Pair Programming – Cubic Prediction Model Results for Random Pairs and Individual Junior Programmers

[1]MARY ADEBOLA AJIBOYE, [2]MATTHEW SUNDAY ABOLARIN,
[3]JOHNSON ADEGBENGA AJIBOYE, [4]ABRAHAM USMAN USMAN, [5]SANJAY MISRA

[1]Abuja Electricity Distribution Company (AEDC), ICT Department, Niger Regional Office,
Minna, NIGERIA
[2]Department of Mechanical Engineering, Federal University of Technology, P.M.B 65,
Minna, NIGERIA
[3]Department of Electrical and Electronics Engineering, Federal University of Technology, P.M.B 65, Minna, NIGERIA
[4]Department of Telecommunication Engineering, Federal University of Technology, P.M.B 65,
Minna, NIGERIA
[5]Department of Applied Data Science, Institute for Energy Technology, Halden, NORWAY

Abstract: Due to the rapidly evolving technology in the dynamic world, there is a growing desire among software clients for swift delivery of high-quality software. Agile software development satisfies this need and has been widely and appropriately accepted by software professionals. The maintainability of such software, however, has a significant impact on its quality. Unfortunately, existing works neglected to consider timely delivery and instead concentrated primarily on the flexibility component of maintainability. This research looked at maintainability as a function of time to rectify codes among Individual Junior and Random pair software developers. Data was acquired from an experiment performed on software developers in the agile environment and analyzed to develop the quality model metrics for maintainability which was used for prediction. One hundred programmers each received a set of agile codes created in the Python programming language, with deliberate bugs ranging from one to ten. The cubic regression model was used for predicting time spent on debugging errors above ten bugs. Results show that the random pair programmers spent an average time of 21.88 min/error while the individual programmers spent a lesser time of 16.57 min/error.

Keywords: Agile

Received: June 21, 2022. Revised: August 22, 2023. Accepted: October 5, 2023. Published: November 6, 2023.

1. Introduction

Software engineering circles have talked about how software development should be structured to provide faster, better, and less expensive solutions. Numerous recommendations for improvement have been made. This includes a wide range of practical tools, techniques, and practices, as well as the standardization and measurement of the software process [1]. The majority of the recommendations for improvement have been made by skilled software professionals who have each established their own strategies and techniques to address the anticipated change.

The agile manifesto was created in February 2001 in Utah by a group of seventeen software engineers who met to discuss the issues facing the software industry. The term "Agile methods" refers to a group of several methods and practices that share the same ideals and fundamental tenets. These rapid development techniques are actually a response to the conventional plan-based approach, which primarily emphasizes an organized, effective, and reasonable engineering-based strategy [2]. According [3] agile processes promote sustainable development. Agile, as it is commonly known, is a point-in-time iterative methodology that promotes a quick and adaptable response to change by anticipating interactions throughout the development cycle. [4].

In the traditional approach to software development, problems can be fully specified. Optimum and predictable solutions are proffered to every problem which involves rigorous planning, codified processes, very thorough and meticulous reuse of codes thereby making the development activity efficient. The agile software development process, on the other hand, approaches the problem of an unpredictable world by putting more of an emphasis on people and their ingenuity than on processes [5]. According to [6] Agile focuses on delivering individual parts of the software.

The pairing of the pair programmers has a significant impact on pair programming's overall success [7]. Based on the programmers' experience, some studies have discussed the effectiveness of pair programming. Only a few of the research on pair programming that discussed the impact of programmer expertise provided precise metrics for the concept used. [7] used two indicators for categorizing programmer expertise. These indicators are assessment by the project managers and the results of a pretest programming task. [8] decided on programmer's expertise by computing the student's weighted Grade Point Average (GPA) in programming courses taken at the university. [9] discovered that, the performance of programs increases with the number of programmers' years of experience, and decreases with the number of years of experience.

According to [10], Despite the success of the agile approach in smaller projects, there haven't been many thorough analyses of its application to the development of large-scale software. Although [11] conducted an experiment on the cost of developing software with input primarily consisting of the working time spent on development, the work did not consider the time spent on debugging of codes. Therefore, this work seeks to fill in the research gap.

2. Methodology

In this work, the basis of pair composition is programmer expertise which is based on previous experience of pair programming. The term Random pair and Individual Junior were used in this study to produce a binary representation for the levels of programming expertise in a project. A junior programmer is a person with less than five years of project-related experience. This is in line with common practice in Software Engineering research [12].

