Collaboration Leads to Success

A Study of the Effects of Using Pair-Programming Teaching Technique on Student Performance in a Middle Eastern Society

Mamoun Nawahdah and Dima Taji
Faculty of Engineering and Technology
Birzeit University
Birzeit, Palestine

Tomoo Inoue
Faculty of Library, Information and Media Science
University of Tsukuba
Tsukuba, Japan

***Abstract*—In pair-programming, two developers share a computer to work together on developing one piece of code. To test Pair-programming effects on student performance in a Middle Eastern society where some interaction restrictions are found, we devised an experiment that was carried out over an entire academic year consists of two semesters. The experiment targeted two sections per semester of an advanced computer programming course. The students of one of the sections worked in pairs during the lab sessions, applying pair-programming rules and techniques. The other section had students who worked individually, as it is the norm in most programming labs. Through this experiment we revealed that pair-programming has the potential to increase the students' confidence, their enjoyment of the course, and improved the course's completion rate. In addition, the students in the pair-programming section showed that they were able to individually produce code of better quality than the students in the traditional section.**

Keywords— Pair-programming; Extreme Programming; Agile; Computing Education; Teaching/Learning Methodologies

I. INTRODUCTION

Trends in teaching techniques in computer science courses are trying to bridge the gap between the university environment and work environment. In universities, students are usually instructed to do their work and assignments individually, and collaborations are most often considered as cheating attempts. When working in software companies and other institutes, team players are the ones that usually thrive, and produce better results. This pushed for incorporating collaborative work in the teaching of computer science courses in universities around the world, to facilitate the transition of computer science students from an individual-centered environment to a group-centered environment.

In 1999, the notion of Extreme Programming (XP) was introduced, and one of the more controversial practices it introduced was pair-programming. Pair-programming is *"the practice whereby two programmers work together at one computer, collaborating on the same algorithm, code, or test"*[1]. In pair-programming, two programmers will sit next to each other on one computer, looking at the same screen, as shown in Fig. 1. They use one keyboard and one mouse, to manipulate the computer and type their code. The programmers work together, taking turns to type, and continuously discuss their code, improve it, revise it, and debug it.

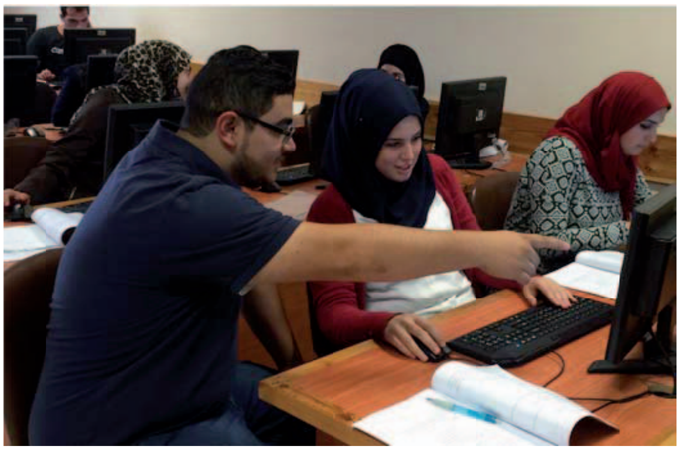


Fig. 1. Two students practicing pair-programming.

Because of the success of this technique in the software development industry, and due to its applicability in labs, this technique was attempted, as a teaching technique, in a number of universities worldwide[2-10]. The experiments where pair-programming was applied in teaching produced various degrees of success, and concentrated on a wide number of parameters, and environment variables.

Programming courses are reputed with having low averages, with failure rates varying between 30% to 50% worldwide[11]. The literature shows that the application of pair-programming as a teaching technique has promising results of improving the education of computer science, if applied correctly[3]. In an attempt to acclimate students to working within a team, and to improve their learning abilities, and their programming quality, we applied pair-programming as a teaching technique to one advanced programming course at our university. Even though similar experiments in the application of pair-programming in education were carried out, it has not, to our knowledge, been applied in a conservative Middle Eastern society so far. A Middle Eastern society has known restrictions on gender interaction and social behaviors norms. This might impose some limitations to the extent of interaction between students in the same class.

