Measuring the Effects of Virtual Pair Programming in an Introductory Programming Java Course

Nick Z. Zacharis

***Abstract*—This study investigated the effectiveness of virtual pair programming (VPP) on student performance and satisfaction in an introductory Java course. Students used online tools that integrated desktop sharing and real-time communication, and the metrics examined showed that VPP is an acceptable alternative to individual programming experience.**

***Index Terms*—Achievement, Java, Java course, satisfaction, virtual pair programming (VPP).**

I. Introduction

Although group work is the norm for professional software development, in traditionally taught programming courses, students are still required to work individually, very often experiencing high levels of frustration and isolation. Pair programming (PP) is a collaborative programming style in which two programmers sit side-by-side at the same computer; discuss what they are about to program; analyze, design, and then write code; and test and debug it, with both continuously participating in the development effort. Using one keyboard and one mouse, the two programmers assume either the role of the "driver" or the "navigator." The driver is typing the code and always explains to the navigator what s/he is doing and what s/he is thinking. The navigator watches carefully and asks the driver to clarify when something is unclear, always thinking of alternatives and the implications of the developing code. The partners regularly switch roles, and both programmers review the design and code in real time, resulting in a highly interactive, adaptive development process [1].

Research has attributed many benefits to the use of PP versus individual programming in terms of productivity, quality, trust and teamwork, knowledge transfer, and enhanced learning. Nosek reports from an experiment with professional programmers where five pairs and five individuals worked on the same algorithm (a database consistency check script) for 45 min [2]. Pairs completed their work in 30 min (12 min less than individuals), spending overall 42% more effort (effort is twice the time of the pair), and produced better code than did individuals. The pairs were also more confident in their final solutions and all enjoyed solving the problem collaborating together. Williams also conducted numerous studies on the efficacy of pair programming for instruction and found that pair programming is an effective lab experience [3]. Students in paired labs produce better code (with only 15% more effort and also 15% fewer defects), have a positive attitude toward collaborative programming settings, and show at least similar performance on the exams when compared to solo programming students.

Many researchers have also experimented with the implementation and efficiency of distributed PP, using either screen-sharing applications (such as Microsoft NetMeeting, Symantec PCAnywhere, or VNC), or domain-specific collaborative environments that integrate editors and communication tools. Baheti *et al.* measured the quality and productivity of distributed pairs that used tools like VNC, NetMeeting, and instant messengers and found that distributed pairs produced comparable code to that of collocated teams, with respect to the productivity (lines of code per hour) and quality (test subjects' grades), and showed a higher level of communication and collaboration [4]. Hanks also compared two groups of students, one remote and one collocated, using VNC with a modification that allowed a second cursor for the navigator and found no statistically significant differences in students' grades on assignments and final examination [5].

In the study presented here, students used virtual pair programming (VPP) to complete four homework programming assignments, each of two weeks duration; they were not collocated in the laboratory. Pairs used groupware technologies and wrote code in the convenience of their own home, collaborating with their partner in real time when they had free time to concentrate on their task. The course organization and the arrangement of the experiment on the effects of VPP, as well as the results on students' performance and satisfaction, are described in the following sections.

II. Course Organization and Methodology

Introduction to Computer Programming, COMP 120, is a beginning-level programming course at the Technological Educational Institute of Piraeus, Greece. The primary goal is to teach students basic elements of programming, object-oriented programming, and problem-solving skills. This course is taught in the second semester of the first year and is appropriate for students with no prior programming experience. There are no strict prerequisites, but a basic background in math and computer skills is required. Students are supposed to feel comfortable using a computer as an everyday tool (for example, using a Web browser, writing e-mail, using word-processing applications, downloading and installing software). The Java language is used to introduce foundations of structured, procedural, and object-oriented programming.

An experiment was conducted in the COMP 120 course, among the 129 students enrolled in the Fall 2007 semester, aiming to assess the effectiveness of VPP in an effort to move the course to a more learner-centered and collaborative direction. Traditionally, the course is taught over 12 weeks, with two 1-h lectures and one 2-h lab each week, and student grades are based on one midterm exam, one final exam, and eight homework programming assignments completed by students working on their own. In each lab, there are about 20 students working independently to modify the code and extend the functionalities of a sample program, after explanations are given and goals set by the instructor.

