

Pair Programming in Graduate Software Engineering Course Projects

Shaochun Xu and Václav Rajlich
 Department of Computer Science, Wayne State University
 Detroit, MI48202 {shaochun, rajlich}@wayne.edu

Abstract - Pair programming has been used in undergraduate classes in order to develop student skills and to enhance student learning. Experiments with such an approach have demonstrated positive effects. This paper investigates the effects of pair programming in the graduate software engineering class by conducting a case study with six students who were assigned to work on incremental changes on an open source application either as pairs or as individuals. The results of the case study showed that paired students completed their change request tasks faster and with higher quality than individuals. They also wrote less lines of code and used more meaningful variable names. Pair programming could be an effective and useful approach for graduate software engineering classes.

Index Terms - Individual Programming, Graduate Class, Pair Programming, Software Engineering Education

INTRODUCTION

Software engineering is an important part of computer science curriculum at both undergraduate and graduate levels. Undergraduate classes often deal with the fundamentals of software development such as waterfall development model, whilst graduate level offers advanced topics including software evolution and maintenance.

The undergraduate software engineering class usually involves a project that gives students a chance to practice what they have learned in the theoretical part of the class. The traditional way of organizing the project is to form groups with 4-6 students and to use the waterfall software development life cycle to develop a software prototype [6].

Pair Programming (PP), one of the important practices of eXtreme Programming (XP), is a collaborative programming where two people work side-by-side in design, implementation, and testing while using one computer [1]. Pair programming has been used in industry and promising results have been reported. Nosek studied 15 professional programmers in both pair programming and individual programming situations and found that all pairs outperformed the individuals in terms of quality and time spent [12]. William (2000) did a survey of professional programmers and found that 100% were more confident in their solutions when using pair programming [20]. However recent studies demonstrates that the pair programming should not be used for either very large systems [19] or trivial problems [1] [10].

Pair programming has been applied in undergraduate courses and also demonstrated positive effects [20]. Students in pairs performed much better and produced higher quality work than when they worked individually. Students also benefited from their partners' ideas and suggestions, and their understanding of project's requirements and other knowledge broadened [22]. However, pair programming has rarely been used in graduate classes; particularly in graduate software engineering course projects.

In order to investigate the effects of pair programming in a graduate software engineering course, we designed a case study where the participants worked either in pairs or individually and made incremental changes to an existing large open source program. That allowed us to make a comparison between the pair programming and traditional individual programming. We also conducted a survey on pair programming.

We discuss projects in software engineering classes in the next section, followed by a discussion on pair programming in classroom projects. After that, our case study design and the survey results are described. This paper is concluded, in the last section, with issues for further investigation.

PROJECTS IN SOFTWARE ENGINEERING CLASSES

Software engineering in industry is a process of collaboration because most software is developed by a team. That is the reason why there is a team project component in the software engineering classes. According to [2], projects can allow student to learn developing complex systems, to experience simulated real-world development environment, to get a chance to learn from each other, to share their skills and knowledge, and to learn how to handle scheduling and other problems.

Sommervill [15] pointed out the critical goals of software engineering courses: to make students practice "programming in the large", to let them practice concepts learned in classroom, and to learn how to use software tools.

Peslak et al. [13] proposed to use collaborative methods to simulate the industrial environment.

The projects in undergraduate software engineering class are often run with a scaled down waterfall model in order to illustrate the basic concepts such as requirement, design, coding and testing. According to Gnatz (2002), the course projects using a group larger than 4 people often fail as a result of issues such as scheduling, and personal conflicts [6]. The failure of projects mainly comes from inefficient

exchange of information since too many people in a group sometimes cause substantial communication overhead. In fact, the projects are often completed by a few advanced students in the group, and the rest of the team learns less [6].

While previous papers reported experience with waterfall development, software evolution and maintenance were listed among the topics of the body of software engineering knowledge (SEEK) proposed by IEEE-CS and ACM [8], which is the basis for the education of software engineers. Since software maintenance takes substantial part of the software costs and most software engineers are actually maintaining the existing system, it is beneficial for students to work on evolution and maintenance projects. Pair programming described in the next section exclusively deals with software evolution or maintenance.

