Middle-East Journal of Scientific Research 21 (12): 2231-2237, 2014
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2014.21.12.21843

# An Empirical Comparative Analysis of Programming Effort, Bugs Incurrence and Code Quality Between Solo and Pair Programmers

*Olalekan S. Akinola*

Department of Computer Science, University of Ibadan, Nigeria

**Abstract:** Pair programming has been adjudged the better approach to programming especially in learning environments. However, little or no empirical evidence exists to support this claim in terms of effort expended in programming, bugs incurrence and quality of code obtained. This study was therefore designed to compare the solo and pair programming with these factors. Sixty volunteered student-programmers were randomly assigned to pair and solo programming groups. The solo group consists of 30 participants while the rest were randomly paired into 15 pair programming subgroups. The two groups were given the same programming task without time limit placed on them. The effort (times) expended on analysis, programming and debugging as well as number of bugs incurred by the groups and their effectiveness score were obtained and analyzed. Briefly, the result shows that pair programming performs better than solo programming in terms of the factors analyzed.

**Key words:** Solo and pair programming · Extreme programming · Software quality assurance

## INTRODUCTION

At one extreme end of computer programming is a single person doing all the thinking, design and implementing a program. This is *solo programming*. But two heads, they say, are better than one. Pair programming, dates at least from 1970 [1], is a style of programming in which *two* programmers work side-by-side at *one* computer with one keyboard and one mouse, continuously collaborating on the same design, algorithm, code, or test [2-4]. The emergence of agile methodologies and Extreme Programming, XP [5] has popularized the pair programming practice.

An agile approach is one that values: "Individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to change over following a plan" [3]. Extreme Programmers work together in pairs and as a group, with simple design and obsessively tested code, improving the design continually to keep it always just right for the current needs [6].

Pair programming is thus component of both XP and Agile approaches of software development. The two participants in pair programming are tagged the driver and the *navigator*. The one that types at the only computer computer available is the *driver*. The other partner that observes and 'criticise' the work of the driver-looking for tactical and strategic defects is called the *navigator*. Some tactical defects might be syntax errors, typos and calling the wrong method. Strategic defects occur when the driver is headed down the wrong path-what driver and navigator are implementing just won't accomplish what it needs to accomplish [7].

The strategic, longer-range thinker of the programming pair is the navigator. Because the navigator may not be deeply involved with the design, algorithm, code or test, he or she can have a more objective point of view and can better think strategically about the direction of the work. In pair programming, the driver and the navigator communicate effectively, brainstorm at any time the situation calls for it and periodically there can be a switch of roles between the driver and the navigator.

Pair programming has been adjudged to be a better alternative to solo programming. Jo *et al*. [8] opine that pair programming is better than solo programming especially when the complexity of the programming task is low. They further submit that pair programming yields code solutions of higher quality if the complexity of the task is high. On the code testing side, pair programmers

**Corresponding Author:** Olalekan S. Akinola, Department of Computer Science, University of Ibadan, Nigeria.

delivered correct and higher coverage test sets [9]. Lech [10] is of the opinion that pair programming makes testing more rigorous, thorough and effective.

**Solo Versus Pair Programming in Educational and Software Development Environments:** Pair programming has been used in educational environments and has a lot of potential benefits to learners of computer programming [11-14]. For instance, Charlie *et al*. [15] opined that academic achievement is enhanced when an individual learns information with others. In a study carried out by Linda *et al*. [16], pair programming has been shown to assist female students to learn better in programming and leads to women retention in Computer Science. Further results from their study however show that there was no significant difference in pass rates between paired and solo students. Brian *et al*. [2] assert that the transition from paired to solo programming is easy for students.

McDowell *et al.* [13] noted that applying pair programming method to first year students resulted in a greater percentage of students who successfully completed the course. Quantitative studies of Nagappan [11], Hanks *et al.* [12], McDowell, *et al.* [13] and others that compared the performances of pair programming students and solo students showed that the former were more likely to perform better and turn in solution of higher quality.

On the other hand, some studies, [17- 20] have suggested that it is not obvious that pair programming is better than solo programming. For instance Charlie *et al*. [15] study submitted that despite the fact that pair-programming results in improved programs, when used to teach programming it appears not to affect the extent to which students master course material and are able to independently apply their knowledge to new problems. Tessem [18], Gittins and Hope [17] also showed that some students found the experience irritating, inefficient and exhausting. According to Mathias [21], Single developers are as costly as programmer pairs, if both programmer pairs and single developers with an additional review phase are forced to produce programs of similar level of correctness. VanDeGrift [19] showed that the students complained about working among people with different personalities and skill levels.