In this semester under study, half of the students (65) completed all their assignments individually as usual (solo section), while the others (64) used PP and collaborated upon the last four assignments (VPP section). During the first four weeks, all students completed homework assignments individually, and they had a strict deadline of one week to submit their solutions. Since the course uses an "objects first" approach, and concepts such as classes, objects, and methods are introduced as early as the first week, weekly feedback and appropriate scaffolding at the beginning is crucial to ensure that individual students understand object-oriented programming as it applies to Java.

After the midterm exam at the end of the fifth week, students in the VPP section were randomly assigned a partner according to their grades in the four first assignments to ensure that each team included members with approximately equal prior knowledge and abilities. All VPP students attended an orientation lesson in the laboratory and collaborated in pairs on short projects, using NetMeeting in order to become familiar with its desktop-sharing (running NetBeans IDE), chat, and video conference features. Students in both sections, VPP and solo, had to record the time needed to complete each programming assignment (successfully running all the accompanied test cases), the lines of code

Manuscript received June 11, 2009; revised January 05, 2010; accepted April 02, 2010. Date of publication May 17, 2010; date of current version February 02, 2011.

The author is with the Technological Educational Institute of Piraeus, Athens 12244, Greece (e-mail: nzach@teipir.gr).

Digital Object Identifier 10.1109/TE.2010.2048328

TABLE I
RESEARCH HYPOTHESES

H1	Students who use VPP on programming assignments will produce better programs faster than solo programmers.
H2	Students who work virtually in pairs will earn exam scores equal to or higher than solo programming students.
H3	Students in pairs enjoy PP and will have a positive attitude towards collaborative programming settings.

TABLE II
LINES OF CODE AND DEVELOPMENT TIME

LOC				Development time		
HW	VPP	Solo	Δ LOC %	VPP	Solo	ΔT%
5	134.30	139.20	3.52	5.97	6.86	74.09
6	144.60	152.80	5.37	5.28	6.70	57.49
7	162.80	175.90	7.45	4.98	6.84	45.48
8	169.40	184.50	8.18	5.20	6.96	49.27

per hour (only executable lines and data declarations), and the number of defects/bugs.

III. RESEARCH HYPOTHESES AND METRICS

This experiment was aimed at examining the effects of pair programming in introductory programming courses. The hypotheses that were tested are listed in Table I.

For the testing of H1, two factors were examined: code productivity and software quality. Generating more code, faster, and that being of high reliability, is a real challenge, especially for novice programmers; pair programming, according to previous research, motivates students to succeed in this difficult effort. Data from midterm and final examinations was used to compare the performance of VPP and solo students and conclude on H2. A survey of students' perceptions of PP was administered in class before the final test and gave VPP students the opportunity to evaluate the new technique. Statistical analysis of the survey responses gave the instructor data with which to test H3.

IV. RESULTS

A. Code Productivity

In this experiment, project productivity was calculated as the amount of work, measured as lines of code (LOC), divided by the effort used, measured as the development time for each assignment. Measuring project development time means just summing up all hours spent on design, programming, testing, and bug fixing. The numbers of LOC and the time elapsed for the development of each programming assignment are presented in Table II.

On average, the number of LOC written by pairs was approximately 6% less than the number of LOC produced by solo students, but the t-test showed the difference was not significant at the 0.05 level. On the contrary, significant difference was found between the person-hours needed for each assignment. Although the VPP teams had better development times in all cases, comparing pair effort (doubling the time of the team) with that of individuals, an average of 57% more effort was needed from pairs for writing the same amount of LOC. This increment in pairs' effort lies near the findings of Nosek [2]. This result implies that VPP is rather an expensive technique, at least as implemented by novice programmers. Since collaboration is the goal of any student-centered approach to learning, the increased effort of pairs is not a big problem. On the other hand, a reliable metric should be used to determine the result of collaboration, including the cost in development time.

