Original Article

INVESTIGATING THE IMPACT OF DYNAMIC PAIRS ON REAL TIME PHARMACEUTICAL SOFTWARE QUALITY ASSURANCE

Dr. Uma Kannan[1], Dr. Rajendran Swamidurai[2]

[1]Alabama State University, Montgomery, AL, USA.
[2]Alabama State University, Montgomery, AL, USA.

DOI: 10.47750/pnr.2023.14.02.221

Abstract

Many successful technological firms today are built on pair programming. Pair programming is a technique for inspecting code as it is written; it is also a real-time problem solving and quality assurance activity. Two people, a driver, and an observer, use a single keyboard and mouse to program all production code on a single system in pair programming. The driver is responsible for implementing the current module at hand. The navigator is an online reviewer tasked with ensuring the quality of the code. We were unable to determine the true impact of pair programming on software quality assurance because many pair programming studies were not conducted as described in Extreme programming and the empirical evidence of pair programming is contradictory. We investigated the effect of dynamic pairs (pair rotation) on real-time software quality assurance in terms of program accuracy and code quality in this study. Our research indicates that pair programming with frequent pair rotation produced a higher quality software system and code in less time than pair programming without pair rotation, and produced higher quality software and code than individual programming in approximately the same amount of software development time.

1. Introduction

Pair programming [1] is a form of collaborative programming in which two people, one of whom is designated as the "driver" and the other as the "observer," work together to write all of the production code on a single computer while sharing a single mouse and keyboard. Pair programming allows real-time problem solving and quality assurance by inspecting code as it is created [2]. During pair programming the driver focuses on operational matters, such as developing the code for the current module. The observer simultaneously focuses on tactical and qualitative issues, such as identifying errors as the driver types the code and potential extra testcases for the current module to enhance the quality of the code [1].

Pair programming is being marketed as a practical alternative to the more traditional method of individual programming, with its proponents claiming that the practice results in higher quality code, increased productivity, and a significant reduction in the overall costs associated with the development of software in a shorter period of time [3]. Despite the fact that the notion of pair programming is appealing, there is conflicting empirical evidence supporting the benefits of pair programming. While studies by Wilson et al. [4], Nosek [5], Williams [6], McDowell et al. [7], and Xu and Rajlich [8] support both the costs and advantages of pair programming, those by Nawrocki and Wojciechowski [9], Vanhanen and Lassenius [10], Arisholm et al. [3], Rostaher and Hericko [11], and Hulkko and Abrahamson [12] shows that there is no noticeable difference between programming alone and programming as a pair.

Pairing (pair programming) is one of the 12 practices described in the Extreme programming [1], and it is a dynamic process, according to its creator Kent Beck. Pairing is a continuous conversation between two developers/programmers who are attempting to do multiple things such as analysis, design, code, and test at once. Nevertheless, the bulk of pair programming studies undertaken to date have been static pair programming trials, in which the same two individuals were put up to solve one or more programming problems. According to a number of academic studies [4,5], partnering helps the team quickly share system knowledge as it is being developed. However, the question of how the system knowledge would be distributed among the team members with static pairs (i.e., without rotating the partners) emerges in this situation. [18]

According to the findings of several research studies conducted by Arisholm et al. [3], Williams et al. [6], Hulkko and Abrahamsson [12], Muller and Tichy [13], and Lui and Chen [14] pair programming is beneficial for the resolution of difficult programming tasks. The findings of the experiment conducted by Vanhanen and Lassenius [10] demonstrate, on the other hand, that pair programming does not facilitate the resolution of difficult programming tasks. Additionally, a number of studies, including Arisholm et al. [3], Hulkko and Abrahamsson [12], and Muller [15] show that the correctness of programs created by pairs and individuals was the same. These raises concerns about the observer's function and value as a person doing in-process quality control during the pair programming process.

Due to the fact that the majority of the pair programming experiments were not carried out in the manner that was described (especially about rotating the pairs) in Extreme programming and that the empirical evidence of pair programming is mixed, we were not able to get a clear picture of the impact that pair programming has on the process of software development [18] as well as on the real-time quality control of software development.

