Evaluating the Effects of Virtual Pair Programming on Students' Achievement and Satisfaction

doi:10.3991/ijet.v4i3.772

Nick Z. Zacharis
Technological Education Institute of Piraeus, Athens, Greece

***Abstract*—Pair programming is a lightweight software development technique in which two programmers work together at one computer. In literature, many benefits of pair programming have been proposed, such as increased productivity, improved code quality, enhanced job satisfaction and confidence. Although pair programming provides clear pedagogical benefits, its collocation requirement and the limited time during a lab session are serious barriers in the full deployment and evaluation of this programming technique.**

This paper reports on a study that investigated the effectiveness of Virtual Pair Programming (VPP) on student performance and satisfaction in an introductory Java course where students worked collaboratively in pairs on homework programming assignments, using online tools that integrated desktop sharing and real time communication. The results of this study support previous research findings and suggest that VPP is an effective pedagogical tool for flexible collaboration and an acceptable alternative to individual/solo programming experience, regarding productivity, code quality, academic performance and student satisfaction.

***Index Terms*—Virtual Pair Programming.**

I. INTRODUCTION

Originating from industrial settings where collaborative software development is the norm, pair programming has come recently in the centre of interest of computer science educators, in an effort to transform the solitary activity of learning to program into a collaborative problem solving process. The principal idea behind problem-based learning is that students are presented with a problem and they begin working in small groups and negotiate in defining the problem precisely, assess what they know, identify what they need to know, plan how to proceed, gather information, collaborate on the evaluation of hypotheses, brainstorm possible solutions, choose one solution, look back and reflect and arrive at clearly stated solutions.

Collaborative exchanges in problem solving tasks extend cognitive activity and team members are able to monitor individual thinking, cope with different opinions, give and take feedback that results in clarification and change, and provide social support and encouragement to each other [1]. Pair programming, as well as test-driven development and refactoring, lies at the core of extreme programming, a discipline of software development based on values of courage, communication, feedback, and simplicity. Extreme programmer teams initially build code according to a simple design and through testing, continuous feedback and design improvement, they keep it that way.

Pair programming especially is a practice in which two programmers work at one computer, collaborating on the same design, algorithm, code or test. Sitting side-by-side and assuming the roles of "driver" and "navigator" or "observer" the two programmers discuss about the current implementation, possible alternatives and errors, searching for a better algorithm to use, optimising parts of the code, creating functional tests for every piece of code and finding better functions or libraries to call. The driver has the control of the keyboard and actively implements the code while the navigator looks at the driver's work and identifies tactical and strategic defects and issues [2]. From time to time, the developers switch their roles, so that both equally develop code.

II. WHY TO USE PAIR PROGRAMMING?

Williams at NCSU, examining the effectiveness of student pair programmers, references many studies that have shown that pair programming creates an environment conducive to more advanced, active learning and collaboration, leading to students being less frustrated, more confident, and more interested in information technology [3][4]. Although computer programming is known to be a complex skill that is difficult to master, pair programming has been shown to produce improved outcomes: better quality software, faster code production, fewer defects, increased programmer confidence in solutions and greater enjoyment [4][5][6].

Williams and Kessler observing effective pairs concluded on several behaviors that tend to happen naturally and contribute to the achievement of the benefits of pair programming: pair pressure, negotiation, courage, reviews, debugging, learning and pair trust [6]. The collective ownership of all that is produced, the need not to let down their partner, the encouragement when the other is stuck and the collaborating review and testing keep partners much more focused on the task at hand and help them improve their programming skills.

III. VIRTUAL PAIR PROGRAMMING

Communication is the core function of negotiation and cooperation that allows information and expertise to be exchanged between team members. The close physical proximity, considered a basic element in the delivery of

prompt responses during pair programming, is frequently a serious limitation e.g. when teams have geographical barriers or scheduling conflicts that prevent them from being collocated. The inevitability of distributed work in industry and education forces software developers to adopt online technologies and implement VPP in order to have the benefits of pair programming in the online environment.

