. With tasks distributed amongst the students, one student would design and build the database, one or two would work on the forms, one would specialise in writing the PL/SQL code, one or two would develop the reports, and one or two students in every group would write the documentation. This allowed students who lacked the assumed knowledge to find a role within the groups. Consequently, a proportion of the students were able to complete the course without having undertaken any work in any of the core areas. On the other hand, it did tend to reflect the typical nature of group work within industry, with different people bringing their own unique skills to the project.

It should be noted here that this may have been a desirable outcome in a traditional capstone project course. Unfortunately the intent of this subject was not to provide a capstone project, but to teach systems design skills..

Unfair Distribution of Workload

Due in part to the habit of students to specialise in particular areas, there was a tendency for the workload to be unfairly distributed amongst the students. Some students, in particular those working on the forms, tended to have to work long hours trying to complete their tasks, while other students – such as those writing the documentation

– tended not to come into the picture until late, and when they did become involved they often found the work easier and quicker to complete. This is not something that is unusual to this course: Zobel and Hamilton (2002, page 124) identify that, in group projects, "[w]eaker students can hide behind the collective and fail to contribute …" – something that was clearly a problem here.

Plagiarism

Although difficult to measure, there was a strong suggestion that instances of plagiarism occurred within the course. Interestingly, this plagiarism may not always have been in accordance with the wishes of the majority of members of the groups. As the students tended to work on their own, due, in part, to their tendency to specialise, it was possible for a student to plagiarise without the other group members being aware. Zobel and Hamilton (2002) suggest that this may be a common problem with group projects, when they note that students may "… share the blame with unwitting colleagues when their contribution is found to be stolen" (page 124). Anecdotal evidence (provided by students some time after they had completed the course) tended to support this view, with a number of students expressing dissatisfaction with the manner by which some of their group members completed their roles. This is not in keeping with some suggested strategies to reduce plagiarism, where group work is often recommended. For example, Born argues that group assessments have:

> "… a "deterrent" effect on cheaters. It would be difficult for someone in a group to cheat while members are watching. Students who do not know each other before are also likely to report [an] offense when they come across it." (Born 2003, page 223)

Perhaps because students tended to compartmentalise the project within their groups, it was difficult for team members to be "watching" one another, and therefore individual instances of plagiarism could occur in spite of the wishes of the group as a whole.

Computer Access

There were never enough computers for the students. At best there would be slightly more than one computer per group, but each group typically required 2-3 computers at a time. As a result many groups adopted a practice of beginning their work late at night and working through until the early morning, while others worked "shifts" – with 2 members always in the lab at any given time. Some students even took to "locking" their computers while they were out for extended periods of time.

Because students were unable to work in any other lab, this became a major issue – an overcrowded lab, containing large numbers of overworked, tired, and overstressed students, is unlikely to engender a positive working environment.

Teaching Load

The lack of qualified teaching staff caused major problems. With typically only one, and at best two, staff having a solid grasp of the technical aspects of the project, it was found that they needed to spend considerable time with the students. Indeed, staff were known to work 16 hour days as the deadline approached, spending almost all of this time within the Oracle Lab.

During the time spent in the lab, the staff members would assist the students in:

- Fixing the database design.
- Helping the students recover from major accidents.
- Coding issues.
- User Interface issues.
- Fixing issues specific to the software and installation.

Many of these issues could perhaps have been ignored. However, the project was not designed solely as a summative assessment, and it was felt that this extra assistance would encourage the students to push themselves beyond the defined course parameters, improving the formative aspects of the project. Nevertheless, turning these problems into educational experiences for the students resulted in yet another issue for teaching load: a problem that would take 5-10 minutes for an experienced staff member to repair would, in practice, take 20-30 minutes while the staff member explained to the students what was being done.

Stress

All of these factors combined to make for a very stressful situation for both staff and students. Staff found entering the Oracle Lab to be an ordeal, as it was very difficult to extricate themselves once they had ventured within. It didn't seem to matter how long they stayed, nor how many questions they answered – there was always another issue to address. The students were also working long hours (even if this was mostly so that they could get access to a computer), with unfamiliar tools and with insufficient access to staff. The projects themselves also tended to be quite difficult, often possessing core requirements that were extremely difficult to implement. Furthermore, the students often attempted to implement unusual and, occasionally, inappropriate solutions.