To gain a more accurate understanding of the benefits of pair programming in software development, it is hypothesized that frequent pair rotation during software development improves software quality compared to pair programming without pair rotation and individual programming.

2. Literature Review

2.1. Pair Programming and Program Correctness

In pair programming, the quality of the product or the accuracy of the program is measured in terms of defect density, which is measured in terms of the number of test cases passed [6,8] and/or relative defect density (defects/KLOC) [6,12].

2.1.1. Static Pair Programming and Program Correctness

Williams et al. [6] and Xu and Rajlich [8] both demonstrate that pairs of programs passed more test cases than single programmers did. Matthias Müller [16] demonstrates that the degree of correctness of programs created by pair groups and review groups is comparable. According to Arisholm et al. [3], the pairings did not generate more accurate programs than individuals.

According to Vanhanen and Lassenius [10], programs developed in pairs had fewer faults than those written by individuals after coding and unit testing, but more defects than those written by individuals after system testing and bug fixing. Although Hulkko and Abrahamson [12] demonstrate that pairs created code with a higher defect density, Williams et al. [6] claim that pairs programs had a lower defect density.

2.1.2. Dynamic Pair Programming and Program Correctness

The research of Nawrocki and Wojciechowski [9] reveals that the approach of dynamic pair programming is more predictable than the technique of individual programming, and that the amount of rework for dynamic pair programming is slightly less than for individual programming.

2.2. Pair Programming and Code Quality

In pair programming, the code's quality is evaluated based on its functionality, the amount of software components it contains, and its readability, the number of comments it contains, as well as its elegance and readability [4,5,8].

2.2.1. Static Pair Programming and Code Quality

The results of the experiment conducted by McDowell, Werner, Bullock, and Fernald [7] demonstrate that pair programming increases the functioning and readability of programs. Hulkko and Abrahamsson [12] found that pair programming produced code with a weaker adherence to coding standards; yet, the code was more readable. Xu and Rajlich [8] demonstrate that the programs created by pairs were more legible and elegant, however Wilson et al. [4] and John Nosek [5] demonstrate that there was no statistically significant difference between the individual and pair programmer's codes in terms of readability.

In terms of functionality, the John Nosek [5] experiment demonstrates that pair programs were more functional than individual programs, however in the Wilson et al. [4] experiment, individual programs were more functional than pair programs. Moreover, Vanhanen and Lassenius's [10] research demonstrates that pair programs are less functional than individual programs.

2.2.2. Dynamic Pair Programming and Code Quality

According to the research of Nawrocki and Wojciechowski [9], dynamic pair programming produces more stable solutions than individual programming.

2.3. Pair Programming and Software Development Time

2.3.1. Static Pair Programming and Software Development Time

John Nosek [5], Williams et al. [6], Nawrocki and Wojciechowski [9], Rostaher et al. [11], Matthias Müller [16], Xu and Rajlich [8], and Arisholm et al. [3] demonstrate that pair programmers need more time than individual programmers to complete a task. Moreover, Nawrocki and Wojciechowski [9] and Rostaher et al. [11] demonstrate that pairs of programmers required nearly twice as much time as individuals.

2.3.2. Dynamic Pair Programming and Software Development Cost

The research of Nawrocki and Wojciechowski [9] indicates that dynamic pair programming is more expensive than solo programming. It also reveals that there is essentially no difference between dynamic pair programming and individual programming methodologies in terms of development time.

3. Experiments

Three empirical tests were undertaken to validate the impact of static and dynamic pair programming. Although the participants were university students, the research was not part of any course. The subjects were placed into three groups: static pair programming (SPP), dynamic pair programming (DPP), and individual programming (IP), the control group. All experimental groups solved three programming problems of varied difficulty using Java and the Eclipse integrated development environment (IDE).