A. Issues and solutions for VPP support

Rapid advances in computer networks and internet technologies have made it possible for developers from different geographical locations to form virtual teams in a distributed setting and jointly develop the same artefact of software in a collaborative way [7]. To allow effective implementation of VPP it is necessary to introduce team awareness support that is closely comparable to collocated pair programming practices, where physical proximity facilitates easy and quick communication, collaboration and coordination.

Dourish and Bellotti define team awareness as "an understanding of the activities of others, which provides a context for your own activity" [8]. Team awareness allows each user to be informed about others users' activities and track the changes that other collaborators have made to a group project. Maintaining awareness in virtual settings requires additional effort and the extensive use of various communication tools such as newsgroups, MUDs, email, text chat, and instant messaging [9].

Effective real time communication is necessary for the virtual pair members to obtain information on how each other react on something the other says or does. To have a physical sense of a reaction, information on body gestures, the faces or even the tone of the voice of the other person is required.

Collaboration is a term that can be used to indicate any form of interaction between software developers, in co-located or distributed settings, working on the same set of artefacts. In the context of VPP, collaboration involves regular synchronous meetings, planning and negotiation, design decisions that are translated into code, regular code integration, and ongoing communication between pair members during the development lifecycle. A crucial point for successful collaboration in general, is the manner in which individual work is related to the team as a whole.

The collaboration among people who are engaged in a common task requires the coordination of the activities related with the task and of the resources used during its execution. Concerning coordination, being in two different locations, the challenge for virtual pairs consists on how to synchronize their availability, adjust the time differences, and integrate their activities [10].

Successful implementation of VPP assumes the existence of a collaborative environment designed to support communication and awareness with a collection of tools that allow pair members to access, manage, and share information through an integrated information infrastructure. In the domain of computer-supported cooperative work (CSCW), virtual collaboration is supported mostly by groupware applications that provide the tools for synchronous/real-time team activities such as pair programming.

There are two basic approaches for VPP via the Internet. The first one is to send screen-buffer information through the network, broadcasting the display of any application from a member to all the others. Using screen sharing applications such as Microsoft NetMeeting, Symantec PCAnywhere, VNC (or one of its derivatives RealVNC, TightVNC, etc) or the built in tool "Remote Assistance" of Windows XP, virtual pairs can view a common desktop and control remotely an IDE (Integrated Development Environment) compiler, writing code in turns. Free and reliable instant messaging/ video conferencing applications like Paltalk, Skype and ooVoo, integrating video, audio and text chatting, accompany screen sharing applications and connect pair members providing presence awareness and immediate interaction. Free web conferencing applications such as Microsoft NetMeeting (installed in both members' boxes) and Dimdim or vRoom from Elluminate, combining audio/ video conference with desktop shearing and interactive whiteboard, provide an integrated information facility for effective collaboration.

The second approach toward VPP is the design of collaborative environments that are domain-specific and integrate collaborative editors with versioning and configuration control tools (such as CVS), databases, videoconference facilities and email systems [10]. Stand alone programs like SubEthaEdit, UNA or NetEdit, and any IDE that support collaboration, such as Eclipse with various plug-ins such as Sangam, QuickShare or Eclipse Wiki Editor, can be used to facilitate the collaborative aspects of VPP. In contrast to desktop sharing systems, these collaborative editors are event driven programs, transmitting only messages that are important for pair programming, and thus they don't need high bandwidth connections. Using a stand alone collaborative editor like UNA, all users who have opened a particular project space see the same set of opened files, the same chat history, the same notes, the same whiteboard, and can freely move in and edit all parts of a document, without locking. Workspace awareness, i.e. the knowledge of the state or actions of other participant, is evenly high in collaborative editors using plug-ins like Sangam, DocShare or Jazz for Ecliplse IDE. These plug-ins benefit from all platforms features such as syntax highlighting and auto-indentation, adding their own capabilities for version control, tracking of local changes and defects, simultaneous editing even in the same line, easy floor handling and role switching.