To emphasise the amount of stress, some students who returned to the university many months after the project was complete refused to go anywhere near the Oracle Lab – their aversion to that part of the university was surprisingly intense.

ADVANTAGES TO THE COURSE, 2001-2005

Although it is clear that there were significant problems with the course as it was offered, it should also be stated that there were a series of distinct advantages. Staff identified three key advantages with the course as it was offered between 2001 and 2005.

Community

As the projects were identical, it was found that students tended to assist other groups as well as their own, and this behaviour was always encouraged. A student who was shown the solution – or at least a valid approach – to a problem by a staff member would typically be willing to share his or her new knowledge with the other groups. This had the advantage of significantly improved the teaching load.

Nevertheless, on occasion this would lead to a negative effect, when students who had particular skills would choose to spend a large portion of their time instructing other groups, to the detriment of their assigned team.

Standards

The students tended to work to a very high standard, with the final project generally being of extremely high quality. Often students claimed that it was this course where they learnt the most about Information Systems. This may not be unusual for a capstone project, and indeed, could well be an underlying rationale for the development of such a course in any IS program: but it was nevertheless rewarding to all involved.

Experience of Complex System Development

As is typical with a capstone project, this course encouraged the students to bring together many of the skills that they had learnt to date, and to apply them within a complex team-based environment. They had the opportunity to discover the effect that design decisions have on the development process, to experience the impact that multiple team members can have on all aspects of systems development – both positive and negative – and to be exposed to a large-scale database system. In general, it can be reasonably claimed that students gained all of the advantages that would be expected from a capstone project.

SYSTEMS DESIGN: 2006

2006 saw the introduction of what was intended to be the new, improved, Systems Design course. Many of the changes were to the course content, with a move from relational design to Object Oriented design, and these were perhaps the changes that made the most impact to the staff teaching the course. Nevertheless, it was also felt that this would be an opportunity to revisit the design of the major project. The changes to the project were based, in part, on a desire by the staff to rectify some of the problems which had been encountered in the past. However, a number of other factors were significant:

1. The number of students per semester dropped significantly from previous offerings. In 2006 the course was offered in two semesters, and the student numbers were split across both.
2. Changes to the University assessment policy meant that the project could no longer make up the bulk of the assessment. A final examination was introduced, so the project was dropped to 50% of the final grade, down from 70% in previous years.
3. It was necessary that Oracle be employed, but some problems were encountered with the installation.

4. The Oracle Lab computers were in a poor state of maintenance. Although the computers had been upgraded, only 16 were functioning at any one time, instead of the previous 18. On the plus side, it did increase the amount of space available on a per-computer basis.
5. Other courses started using the Oracle Lab for non-Oracle projects.
6. The staff were reduced to two: one lecturer/tutor, and one dedicated tutor.

The New Approach

The new approach involved three key aspects:

- The project was simplified. It wasn't much smaller than that offered in previous years, but the documentation was significantly reduced (only system documentation being required), and the project itself was less complex than was required in the past.
- Previously the students had self-selected their groups. Although this was offered, students who did not select their own groups by the deadline were placed into groups automatically. (Approximately 60% of the students were placed into groups by the staff).
- Rather than employ the large groups which had been used in the past, a paired programming approach was employed.

The complexity of the project had been identified as a major factor in the earlier problems. It was felt that by reducing the complexity the overall workload would be reduced – both for staff and students. It didn't. By the end of the semester it was clear that the project wasn't the major difficulty faced by the students. Instead the students were suffering at the hands of Oracle.

In the end it became clear that the major change was, in fact, the introduction of paired programming. While it was acknowledged that paired programming can have pedagogical value in terms of learning outcomes, the major arguments for its introduction were based on assessment and resource concerns.

- The habit of students to specialise had been an acknowledged problem in the past, and was seen as being detrimental to the assessment process. By using paired programming it was hoped that students would be exposed to – and assessed on – all aspects of the design and implementation process.
- Paired programming requires both partners to be present at the same time. While this may not result in equal contributions of effort, it should, at the very least, result in equal contributions of time. It was hoped that this would allow for fairer assessment of group work.
- With the reduced number of working computers, it was felt that sharing one computer between two people would result in better resource allocation than what was encountered in previous years.

Although paired programming was not expected to improve the teaching load, (it was felt that the simplified project would have the most impact on this issue), there was nevertheless an expectation that the pairs would be easier to work with than the larger groups. When combined with the reduced student numbers, the overall belief was that there should be a significant reduction in the amount of time that individual staff members had to spend with students.

