Investigating the Impact of Peer Code Review and Pair Programming on Test-Driven Development

Rajendran Swamidurai
Department of Mathematics and Computer Science
Alabama State University
Montgomery, USA
rswamidurai@alasu.edu

Brad Dennis, Uma Kannan
Dept. of Computer Science and Software Engineering
Auburn University
Auburn, USA
dennibc, uzk0002 @ auburn.edu

Abstract— **Pair programming and peer code review are two collaborative inspection methods that improve the quality of software. These light-weight review techniques are low-cost alternatives to formal inspections and rigorous reviews that commonly remove 90% of software defects before even the first test case is run. While pair programming has many advocates, evidence suggests that pair programming is not as useful as claimed and the requirement of the pairs to co-locate is hindrance on many development projects. Peer code reviews, however, have been shown to be just as effective as pair programming and are a better fit for many of today's software efforts that are being developed collaboratively, but asynchronously, in the cloud. In this paper we demonstrate peer review's effectiveness as compared to pair programming in the context of Test Driven Development (TDD), a popular agile programming technique that's rapidly gaining mainstream acceptance. The empirical evidence also shows that equal quality programs can be produced at a lower cost (28% less than pair programming) using a peer review technique in TDD as compared with traditional pair programming**

Keywords— ***Pair programming; PP; peer review; peer code review; code review; collaborative programming; test driven development; agile development; Extreme programming; empirical software engineering.***

I. INTRODUCTION

Pair programming [1] is a way of inspecting code while it is being written and is an activity that provides real-time problem solving and real-time quality assurance [2]. Its advocates claim that using pair programming achieves a considerable reduction in overall software development cost in lesser duration at the price of slightly increased personnel hours compared to traditional individual programming [3]. The idea of pair programming is attractive, but the technique does have drawbacks [4]. First, pair programming requires two developers be at the same place during the entire development duration. This co-location is often not realistic in busy organizations as well as in open source software development; since the developers may be matrixed concurrently to a number of projects in the earlier case and due to the distributed nature of the projects in the latter case. Second, it requires an enlightened management that believes that pair will produce better result than individuals since software products are measured more by tangible properties than by intangible properties. Third, the empirical evidence of the benefits of pair programming is mixed: the works of Wilson et al. [5], Nosek [6], Williams [7], McDowell et al. [8], and Xu and Rajlich [9] support the costs and benefits of pair programming. While experiments by Nawrocki and Wojciechowski [10], Vanhanen and Lassenius [11], Arisholm et al. [3], Rostaher and Hericko [12], and Hulkko and Abrahamson [13] show that there is no significant difference between the pair programming and solo programming. Boutin [14] also suggests that many developers are forced to abandon pair programming due to lack of resources (e.g. due to small team size). Boutin observed that abandoning the pair programming in the middle of the project hindered the integration of new modules to the existing project. Pair programming helps in knowledge sharing among partners and reduces the risk of project failure by denying specialization and sharing responsibility. These benefits, however, come at the expense of almost doubled personnel cost [15]. Moreover, pair programming has been demonstrated to be more effective than traditional single-person development only if both members of the pair are novices to the task at hand. Novice-expert and expert-expert pairs have not been demonstrated to be effective [3, 16, 17].

Rigorous reviews are more effective than any other error-removal strategy, including testing [18]. They commonly remove up to 90 percent of errors from a software product before the first test case is run [18]. But rigorous reviews, like software inspection, require a disciplined and structured approach [19]. Moreover, the software inspection is economically feasible only in large scale software developments; that is, achieving quality software through software inspection is an expensive process for smaller and less critical software [19]. A Porter and Johnson [20] study suggests that inspection is effective and can be done by individuals. A related study by Porter et al. [21] shows that there was no observable difference in the effectiveness of the two, three and five person inspection teams. The Porter and Johnson [20] study also reveals that the results of the inspection teams that do not meet are just as effective as those that do [22]. This suggests that effective inspections can be conducted with only one reviewer either asynchronously, e.g. remotely without a meeting, or synchronously. The one person software inspection formats that meet these criteria are peer code review and pair programming. Pair programming and peer code review are two similar (two-person or four-eye

978-1-4799-6585-4/14/$31.00 ©2014 IEEE

principle) collaborative techniques that are used to develop quality software [23, 24]. As both the techniques implement the four-eye (two-person) principle, they may be used as alternatives for mitigating the software defects [15, 24 - 26].