TABLE III
PRODUCTIVITY MEASUREMENT IN LINES OF CODE PER HOUR

LOC/hr	VPP		Solo		T-test	
HW	mean	sd	mean	sd	t-value	p-value
5	22.5	2.70	20.3	3.60	3.37	0.001
6	27.4	3.90	22.8	3.80	5.50	0
7	32.7	4.54	25.7	4.78	7.01	0
8	32.6	4.04	26.5	5.45	6.20	0

TABLE IV
DEFECTS PER 1000* LOC

Dfs/KLOC	VPP		Solo		Difference	T-test
HW	mean	sd	mean	sd	Δ mean %	t-value
5	53.61	15.47	96.98	27.65	80.90	9.87
6	67.43	18.32	115.3	32.73	71.02	9.22
7	49.75	13.71	80.73	26.21	62.25	7.64
8	71.13	21.62	115.7	35.22	62.68	7.68

In Table III, results of a t-test analysis for productivity, measured in LOC/h, show significant differences ($p < 0.05$) between VPP and solo sections, with pair students more productive than solo programmers. Although a single dimensional measure such as LOC/h gives a good picture of individual or pair productivity, it is a poor measure if the desired level of software quality is not taken into account.

B. Code Quality

Software quality measurements are related to the absence of defects that would cause a program to behave unpredictably or stop successful execution. Empirical studies have found that defect density (number of defects normalized to the lines of code) is significantly related to the LOC count [6], [7]. Large Java programs have more lines of code and, consequently, more variables, more operators, more statements, and more complexity. Long methods or excessively deeply nested conditionals increase cyclomatic (or conditional) complexity, require extra mental effort from the reader to grasp their logic, and tend to correlate to defects [7]. During the semester, students were reminded repeatedly to minimize the number of method parameters since objects with methods that take many variables have low encapsulation and, thus, low cohesion.

From Table IV, it is apparent that pairs produced code of higher quality with about 50% fewer defects ($p < 0.05$). Although students were instructed to count only logical/design-type defects or syntax-like defects that were not flagged by the compiler, it was difficult to assess the seriousness of those defects or if they were discovered before or after testing. In any case, given that all students had the same previous programming background, it seems that VPP members, through collaboration and continuous code reviewing, improved code quality. This result agrees with the findings of many previous studies that have reported smaller defect counts for pair programming [1]–[3].

C. Students' Performance

Students' grades on their programming assignments were used as a direct measure of their ability to program. In Table V, mean scores and standard deviations in each of the four assignments are given for VPP and solo students, while a t-test analysis indicates the differences between them.

Although VPP students achieved better scores, there were no statistically significant differences in any of these assignments between them and solo students at the 0.05 level. Both sections showed a progressive improvement in their scores, partly due to the nature of the assignments, which integrated classes used in previous corrected assignments, and

TABLE V
STUDENTS' SCORES IN EACH PROGRAMMING ASSIGNMENT

Scores	VPP		Solo		T-test	
HW	mean	sd	mean	sd	t-value	p-value
5	74.3	13.3	68.8	14.5	1.86	0.067
6	76.5	26.2	72.9	24.4	0.65	0.517
7	79.6	28.3	75.8	27.2	0.73	0.467
8	81.3	27.2	78.4	27.1	0.45	0.621

TABLE VI
STUDENTS' SCORES ON EXAMINATIONS

Scores	VPP		Solo		T-test	
Exam	mean	sd	mean	sd	t-value	p-value
Midterm	65.4	12.3	66.2	13.5	0.29	0.77
Final	78.7	16.7	75.9	14.6	0.81	0.422

TABLE VII
STUDENTS' RESPONSES TO SURVEY QUESTIONS

	SA	A	N	D	SD	mean	sd	Positive %
Q1	28	31	5	0	0	4.36	0.62	92.19
Q2	32	29	2	1	0	4.44	0.63	95.31
Q3	37	23	2	2	0	4.48	0.71	93.75
Q4	21	33	4	4	2	4.05	0.96	84.38
Q5	27	25	5	6	1	4.11	1.00	81.25

partly due to continuous code exchanging and discussion through the class forum.