3.1. Subjects and Group Assignment

Ten computer science seniors took part in the second and third control investigations, whereas 18 undergraduate seniors and graduate students volunteered for the first control experiment. Prior to the start of the actual control experiments, each subject was required to independently solve two programming problems in order to evaluate their level of programming proficiency. They were also required to respond to a survey questionnaire in order to provide demographic data regarding their experience with programming, software testing, test-driven development, and previous pair programming. Six metrics were gathered from the pretest and survey, and based on these six factors, a score was given to each subject as follows:

Score=$\mathbf{A}*\mathbf{6}+\mathbf{B}*\mathbf{5}+\mathbf{C}*\mathbf{4}+\mathbf{D}*\mathbf{3}+\mathbf{E}*\mathbf{2}+\mathbf{F}*\mathbf{1}$

Where,

A: Java Programming Skills/Knowledge (1-below average, 2-average, 3-good, 4-very good, 5-excellenet)

B: Programming Analysis Skills from the pretests' results average. Analysis Score = (Test1+Test2)/200

C: Programming Experience (1-less than one year, 3-one to five years, 5-more than five years)

D: Knowledge of black-box testing (1-Yes, 0-No)

E: Knowledge of JUnit (1-Yes, 0-No)

F: Knowledge Test-driven development (1-Yes, 0-No)

The subjects were separated and put into groups of five based on their score. Three experimental groups—dynamic pair programming (DPP), static pair programming (SPP), and individual programming (IP)—were formed by randomly selecting participants from each group.

3.2. Hypothesis

We have tested the following hypotheses to validate the experiment:

- H01 (Program Correctness DPP Vs SPP): The Program Correctness of dynamic pair programming is equal or higher than static pair programming in average.
- Ha1 (Program Correctness DPP Vs SPP): The Program Correctness of dynamic pair programming is less than static pair programming in average.
- H02 (Program Correctness DPP Vs IP): The Program Correctness of dynamic pair programming is equal or higher than traditional individual programming in average.
- Ha2 (Program Correctness DPP Vs IP): The Program Correctness of dynamic pair programming is less than traditional individual programming in average.
- H03 (Code Quality DPP Vs SPP): The Code Quality of dynamic pair programming is equal or higher than static pair programming in average.
- Ha3 (Code Quality DPP Vs SPP): The Code Quality of dynamic pair programming is less than static pair programming in average.
- H04 (Code Quality DPP Vs IP): The Code Quality of dynamic pair programming is equal or higher than traditional individual programming in average.
- Ha4 (Code Quality DPP Vs IP): The Code Quality of dynamic pair programming is less than traditional individual programming in average.

- H05 (Cost DPP Vs SPP): The overall software development cost of dynamic pair programming is equal or higher than static pair programming in average.
- Ha5 (Cost DPP Vs SPP): The overall software development cost of dynamic pair programming is less than static pair programming in average.
- H06 (Cost DPP Vs IP): The overall software development cost of dynamic pair programming is equal or higher than traditional individual programming in average.
- Ha6 (Cost DPP Vs IP): The overall software development cost of dynamic pair programming is less than traditional individual programming in average.

3.3. Software Development Cost

To determine the overall cost of software development, we compared the entire development time (in minutes) of each step. The overall software costs for IP (individual programming) and PP (pair programming) were determined using the following formulas:

$$Cost_{IP} = Time_{Design} + Time_{Coding} + Time_{Test}$$

$$Cost_{PP} = 2\ (Time_{Design} + Time_{Coding} + Time_{Test})$$

3.4. Program Correctness

The number of errors discovered (defect density) during acceptance testing is used to assess system quality, program correctness, and software process efficiency.