They were further grouped to work as pairs, where two programmers work together on a task and individuals. According to [13], the pairing was based on their knowledge and experience of pair programming. The grouping is as shown in Table 1:

TABLE 1. GROUPING OF SOFTWARE PROGRAMMERS

Grouping	Remark
Random pairs	Regardless of how long they have been working as pair programmers.
Individual Junior	Agile approach experience of fewer than five years.

The codes were extracted from an existing development project; Smart Revenue System. The forked link is https://github.com/ajiboyemary/phd2/blob/master/controllers/api.py
Errors from one to ten were introduced into the python codes as and these were given to one hundred individual junior programmers and same set of codes were also given to one hundred pair programmers randomly (years of experience not considered) and time taken to debug various number of errors was acquired as recorded in a log file.

2.1 Statistical Tools

Different types of variance analysis are provided by the Analysis of Variance (ANOVA). Analysis of variance on parametric data taken from a known population for three or more samples can be achieved. In this work, ANOVA was used to compare the average time spent on an error and the time spent on each of the project containing different number of bugs between the different pair programmers and the different individual programmer. Significant means were separated using the Duncan Multiple Range Test (DMRT). DMRT is sensitive and used for separation of means within the range of 3 and 10 samples.

Duncan created the multiple range test in 1955, which is a widely used method for comparing all pairs of means. The computation of numerical bounds that enable the assessment of the difference between any two treatment means as significant or non-significant is required for the use of Duncan's Multiple Range Test (DMRT). The computation of a number of values, each of which corresponds to a particular set of pair comparisons, is necessary for DMRT. The mean difference's standard error is what matters most. Using the estimated variance of an estimated elementary treatment contrast from the design, this is simply to be calculated. According to the preferences of the character being studied, rank all the treatment options for DMRT application in either descending or ascending order.

2.2 Correlation Coefficient

The correlation coefficient measures how much two measurements X and Y, differ from one another. Each set of measurements is examined via correlation analysis to see whether there is any tendency for them to move in tandem. Between -1 and +1, the correlation coefficient can take on any value. When big values of one variable tend to be correlated with large values of the other, a positive correlation is obtained. When small values of one variable tend to correlate negatively with big values of the other, and when the values of both variables tend to be unrelated, the correlation is close to zero (zero). Bivariate correlation given in Equation 1, was used to check the relationships between the number of bugs in projects and the time spent to correct the errors. Bivariate shows relationship between two variables *x* and *y*.

$$Correl(X,Y) = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2\sum(y-\bar{y})^2}} \quad (1)$$

2.3 Regression Models

The Regression analysis analyzes how the values of one or more independent variables affect the value of a single dependent variable. The outcomes can be used to forecast how a brand-new, untested data collection will perform. Bivariate analysis, where exactly two measurements are made on each observation was used.
The cubic regression model for the Random pair and individual junior programmers are shown in Table 2.

TABLE 2. CUBIC REGRESSION MODEL OF THE NUMBER OF BUGS DEBUGGED IN A PROJECT AND THE TIME TAKEN FOR THE DEBUGGING

Pair type	R	R^2	Significance of the relationship
Random	0.859	0.738	*
Individual (Junior)	0.661	0.437	*

Y (Dependent variable): Time spent on debugging (min)
X (Independent variable): Number of bugs in a project
* Significant at 5% level
NS: Not significant at 5% level

Cubic regression model was used to generate metrics with time spent on error as dependent variable *EY(t)* and number of bugs as independent variable t for each of the pair of random programmers and individual junior programmers as shown in Equation 2.

$$EY(t) = \beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3 \quad (2)$$

According to [14], the equation for the best fit model (cubic model) is shown in Table 3 revealing the model metric equations. The point at which a change in the time spent on error was experienced based on cubic regression model for both groups.