B. Previous experiments on VPP

Baheti et al studied the effectiveness of VPP measuring the quality and productivity of distributed pairs. They used tools like VNC, Microsoft NetMeeting and instant messengers and found that VPP was a feasible way of developing object-oriented software and as effective as collocated pair programming. The virtual 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 [11].

Hanks conducted an experiment on VPP using VNC with a modification, to allow for a second cursor that can be controlled by the navigator to point at areas of the screen without affecting the driver's state. Two groups of students, one remote and one collocated, were compared on performance on assignments and final exam. No statistical significant differences were found on students' achievement between collocated and virtual pairs [12].

Stotts et al experimented for 5 weeks with 4 distributed pairs of graduate students, having two of them work as virtual synchronous pairs (utilizing VPP) and the remaining two work as more traditional virtual teams (no pair programming). All pairs used Microsoft NetMeeting for code development and Yahoo Messenger for voice communication. The number of test cases passed was the metric used for program quality while productivity was calculated as the mean total time for development, without any concern for lines of code measures. The researchers found that VPP teams wrote 70% more unit test cases than the non-pair programming virtual teams, satisfied all the test cases and competed their projects in about 60% less time [13].

Natsu et al conducted an experiment with 9 pairs (5 distributed and 4 collocated in the laboratory) using their COPPER system, a synchronous source code editor that allows two distributed software engineers to write a program using pair programming. After a 90 minutes session students evaluated the usability of the tools provided by the editor, the usefulness of the floor control and the awareness mechanisms and the adequacy of the communication. All pairs had near equal perception about the usefulness of the system, indicating the feasibility of VPP with tools that provide the means for simultaneous code editing and presence and workspace awareness [14].

The experiments described above indicate that effective collaborative software development is possible with a few simple, widely-available tools (screen sharing, Internet-based audio communications) [13]. The aim of the present study was to evaluate the implementation of VPP in an introductory programming course and the impact that would have on students' achievement and satisfaction. In the following sections we describe the course organization, the experimental design, the research questions, the metrics that were used, and the results that were obtained.

IV. COURSE ORGANIZATION

Introduction to Computer Programming COMP 120 is a beginning level programming course. 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 (e.g., using a web browser, writing email, using word processing applications, downloading and installing software).

The Java language is used to introduce foundations of structured, procedural, and object-oriented programming. Topics include I/O, data types, operators, operands, expressions, conditional statements, iteration, recursion, arrays, functions, parameter passing, and returning values. Students are also introduced to classes, objects, object references, inheritance, sorting, polymorphism, exception handling, searching, Java Collections, and Applets. COMP 120 is required for computer science majors, electrical engineering majors, and also for the Computer Programming Certificate, a program designed to enable students with undergraduate degrees or working professionals to upgrade their programming skills or make a career change.

V. THE EXPERIMENT

An experiment was conducted in the COMP 120 course, among the 129 students enrolled in 2007 fall semester, aiming to assess the effectiveness of VPP in an effort to move the course to a more learner centred and collaborative direction. Traditionally, the course is taught during 12 weeks, with two hour lectures and one two-hour lab each week, and student grades are based on one midterm exam, one final exam and 8 homework programming assignments completed by students working on their own. In each lab room there are about 20 students working independently to modify the code and extent the functionalities of a sample program, after the explanations and goals given by the instructor. Usually, the lab time is never enough for the majority of students to finish the tasks assigned them, such as compiling the code, testing it, debugging it and refining it, and much work remains to be done at home, augmented by additional tasks defined in every homework programming assignment after the end of the lab period.

This semester, half of the students (65) completed all their assignments individually as usual (solo section), while the others (64) used pair programming and collaborated upon the last 4 assignments (VPP section). During the first 4 weeks all students completed homework assignments individually and they had strictly 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, providing 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 5th week, students in VPP section were randomly assigned a partner according to their grades in the 4 first assignments to ensure that each team included members with approximately equal previous knowledge and abilities. In general the literature indicates that small differences in cognitive level are more conductive to effective collaboration and cognitive growth than larger differences [15]. Although other studies support precisely the opposite and suggest the novice expert pairing, we chose the first approach as more suitable in a freshman class.