The nearly doubled cost appears to outweigh the proposed advantages of pair programming. Asynchronous reviews may be a reasonable alternative for producing code of similar quality to that of pair programming but with a fraction of the cost [24]. While experiments [15] comparing pair programming to peer review in traditional test first programming environment exists, there is no study to compare the pair programming with peer review in test-driven development environment. The aim of the study is to clarify which of the one person inspection techniques, pair programming or peer-review, has the greater impact on test-driven development in terms of software development cost.

II. Related Works

James Tomayko [22], of Carnegie Mellon University, conducted an experiment in 2001 to compare pair programming with software inspection. The subjects were eight computer science graduates divided into two teams of four. The experimental groups (pair programming groups) used the extreme programming and the control groups (inspection groups) used the Team Software Process (TSP). The aim of the experiment was to evaluate whether pair programming reduced defects more than the formal inspections prescribed by TSP. The results suggest that pair programming is more effective than formal inspection in reducing defects.

Matthias Muller [15], University of Karlsruhe, Germany conducted two experiments to compare pair programming with peer review. The first experiment was conducted in 2002 then repeated in 2003. The subjects were divided into control groups (individuals) called review groups and experimental groups (pairs). In the review group an individual programmer developed the program, compiled it, had it reviewed by an unknown reviewer, and then conducted testing. In the pair programming group, all the development activities were carried out by two programmers sitting in front of the same computer. The students were asked to solve polynomial and shuffle-puzzle problems using Java in both experiments. The purpose of the studies was to find the cost of pair programming as opposed to peer review methods. The results reveal that there was no difference in program correctness, and for a similar level of correctness there was no difference in development cost. These experiments suggest that pair programming and individual inspection may be interchangeable in terms of cost.

Phongpaibul and Boehm [19] conducted four control experiments in 2005 and 2006 to compare the software development with inspection and pair programming. The subjects of the first two and fourth experiments were students, whereas professional developers were used in the third experiment. The results show that the total development effort of the pair development group was less than the inspection group with the same product quality.

These experiments suggest that 1) pair programming is more effective than formal inspection in reducing defects, 2) pair programming and individual inspection may be interchangeable in terms of cost and 3) the total development effort of the pair development group was less than the inspection group with the same product quality.

III. The Experiment

A formal experiment was held at the Auburn University to validate the effectiveness of the pair programming (PP) versus peer review (PR) process in 2009. While the subjects were students, the study was not part of any university course. The subjects were divided into two groups: the pair programming (PP) group and peer review (PR) group. Both the experimental groups (PP group and PR group) used Java and the Eclipse integrated development environment (IDE) to solve three programming problems of varying complexity.

A. Subjects

Ten undergraduate and graduate students from the Software Process (COMP5700/6700) class volunteered to participate in the study. All participants had already taken software modeling and design and computer programming courses such as C, C++ and Java.

B. Experimental Tasks

The subjects were asked to solve the following three programming problems using Test Driven Development (TDD) with Java in the Eclipse IDE.

Problem1: Write a program which reads a text file and displays the name of the file, the total number of occurrences of a user-input string the total number of non-blank lines in the file, and count the number of lines of code according to the LOC Counting Standard used in PSP, Personal Software Process [27]. You may assume that the source code adheres to the LOC Coding Standard. This assignment should not determine if the coding standard has been followed. The program should be capable of sequentially processing multiple files by repeatedly prompting the user for file names until the user enters a file name of "stop". The program should issue the message, "I/O error", if the file is not found or if any other I/O error occurs.

Problem2: Write a program to list information (name, number of methods, type, and LOC) on each proxy in a source file. The program should also produce an LOC count of the entire source file. Your program should accept as input the name of a file that contains source code. You are to read the file and count the number of lines of code according to our LOC Counting Standard. You may assume that the source code adheres to the LOC Coding Standard. This assignment should not determine if the coding standard has been followed. The exact format of the application-user interaction is up to you.

- A "proxy" is defined as a recognizable software component. Classes are typical proxies in an object-oriented systems; subprograms are typical proxies in traditional functionally-decomposed systems.
- If you are using a functionally-decomposed (meaning, non-OO) approach, the number of methods for each

978-1-4799-6585-4/14/$31.00 ©2014 IEEE

proxy will be "1". If you are using an OO approach, the number of methods will be a count of the methods associated with an object.

Probelm3: Write a program to calculate the planned number of lines of code given the estimated lines of code (using PSP's PROBE Estimation Script). Your program should accept as input the name of a file. Each line of the file contains four pieces of information separated by a space: the name of a project and its estimated LOC, planned LOC, and actual LOC. Read this file and echo the data to the output device. Accept as input from the keyboard a number which represents the estimated size (E) of a new project. Output the calculations of each decision and the responding planned size (P), as well as the PROBE decision designation (A, B, or C) used to calculate P. For each decision, indicate why it is/isn't valid. The exact format of the application-user interaction is up to you.