Being a conservative society does not look favorably on pairs formed of students from different genders. Therefore, male students often prefer to work with other male students, and female students with other female students. This usually limits the exposure of a student to his or her peers, which leads to limiting their experience and knowledge. In addition, from our experience in higher education, it was noted that students regard any group activity as an opportunity to have the workload fall on one or more students, but not the entire group. However, pair-programming's success depends on having both members of the pair work equally to obtain the best results. Having students understand that they will have to work with each other, regardless of who their partner is, or where they are from, was a hitch that we had to overcome. This makes the experiment conducted in our university stand out from other experiments, due to the social norms that distinguishes the Middle Eastern society from other societies.

The main research question is does pair-programming have the potential of improving the performance, in terms of grades, and the enjoyment of students, and allow them to produce quality software code, within the local restriction?

Despite the mentioned society restrictions, this study revealed that pair-programming had the ability to improve the course completion rate, the students' enjoyment of the course, and quality of the code that they produced by the end of the semester.

II. Literature Review

Programming has always been considered a solitary activity[16]. This was the way it was taught, and the way it was practiced until 1999, when Kent Beck created Extreme Programming (XP), and listed pair-programming as one of its twelve practices[17]. Solo programming was associated with the waterfall development cycle in software engineering, suggested in 1970, in which a development team would meet with the customer once to get all the requirements of the system, and then design it, and implement its design[18]. Nowadays, most software development institutes give a lot of importance to the ability to work in teams, and research is always looking for methods that will improve the programmers' efficiency, productivity, and quality of work.

A. Agile Software Development

Agility may best be defined as "*the ability to both create and respond to change in order to profit in a turbulent business environment*"[20], which is an aspiration to most software development teams. Judging from this definition, as well as the Manifesto for Agile Software Development[21], flexibility is a requirement to successful software development. Many practices are being adopted by software development teams in order to achieve the concepts of flexibility, interaction, and collaboration, among others. Of the practices listed by Kent for Extreme Programming, pair-programming appears to stand out as a different and somewhat controversial practice.

B. Pair-Programming

Pair-programming may be defined as "*the practice whereby two programmers work together at one computer, collaborating on the same algorithm, code, or test*"[1]. Not only do the programmers work at one computer, but "*all production code is written with two people looking at one machine, with one keyboard*"[22].

A programming pair consists of a driver and a navigator. The driver's task is to actively type the code, and handle the keyboard, mouse, and any other input devices that are relevant. The navigator has to follow up with what is being typed on the screen, to catch any syntax mistakes, errors, or shortcomings of the code, in order to be able to correct and suggest better methods and solutions.

A key point that distinguishes the pair-programming practice is that the pair should always switch roles. Pair-programming is most beneficial when this happens regularly. When the two members of the pair each assume a role and stick to it for an extended period of time, the efficiency of this technique decreases, and becomes less apparent.

Some of the advantages of pair-programming that are repeated throughout the literature include improving design quality[1, 2, 6, 16, 19, 23], reducing defects[16, 19, 23], contributing to pair members' skills[8, 16, 24], improving team communications[15, 16, 19], and resulting in simpler code that is easier to extend[16]. In addition, some researchers have found that people working using pair-programming tend to spend more effort on the tasks they undertake[4].

C. Pair-Programming in Education

Research shows that when students take a programming course, they are usually in their first year of university, when it is important to get them used to help them gain the skills needed during their university years as well as integrating into the industry[3]. This is why most experiments carried out with pair-programming as a teaching technique is applied to introductory level courses[3, 5, 8, 14]. However, a number of researchers attempted to carry out the experiments on second-year and even advance students as well[6, 10].

Conducting a pair-programming experiment involves two issues to be considered: Pair formation and work assessment. Pair formation means the way students are paired. Pair formation could affect the dynamics of the pair, and might result in an effect on the outcome of the experiments. The methods followed for the formation of pairs differed among experiments. They varied between forming pairs according to their level of programming experience[3, 5, 6], random formation[5, 10], letting students form their own pairs[3, 5], or a combination of having students select a number of potential partners, and then assigning one of them according to experience[8]. Concerning the work assessment, in most the reviewed experiments, students were required to finish a number of assignments for which they were graded[3, 5, 6, 8, 10]. In addition, at least a final exam was given to assess students' performance in the course[8, 10]. A number of researchers collected feedback from their students as well, through questionnaires and surveys[3, 5, 6]. These questionnaires aimed to measure different parameters, such as students' confidence level after pair-programming, their perception of their compatibility with their partner, the effect of pair-programming on their understanding the exercises and course material, and their enjoyment of the course. Other researchers as in [27] opted for recording students while they were working in pairs, and analyzing the videos for certain parameters.