However, as argued by Bryant [22] there is evidence to suggest that pair programming in some situations appears to be more engaging, useful and enjoyable. Mustafa [23] also asserts that collaborative programming could be a chance to dissipate gender differences in attitudes towards programming. Alistair and Laurie [24] summarized the significant benefits of pair programming as follows:

- Mistakes are caught as they are typed;
- Time taken to code is usually small;
- The defect or bugs incurred are usually lower compared with the solo programming;
- There is thus a code review taking place in pair programming process;
- Pair programming produces better design and quality solution as a result of brainstorming and pair relaying;
- People learn better about the system development process; and
- Team building usually results; with people working together and better information flow.

Rajendran and David [25] assert that industry and academia have turned their attention and interest toward pair programming in recent years. It has been widely accepted as an alternative to traditional individual programming.

In the present study, we employed an experimental proof approach to determine if there is any significant difference in the performance of solo and pair programming approaches; by studying the time taken for program comprehension (analysis), time taken for coding and debugging, number of bugs incurred by the groups and the outcomes of the participants in the experiments.

## MATERIALS AND METHODS

**Subjects:** Sixty students who had passed through Structured Programming (CSC 232) course using Java programming language as a tool volunteered and participated in this study. The students were taken through the concepts of Structured programming and Object Oriented Programming (OOP) paradigms, including files and database handling in Java for twelve weeks in 2013/2014 session, at the Department of Computer Science, University of Ibadan, Nigeria. The experiment actually took place at a practical class lasting five hours at the end of the 11$^{th}$ week.

**Experimental Design and Experimental Artifact:** The experimental artifact used was a programming problem on computerizing an admission system in a hypothetical high school. The participants were given the problem specification to write a Java program to solve. The 60 participants were randomly divided into two groups. The first group consisting 30 participants worked as solo group. They individually worked on the problem from the comprehension (analysis) to the implementation phase. The second group consists of 30 participants who were also randomly assigned to 15 pair groups. The two groups worked independently on the same artifact without any time restriction placed on them. The groups were asked to record all their analyses as well as the times taken to do the analysis, time taken to code and debug and all the bugs reported by the compiler.

During the analysis phase, the groups were expected to comprehend the problem and analyze it for the input and output variables as well as the process logic needed to solve the given problem. The program design on appropriate data and control structures needed to solve the problem was also carried out by the participants. The researcher ensured proper compliance of the groups to the experiment's rules and collated the reports for analysis.

A two-group between-subjects research design approach was used in this study. The participants were randomly assigned to the two levels of the independent variable (Solo or Pair). In this design, each participant was assigned to only one group and consequently, the two groups were independent of one another. The problem specification artifact is given as Appendix at the end of this paper.

**Variables:** One major independent variable was manipulated in the study: The programming group; Solo or Pair.

**The Four Dependent Variables Measured Were:**

- Comprehension (Analysis) time: the time spent by a group or an individual in understanding the problem and documents the analysis report.
- Coding time: The time spent by a group or an individual to translate the analysis into a chosen computer programming language and to debug the program completely of errors /bugs.
- Number of Bugs: The numbers of errors recorded during compilation.
- Effectiveness: Score obtained at the end of the exercise. This is a measure of the accuracy and efficiency of the code produced. This was measured over 10 marks.

**Statistics Test:** The data obtained in this study was subjected to independent samples t-test at $p = 0.05$. Descriptive statistics were also employed in the analysis of the data.

## RESULTS

Four major results were obtained in this study as stated in section 3.3.

**Comprehension (Analysis) Time:** The time taken for the solo programmers to comprehend and analyze the problem ranged from 25 (minimum) to 120 (maximum) minutes with mean time taken of 56.27 ± 4.11 Standard Error of Mean (SEM). On the contrary, pair programmers took between 15 to 60 (mean = 38.00 _± 3.46 SEM) minutes for their analysis.

The independent *t*-test statistic result ($p = 0.006$) showed that there was a high significant difference between the times taken by both Solo and Pair programmers in the study.

**Coding and Debugging (Programming) Time:** The time taken by the Solo programmers to do the coding and debugging ranged from 163 to 276 minutes (mean = 217.37_± 5.17 SEM), while the Pair groups took between 125 to 260 (mean = 185.00 _± 9.94 SEM) minutes for these tasks. The *t*-test showed a high significant difference between the two groups ($p = 0.03$).