FINDINGS

Things did not work out as planned. It can be argued that things never work out as planned, especially when teaching, but while it was felt that the paired programming model was, overall, an advantage, the strengths and weaknesses of the model did not necessarily coincide with those that were predicted before the semester began.

Student Specialisation

Staff reported that there were no apparent instances of specialisation – students tended to be involved in all aspects of the design and development process. Students worked together, with only a few cases where the students chose – for a time – to work independently. When they worked independently, it often was to try to develop solutions to the same problem. Upon return they would compare each solution and implement the preferred approach. There were certainly times when the students did separate to work on independent tasks, but these were very much in the minority.

Overall, this was particularly satisfying for staff, who felt that the learning outcomes were better as a result than those which emerged from the previous model, and that the final assessment better reflected the effort put in by both members of the group.

Distribution of Workload

The assignment of pairs was considered to be a key issue. Wills, Davis and Cooke (2004) identify one approach, whereby students are be matched based on equivalent skill levels and genders. They state:

> "It is important to put students of similar abilities and same genders together, to stop one of the pair taking over" (Wills et al, 2004, p3)

Rather than using this approach, and thus ignoring these concerns, it was decided that students would be randomly assigned to partners. So neither skill levels nor gender played a part in the pairings.

The results proved to be highly satisfactory, as there were no gender or skill-based workload issues. All of the pairs appeared to work well together, with all the students taking turns as "navigator" and "driver". The mixed gender groups performed as well as the single gender groups, both in terms of workload distribution and project standards, and all of the students were able to apply their respective skills to the project.

Plagiarism

As it was not expected that the paired programming model would have any impact on the level of plagiarism, this was not considered as a factor for employing the paired programming model. Nevertheless, it would appear that it may have had an impact – the close working environment meant that it would be difficult, if not impossible, for one member of the group to plagiarise without the willing participation of their partner. This is, perhaps, in keeping with both Zobel and Hamilton (2002) and Born (2003) who suggest that group work allows plagiarism when students are able to slip their contributions into the mix – the corollary of this being that plagiarism becomes more difficult if the group members "monitor one another" (Rowe, 2004).

The possibility that paired programming is a means of reducing the level of plagiarism when compared to both individual and group teaching is one that is worth further exploration.

Computer Access

The biggest success for the paired programming model was arguably the computer access, although at least part of this success can be laid at the feet of the reduced student numbers. In previous offerings, the Oracle Lab became full with only 4-5 groups present. In 2006, all 15 groups could use the lab at the same time. As a result students were able to work normal hours, with only a few students present after 11pm, and student numbers were much reduced on weekends.

In situations where the paired model "broke down", resulting in two computers per group, students were surprisingly ready to surrender their second computer if it was required by another pair, invariably without any prompting from staff.

Community

In previous offerings a strong sense of community was developed, both within groups and across the student body as a whole. To some degree this was due to a sense of shared suffering. However, it can be argued that this was also the product of the willingness of individual group members to assist other groups when required.

In 2006 this was not the case. Certainly the pairs tended to work well (with one or two notable exceptions), but there was comparatively little interaction between groups. It is not entirely clear why, but two possibilities present themselves. The first is that the overall level of suffering was reduced: stress levels were down, the students had a less complex project to develop, and the contact hours were kept to manageable (and indeed sane) levels. The second derives from the nature of paired programming. In large groups – especially large groups with a clear demarcation of responsibilities – it is not unusual to have one or two members free to wander and help others. In paired programming the individual members cannot leave their groups in order to interact with others – doing so is tantamount to abandoning their partners. Thus, while individuals were allowed to, on occasion, live their own lives, group-to-group interaction was not something that featured highly in the course.

Teaching Load

Teaching load was the most surprising outcome from the employment of a paired programming model. It has been argued by other researchers, such as Willis, Davis and Cooke (2004) that paired programming decreases the teaching load.

> "The next observation was that the amount of work undertaken by the postgraduate demonstrators was significantly reduced, that is, they had to answer fewer questions. However the questions they

> did have to answer were generally of a higher level of difficulty than previously, so they spent the same amount of time in answering questions as in previous years." (Willis et al, 2004, page 3)

Whether or not reducing the number of questions while increasing their difficulty equates to reducing the workload is something that is open to debate. Nevertheless, staff clearly envisioned that the paired programming model, in conjunction with the reduced number of students, would result in a net reduction in teaching load.