3.5. Code Quality

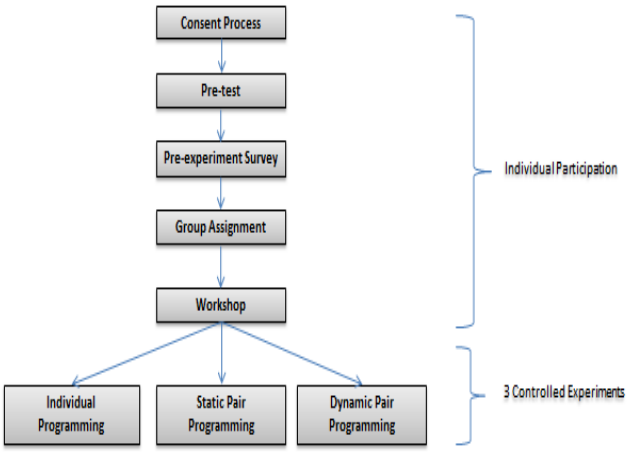
The functionality of the code, as well as the number of software components contained in the program, are used to assess its quality.

3.6. Experiment Procedure

The experiment consisted of the six stages shown in Figure 1. Stage-1: The subjects were given an informed consent form certified by the university's Institutional Review Board and were offered the option to volunteer for the experiment. The researcher informed students about the experiment, answered any questions, and provided one day to study and return the signed consent forms; Stage-2: respondents were asked to independently solve two programming problems in order to evaluate their programming skills; Stage-3: Each respondent was requested to complete a survey that collected information such their experience with programming, software testing, test-driven development, and previous pair programming experience prior to this experiment; Stage-4: The subjects were separated and put into groups of five based on their score. Three experimental groups—dynamic pair programming (DPP), static pair programming (SPP), and individual programming (IP)—were formed by randomly selecting participants from each group; Stage-5: A workshop was given to introduce the ideas of pair programming, unit testing, and acceptance testing to the participants. The pair programming groups then participated in a pair-jelling exercise, often known as a pair programming practice session. This exposed the programmers to pair programming and allowed them to comprehend its practices. This workshop was held prior to the controlled experiments; and Stage-6: Each experimental group was given three programming exercises. The dynamic pair programming group performed the initial experiment in pairs. After the initial trial, the dynamic couples rotated within their own group (i.e., each DPP duo swapped partners with another DPP pair). The second experiment was completed by the newly rotated pairings. The pairs

rotated once more to conduct the third trial. For all three tests, subjects in the static pair programming group were randomly partnered with the same partner. Subjects in the IP group were instructed to do all three experiments independently.

Fig. 1. Experimental Procedure



4. Results

Table I, below summarizes the control experiment quality metrics data.

Table I. Experiment data Summary (Quality Metrics)

	Static Pairs		**Dynamic Pairs**		**Individuals**	
	#Functions	**#Errors**	**#Functions**	**#Errors**	**#Functions**	**#Errors**
Problem1	6	5	5	5	4	3
	1	3	9	4	8	5
	7	5	3	4	3	6
	--	--	--	--	NA	NA
	--	--	--	--	NA	NA
Problem2	4	2	15	4	4	12
	16	2	19	4	4	2
	7	7	8	2	8	6

	--	--	--	--	NA	NA
	--	--	--	--	NA	NA
Problem3	5	2	16	2	8	14
	11	3	31	2	10	5
	10	3	16	6	8	6
	--	--	--	--	NA	NA
	--	--	--	--	NA	NA

*NA – Data Not Available

Table II below summarizes the control experiment cost metrics data.

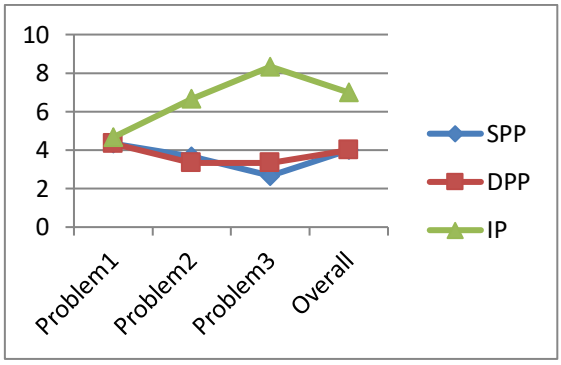
Table II. Cost Metrics Data Summary (Time in minutes)