All VPP students were given a brief introduction to pair programming technique and instructed to switch regularly roles between driver and navigator and respond constructively to feedback, in order to keep an objective view about the direction in which the program is going and look for the strategic implications of the developing code. They were cautioned to avoid the temptation to break projects into smaller parts to be completed independently and tolerate partner's questions or comments as a necessary part of the process, without becoming annoyed or upset. As pair programming requires real time collaborative effort, they had to spend over 70% of the total time on an assignment interacting synchronously with their partner. All VPP students attended an orientation lesson in the laboratory rooms and collaborated in pairs on short projects using NetMeeting in order to become familiarized 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 assign-

ment (running successfully all the accompanied test cases), the lines of code per hour (only executable lines and data declarations) and the number of defects/bugs. Instructor made clear to students that all the data that would be gathered was to be used only for evaluating the programming technique of pair programming and the only thing that they had to worry about was as always the quality and functionality of their programs. Although all students in pairing section were assigned partners from their class in order to avoid schedule conflicts, they were allowed to re-pair if scheduling difficulties arose in meetings or even to work alone if insurmountable problems occurred.

VI. 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.

To decide on H1 we examined two factors: code productivity and software quality. Generating more code faster and of high reliability is a real challenge especially for novice programmers and pair programming according to previous research motivates students to succeed in this difficult effort. Data from midterm and final examination were used to compare the performance of VPP and solo students and conclude on H2. A survey about students' perceptions of pair programming 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 us data to test H3.

VII. RESULTS

A. Code productivity

Over time, there have been many attempts to define metrics that effectively measure software development productivity. Most of them are amazingly complicated and very difficult to apply. In this experiment project productivity was calculated as the amount of work, i.e., lines of code, divided by the effort used, i.e., the development time for each assignment. Measuring project development time means just summing up all hours spent on design, programming, testing and bug fixing. In Table II are presented the numbers of lines of code (loc) and the time elapsed for the development of each programming assignment.

The number of loc written by pairs was approximately 7% less than the number of loc produced by solo students but the difference was not significant at 0,05 level. On the contrary, significant difference was found between the completion times of each assignment. Although VPP teams had in all cases better development times, 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, who reported that the pairs spent on the average 42% more effort than individuals (the difference was not statistically significant) [16]. This result implies that VPP is rather a time consuming technique at least as implemented by novice programmers.

TABLE I.
THE RESEARCH HYPOTHESES

H1	Students who use VPP on programming assignments will produce better programs faster than solo programming students.
H2	Students who work virtually in pairs will earn exam scores equal to or higher than solo programming students.
H3	Students in pairs enjoy pair programming 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***	***Δ time %***
5	134,30	139,20	3,65	5,97	6,86	74,09
6	144,60	152,80	5,67	5,28	6,70	57,49
7	162,80	175,90	8,05	4,98	6,84	45,48
8	169,40	184,50	8,91	5,20	6,96	49,27

TABLE III.
PRODUCTIVITY MEASUREMENT (LOC 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

In Table III, results of a t-test analysis for productivity measured in loc/hr, show significant differences ($p < 0,05$) between VPP and solo sections, with pair students to be more productive than solo programmers. This result is in agreement with the findings of the experiments made by Baheti et al, who found that collocated pair teams performed better than solo programmers but did not achieved statistically significantly better results than VPP teams, and Lui and Chan, who concluded that pair programming achieves higher productivity when a pair writes a more challenging program that demands more time spent on design [11][17]. Although a single dimensional measure of productivity, such as loc/hr, gives a good picture of individual or pair productivity, it is evidence that productivity is a poor measure if desired level of 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. As each line of code is a potential point of failure and takes time to plan, type, review, and debug, fewer lines of code reduce failures and increase coding speed. On the other hand, clever code takes longer to plan, review, and debug and can also have more points of failure per line of code. During the course, students instructed that variables, operators and

statements are the real points of failure and reducing those is what truly reduces defects.