III. Research Methodology

Experiment Design

In an attempt to improve the level of students programming ability, in terms of their grades, their confidence, their pass rate, and their enjoyment we applied pair-programming method in an advanced programming course. This course is preceded by an introduction to programming course that is taught in C, which gives the basic of procedural programming. The advanced programming course is offered for sophomore-level students, and covers the basics of Object-Oriented programming and is taught in Java language. The course includes two one-hour lectures, and one three-hours lab per week, and spans over a 15-week semester.

The experiment was carried out twice during each of the fall and spring semesters of the educational year 2014-2015. In each semester, the experiment was conducted on two sections, one of which was a pair-programming based section, and the other was a control section where the course was taught the traditional way.

In both semesters, the two sections were taught by the same instructor and teacher assistant (TA). The lectures were given to both sections using the same methodology, and pair-programming was applied only during the lab sessions. Both sections in each semester contained the same number of students. In the first semester, both sections had 30 students each (30 males and 30 females), and in the second semester, both sections had 29 students each (32 males and 26 females). Because all those parameters were identical, we were able to select the pair-programming section randomly in both semesters with a coin toss.

The students were asked to work in pairs during the lab only. They were presented with the programming problems in their lab workbook, and asked to solve them while working on one computer. Students were instructed to switch roles constantly, which usually happened in between exercises. Students were also encouraged to discuss the problem before starting to solve it, and to avoid asking the instructor or the TA for help, unless they both fail to reach a solution.

The control section was not given any specific instructions. They continued with the lab sessions regularly after the fourth lab, with every student working on the assigned exercises on their own. Like the students of the pair-programming section, they were encouraged to try to reach the solution on their own.

Pair Formation

The literature showed two main methods of pair formation stood out for having more merits than others; random selection, and students selecting their own partners. As the experiment was spanning over two semesters, both methods were tried, in an attempt to determine which was more beneficial to the participants.

During the first semester, students were asked to select their own partner for the duration of the semester, following the methods illustrated by Teague[3] and Khan[5] in their research. As most of the students knew each other for around a year, they opted to select a friend rather than a work partner. On the other hand, students who did not have friends in the same section ended up in random pairs.

During the second semester, the students were distributed into pairs by the TA and the instructor. This was in accordance with the experiments presented by Khan[5] and Mendes[10]. The factors that were taken into consideration when distributing the pairs are their preference in working with a partner of the same or opposite gender, and the partner's academic level. The aim was to try to find the most compatible pairs according to the students' preferences.

Data Collection

The data collected throughout the semesters were in the form of questionnaire, course work assessment, and observations done by the instructor and TA during the labs.

- Initial Questionnaire: Before starting the experiments, a questionnaire was designed and distributed among the students. The questionnaire was designed after studying several similar questionnaires that were designed for similar experiments[3, 12, 25], while taking into consideration the background and mentality that distinguished our students from those in other countries. The questionnaire aimed to give us a general idea of the students' academic background, and their preference as to working in groups. The questionnaire also asked about the students' partner preference from an academic aspect, as well as their gender.
- Work Assessment: During the semester, the students took four quizzes and submitted four assignments. In addition, a midterm exam, a practical final exam and a written final exam were taken into consideration. The midterm exam took place during the semester, and the practical and written finals are scheduled at the end of the semester. Finally, the drop rate and the attendance and average absence from lectures and labs were taken into consideration.
- In-lab observations: Notes and observations were made both by the instructor and the TA during the labs in both sections. These observations pertained to the interactions between the students, the number and type of questions asked, the time required to complete tasks, and the degree of enjoyment of the labs session. In addition, in the lab sessions of the pair-programming section, it was observed how often students switched roles, how helpful and attentive were the navigators, and to what extent the tasks were discussed among each pair.

IV. Results and Discussion

Results

This section presents the results of the data that was collected and statistically analyzed during the two iterations of the experiment. This includes data collected from the questionnaire given to the students, and the code collected from students, as well as the grades of the students.