PAIR PROGRAMMING IN CLASSROOM PROJECTS

Several researchers used pair programming in computer science undergraduate courses and found it to be beneficial to students. William et al. [21] performed a case study in an introductory computer science course with 41 undergraduate students who formed two groups: one that worked in pairs and the other one that worked individually. The students were asked to complete four programming assignments. They found that paired students completed assignments faster and with higher quality. The defects of their programs were 15% lower and they also appeared to learn faster than the students who worked individually [21].

Hedin et al. (2003) used pair programming to teach second year software engineering class that had 107 students and reported a very positive result [7]. McDowell et al. (2002) found that students who worked in pairs implemented better programs and performed significantly better in exams than those worked individually [9]. They also noticed that fewer students in pair programming dropped the course than those who worked individually.

Williams et al. (2003) conducted a mass case study in introductory programming courses with over 1200 students. They found that paired students were more likely to complete the introductory course with a grade C or better, and their exam and project scores were better than those who worked individually [22].

The learning process involved in pair programming has been closely studied recently. Williams and Upchurch [20] conducted a case study in order to understand how the teaching and learning are enhanced by pair programming. They found that the students communicated with each other more effectively, appeared to learn faster, and were happier. The positive effect of pair programming on knowledge sharing during program design was also noticed by others [3].

The benefits of pair rotation have also been observed [17] [3]. The researchers found that students can learn from several partners, while the teachers have to deal with dysfunctional pairs.

However some disadvantages were also reported. Nawrocki and Wokciechowski [11] claimed that pairs spend nearly twice as much total effort than individual programmers.

The case study conducted by Gallis et al. [5] has shown that pair programming is inefficient for trivial programming tasks. The undergraduate students only have basic programming knowledge and they can not conduct pair programming on complex projects.

Schneider and Johnston (2003) noticed that due to the differences of experience between the two programmers in a pair, the stronger one did most of the job and the weaker one often failed to learn [5, 14], which often happens in undergraduate classes. Schneider and Johnston also observed that in order to fully understand the important elements in software development as well as the consequences of agile practices, students must reach certain level of maturity. Otherwise, agile practices such as pair programming might be misused [14].

Pair programming was also used in graduate classes, although not as frequently. Muller and Tichy performed a case study using pair programming with graduate students who were required to take a practical training course [10]. 12 participants completed three simple tasks and one project in pairs. They found it easy to adapt to the pair programming in graduate classes. However, some students commented that it was a waste of time to watch their partners to deal with trivial programming problems.

Stotts et al. (2003) did a case study with graduate students and compared the effects of pair programming in a distributed environment and found such an approach feasible and effective [18].

Based on that we designed our case study to see how well the graduate students actually perform in graduate software engineering course projects when using pair programming technique.

CASE STUDY DESIGN

In our design, we were led by our belief that the graduate students have sufficient knowledge of the programming that is required for pair programming, and also have an experience in cooperation that lessens the possibility of failure.

The case study design is based on the following null hypothesis: "Pair programming is not useful for graduate software engineering class" that is falsified by the case study. The rest of this section describes various facets of the case study design.

I. Participants

Six graduate students from Department of Computer Science, Wayne State University, participated in the study through their course project. The case study lasted for the whole semester. The students were experienced in programming in C, Java, or C++, but they have never performed pair programming before, nor did they have experience in software evolution. The programming experience of all the six students is shown in Table I. In the pair group, there are 3 Master students and one Ph.D. student. Two male students and two female students formed two pairs respectively. Since all of them had similar experience in OO programming and had never used pair programming technique before, there is no obvious bias in

such pairing. In the individual group, both are master students, one male and one female.