From Table IV is apparent that pairs produced code of higher quality with about half 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 conclude 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 which have reported smaller defect counts for pair programming [3][4][5][6].

TABLE IV.
DEFECTS PER KLOC

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

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 0,05 level. The better degrees that achieved on average the students that worked in pairs reflect the higher code quality (more compact code for the same functionality and fewer post-delivery defects), maintainability (program's ability to stay the same or to adapt to change), performance, and documentation quality. Both sections showed a progressive improvement in their scores partly due to the nature of the assignments that integrated classes used in previous ones and had been corrected and partly due to continuous code exchanging and discussion through the class forum.

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

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). Both midterm and final examinations were written individually in-class tests designed to assess students' comprehension and problem solving ability on short programs, methods, or classes. It is surprising that although both sections had almost identical performance in midterm examination, VPP students performed better in final examination. This is a strong indication that on average pair members benefited from collaboration in gaining better understanding of fundamental programming concepts and object oriented techniques.

D. Students' attitude and perceptions

As a whole, the 64 students in VPP section had a highly positive attitude toward pair programming. Students ranked the following statements with strongly disagree = 1, disagree = 2, neutral = 3, agree = 4, 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 under continuous communication with my partner.
- Q5: Pair programming increased my confidence in my solutions to programming assignments.

TABLE VII.
STUDENTS' SCORES ON EXAMINATIONS

	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

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 and strongly agree) and positive attitude (summing answers of strongly agree SA and agree A) was over 80% in every question. Students enjoyed pair programming (92%) instead of programming alone and they felt (95%) that their partners' pressure had significant impact on motivating them stay on task. These two findings are very encouraging taking into consideration the unavoidable differences in time schedules or in personali-

ties or even in skill levels. One reason that justifies this result is that students in pairs had similar abilities. Research has shown that pairing students of similar abilities motivates them work most effectively and compatibly with their partners [18]. Another reason may be the flexibility of the distributed programming they experienced using tools that permitted them to work anytime anywhere.

Students agree (93%) that real time interaction helped them to look deeper in programming concepts and gain knowledge from the continuous reviewing process. Defect removal was also much more efficient (84%) resulting in higher confidence in the produced code (81%). Students' perceptions are in accordance with previous studies reported that students like pair programming, believe that this programming style improves software quality and feel more confidence in their solutions [4][5][6][11].

VIII. CONCLUSIONS

The findings of this study confirm the three hypotheses we set up at the beginning. Testing hypothesis H1 we conclude that students who used VPP on their assignments produced code of better quality, with about half fewer defects and were more productive in Loc/hr. Comparing students' performance based on the grades they achieved in each programming assignment and on midterm and final test, we found no difference between VPP and solo programming students. This result confirms the H2 hypothesis that VPP students would have at least equal scores on programming projects and exams as their solo counterparts. Examining students' satisfaction towards pair programming through their responses in the survey questionnaire, we confirm hypothesis H3 that students in pairs perceive pair programming as a positive learning experience.

Students in pairs used on line technologies that provide desktop sharing and real time communication (at least audio and chat) and collaborated when it was convenient for both partners. In conversations with VPP students, the instructor was being informed frequently about the tools they were using and the difficulties they were experiencing. 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 results of this study suggest that VPP was an effective pedagogical tool for flexible collaboration and an acceptable alternative to individual programming experience. Our intention is to conduct more experiments like this so that we can draw conclusions about pair programming technique in general and its virtual/ distributed form particularly.

REFERENCES

[1] M. Alavi, "Computer-mediated collaborative learning: An empirical evaluation, " *MIS Quarterly*, vol 18, pp. 159-174, 1994. (doi:10.2307/249763)

[2] G. Canfora, A. Cimitile, G, Di Lucca, C.Visaggio, "How distribution affects the success of pair programming", *International Journal of Software Engineering and Knowledge Engineering*, vol 16, pp. 293-313, 2006. (doi:10.1142/S0218194006002756)

[3] L. Williams, C. McDowell, N. Nagappan, J. Fernald, L. Werner, "Building Pair Programming Knowledge through a Family of Experiments," ISESE, pp.143-152, International Symposium on Empirical Software Engineering (ISESE'03), 2003.