- Initial Questionnaire

This questionnaire was given to the students at the beginning of the semesters aimed to collect some demographic information about the students, their academic level, and their preferences to working within a team.

Regarding the students' preferences when it comes to working in teams, very few students indicated that they preferred to work individually, with 17% in the pair-programming section, and 20% in the traditional section. In addition to that, the vast majority of the students preferred to select their own partner. In the pair-programming section, 87% indicated that they preferred to select their partner, as did 93% of the traditional section. When it came to the partner gender most students did not have an issue with working with partners from either gender. However, most students indicated that they would prefer to work with a partner with the same programming level as they are. 53% of the pair-programming section students indicated that they would prefer to work with a partner in the same programming level as they were, and 23% of the students indicated wanting to work with a partner in a different programming level, whether it be higher or lower. In the traditional section, 56% of the students indicated wanting to work with a partner at their programming level, and 23% of them preferred to work with a partner with a different programming level than theirs.

- Code Quality

To understand the effect that pair-programming had on the quality of the code that was produced by the students, a sample of the students' code was collected and analyzed using SourceMonitor[1] application. Nine programs were collected from each of the pair-programming and traditional sections, and run through the software. SourceMonitor gives statistics about a number of parameter per program; It also counts the number of lines, statements, calls, and classes in the code. In addition, it gives the percentage of branches and comments. Moreover, it offers the ration of methods to classes, and statements to methods. Finally, it calculates the average and maximum complexity and depth. In addition to the statistics produced by SourceMonitor, the number of methods in each program was counted, as well as the number of syntax errors the software had. Fig. 2 shows that the code written by pair-programming section was shorter on average, by about a third (T(9)=-1.14, p<0.15), with fewer statements (T(10)=-1.11, p<0.15), and fewer calls (T(11)=-1.12, p<0.15). This resulted in significantly less errors (T(8)=-1.13, p<0.15), as shown in Fig. 3.

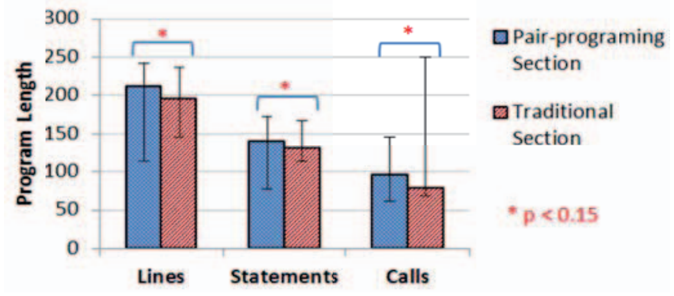


Fig. 2. Code Statistics from SourceMonitor Regarding Program Length.

[1] Available from: http://www.campwoodsw.com/sourcemonitor.html

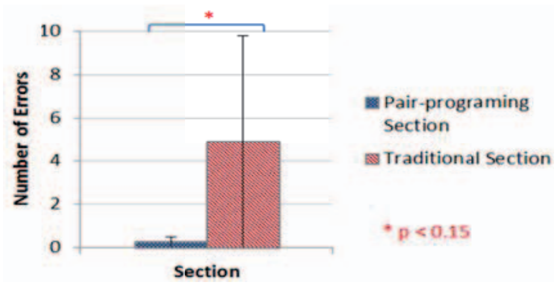


Fig. 3. Number of Errors per Program.

- Course Assessment

The course assessment was measured firstly through the grades that the students received during the semester, and secondly through the drop and completion rates of the students in the course. The grades were the results of four quizzes, four assignments, a practical exam, a midterm and final exams.

The quizzes were given during the lab, and often contained a written part, and a programming part. Fig. 4, shows that the pair-programming section's grades were significantly better in the first (T(108)=3.57, p<0.001) and third (T(94)=1.56, p<0.15) quizzes, as well as in the quizzes total grade (T(102)=3.11, p<0.001), but not in the second (T(94)=-0.15) and fourth (T(66)=1.37) quizzes.