**Number of Bugs / Errors Incurred at Compilation:** The total number of bugs/errors (Syntax, semantic and logic) incurred by Solo programmers ranged from 10 to 60 (mean = 21.17 ± 2.10 SEM) while the Pair programmers incurred between 2 and 50 (mean = 17.53 ± 3.74) bugs. The *t*-test showed no significant difference ($p = 0.37$) between the Solo and Pair groups in their bugs incurred.

**Effectiveness:** The effectiveness gives the score obtained (out of a maximum of 10) by the participants in the exercise as a measure of their performance. Results showed that Solo programmers scored between 2.0 and 6.0 (mean = 3.93 ± 0.17) while Pair programmers had between 4.0 and 8.0 (mean = 5.57 ± 0.29). The independent *t*-test showed a high significant difference between the effectiveness of the two groups ($p = 0.00$).

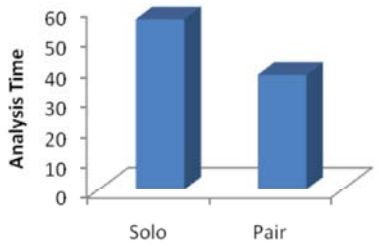


Fig. 1: Mean Analysis Time taken by the Participant Groups

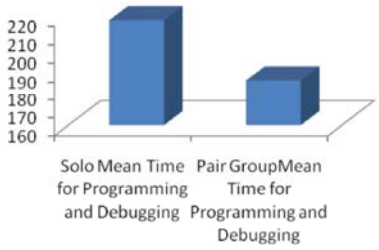


Fig. 2: Mean Time Taken for Programming and Debugging

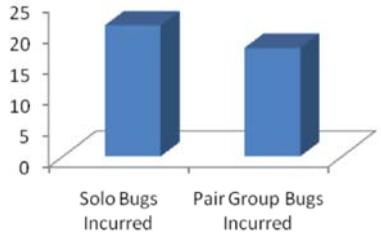


Fig. 3: Number of Bugs Incurred by Solo and Pair Programmers

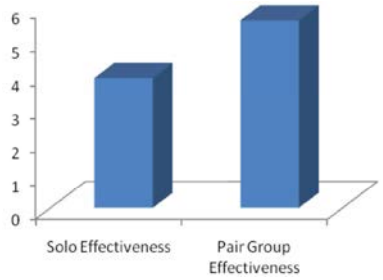


Fig. 4: Effectiveness of the Solo and Pair Programmers

**Discussion of Results:** Solo programming has been the usual programming practice from time immemorial. However, it has been criticized for a number of demerits such as having long time duration for program development, high bug incurrence rate, poor quality output, among others [24]. This study was conducted in order to verify the efficacy of pair programming in imparting programming knowledge to students and in software development environments.

**Analysis Effort:** The first step in programming is the analysis of the problem at hand. The program must be understood (comprehended) and carefully analysed for the input and output variables expected as well as the process logic to follow in solving the problem at hand. At the analysis phase, program design also comes into play. The design of the inputs and outputs, user interface, data structures, file/database design are carried out. Ideally, it is expected that once the analysis and design are thoroughly done by a programmer, then the problem would be solved with a greater level of accuracy and at a lesser time. The effort expended by the programmer in the analysis and design is likened to the total time spent doing the analysis.

Results from this study indicate that solo programmers expended more effort (average of 56 minutes) on the analysis/design than the pair programmers, who expended less effort on the same task (average of 38 minutes). This accounts for the high significance difference ($p < 0.05$) between the variables. The results suggest that pair programmers spent less time in the exercise possibly due to the "two-brain effects". Pair programming is thus recommended for good program analysis and design in software development exercise.

**Programming Effort:**Following the analysis and design phase is the real programming, which is usually tagged as the implementation phase. Debugging of errors reported by compilers is carried out by programmers by inspecting the locations of the bugs and fixing them. The effort expended (time taken) by the participants for this phase was recorded, analysed and compared.

The study shows that solo programmers spent more time (average of 217 minutes) for their programming than their pair counterparts who spent on the average, 185 minutes on the same task given to them. This accounts for the high significant difference between the times spent for programming by the two independent groups ($p < 0.05$).

The program implementation phase is also very critical in software development. The quality of the prospective software is at stake once it is full of bugs and if wrong logic is used to implement it. Programmers often spend enormous time implementing a piece of code. Results from this study show that pair programming could assist in reducing time spent on implementing programs especially if it is well planned and conducted. The results obtained are in support of the submissions of Constantine [26], Beck [5], Alistair and Laurie [24] and Jo *et al.* [8].