TABLE I
PROGRAMMERS' EXPERIENCE

	Pair 1 (A)	Pair 1 (B)	Pair 2 (A)	Pair 2 (B)	Single 1	Single 2
Years using Java language	3	1	2	5	3	2
Years using C++ language	5	4	1	2	2	2
Other languages known	Matlab	C, VB	C	Perl, Clips	C	C
Total lines of code having written in java	>1000	500	<1000	2000 - 3000	2000	>1000
lines of biggest programs having previously studied	>3,000	>1,000	5,000	>1,000	1000	1500

II. Experimental Tasks

The material for this case study is an open source project, JAdvisor [16]. JAdvisor is a class scheduler, course planner, and course search program written in Java. JAdvisor allows college students to view their schedules graphically and to create an optimal schedule. We selected this open-source application since it has reasonably big size with 31 classes, and satisfies Beck's criterion that the tasks for pair programming should take at least several hours [1]. JAdvisor application was also new to all the programmers.

The students were working on six distinct change requests that varied in difficulty. Here are the change requests:

Change request 1: Color coding schedule. JAdvisor allows the user to input block time to the schedule; however, there is no specific drop down menu to do so and each classification shows up with same color and therefore it is hard to recognize. We are to add a classification drop down box to the block time menu. It should display various classifications like study time, food break time, homework time, etc., with different color blocks on the schedule. With this way the user can easily identify their allocated times.

Change request 2: XML output. Current version of JAdvisor does not support XML output for class schedule, but we need it to output the schedule of the user. The XML structure must be clear and readable. Once this structure is documented the user should be able to save their output with the XML format, and also the program should be able to read the XML format files into both the planner and schedule tabs.

Change request 3: Planner wizard. JAdvisor has no planner wizard for all the required courses for a degree. We are going to create a planner wizard that allows the user to enter all courses necessary for their degrees. The information should be

saved in XML format when the user chooses. These courses should be shown in the planner tab on the right hand side. If a course has been taken, the course name should appear in red color and the user should not be allowed to select it.

Change request 4: Planner duplication. The planner is supposed to allow the user to plan all four years of their curriculum, but it allows for duplicates. Since students usually do not repeat their courses, JAdvisor should ask the user to overwrite if they do add a duplicate course either in the same semester or future semesters.

Change request 5: Pop-up dialog. The planner does not allow the user to select a course and then to display its information. We are going to add a pop-up dialog window which allows the user to click on a course in the current term and then a pop-up dialog should be able to display the course information.

Change request 6: HTML input. The current version of JAdvisor allows the user to output HTML, but it cannot read HTML files. The program should be able to read HTML files and fill in the scheduler as if they were saved.

III. Procedure

Prior to the case study, all subjects were given lectures on basic concepts of pair programming and incremental change such as concept location, change propagation and impact analysis. They were taught with how to do incremental change and how to conduct pair programming. They were also provided with JAdvisor website and the change requests.

One of the authors acted as the mentor who monitored the programming process for all the pairs. The programmers were required to document the time they used.

One pair and one individual programmer worked on change requests 1, 2 and 3, and the other pair and the other individual programmer worked on change requests 4, 5 and 6. They worked on those change requests in sequence.

After the case study, programmer pairs and individual programmers were asked to write a report on the process.

RESULTS AND DISCUSSION

In this section, we summarize the results of experiments and student survey on pair programming.

I. Case Study Results

All six participants have completed their work. In Table II, we presented the time the participants spent on each change request. Please note that the times listed in Table II are the duration that the pairs and individuals spent on the tasks, therefore the "total time cost" for a pair is the double of the time indicated. In all cases it is bigger than the total time cost spent by individuals who worked on the same tasks. On the other hand, programmer pairs worked on the tasks for a significantly shorter duration than individual programmers. This indicates the pairing can reduce the time of software evolution, which is consistent with the result for pair programming in software development reported by [4, 21]. However it may increase the cost of the development as the "total time investment" for two programmers is higher.