This proved not to be the case. Overall, the teaching load was on a par with that experienced in previous years. Certainly the number of hours which the staff spent working with the students was equivalent to those spent in previous years, although there was less emphasis on the late nights – if only due to the simple fact that students no longer had to arrive late in order to gain access to the lab. This represents a significant problem when it is considered that the student numbers were significantly decreased, suggesting that the time per student was higher than in previous years.

The four main explanations that offered themselves were:

1. The number of groups remained the same. Thus time-per-group remained consistent with previous offerings, even if time-per-student increased. Nevertheless, it should be noted that the reduction in the complexity of the projects should have helped address this problem.
2. As identified by Willis et al (2004), potentially the questions became richer, so that the time spent on individual questions was increased, even if the overall number of questions decreased. But this does not seem to be the case – the questions were, in general, on a par with those previously encountered.
3. The teaching load had reached "saturation point" in previous years – the reason it wasn't higher in 2001-2005 was that the staff were unable to spend any more time with the students. Thus the 2006 offering represented a decrease in teaching *need,* but this failed to reach a point where the need was lower than the amount of time on offer.
4. The lack of inter-group communication limited the dissemination of knowledge through "unofficial" channels, resulting in a greater reliance by students on the staff as the primary source of information.

Overall, it would appear that both 3 and 4 were the primary causes. It would be difficult to quantify student need in any circumstances, but there is a clear possibility that the amount of time spent with students was insufficient to meet their requirements from 2001 to 2005, and thus reducing the complexity of the task in 2006 was not enough, in and of itself, to produce a net improvement in teaching load. However, it is possible to get some sort of idea about the amount of knowledge flow between groups – and this was clearly less than in previous years. Staff found that they had to address the same concerns with all of the groups, and this was a significant change from what they had encountered between 2001 and 2005. Indeed, it could be argued that this resulted in the exact opposite of what Willis et al (2004) described: rather than the questions becoming more complex, the questions became less complex but more numerous, and certainly more repetitive. (It should be noted here that Willis et al went from individual projects to paired projects, while this course moved from large group projects to paired projects. Unless Willis et al also witnessed an increase in student numbers, it may be that time-per-group is the significant factor, not time-per-student).

Stress

In general, it would appear that stress levels for students were much reduced, although it was unclear whether or not the stress levels for staff changed significantly. The working environment was much improved, with fewer late nights, easy access to computers, and less desperation to meet deadlines. The major stress factors would appear to be internal to the groups and to be based on the ability of the pairs to work together: internal dynamics of one of the groups was such that their stress levels were at least on a par with that experienced by groups in previous years, but this would appear to be an aberration, as most of the pairs worked well together.

Standards

The overall standards of the final assignment were as high as had been experienced in previous years. This was surprising, as it was felt that the smaller groups would be unable to achieve as much as that managed by the larger teams. This may have been a result of the strength of the paired programming model. However, this is not a conclusion that can necessarily be supported by the current data set, as this may have had as much to do with the reduced complexity of the project and the smaller number of required deliverables as with the development model which was employed.

Nevertheless, as identified by other researchers, the paired programming model will tend to favour the novice, or less-skilled, members of the pairs (Lui & Chan, 2006). This is reinforced when the potential compositions of the pairs is taken into consideration. For arguments sake, we can assume that participants in a paired programming project will be highly skilled, skilled or unskilled. It can be further assumed that two highly skilled people, when

working together, will produce a high quality product, and that, on average, two unskilled participants will produce a comparatively poor quality product, *ceteris paribus.* It can also be assumed that a highly skilled participant, when paired with a less-skilled partner, will improve the quality of the final product. This would result in a model similar to that envisioned in Table 2.

	Highly Skilled	Skilled	Unskilled
Highly Skilled	High Quality	High Quality	High Quality/Acceptable
Skilled	High Quality	Acceptable	Acceptable
Unskilled	High Quality/Acceptable	Acceptable	Poor Quality

Table 2. Paired project quality matrix.

As can be seen from the table, this should result in a higher incidence of high quality or acceptable products than would be expected if the students worked individually. This was reflected in the final grades for the project: there was a clear predominance of Distinctions or greater (56% of the student results) and Credits (38%) when compared to Passes (6%). Potentially this could be addressed through the use of stricter assessment standards, but a deliberate choice was made to apply the same standards in 2006 (with the smaller groups) as had been applied in previous years.