**Bugs Incurrence:** Bugs are errors incurred by programmers during programming. Bugs are serious threats to quality of software as well as its acceptance by clients. This study compares the level of bugs incurrence by the solo and pair programmers in order to determine which approach will reduce the number of bugs in programming. The mean bug incurred by solo programmers was 17 as compared to the pair programmers (21). This small marginal difference, however, was not significant ($p > 0.05$), indicating that "the difference between the two groups was not large enough to attribute to anything other than chance" [15]. This result suggests that despite the fact that pair-programming results in improved programs, it does not mean that bugs will not still occur.

**Programmers' Effectiveness:** The quality of the code outputs produced by the solo and pair programmers was determined by scoring them based on accuracy of their programs to specification, use of suitable data structures and data structures, number of lines of code produced (efficiency) and documentations (white spaces and comments) of the programs. Results from the study show a high significant difference ($p < 0.05$) between the two programming groups with pair programmers performing better. On the average, solo programmers scored 4 while pair programmers 6 out of maximum of 10 marks allotted to their effectiveness. This result further buttresses the fact that pair programming has higher advantage than solo programming in software design and implementation. The results are not far different from the claims of other researchers [10, 27] who had done extensive works in this domain.

## CONCLUSION

The performance of programmers using solo or pair programming approaches was examined in this study. Pair programming has been shown to perform better in terms of programmers' efforts expended on analysis and implementation, bugs or errors incurred and their final effectiveness in producing quality programs.

## ACKNOWLEDGEMENT

The researcher appreciates the voluntary and dedication of the students that participated in this study.

## REFERENCES

1. Edgar Acosta Chaparro, Aybala Yuksel, Pablo Romero and Sallyann Bryant (2005). Factors Affecting the Perceived Effectiveness of Pair Programming in Higher Education, *17th Workshop of the Psychology of Programming Interest Group,* Sussex University, June 2005, pp: 5 - 18. www.ppig.org.
2. Brian Hanks, Sue Fitzgerald, Renée McCauley, Laurie Murphy and Carol Zander (2011). Pair programming in education: a literature review, *Computer Science Education,* 21(2): 135-173.
3. Sallyann Bryant, Benedict du Boulay and Pablo Romero, 2006. XP and Pair Programming practices, *PPIG Newsletter*, September 2006, pp: 1 - 6.
4. Williams, L. and R. Kessler, 2003. Pair Programming Illuminated. Reading, Massachusetts, Addison Wesley.
5. Beck, K., 2000. Extreme Programming Explained: Embrace Change. 2000, *Reading*, Massachusetts: Addison-Wesley.
6. Ron Jeffries, 2001. What is extreme programming? http://xprogramming.com/book/whatisxp/ visited in August 2014.
7. Laurie Williams, 2004. Software Reviews and Pair Programming, pp: 11-32.
8. Jo E. Hannay, Tore Dybå, Erik Arisholm and Dag I.K. Sjøberg, 2009. The effectiveness of pair programming: A meta-analysis, *Information and Software Technology,* 51(7): 1110-1122.
9. Otávio Augusto Lazzarini Lemos, Fabiano Cutigi Ferrari, Fábio Fagundes Silveira and Alessandro Garcia (2012). Development of auxiliary functions: should you be agile? an empirical assessment of pair programming and test-first programming, *Proceedings of the 34th International Conference on Software Engineering,* IEEE Press Piscataway, NJ, USA, pp: 529-539.
10. Lech Madeyski, 2008. Impact of pair programming on thoroughness and fault detection effectiveness of unit test suites, *Software Process: Improvement and Practice,* 13(3): 281-295.
11. Nagappan, N., *et al.,* 2003. *Improving the CS1 experience with Pair Programming*. In *SIGCSE*. 2003.
12. Hanks, B., *et al*., 2004. Program Quality with pair programming in CS1. in1 *ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. 2004. Leeds, United Kingdom: ACM Press.