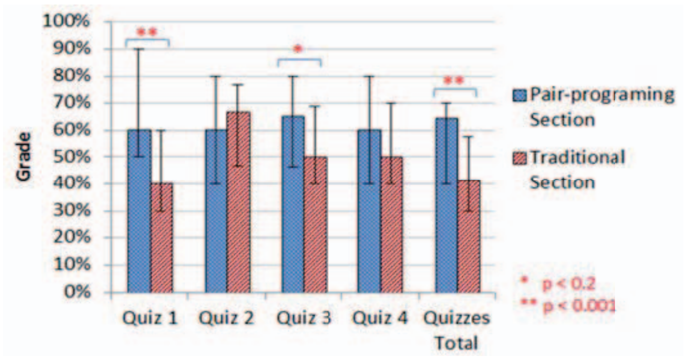


Fig. 4. Students Performance in Quizzes.

Fig. 5 shows the students' assignment results. The only significant difference was in the third assignment A3 (T(86)=1.54, p<0.2), and the assignments total (T(98)=1.67, p<0.1). However, in the first assignment A1 (T(98)=1.02), second assignment A2 (T(88) = -0.22), and fourth assignment A4 (T(50) = 1.27), the differences were not of significance.

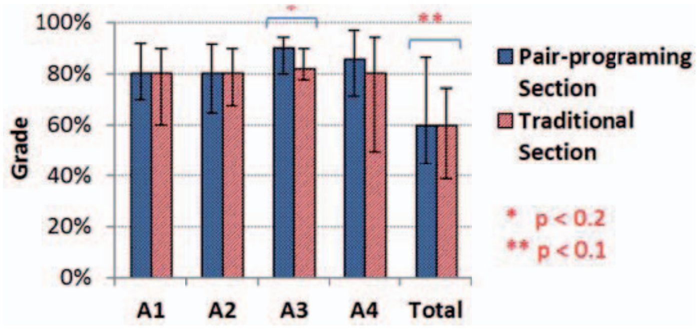


Fig. 5. Students Performance in Assignments.

At the end of the semester, a practical exam was given to the students. This exam was a programming exercise that covers all the programming concepts that were introduced throughout the semester. The students' performance in the quizzes, assignments, and the practical exam is combined to create their lab total. The lab work has a weight of 35% of their semester work. Fig. 6 shows the lab performance of the students of the pair-programming section and the traditional section. This results shows no significance between two sections with respect to practical exam (T(81)=0.57). However, the pair-programming section performed significantly better in lab total work (T(101)=2.43, $p<0.001$).

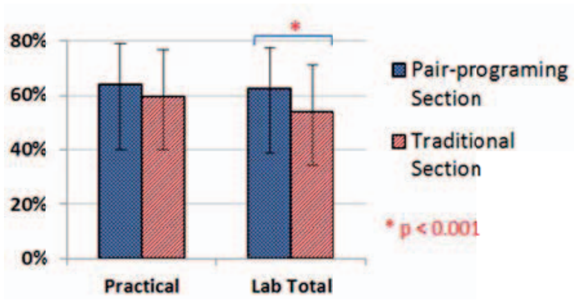


Fig. 6. Students Performance in the Practical Exam and Total lab grades.

The written exams during the semester were distributed on a midterm and final exams. Both sections took the same exams at the same time. The students in the pair-programming section performed better in the midterm (T(95)=2.51, $p<0.001$). However, in the final exam, the difference between both sections was not significant (T(87)=-0.38). The students' final grade was made up from the lab total, and the midterm and final exams. Fig. 7 shows that the pair-programming section performed significantly better in the overall result of the course (T(88)=1.61, $p<0.15$).

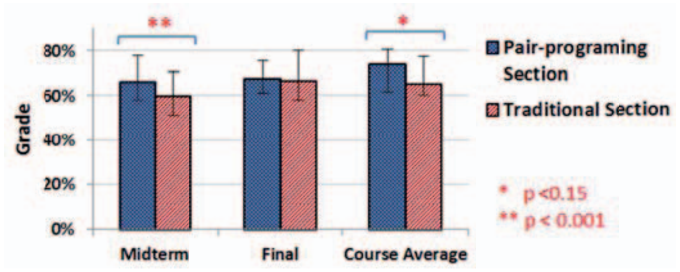


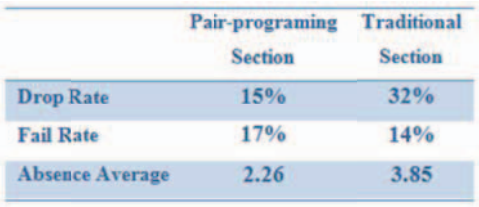
Fig. 7. Studnets Performance in Written Exams and Final Course Average.