TABLE 3. CUBIC REGRESSION MODEL EQUATIONS

	Equation
Random	$Y = 0.461\,x^3 - 7.262\,x^2 + 47.868x - 40.528$
Individual (Junior)	$Y = 0.346\,x^3 - 5.904\,x^2 + 45.182x - 13.166$

R-square (R2): is a metric that expresses how well the anticipated values match the collection of measured data. The correlation between the actual response and the projected response is measured by R-square. It is also known as the coefficient of multiple determinations and the square of the multiple correlation coefficients given in Equation 3.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2} \quad (3)$$

where, y_i is the measured data, $\hat{y}_i$ is the predicted data and $\bar{y}_i$ is the data's mean as measured. For models without a constant, R^2 can take on any value between 0 and 1, but it can also be negative, which shows that the model is inapplicable to the data. A value that is nearer 1 means that more of the variance is explained by the model. On the average, cubic model gave the highest R^2 value of 0.644 in comparison to other models. Therefore, the patterns of the relationship between the dependent and independent variable were obtained by calculation using the best fit model.

3. Results of Debugging Time

Results for the cubic prediction model for Random pair and individual junior programmers is hereby presented. Figure 1 compares the time taken by a Random pair of junior programmers and an individual junior programmer to correct errors ranging from 1 to 10. The outcome demonstrates that, despite following a similar pattern, the time spent by each junior programmer was consistently higher than that of the random pair programmers.

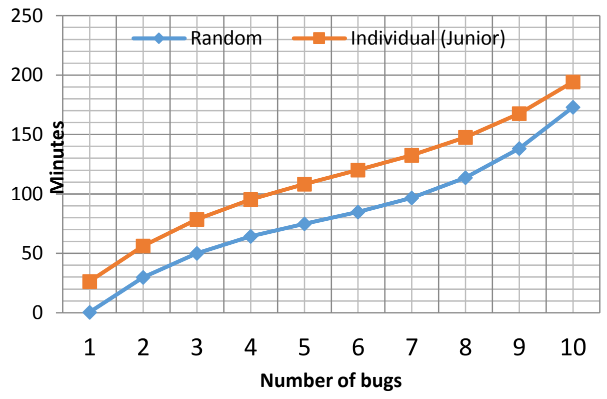

Figure 1 Comparison of Random pair and Individual-Junior for 1-10 bugs

However, in Figure 2 and after the eleventh bug, the pattern altered, and the random programmer's debugging time began to exceed that of the individual junior programmers.

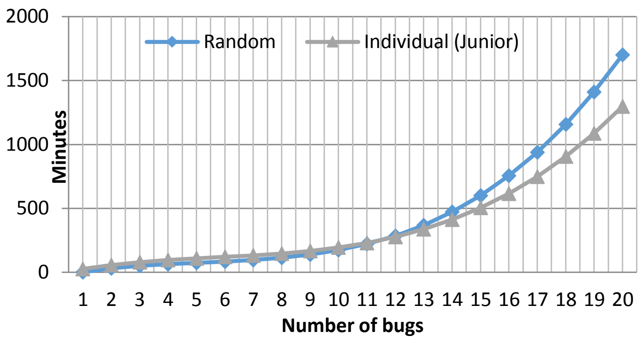

Figure 2 Comparison of Random pair and Individual Junior for 1-20 bugs

This trend continued for up to 100 bugs as shown in Figure 3 and Figure 4. For random pair, the time spent on debugging 10, 20, 30 and 100 errors are 173mins, 1700mins and 7307mins and 393196mins respectively while those of the individual junior are 194mins, 1297mins, 5371mins and 291465mins respectively. The bugs were divided into groups of 10, and the averages of each group for several programmers were acquired to determine how long it took to fix each bug.