In the end it was felt that the change to paired programming did not significantly reduce the quality of either the work submitted or the learning outcomes.

External Students

This course has never been known to do a good job at addressing the needs of external students, and this has been highlighted as an issue that will need to be addressed in the future. Certainly 2006 was no exception. Although the paired programming model would appear to have provided a positive outcome, it was of no help to the external students, who were unable to work with this approach. The team-based model employed in the past may have worked better for external students, but this is simply a matter of conjecture, as no previous external student worked as a member of a group.

FUTURE DIRECTIONS AND CONCLUSION

Overall, it seems that the use of paired programming was a positive addition to the course, and as such it will probably be retained for future offerings. Still, notwithstanding the overall success of the approach, it wasn't without problems. It was surprising, and perhaps worrying, that the overall teaching load had failed to decrease significantly, as this had always been one of the major concerns with the subject. Furthermore, the loss of communication between groups has been identified as a possible area of concern.

One suggestion to improve the workload issues is to look at a means to provide for the spread of information that is not so reliant upon explicit student-to-student communication. Suggestions include the use of Wikis to store knowledge of value to all students, greater reliance on discussion forums rather than face-to-face problem solving, regular guided discussion groups where students are encouraged to share knowledge and experience, and encouraging "group explanations", where all students in the lab are pulled together to be shown the same information.

The potential reduction in incidences of plagiarism is an issue that warrants further exploration. It may be the case that paired programming provides a potential methodology that will overcome some of the problems that others have identified with group work. However, in order to explore this issue further, several questions will need to be addressed:

1. If students self-select, might they self-select for plagiarism?
2. How difficult is it for a student to "sneak" plagiarised work into a paired project?
3. How much of a sacrifice is the use of paired programming in terms of assessment outcomes?
4. How much of an impact does the lack of inter-group communication have on the incidences of plagiarism? Does this represent a net positive or a net negative?

It would also be worth examining the impact of paired programming on the overall standards of submitted work, and whether or not it leads to a fair breakdown of marks. Current figures suggest that it may inflate the results of the poorer performing participants, and this is certainly an issue that will warrant further research.

In conclusion, the 2006 offering was significantly improved over previous years, largely due to the introduction of paired programming. This, however, should be noted in terms of the overall picture: the course may have been improved, but it wasn't exactly looking good to begin with.

REFERENCES

Born A.D. (2003) How to Reduce Plagiarism, *The Journal of Information Systems Education*, 14(3): 223-224.

Cockburn, A. (2000) *Agile Software Development* Addison-Wesley, Boston, MA.

Todd, K. Mardis, L. & Wyatt, P. (2005) We've come a long way, baby!: but where women and technology are concerned, have we really? *SIGUCCS 2005*: 380-387

Lui, K. M & Chan, K. C. C. (2006) Pair Programming Productivity : Novice-Novice vs Expert-Expert, *International Journal of Human-Computer Studies* 64 : 915-925

McDowell, C, Hanks, B, & Werner, L. (2003) Experimenting with Pair Programming in the Classroom *Proceedings of 8th Annual conference on Innovation and Technology in Computer Science Education*, 60-64.

Rowe, N. C. (2004) Cheating in Online Student Assessment: Beyond Plagiarism. *Online Journal of Distance Learning Administration* 7(2)

Werner, L.L., Hanks, B., McDowell, C., Bullock, H. and Fernald, J. (2005) Want to Increase Retention of Your Female Students? *Computing Research News* 17(2): 2.

Wills G, Davis, H, & Cooke E (2004) Paired Programming for Non-Computing Students. *Fifth Annual LTSN-ICS Conference*, Ulster, UK.

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

Van Der Vyver G & Lane M (2003) Using a Team-Based Approach in an IS Course: An Empirical Study. *Journal of Information Technology Education* 2:393-406.

Williams, L, Kessler, R. R, Cunningham, W. & Jeffries, R. (2000) "Strengthening the Case for Pair Programming" IEEE Software, 17(4): 19-25.

Zobel, J. and Hamilton, M. (2002) Managing Student Plagiarism in Large Academic Departments, *Australian Universities Review* 45(2):119-126.

COPYRIGHT