The drop rate and absence rate in pair-programming section was less than half of the traditional section, as shown in Table 1. However, this affected the fail rate, making it a little higher in the pair-programming section (17%) than in the traditional section (14%). More students tended to miss class in the traditional section, with absences averaging near four students per class, where in the pair-programming section this rate was a around two students per class.

Discussion

This discussion attempts to put the data that was produced by the experiment into the context of the experiment, and the society. In addition, it offers the insights and experience of the researchers regarding the results.

TABLE I. THE STUDENTS DROP AND FAIL RATES, AND ABSENCE AVERAGE PER SEMESTER



	Pair-programing Section	Traditional Section
Drop Rate	15%	32%
Fail Rate	17%	14%
Absence Average	2.26	3.85

Before the experiment started, it was crucial to know whether the students were willing to work in pairs. Luckily, most students did not have any issues working with a partner, and the larger percentage of them (around 61% of all students) preferred working with a partner to working individually. The students' preference to selecting their partners or having a partner assigned by the instructor or TA was measured to direct the selection of the pair formation method. The majority of students (90% of the students in both sections) indicated that they preferred to select their own partner.

From observations, and experience, the parameters regarding the partner, whether they were the gender or programming level, did not have a clear effect on the students' performance. However, it was noted in one pair, where a male students and female students were paired together, that the male students, who has registered for the course during previous semesters, exhibited more commitment to the course, and put more effort during the lab, than he did in previous semesters. This could be the result of the student's act of proving his abilities in front of his female partner.

Since the students in the first semester selected their own partner, they tended to select a friend to work with, and they got along well together. However, some conflicts issued between a few pairs, and one student explicitly declared that if they were to work in pairs in other courses they would rather not work with the same partner.

When comparing the students' preferences in partners that they indicated in the initial questionnaire to the characteristics of the partner that the students selected, the selection was very random. There was no pattern or similarity between the indicated preference and the selected partner. This is attributed to the students selecting their friends to be their partners.

In the second semester, pairs were selected by the instructors randomly. The change in pair formation method did not seem to have an effect on the pair dynamic within the lab session. The students in the pairs got along together, and were working together to solve the exercises. However, the instructor and TA were approached with a request to switch partners by a couple of students.

From trying out two different methods of pair formation, and considering the results that were obtained, both approaches to pair formation achieved similar results. For this reason, we recommend following the approach which lets students select their own partners. Through this approach, students are already comfortable with their partner, and it reduces the overhead that pairing the students requires, which is a concern that was expressed by Gupta et al. in [13].

Measuring the code quality was a key to this experiment, because it is not enough to measure students' grades, and their perception of pair-programming. Referring to SourceMonitor results the following observations were made:

- Pair-programming students wrote shorter code as indicated in Fig. 2. Keating explains in [26] that shorter and simpler code is generally a better code. Also pair-programming students wrote code with more classes (7.63 classes per program on average), when compared to the code written by students in the traditional section (6.89 classes per program on average). This indicates that pair-programming students were able to modularize their code better than the traditional section students.

- On average, pair-programming section students' code contained less than one error (syntax or compilation) per program as illustrated in Fig. 3.

- Pair-programming section students commented their code more than traditional section students. SourceMonitor indicated that pair-programming section code had 10% comments, while the traditional section code had only 6%. Furthermore, a large percentage of the comments in the traditional section code were auto generated comments, which were auto generated by API.

The course assessment depended on the grades of four quizzes, four assignments, a practical exam, a midterm exam, and a final exam. These grades were distributed the same throughout both semesters. Regarding quizzes, the averages varied throughout the semesters, with the pair-programming section getting higher averages at times. However, the results of the t-test that was administered on this data, and illustrated in Fig. 4, showed that the differences between the two sections' averages were significant in the first and third quizzes, as well as in the quizzes total. The reason that the differences were not always significant is the number of students who missed the quizzes due to skipping lab sessions in the traditional section. This meant that mostly serious and hardworking students were present for the quiz, resulting in a better average than what would have been had all the students been present.