	Static Pairs		**Dynamic Pairs**		**Individuals**	
	Delivery Time	**Cost**	**Delivery Time**	**Cost**	**Delivery Time**	**Cost**
Problem1	296	592	132	264	318	318
	175	350	125	250	184	184
	433	866	171	342	152	152
	-	-	-	-	270	270
	-	-	-	-	242	242
Problem2	272	544	252	504	227	227
	240	480	173	346	417	417
	214	428	244	488	290	290
	-	-	-	-	145	145
	-	-	-	-	320	320
Problem3	156	312	70	140	150	150
	255	510	136	272	345	345
	155	310	128	256	59	59
	-	-	-	-	195	195
	-	-	-	-	285	285

4.1. Program Correctness

The number of errors discovered during acceptance testing [9] is an indicator of system quality (or program correctness) as well as software process efficiency. The number of reworks for each experimental group is depicted in Figure 2. In a perfect world, this number would be zero [9]. The DPP and SPP groups have a similar rate of acceptance errors, while the IP group has a higher rate than the pair programming groups, as shown in Table I and Figure 2. In terms of program correctness and system quality, the DPP is the most efficient process while the SPP is the least efficient.

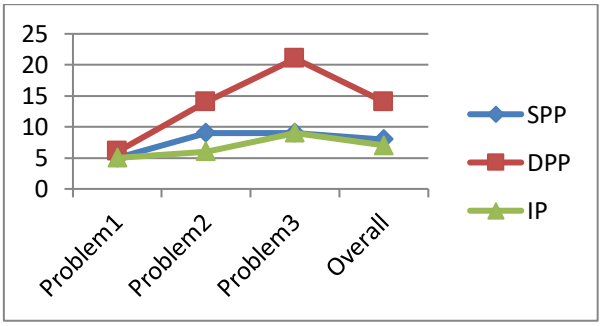
Fig. 2. The Defect Density



4.2. Code Quality

We have analyzed the code's functionality and the number of software components it contains to determine the code's quality. Table I and Figure 3 depict the number of software components present in each experimental group. This analysis indicates that the code produced by the DPP group is of significantly higher quality than that of the SPP and IP groups, and that the code produced by the SPP group is of higher quality than that of the IP group.

Fig. 3. The number of software components implemented.



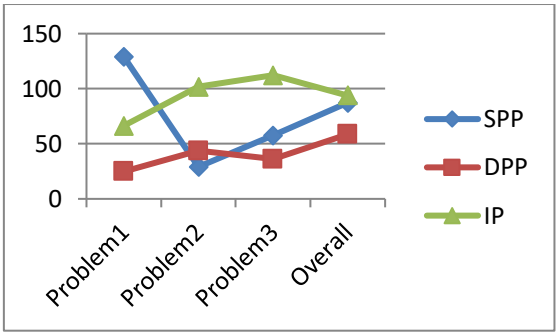
4.3. Predictability

As shown in Figure 4 and Table III, we have also analyzed the standard deviation of development times for each of the groups. This analysis suggests that the variance in development time for DPP is less than that of SPP and IP. It indicates that dynamic pair programming is more predictable than static pair programming and individual programming techniques, and that the SPP is the method with the greatest degree of unpredictability.

Table III. Standard Deviation of Development Time

	SPP	**DPP**	**IP**
Problem1	129.08	24.79	66.40
Problem2	29.05	43.49	101.86
Problem3	57.45	36.02	112.28
Overall	87.09	58.64	93.82

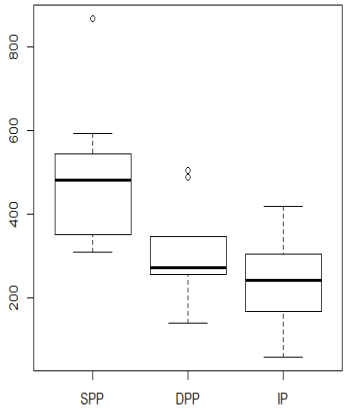
Fig.4. Standard Deviation in Development Time



4.4. Software Development Cost

Figure 5 depicts a box plot of the total software development costs for each of the three experimental groups.