- Your software should gracefully handle error conditions, such as non-existent files and invalid input values.
- Round P up to the nearest multiple of 10.

C. Hypothesis

We have tested the following hypothesis to validate the peer code review against the traditional pair programming:

- H01 (Cost PP vs. PR): The overall software development cost of pair programming is equal or higher than peer review in average.
- Ha1 (Cost PP vs. PR): The overall software development cost of pair programming is less than peer review in average.

D. Software Development Cost

To study the cost of overall software development we compared the total development time, measured in minutes, of all the phases. Pair programming (PP) consists of design, coding and test phases and the peer review (PR) groups had an additional review phase. The total software development costs for PP and PR were calculated as follows:

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

$$Cost_{PR} = Time_{Design} + Time_{Coding} + Time_{Review} + Time_{Test}$$

E. Experiment Procedure

1) *Consent Process:* An Auburn University Institutional Review Board approved informed consent form was handed out to the students in the spring 2009 Software Process course and they were given the chance to volunteer to participate in the experiment. The researcher provided information to students about the experiment, answered any questions, and requested that the forms be returned the following class; so students had at least one intervening day to review all aspects of consent. The researcher returned the following class and answered any more questions raised and collected the consent forms.

2) *Pre-Test:* In the pre-test the subjects were asked to solve two programming problems individually in order to measure their programming skills.

3) *Pre-Experiment Survey:* Each subject was asked to complete a survey which collected information such as age, class level, programming languages known, experience level, and pair programming experience.

4) *Assigning the Subjects to Experimental Groups:* Based on the pre-test's result and survey, the subjects were divided into groups of four. The subjects were randomly selected from each group and assigned to the two experimental groups: Pair programming (PP) group and Peer review (PR) group.

5) *Workshop:* A workshop was held for the subjects that explained the concepts of peer review, pair programming, and unit and acceptance testing. Then, a pair programming practice session, known as pair-jelling exercise, was held for the pair programming group and a similar peer review practice session was held for the review group. This enabled the programmers to be exposed to pair programming (and peer review) and understand its practices. This workshop was held before the control experiments were conducted.

6) *Control Experiments:* Three programming exercises were given to each experimental group. The pair programming group paired-up to do the first experiment. After the first experiment the pairs rotated within their own group (i.e., A PP pair interchanged his/her pair with another PP pair). The new rotated pairs completed the second experiment. Once again the group's pairs rotated to do the third experiment. The peer review group developers were asked to complete the design and coding phases; after the coding phase the program will be given to a reviewer for code review; after the code review the developer will conduct the test.

Figure 1 summarizes the experimental procedure.

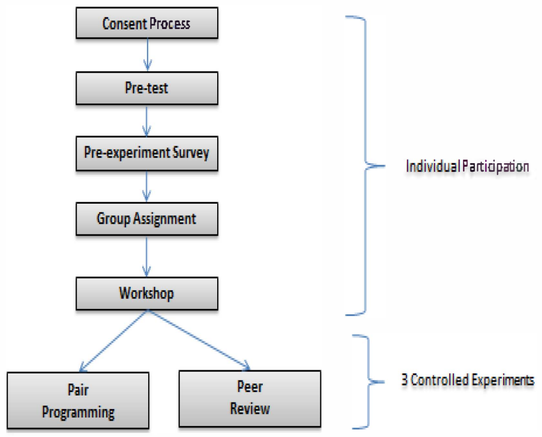

Fig. 1. Experimental Procedure

IV. RESULTS

Table I, below summarizes the control experiment data.

TABLE I. SOFTWARE DEVELOPMENT COST MEASURED IN MINUTES

Problem	Group	Peer Review	Pair Programming
Problem1	1	110	264
	2	123	250
	3	162	342
	4	124	-
Problem2	1	86	504
	2	280	346
	3	170	488
	4	138	-
Problem3	1	174	140
	2	651	272
	3	469	256
	4	265	-

Figure 2, shows the box plots for total software development cost for both pair programming and peer review groups.