TABLE II
COMPARISON OF TIME (HOURS) USED BY PAIR AND INDIVIDUAL PROGRAMMERS

Subjects	Tasks completed in order	Pairs	Individuals
Pair 1/ Individual 1	Change 1	12	15
	Change 2	9	12
	Change 3	9	15
Pair 2/ individual2	Change 4	13	15
	Change 5	11	15
	Change 6	7	14

Change requests 1, 2 and 3 were done by one pair and change requests 4, 5 and 6 were done by second pair in sequence. From Table II, we can see that the time spent by the pairs on the change requests became shorter along the process; however, this phenomenon is not shown by the work of the individuals. This indicates that the process of learning is faster for the pairs than for the individuals. This agrees with the results claimed by Williams et al [21], who stated that pairing forces the two programmers to think aloud, making their comprehension strategy explicit, which enhanced the learning process.

TABLE III
CHANGE REQUEST RESULTS FOR PAIRS

Change requests	Pair		
	Lines of code	Numbers of classes changed	Numbers of classes changed unnecessary
1	110	3	1
2	87	3	0
3	56	1	0
4	64	4	0
5	82	3	0
6	67	3	0
Total	466	17	1

TABLE IV
CHANGE REQUEST RESULTS FOR INDIVIDUALS

Change requests	Individuals		
	Lines of code	Numbers of classes changed necessary	Numbers of classes changed unnecessary
1	125	3	0
2	104	3	0
3	86	1	2
4	102	3	1
5	78	3	0
6	75	2	0
Total	570	15	3

Tables III and IV show the numbers of lines of code added for each change request, the numbers of classes which have been correctly changed, and the numbers of classes which should not be changed but were modified by the programmer pairs or individuals. For six change requests, pair programmers added a total of 466 lines of code, but there are 570 lines of code written by individual programmers. The pair programmers also did most of the job correctly since they completed more required change propagations than individual programmers. The individual programmers failed to find 3

change propagations, and did unnecessary change in 3 classes. But the pairs only failed to find one change propagation out of total eighteen. Also the programs written by pairs have more meaningful variable names, which proves that with the help from the partner, programmer pairs are able to write higher quality code, which supports the same finding by [21].

II. Survey Results

A survey on pair programming was conducted with the survey questions listed in Table V through Table VII.

TABLE V
RESULT OF QUANTITATIVE QUESTIONS FROM THE SURVEY FOR PAIRS

Questions	Lowest (0.0)	Highest (5.0)	Answer (average)
How effective do you think pair programming was for the project?	Not effective	Very effective	4.00
Did you and your partner contribute equally to the project?	Very unequal	equal	3.20
What is your rating of your performance?	Did very little	Did most of work	3.8
What is your rating of your partner's performance?	Did very little	Did most of work	3.9
Do you think you learned more or less than you would have if you had worked on your own?	Much less	Much more	3.8
How do you think the time that you personally spent on this project compares to the time it would have taken you to do it on your own?	Pairs much slower	Pairs much faster	4.0
Would you like to use pair programming during your future graduate course project?	Not at all	Very like	4.6

In general, students thought that pair programming is an effective way for the course project. All students were satisfied with the performance by themselves and their partners; however, they thought the contributions from the two persons in the pair were different, which is consistent with the two roles in the pair: one is the driver who controls the keyboard and the other one is observer performing code review and helping driver to make decisions [1]. It seems that a role rotation is needed in order to further promote learning between the two programmers in the pair as mentioned in [17]. The students also believed that pair programming can actually saves their time in the projects, which corresponds to the shorter project durations that were indicated in Table II. The two pairs reported that they enjoyed the pair programming technique more than them programming alone and are willing to apply it in their future projects. It seems that the programmer pairs had higher motivation to finish the tasks than individual programmers, which also contributes to the time difference on the tasks by the pairs.

The two individuals who participated in the case study were also conducted a survey via email. The questions and results are shown in Table VII. It seems that the two

individuals expected positive effects in applying pair programming in the course projects and they were also interested in using it in near future.