Likewise, the assignment results, as illustrated in Fig. 5, were not substantially different between the pair-programming and traditional sections, with the exception of the third assignment. This is understandable, since students worked on assignments at home, taking their time, and using whatever resources they required. In addition to that, the number of students that submitted their assignments was more in the pair-programming section, while students in the traditional section preferred to forego the submission of assignments they had trouble solving, rather than trying to find a solution or submitted work that was not complete. This might be an indication of the students' persistence and confidence in their ability to solve problems, which pair-programming is believed to enhance. However, the assignments total showed a significant difference between the pair-programming and traditional sections. This could be due to the fact that several students in the traditional section did not submit one or more assignment, as mentioned previously, resulting in a low assignments total, even though individual assignments had good grades.

The practical exam results, shown in Fig. 6, were fairly close in both semesters, and the t-test showed that they did not have much difference. This is due in the most part to the withdrawal of students from the traditional section. The weakest and most unconfident students were noticed to have withdrawn from the course during the period between the midterm exam and the practical exam. Therefore, the number of weak students in the pair-programming section was more than those in the traditional section, which affected the averages, and the significance of the difference in the results. The lab total were significantly better in the pair-programming section which can be reasoned by the fact that the students in the pair-programming section had more confidence in their work and were getting better grades than the students in the traditional section. In addition, they did not miss out on quizzes and assignments throughout the semester, resulting in better totals than those in the traditional sections.

It can also be noted that according to Fig. 7, the results of the pair-programming students in the midterm exam were significantly better than those of the traditional section students. However, since most withdrawals take place between the midterm and the practical exam time, the results of the final exam were not significantly different. Nevertheless, in the overall total of the semester, the difference between the pair-programming section's results, and the traditional section's results is significant with an 85% confidence.

Due to their higher confidence in their abilities, the number of students who withdraw from the course in the pair-programming section was almost half of the number of students who withdrew from the course in the traditional section. This can be attributed to the increase in the students' confidence in their ability to complete the requirements of the course, and their will to commit to it. This result comes in accordance with the findings of [2, 7-9, 14]. Likewise, the low absence rate can be caused by the enjoyment the students had in the pair-programming labs and classes. However, since less students drop out of the course, the chances of having students who are weaker increase, making it more likely to have some students, although still a low percentage, fail the course. This resulted in having the fail rate a little bit higher in the pair-programming section than in the traditional section.

General impressions are more difficult to measure, but informal chats with the students during the semester and the instructor and TA's observation indicated that:

- Students in the pair-programming section were enjoying the lab more.
- Students in the pair-programming section had more confidence in the code they were producing, whether it was for in-class tasks, quizzes, or assignments.
- Students in the pair-programming section often came to the lab with prior knowledge of the tasks that they were going to take, and prepared accordingly.
- Students in the pair-programming section were more interested in coming up with different ideas for programs to write.

V. Conclusion

The work described in this research was concerned with the implementation of pair-programming as a teaching technique in a Middle Eastern university. Our goal was to find out whether pair-programming will improve the learning experience of students within local interaction constraints. This research concludes that pair-programming has a potential to improve the overall performance of the students despite the restrictions. One of the aspects that can be improved by pair-programming is the quality of the code that is produced by the students. Students in the pair-programming section usually produced a code that had fewer errors, and was simpler and of better quality. Another aspect that may be improved is the students' performance in terms of grade. Students in the pair-programming section scored significantly better in a number of exams and assignments, and more importantly in the overall averages of the course. Moreover, pair-programming has the ability to increase the students' enjoyment in programming courses. Pair-programming technique forces students to interact with each other, allowing them to socialize within the classroom. This leads to students feeling more relaxed, allowing them to enjoy the lab sessions more. Finally, pair-programming has the potential to increase the students' completion rate and reduced the absence rate. Students who are enjoying the course, and have more confidence in their programming abilities tend to avoid dropping the course, and try to complete all the requirements needed to pass the course.

References

[1] A. Begel and N. Nagappan, "Pair programming: What's in it for me?," in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, (New York, NY, USA), pp. 120–128, ACM, 2008.