13. McDowell, C., *et al*., 2003. The impact of pair-programming on student performance, perception and persistence. in *ICSE '03: In Proceedings of the 25th International Conference of Software engineering,*. Portland, Oregon: IEEE Computer Society.
14. Williams, L., *et al*., 2000. Strengthening the case for pair-programming. *IEEE Software*, 17(4): 19-25.
15. Charlie McDowell, Linda Werner, Heather Bullock and Julian Fernald (2002). The Effects of Pair-Programming on Performance in an Introductory Programming Course, *Proceedings of the 33rd ACM Technical Symposium on Computer Science Education,* February 7-March 3, 2002.
16. Linda, L. ,Werner, Brian Hanks and Charlie Mcdowell, 2004. Pair-Programming Helps Female Computer Science Students ACM Journal of Educational Resources in Computing, Vol. 4, No. 1, March 2004, Article 3.
17. Gittins, R. and S. Hope, 2001. A study of Human Solutions in eXtreme Programming. In *13th Workshop of the Psychology of Programming Interest Group*, Bournemouth UK.
18. Tessem, B., 2003. Experiences in Learning XP Practices: A Qualitative Study. in Extreme Programming and Agile Processes in Software Engineering*, 4th International Conference*.. Genova, Italy: Springer.
19. Van De Grift, T., 2004. Coupling pair programming and writing: learning about students' perceptions and processes, in *Proceedings of the 35th SIGCSE technical symposium on Computer science education*. ACM Press: Norfolk, Virginia, USA.
20. Melnik, G. and F. Maurer, 2002. Perceptions of Agile Practices: A Student Survey. In *Extreme Programming/Agile Universe*. Chicago, IL.
21. Matthias M. Müller, 2005. Two controlled experiments concerning the comparison of pair programming to peer review, *Journal of Systems and Software,* 78(2): 166-179.
22. Bryant, S., 2004. XP: Taking the psychology of programming to the eXtreme. in *16th Workshop of Psychology of Programming Interest Group*. Institute of Technology, Carlow, Ireland.
23. Mustafa Baer, 2013. Attitude, Gender and Achievement in Compuer Programming *Middle-East Journal of Scientific Research,* 14(2): 248-255, 2013 ISSN 1990-9233.
24. Alistair Cockburn and Laurie Williams, 2002. The Costs and Benefits of Pair Programming, http://www.cs.pomona.edu/classes/cs121/supp/williams_prpgm.pdf, downloaded in August 2014.
25. Rajendran, S. and A. David, 2012. Umphress. Research Article Collaborative Adversarial Pair Programming. *ISRN Software Engineering,* Volume 2012, http://www. downloads. hindawi. com/isrn/ se/ 2012/516184.pdfý Downloaded on 23rd May, 2013.
26. Constantine, L.L., 1995. Constantine on Peopleware. *Yourdon Press Computing Series*, ed. E. Yourdon, Englewood Cliffs, NJ: Yourdon Press.
27. Brain Hanks, 2008. Empirical evaluation of distributed pair programming, International Journal of Human-Computer Studies, 66(7): 530-544.

Appendix: **Programming Experimental Artefact**

**The Problem Specification:**

CSC Grammar School has just concluded an entrance examination test for her prospective students. This has been a yearly routine exercise for the intending students of the school. As a matter of fact, varied number of students attends the test yearly. Prospective students answer questions from 10 different subjects; each being 100 marks. The mean of the scores in the subjects is then computed; 50 being set as the cut-off point for the admission. The management of the school has just decided to employ the use of computers to assist them in computing some statistics from the yearly test results and you have been contacted on this, being a prospective full-fledged computer programmer.

Your analysis of the system shows that the following information is needed from the examination results data:

- Total number of students that participated in an entrance examination test;
- Data and number of students that are admit-able in a session;
- Data and total number of students that fail the test;
- Data and total number of students in the marginal mean score levels (between 48 and 49 inclusive) for possible consideration in the second batch; and
- Data and total number of students in the following mean score categories:

A.0 - 20
B.21 - 40
C.41 - 49
D.50 - 80

**Task:**

- Create a file named "data.dat" in the folder of your java file. The structure of the records in the fiel table should follow the following sample

| S/N | Exam No. | Name | Gender | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 14/01 | Akinola S. O. | M | 54 | 46 | 25 | 39 | 49 | 72 | 73 | 81 | 24 | 28 |
| 2 | 14/02 | Dennis Kay | M | 78 | 29 | 11 | 50 | 58 | 59 | 29 | 98 | 27 | 38 |
| 3 | 14/03 | Mohammed Ayisat | F | 48 | 30 | 48 | 49 | 28 | 38 | 49 | 21 | 87 | 67 |
| | | | | …. | …. | … | …. | ..... | | | | | |

Populate the file with about 10 records or more.

- Implement a java program to read the data from the file and computes all the information needed by the school, using arrays, files and OOP concepts.
- Your outputs, with appropriate sectional information headings, should be directed to an output file to be named "out.txt" and newly created by the program.