TABLE VI
ANSWERS OF QUALITATIVE QUESTIONS FROM THE SURVEY FOR PAIRS

Questions	Answers
Do you feel like you have learned anything just by reading your partner's code?	<ul style="list-style-type: none"> • Yes, I think so • Yes, I agree • No, I do not feel that way • No, I did not learn more
What was the biggest problem you have had to overcome as a paired programmer?	<ul style="list-style-type: none"> • No problem • I do not think there is a problem • The pair should know how to deal with different options • Partners should have similar knowledge
What do you think is the biggest concern in pair programming?	<ul style="list-style-type: none"> • Matching of the two programmers • Having agreement • Not being ready for big problem • Time conflict
What are the advantages of pair programming?	<ul style="list-style-type: none"> • Pair works faster • Pair explores more design and implementation options • Pair has more confidence • Pair can share ideas

Based on the case study and survey results, it seems that pair programming and the large application like JAdvisor can work as graduate software engineering projects since the paired programmers produced higher quality results within shorter time, compared with those working in individual. The programming ability of the pairs was better than that of individuals based on Tables II, III and IV.

TABLE VII
ANSWERS OF QUALITATIVE QUESTIONS FROM THE SURVEY FOR INDIVIDUALS

Questions	Answers
Do you think it would be more efficient if you would have worked with a partner?	<ul style="list-style-type: none"> • Yes, I think it would • Yes, I agree, but not sure
Do you think you learn more if you would have worked with a partner than when you worked alone?	<ul style="list-style-type: none"> • Yes • Yes
How do you think the time that you would spend with a partner compared to the time you spent on your own?	<ul style="list-style-type: none"> • It depends on the partner. If the team was on the same level, it would be more efficient and require lesser time. But if the partner was learning then I don't see any advantage. • Possibly we would use less time by working as pair
How do you think the pair programming impacts a quality of the program?	<ul style="list-style-type: none"> • Better than with a single person project • It should have higher quality
If you are allowed to choose in future course projects, would you prefer to work with a partner or work alone?	<ul style="list-style-type: none"> • With a partner • I love to pair with a friend to see the results

With pair programming, learning has been enhanced greatly, since the knowledge is constantly exchanged between partners, which include those concepts learned in the classroom, tool usage tips, programming language, design

skill, and debugging techniques. Pair programming affects learning and motivation, which has been demonstrated by the result of the survey.

Nevertheless, using the pair programming concept in graduate class projects certainly has some obstacles. The first one is how to pair students. Letting students themselves decide who pair with is often a good option, which can minimize the incompatibility. The second issue is the types of programming problems for the projects. Theoretically, any program problem with medium size is suitable for pair programming. However, some problems may motivate students to learn more than others. The third obstacle is the scheduling problem. If the two students have scheduling problems, they cannot work on the project.

CONCLUSIONS AND FUTURE WORK

Pair programming increasingly attracts attention not only in the industry, but also of academia. Pair programming has been reported to have benefits in software development and in undergraduate classroom.

Because of maturity, a good command of programming knowledge, experience in cooperation, and the moderate size of graduate software engineering class projects, graduate students are ideal candidates to use pair programming in their classroom projects as well.

We performed a case study on pair programming in software evolution at graduate software engineering class with six graduate students, who worked as either pairs or individuals. All the subjects worked on change requests on open source software: JAdvisor.

The case study showed that programmer pairs consume shorter time, but at an increased cost because of the extra labor involved. Programmer pairs completed the tasks with higher quality than those done by individuals since they only missed one change propagation out of eighteen in total, but individual programmers missed three. They have also written higher quality code, such as with less lines of code and more meaningful variable names. According to the survey, all students are satisfied with the performance by themselves and their partners and they are willing to use pair programming in future. Based on the case study and survey results, it seems that it is a good idea to develop graduate software engineering course project with pair programming. Using pair programming technique to organize the projects not only forces students to learn, but also gives them some fun.

For the future work, we would like to replicate this case study in order to collect more data and to further validate our observation. We also plan to conduct the case study on different sizes and different kinds of problems to study those effects.