Fig. 5. Box Plots for Software Development Cost



4.4.1. Dynamic pairs Vs Static pairs

Figure 6 depicts the t-test results for the software development cost. The p-value (0.0108) is highly statistically significant; consequently, we accept the alternative hypothesis that the overall software development cost of pair programming that uses pair rotation is less than pair programming that uses the same pair for all three programs.

Fig.6. Results of the T-test for Software Development Cost (Static pairs vs. Dynamic pairs)

```
Wilcoxon Scores (Rank Sums) for Variable time
    Classified by Variable group

        Sum of Expected Std Dev    Mean
group N Scores Under H0 Under H0   Score
dpp   9  59.0   85.50   11.324752  6.555556
spp   9 112.0   85.50   11.324752 12.444444

        Wilcoxon Two-Sample Test
Statistic (S)               59.0000
Normal Approximation
Z                           -2.2959
One-Sided Pr <  Z            0.0108
Two-Sided Pr > |Z|           0.0217
t Approximation
One-Sided Pr <  Z            0.0173
Two-Sided Pr > |Z|           0.0347
Exact Test
One-Sided Pr <=  S           0.0094
Two-Sided Pr >= |S - Mean|   0.0188
Z includes a continuity correction of 0.5.
```

4.4.2. Dynamic pairs Vs Individuals

Figure 7 depicts the t-test results for the software development cost. The p-value (0.0851) is not highly statistically significant; therefore, we accept the null hypothesis that the overall software development cost of pair programming that uses pair rotation is more than individual programming.

Fig.7. Results of the T-test for Software Development Cost (Dynamic pairs vs. Individuals)

```
Wilcoxon Scores (Rank Sums) for Variable time
    Classified by Variable group

         Sum of Expected Std Dev    Mean
Group  N Scores Under H0 Under H0   Score
dpp    9 136.0  112.50   16.770510  15.111111
ip    15 164.0  187.50   16.770510  10.933333

         Wilcoxon Two-Sample Test
Statistic (S)               136.0000
Normal Approximation
Z                             1.3715
One-Sided Pr >  Z             0.0851
Two-Sided Pr > |Z|            0.1702
t Approximation
One-Sided Pr >  Z             0.0917
Two-Sided Pr > |Z|            0.1835
Exact Test
One-Sided Pr >=  S            0.0869
Two-Sided Pr >= |S - Mean|    0.1737
Z includes a continuity correction of 0.5.
```

Table 4 provides a summary of the statistical test results.

Table IV. Summary of Statistical Experiments

Software Metrics	Test	p-vale	Decision
Program Correctness	Hypothesis 1	--	DPP is equal or higher than SPP
	Hypothesis 2	--	DPP is equal or higher than IP
Code Quality	Hypothesis 3	--	DPP is equal or higher than SPP
	Hypothesis 4	--	DPP is equal or higher than IP
Delivery/ Development Time	Hypothesis 5	0.0108	DPP is less than SPP
	Hypothesis 6	0.0117	DPP is less than IP
Software Development Cost	Hypothesis 7	0.0108	DPP is less than SPP
	Hypothesis 8	0.0851	DPP is equal or higher than IP

5. Summary and Conclusion

The primary goal of pair programming is to transfer knowledge among project team members and achieve quality control while developing software. Every new task should include a pair rotation to achieve this goal. However, the majority of pair programming studies conducted to date have not used the pair rotation. That is, the same two developers were assigned to all programming tasks. We do not have a clear understanding of the effects of pair programming on software development and real-time software quality assurance because the majority of pair programming experiments were not conducted as described in extreme programming and the empirical evidence of pair programming is inconclusive. We compared pair programming experiments with three experimental groups: one using pair rotation (dynamic pair programming), one using pair programming without pair rotation (static pair programming), and one using traditional individual programming. The dynamic and static pair programming methodologies were compared to traditional individual programming as well as to one another in terms of program accuracy, code quality, delivery time, and development cost.