[4] L. Williams, "Lessons Learned from Seven Years of Pair Programming at North Carolina State University", ACM SIGCSE Bulletin, vol 39, pp. 79-83, 2007. (doi:10.1145/1345375.1345420)

[5] K. Beck, *Extreme Programming Explained*: *Embrace Change*, Reading, MA, Addison-Wesley Professional, 1999.

[6] L. Williams & R. Kessler, *Pair Programming Illuminated*, Reading, MA, Addison-Wesley Professional, 2002.

[7] M. Reeves, J. Zhu, "Moomba - A collaborative environment for supporting distributed extreme programming in global software development", In Proceedings of XP 2004, Lecture Notes in Computer Science, vol. 3092, Springer, Berlin, pp. 38–50, 2004.

[8] P. Dourish & V. Bellotti, "Awareness and Coordination in Shared Workspaces", In Proceedings of CSCW'92, Toronto, pp. 107-114, 1992.

[9] C. Gutwin, R. Penner and K. Schneider, "Group awareness in distributed software development"**.** In Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work. Chicago, Illinois, November 2004.

[10] A. Morán, J. Favela, R. Romero, H. Natsu, C. Pérez, O. Robles and A.Enríquez, "Potential and Actual Collaboration Support for Distributed Pair Programming", *Computación y Sistemas*, vol 11, pp. 211-229, 2008.

[11] P. Baheti, E. Gehringer, D. Stotts, "Exploring the efficacy of distributed pair programming", In Extreme Programming and Agile Methods—XP/Agile Universe 2002, Lecture Notes in Computer Science, vol. 2418, Springer, Berlin, pp. 208-220, 2002. (doi:10.1007/3-540-45672-4_20)

[12] B. Hanks, "Empirical evaluation of distributed pair programming", *International Journal of Human-Computer Studies*, vol 66, pp. 530-544, 2008. (doi:10.1016/j.ijhcs.2007.10.003)

[13] D. Stotts, L. Williams, N. Nagappan, P. Baheti, D. Jen, and A. Jackson, "Virtual teaming: Experiments and experiences with distributed pair programming", In Extreme Programming and Agile Methods - XP/Agile Universe 2003, Lecture Notes in Computer Science, vol 2753, Springer, pp. 129-141, 2003.

[14] H. Natsu, J. Favela, A. Moran, D. Decouchant, A. Enriquez, "Distributed pair programming on the web", In Proceedings of the Fourth Mexican International Conference on Computer Science, pp. 81-88, 2003.

[15] R. Slavin , *Cooperative Learning: Theory, Research and Practice*, Prentice Hall, 1990.

[16] J. Nosek, "The Case for Collaborative Programming", Communications of the ACM, vol 41, pp. 105-108, 1998. (doi:10.1145/272287.272333)

[17] K. Lui and K. Chan, "When Does a Pair Outperform Two Individuals?", In M. Marchesi and G. Succi (Eds) Extreme Programming and Aile Processes in Software Engineering - 4th International Conference, XP 2003 Lecture Notes in Computer Science vol 2675, pp. 225-233, 2003.

[18] L. Williams, L. Layman, J. Osborne, N. Katira, "Examining the Compatibility of Student Pair Programmers", In Proceedings of the conference on AGILE 2006, pp. 411-420, 2006.

[19] G. Melnik and F. Maurer, "Perceptions of agile practices: A student survey", In Proceedings of Extreme Programming and Agile Methods 2002, pp. 241–250, 2002.

AUTHOR

Nick. Z. Zacharis is with the Technological Education Institute of Piraeus, Athens, Greece (e-mail: nzach@ teipi.gr).

Submitted 15 December 2008. Published as resubmitted by the author on 9 August 2009.