REFERENCES

[1] Beck, K., *Extreme programming explained*, Massachusetts, Addison-Wesley, 2000.
 [2] Brereton, P., Lees, S., Gumbley, M., Boldyreft, C., Drummond, S., Layzell, P., Macaulay, L., and Young, R., "Distributed group working in

- software engineering education", *Information & Software Technology*, vol. 40, 1998, pp. 221-227.
- [3] Canfora, G., Cimitile, A., and Visaggio, C., "Working in pairs as a means for design knowledge building: an empirical study", in Proceedings of International Workshop on Program Comprehension, 2004, pp. 62-69.
- [4] Cockburn, A. and Williams, L., "The costs and benefits of pair programming", in Proceedings of Extreme Programming and Flexible Processes in Software Engineering, Cagliari, Italy, 2000, pp. 223-243.
- [5] Gallis, H., Arisholm, E., and Dyba, T., "A transition from partner programming to pair programming - an industrial case study", in Proceedings of Pair programming workshop in 17th annual ACM conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), 2002, pp. 1-8.
- [6] Gnatz, M., Kof, L., Prilmeier, F., and Seifert, T., "A practical approach of teaching software engineering", in Proceedings of 16th Conference on Software Engineering Education and Training, 2003, pp. 140-147.
- [7] Hedin, G., Bendix, L., and Magnusson, B., "Introducing software engineering by means of extreme programming", in Proceedings of International Conference on Software Engineering, 2003, pp. 586-593.
- [8] IEEE-CS and ACM, "Software Engineering Education Knowledge (SEEK): Second Draft, available at <http://sites.computer.org/ccse>, Dec", 2002.
- [9] McDowell, C., Werner, L., Bullock, H., and Fernald, J., "The effects of pair-programming on performance in an introductory programming course", in Proceedings of SIGCSE Technical Symposium on Computer Science Education, Cincinnati, Kentucky, 2002, pp. 38-42.
- [10] Muller, M. and Tichy, W., "Case study: extreme programming in a university environment", in Proceedings of 23rd International Conference on Software Engineering, Toronto, Canada, May 2001, pp. 537-544.
- [11] Nawrocki, J. and Wokciechowski, A., "Experimental evaluation of pair programming", in Proceedings of European Software Control and Metrics (Escom), 2001.
- [12] Nosek, J. T., "The case for collaborative programming", *Communication of the ACM*, vol. 41, no. 3, 1998, pp. 105-108.
- [13] Peslak, A., "Teaching software engineering through collaborative methods", *Issues in Information Systems*, vol. 1, no. 1, 2004, pp. 247-253.
- [14] Schneider, J.-G. and Johnston, L., "eXtreme programming at universities - an educational perspective ", in Proceedings of 25th International Conference on Software Engineering, Portland, Oregon, 2003, pp. 594-599.
- [15] Sommerville, I., *Software Engineering*, Addison-Wesley, 1996.
- [16] SourceForge.net, "JAdvisor: <http://jadvisor.sourceforge.net/>",
- [17] Srikanth, H., Williams, J. G., Wiebe, E., Miller, C., and Balik, S., "On pair rotation in the computer science course", in Proceedings of the 17th Conference on Software Engineering Education and Training, 2004, pp. 144-149.
- [18] Stotts, P. D., Williams, L. A., Nagappan, N., Baheti, P., Jen, D., and Jackson, A., "Virtual teaming: experiments and experiences with distributed pair programming", in *Extreme Programming and Agile Methods - XP/Agile Universe 2003, Third XP and Second Agile Universe Conference, New Orleans, LA, USA, August 10-13, 2003. Lecture Notes in Computer Science*, New Orleans, LA Springer, 2003, pp. 129-141.
- [19] Tuck, D., France, R., and Rumpe, B., "Limitations of agile software processes", in Proceedings of the Third International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2002), Alghero, Italy, 2002, pp. 43-46.
- [20] Williams, J. G. and UpChirch, R. L., "In support of student pair-programming", in Proceedings of the 32th SIGCSE technical symposium on Computer Science Education, 2001, pp. 327- 331.
- [21] Williams, L., Kessler, R., Cuningham, W., and Jeffries, R., "Strengthening the case for pair-programming", *IEEE Software* July/August 2000, pp. 19-25.
- [22] Williams, L., McDowell, C., Nagappan, N., Fernald, J., and Werner, L., "Building pair programming knowledge through a family of experiments", in Proceedings of International Symposium on Empirical Software Engineering, 2003, pp. 143-152.