From the statistical test findings, the following conclusions were drawn:

- The quality of the software system (in terms of program correctness) created by the dynamic pair programming group is equivalent to that of the static pair programming group and significantly higher than individual programming.

- The quality of the code (in terms of the number of implemented functions) produced by the dynamic pair programming group is significantly greater than that of the static pair programming group and the individual programming group.

- The predictability of dynamic pair programming is greater than that of static pair programming and individual programming methodologies.

- Demonstrates that programs of comparable or higher quality can be developed at a lower cost (35 percent less overall software development time than static pair programming) and in a shorter amount of time (35 percent less overall software delivery time than static pair programming) using the dynamic pair programming method.

- Dynamic pair programming produces programs of equivalent or higher quality in 34% less time (34% less time than traditional solo programming) than traditional individual programming.

6. References

1. Kent Beck, Extreme Programming Explained: An Embrace Change, Addison-Wesley, 2000, ISBN 0201616416.

2. Roger S. Pressman. Software Engineering: A Practitioner's Approach, sixth edition, McGraw Hill, 2005.

3. Erik Arisholm, Hans Gallis, Tore Dyba, and Dag I.K. Sjøberg, Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise, IEEE Transactions on Software Engineering, Vol. 33, No. 2, Feb 2007

4. Wilson, J., Hoskin, N., Nosek, J., 1993. The benefits of collaboration for student programmers. In: Proceedings 24th SIGCSE Technical Symposium on Computer Science Education, pp. 160–164.

5. John T. Nosek, The Case for Collaborative Programming, Communications of the ACM March 1998/Vol. 41, No. 3

6. Laurie Williams, Robert R. Kessler, Ward Cunningham, Ron Jeffries, Strengthening the Case for Pair Programming, July/August 2000 IEEE SOFTWARE

7. McDowell, C., Werner, L., Bullock, H., Fernald, J., 2002. The effects of pair-programming on performance in an introductory programming course. In: Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education. ACM, Cincinnati, KY, USA, pp. 38–42.

8. Shaochun Xu, Vaclav Rajlich, Empirical Validation of Test-Driven Pair Programming in Game Development, Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06)

9. Nawrocki, J. and Wojciechowski, A., 2001. Experimental Evaluation of pair programming. In: Proceedings of the European Software Control and Metrics Conference (ESCOM 2001). ESCOM Press, 2001, pp. 269-276.

10. Jari Vanhanen and Casper Lassenius, Effects of Pair Programming at the Development Team Level: An Experiment, 2005 IEEE

11. Matevz Rostaher and Marjan Hericko, Tracking Test First Programming – An Experiment, XP/Agile Universe 2002, LNCS 2418, pp. 174-184, 2002.

12. Hanna Hulkko and Pekka Abrahamsson, A Multiple Case Study on the Impact of Pair Programming on Product Quality, ICSE'05, May 15-21, 2005, St. Louis, Missouri, USA.

13. Matthias M. Müller, Walter F. Tichy. Case Study: Extreme Programming in a University Environment. In International Conference on Software Engineering (ICSE), pages 537-544, Toronto, Canada, May 2001.

14. Kim Man Lui and Keith C.C. Chan, Pair programming productivity: Novice-novice vs. expert-expert, Int. J. Human-Computer Studies 64 (2006) 915-925.

15. Matthias M. Müller, A preliminary study on the impact of a pair design phase on pair programming and solo programming, Information and Software Technology, Volume 48, Issue 5, May 2006, Pages 335-344, ISSN 0950-5849.

16. Matthias M. Muller, Two controlled experiments concerning the comparison of pair programming to peer review, The Journal of Systems and Software 78 (2005) 166-179

17. Watts S. Humphrey, PSP(sm): A Self-Improvement Process for Software Engineers, Addison-Wesley Professional, March 2005.

18. R. Swamidurai and D. Umphress, "The Impact of Static and Dynamic Pairs on Pair Programming," 2014 IEEE Eighth International Conference on Software Security and Reliability-Companion, San Francisco, CA, USA, 2014, pp. 57-63, doi: 10.1109/SERE-C.2014.52.