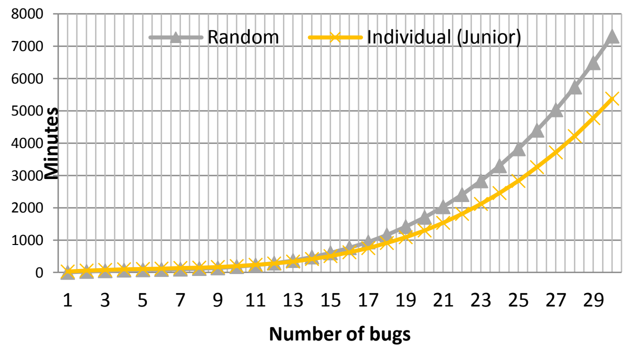

Figure 3. Comparison of Random pair and Individual Junior for 1-30 bugs

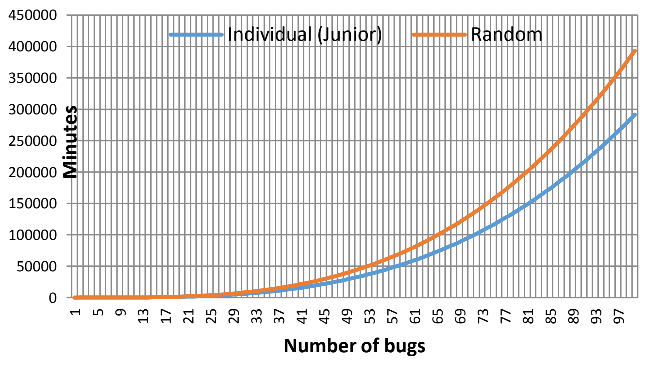

Figure 4. Comparison of Random pair and Individual-Junior for 1-100 bugs

The average time spent in correcting an error in a system software by both programmer expertise is shown in Table 4.

Table 4. Average time spent on debugging system software error

Pair group	Average time spent on error (min / error)
RANDOM PAIRS	21.88 ± 7.37^a
INDIVIDUAL JUNIOR	16.57 ± 3.36^b

± Means standard deviation on the same column with different superscript are significantly different ($p < 0.05$).

4. Conclusion

The average time spent by the programmers on an error showed some level of significant difference ($p < 0.05$). The randomly paired programmers spent the highest average time (21.88 min) on correcting an error which was significantly higher than individual junior programmers. The Individual Junior spent statistically comparable average time on correcting a system software error. While, between 1 and 50 bugs, younger programmers' average debugging time is higher than that of a random pair, while between 50 and 100 bugs, the situation is reversed.

References

[1] P. Kaushal, S. Anju "Review of Agile Software Development Methodologies" International Journal of Advanced Research in Computer Science and Software Engineering, vol. 3 issue 2, pp. 123-133, 2013.

[2] S. Nerur, R.. Mahapatra, G. Mangalaraj "Challenges of Migrating to Agile Methodologies" Communications of the ACM, vol. 48 issue, pp. 72–78, 2005.

[3] R. Healy, T. Dey, K. Conboy, B. Fitgerald "A Novel Technique to Assess Agile Systems for Stability" 24th Intrnational Conference on Agile Software Develoment, XP 2023, Amsterdam, the Netherlands, proceedings. pp. 30-43, June 13-16, 2023.

[4] L. Gonçalves, S. de Oliveira., J. Pacheco, P. Salume "Competências requeridas em equipes de projetos ágeis: um estudo de caso em uma Edtech" Revista de Gestão e Projetos, vol. 11, issue 3, pp. 72-93, 2020.

[5] T. Dyba "Improvisation in Small Software Organizations" IEEE Computer Society, vol. 17, issue 5, pp. 82–87, 2000.

[6] A. Gheorghe, I. Gheorghe, I. Iatan "Agile Software Development" Informatica Economică vol. 24, issue. 2, 2020.

[7] O. Demir, S. Seferoglu "he Effect of Determining Pair Programming Groups According to Various Individual Difference Variables on Group Compatibility, Flow, and Coding Performance" Journal of Educational Computing Research, vol. 59, issue 1, 20 August, 2020.

[8] V. Balijepally, R. Mahapatra, S. Nerur, K. Price "Are Two Heads Better than One for Software Development? The Productivity Paradox of Pair Programming" MIS Quarterly, vol. 3 issue 1, pp. 91-118. 2009.

[9] J. Nosek, "The Case for Collaborative Programming. Communication of the ACM", vol. 41 issue 3, pp. 105-108. 1998.

[10] Edison, H.; Wang, X.; Conboy, K. Comparing Methods for Large-Scale Agile Software Development: A Systematic Literature Review. IEEE Trans. Softw. Eng. 48, pp. 2709–2731, 2022.

[11] M. Sallin, M.Kropp, C. Anslow, R. Biddle "A Novel Technique to Assess Agile Systems for Stability" 24th Intrnational Conference on Agile Software Develoment, XP 2023, Amsterdam, the Netherlands, proceedings. pp. 50-66, June 13-16, 2023.

[12] E. Herel, E. McLean "Effects of using Non Procedural Computer Language on Programmer Productivity" MIS Quarterly, pp. 109-120, 1985.

[13] S. Misra "Pair Programing: Empirical Investigation in an Agile Development Environment, Springer Nature Switzerland AG, pp. 195-1999, 2021.

[14] M. Ajiboye "Development of Quality Model Metrics for Agile Software Engineering" PhD Thesis, Federal University of Technology, Minna, Nigeria, 2016.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.