[2] N. Salleh, E. Mendes, and J. Grundy, "Empirical studies of pair programming for cs/se teaching in higher education: A systematic literature review," *Software Engineering, IEEE Transactions on*, vol. 37, no. 4, pp. 509–525, 2011.

[3] M. M. Teague, "Pedagogy of introductory computer programming: a people-first approach," 2011.

[4] O. S. G´omez, J. L. Bat´un, and R. A. Aguilar, "Pair versus solo programming–an experience report from a course on design of experiments in software engineering," *arXiv preprint arXiv:1306.4245*, 2013.

[5] S. Khan, L. Ray, A. Smith, and A. Kongmunvattana, "A pair programming trial in the cs1 lab," in *Proc. Annual International Conference on Computer Science Education: Innovation and Technology (CSEIT)*, pp. 6–7, 2010.

[6] A. B. Prabhakar, "Applying pair programming for advanced java course: a different approach," in *Proceedings of the 2011 conference on Information technology education*, pp. 319–320, ACM, 2011.

[7] X. He and Y. Chen, "Analyzing the efficiency of pair programming in education," 2015.

[8] C. McDowell, L. Werner, H. Bullock, and J. Fernald, "The effects of pairprogramming on performance in an introductory programming course," *SIGCSE Bull.*, vol. 34, pp. 38–42, Feb. 2002.

[9] N. Salleh, E. Mendes, J. Grundy, and G. S. J. Burch, "The effects of neuroticism on pair programming: An empirical study in the higher education context," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, (New York, NY, USA), pp. 22:1–22:10, ACM, 2010.

[10] E. Mendes, L. Al-Fakhri, and A. Luxton-Reilly, "A replicated experiment of pair-programming in a 2nd-year software development and design computer science course," in *ACM SIGCSE Bulletin*, vol. 38, pp. 108–112, ACM, 2006.

[11] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," *ACM SIGCSE Bulletin*, vol. 39, no. 2, pp. 32–36, 2007.

[12] K. Wood, D. Parsons, J. Gasson, and P. Haden, "It's never too early: Pair programming in cs1," in *Proceedings of the Fifteenth Australasian Computing Education Conference - Volume 136*, ACE '13, (Darlinghurst, Australia, Australia), pp. 13–21, Australian Computer Society, Inc., 2013.

[13] S. Gupta, V. Bhattacharya, and M. Singha, "Pair programming "potential benefits and threats"," *International Journal of Advanced Computer Research (IJACR)*, vol. 3, no. 1, 2013.

[14] L. Porter, M. Guzdial, C. McDowell, and B. Simon, "Success in introductory programming: What works?," *Commun. ACM*, vol. 56, pp. 34–36, Aug. 2013.

[15] S. Wray, "How pair programming really works," *IEEE software*, no. 1, pp. 50–55, 2010.

[16] A. Cockburn and L. Williams, "Extreme programming examined," 2001.

[17] K. Beck, "Embracing change with extreme programming," *Computer*, vol. 32, pp. 70–77, Oct. 1999.

[18] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns, and Java*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd ed., 2009.

[19] A. Sillitti, G. Succi, and J. Vlasenko, "Understanding the impact of pair programming on developers attention: A case study on a large industrial experimentation," in *Software Engineering (ICSE), 2012 34th International Conference on*, pp. 1094–1101, June 2012.

[20] J. Highsmith, *Agile Software Development Ecosystems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[21] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham,
M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," 2001.

[22] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional, 2004.

[23] E. di Bella, I. Fronza, N. Phaphoom, A. Sillitti, G. Succi, and J. Vlasenko, "Pair programming and software defects-a large, industrial case study," *IEEE Trans. Software Eng.*, vol. 39, no. 7, pp. 930–953, 2013.

[24] M. Giri and S. Soni, "Effectiveness of software development process using programmer ranker algorithm in pair programming," 2015.

[25] N. Salleh, E. Mendes, and J. Grundy, "Investigating the effects of personality traits on pair programming in a higher education setting through a family of experiments," *Empirical Software Engineering*, vol. 19, no. 3, pp. 714–752, 2014.

[26] M. Keating, "Good design of impossibly complex chips," 2013.

[27] T. Inoue, "Investigating the relation between behavior and result in pair programming: Talk and work leads to success," The Journal of Information and Systems in Education, vol.12, no.1, pp.39–49, 2014.