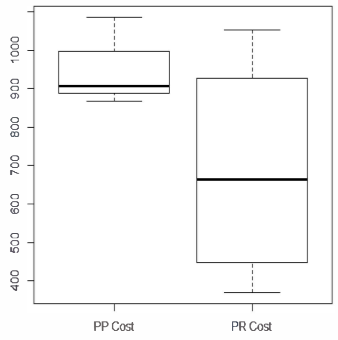

Fig. 2. Box plots for PP vs. PR cost

```
                  TTEST Procedure
                   Variable:  time

group       N     Mean     Std Dev   Std Err    Minimum     Maximum
PP          9     318.0     117.3     39.0939     140.0      504.0
PR         12     229.3     169.4     48.9067    86.0000     651.0
Diff (1-2)        88.6667   149.7    66.0096

group     Method    Mean     95% CL Mean    Std Dev   95% CL Std Dev
PP                  318.0     227.8    408.2  117.3     79.2188 224.7
PR                  229.3     121.7    337.0  169.4     120.0    287.7
Diff (1-2) Pooled   88.6667 -49.4929 226.8   149.7     113.8    218.6
Diff (1-2) Satterth 88.6667  -42.4162 219.7
           waite

           Method         Variances     DF       t Value Pr > |t|
           Pooled          Equal        19        1.34     0.1950
           Satterthwaite   Unequal      18.924  1.42     0.1730

                       Equality of Variances
           Method      Num DF    Den DF    F Value    Pr > F
           Folded F      11         8        2.09      0.3058
```

Fig. 3. T-Test Results

The peer review (PR) groups took 688 minutes on average to solve all the three programming problems; whereas the pair programming (PP) groups took 954 minutes (28% more than PR groups) in average to solve all the three programming problems. The t-test result for hypothesis is shown in Figure 3. Since p>=0 .05 (p=0.1950), we do not have sufficient statistical evidence to reject H01 in favor of Ha1 and conclude that the overall software development cost of pair programming is less than peer review in average.

V. SUMMARY AND CONCLUSION

In this paper, we compared two light-weight and low-cost collaborative inspection methods, traditional pair programming and peer code review, that effectively improve the quality of software. While the concept of pair programming is attractive, evidence suggests that pair programming is not as useful as claimed by its advocates. Moreover, the requirement of the pairs to co-locate is hindrance on many development projects. Peer code reviews, however, have been shown to be just as effective as pair programming and are a better fit for many of today's software efforts that are being developed collaboratively, but asynchronously, in the cloud.

The empirical evidence shows that equal quality programs can be produced with much cheaper cost (28% less than pair programming) using peer review technique as compared with the traditional pair programming technique in the context of Test Driven Development (TDD) environment.

ACKNOWLEDGMENT

The authors gratefully acknowledge Dr. David Umphress of Auburn University for his extended support and guidance throughout the control experiment.

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] Rajendran Swamidurai and David A. Umphress, "Collaborative-Adversarial Pair Programming," ISRN Software Engineering, vol. 2012, Article ID 516184, 11 pages, 2012. doi:10.5402/2012/516184.

[5] 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.

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

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

[8] 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.

[9] 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)

[10] 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.

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

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

[13] 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.

[14] Karl Boutin. Introducing Extreme Programming in a Research and Development Laboratory. Extreme Programming Examined, Addison-Wesley, Chapter 25, pages 433-448.

[15] 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

[16] Don Wells and Trish Buckley, The VCAPS Project: An Example of Transitioning to XP. Extreme Programming Examined, Addison-Wesley, Chapter 23, pages 399-421.

[17] 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.

[18] Robert L. Glass, Frequently forgotten fundamental facts about software engineering, IEEE Software 2001.

[19] Phongpaibul and Boehm, A Replicate Empirical Comparison between Pair Development and Software Development with Inspection, First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007).

[20] Porter, A.A & Johnson, P.M. (1997), Assessing Software Review meetings: Results of a Comparative analysis of two experimental studies. IEEE Transactions on Software Engineering, 23, 129-145.

[21] Porter, A.A., Siy, H.P., Toman, C.A., & Votta, L.G. (1997), An experiment to asses the costs of code inspection in large scale software development, IEEE Transactions on Software Engineering, 23, 329-346.

[22] James. E. Tomayko, A Comparision of Pair Programming to Inspections for Software Defect Reduction, Swets & Zeitlinger Computer Science Education, 2002, Vol 12, No. 3, PP 213-222.

[23] Salleh, N., Mendes, E., & Grundy, J. C. (2011). Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review. IEEE Transactions on Software Engineering, 37(4), 509-525.

[24] Spohrer, Kude, Schmidt and Heinzl, Knowledge Creation in Information Systems Development Teams: The Role of Pair Programming and Peer Code Review, Proceedings of the 21st European Conference on Information Systems.

[25] Müller, M. M. (2004). Are Reviews an Alternative to Pair Programming? Empirical Software Engineering, 9(4), 335-351.

[26] Rigby, P., Cleary, B., Painchaud, F., Storey, M.-A., & German, D. (2012). Contemporary Peer Review in Action: Lessons from Open Source Development. IEEE Software, 29(6), 56-61.

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