Both midterm and final examinations were individual in-class tests designed to assess students' comprehension and problem-solving ability on short programs, methods, or classes. A comparison of midterm and final examination scores for both sections, using t-test analysis, revealed no significant differences between pairs and solo students (see Table VI).

D. Students' Attitude and Perceptions

As a whole, the 64 students in the VPP section had a highly positive attitude toward pair programming. Students ranked the following statements with $\text{Strongly Disagree} = 1$, $\text{Disagree} = 2$, $\text{Neutral} = 3$, $\text{Agree} = 4$, and $\text{Strongly Agree} = 5$.

Q1: I enjoyed programming with a partner more than programming alone.
Q2: Pair programming motivated me to stay on task.
Q3: Interacting with my partner in real time helped me think at a higher level and understand difficult concepts.
Q4: I was more efficient in debugging my code while working in continuous communication with my partner.
Q5: Pair programming increased my confidence in my solutions to programming assignments.

In Table VII, students' rankings in the survey questions are shown in detail. Mean values ranged from 4.05 to 4.48 [all in the area of Agree (A) and Strongly Agree (SA)], and positive attitude (summing answers of SA and A) was over 80% in every question. Students enjoyed VPP (92%) better than programming alone, and they felt (95%) that their partners' pressure had significant impact on motivating them to stay on task. These two findings are very encouraging, considering the unavoidable differences in time schedules or in personalities or even in skill levels.

Students agree (93%) that real-time interaction helped them to look deeper in programming concepts and gain knowledge from the continuous reviewing process. The majority of students perceived that PP increased their efficiency in debugging their code (84%) and their confidence in submitting their programs for assessment (81%). Students' perceptions are in accordance with previous studies that reported that students like PP, believe this programming style improves software quality, and feel more confidence in their PP-derived solutions [3], [4].

V. CONCLUSION

The goal of this experiment was to determine whether students collaborating in pairs through online technologies can produce code of the same functionality and quality as that of students programming alone. The majority of pairs used applications that provide desktop sharing and real-time communication (at least audio and chat), while over 30% of the pairs experimented with collaborative editors, especially with CollabEd, which was proposed by the instructor. In most cases, simple solutions like NetMeeting and the Remote Desktop Sharing feature of Windows, accompanied with free VoIP applications like Skype, worked perfectly.

The findings of this study confirm the three hypotheses set up at the beginning. The test of hypothesis H1 revealed that students who used VPP on their assignments produced code of better quality, had about 50% fewer defects, and were more productive in LOC/h. The comparison of academic performance of solo and VPP students was based on the scores they achieved in four programming assignments, a midterm, and a final exam and has shown that there was no difference between VPP and solo programming students. The examination of students' satisfaction toward PP, through their responses in the survey questionnaire, confirms hypothesis H3, that students in pairs perceive PP as a positive learning experience. Based on these results, this study suggests that VPP is an effective pedagogical tool for flexible collaboration and an acceptable alternative to individual programming experience.

REFERENCES

[1] K. Beck, *Extreme Programming Explained*. Reading, MA: Addison-Wesley, 2000.
[2] J. Nosek, "The case for collaborative programming," *Commun. ACM.*, vol. 41, no. 3, pp. 105–108, 1998.
[3] L. Williams and R. Kessler, *Pair Programming Illuminated*. Reading, MA: Addison-Wesley, 2002.
[4] P. Baheti, E. Gehringer, and D. Stotts, "Exploring the efficacy of distributed pair programming," in *Proc Extreme Program. Agile Methods*, 2000, pp. 208–220.
[5] B. Hanks, "Empirical evaluation of distributed pair programming," *Int. J. Human-Comput. Studies*, vol. 66, pp. 530–544, 2008.
[6] C. Withrow, "Error density and size in Ada software," *IEEE Softw.*, vol. 7, no. 1, pp. 26–30, Jan. 1990.
[7] S. Kan, *Metrics and Models in Software Quality Engineering*. Reading, MA: Addison-Wesley, 1995.