ISSN:2249-5789

Reducing Defects in Software Development Process using Pair Programming

[1] P.Prabu, [2] Dr. S.Duraisamy

[1] Assistant Professor, Department of MCA
prabukiwi@gmail.com
[2] Professor, HOD of Computer Applications,
Sri Krishna College of Engineering and Technology, Coimbatore

Abstract - Pair or collaborative programming is a software development technique where two programmers develop software side by side at one computer. One person types in while the other reviews each line of code making it possible to create and continuously review what is being created. Although there has been a large amount of research investigating pair programming in an industry setting, there has been little work done with pair programming in an academic setting. To increase a body of evidence regarding the real benefits of pair programming, we investigate its relationship with software defects. The analysis is based on 3-months data collected from a large Software company. The team of 17 developers adopted a customized version of extreme programming and used pair programming on a daily basis. We explore and compare the defect rate of the code changed by doing pair and solo programming. The results show that defects appear to be lower for the code modified during pair programming. As a consequence, we formulate a hypothesis that pair programming is effective in reducing the introduction of new defects when existing code is modified.

Keywords: *Extreme Programming, Pair Programming, REAP.*

I. Introduction

There have been claims that pair programming may improve software development under several perspectives. A significant numbers of empirical studies have been conducted and results have been used in support of such claims. The apparent benefits include (a) reducing defect rate [3, 16,18, 19, 21-24], (b) improving design [4], (c) increasing productivity [11, 13, 24], (d) shortening the time-to-market [6,8, 24, 26, 27], (e) enhancing knowledge transfer and team communication [3, 5, 6, 22, 23], (f) increasing job satisfaction [3, 20], (g) facilitating integration of newcomers [9], and (h) reducing training costs [25].

However, there are also studies that do not confirm such outcomes, especially regarding productivity and cost-efficiency. The study in [15] indicates no positive effect of pair programming in term of development times. In [10], pair productivity is varied across projects. A large experiment conducted by [1] shows that neither does pair programming reduce time required to correctly perform change tasks nor increase the percentage of correct solutions. Begel and Nagappan reports in [3] the survey results from Microsoft mentioning high skepticism over pair efficiency.

In particular, the effects of pair programming on the code quality appear to be inconsistent across situations. In [1], the correctness of solutions increases when defects and adjust implementation strategies just when the code is written. Several studies [10, 16, 23, 27] indicate positive effects of pair programming only when using a particular quality measure. Hence, its effectiveness depends on the combination of the features of the subjects, the performed tasks, and the choices of employed quality measures. The diversity of such situational variables creates inconsistency on the results from different studies. In addition, it reflects the need to replicate the studies for each situation to confirm the previous findings.

In addition, it is difficult to generalize the results of existing studies to industrial settings, especially when the representativeness of experimental settings is considered. Most of the studies were conducted in educational settings. Data coming from classroom experiments, especially inexperienced students attending an introductory computer science course, pose serious concerns as they appear very different from the data coming from industry. Students have limited programming experience and knowledge in task domain; while generally it is not the case for professional developers. Moreover, the tasks performed in the experiments are mostly small isolated development tasks. Very few studies focus on maintenance tasks [1, 28] which are an essential area in long term software projects.

To enlarge the body of evidence, our study investigates the relationship between pair programming practices and the defects of produced code in two situations: 1) when it is used for defect corrections, and 2) when it is used for implementing user stories. The analysis is based on several

data sources gathering for about 14 months from a large Italian manufacturing company. The observed team has adopted XP and used pair programming on a daily basis. The differences on the defect rates between the parts of code involved during pair programming sessions and the parts not involved during pair programming are presented as the result. Due to the limitation of observational data we used, the significant results are used to form hypotheses for further study.

II. Background

Werner et al and other three authors [9] investigated the effects of pair programming on members performance and subsequent pursuit of computer science-related degrees among both female and male college members taking an introductory programming course designed for computer science related majors. They collected data on 554 members who attempted the course at the University of California-Santa Cruz. Data was collected from a total of four sections of the course: Fall 2000, Winter 2001 (two sections), and Spring 2001. One of the principle investigators of this study, Charlie McDowell, taught the fall and spring sections of the course. The winter 2001 sections were taught by UCSC faculty members.

In 1998 Nosek [5] conducted an experiment with five pairs and five individual professionals solving a challenging problem. They were asked to write a UNIX script that performed a database consistency check. The programmers were well versed in the Unix script but had not performed that kind of task before. The controlled experiment showed that pair programming shortened the elapsed time and produced better software quality than individual programming. This creates evidence that collaboration improves the problem-solving process and produces more efficient code

In an academic environment, the most cited study is probably that described in [9] in which 13 university students worked individually on a project and 28 chose to work in pairs. The findings showed that code produced by the pairs passed more automated tests over four different programming exercises. This resulted that pair programming in the software development yields better products in less time. The programmers feel happier, more confident.

However, a common feature of the existing studies is that the basic understanding of the effectiveness of pairs in the coding phase alone and they have not addressed in all the phase of software development. This paper focuses on developing a model for software development using Pair Programming and assessing the effectiveness, variation in effectiveness between pairs of varying skills with respect to all phases of software development.

III. Software Development Model Using Pair Programming.

The software development process involves many stages namely requirement analysis, designing, implementation, testing and maintenance.

The development model represented in figure 1 contains all the stages and each and every stage is tuned by the practice of Pair Programming.

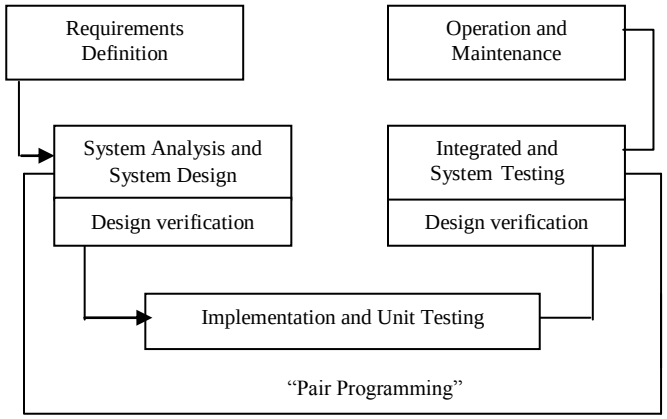

Figure 1: The Development Model

The analysis stage, actually a set model, is the first technical representation of a system. Pairing in the analysis phase is a constructive approach will result in an effective analysis of the system and comes out well defined models.

Software design requires dealing with many levels: implementation, database, business logic, presentation, deployment, interaction with other systems [7]. Pair working is applied to the design phase of the software development and results are given in terms of pair designing to improves performance.

Implementation needs a multi-layered, multidimensional model capable of supporting mental simulations and amount of knowledge required to its suitable organization [6]. One possible method of taming the complexity of software development may be to work collaboratively.

The figure 2, explains the practice of PP, in the implementation phase. Defects identified by the observer, is then there reviewed, the final outcome of process will be defect less or only has least number of defects.

IV. Experimental Setup

In this work the effectiveness of Pair Programming in software development process can be highlighted by conducting experiments between solo versus pair work at each software development phase.

We conducted an empirical analysis among 66 employees those are working as Software Engineer in Valtech. A skill test has been conducted for the employees to test their core conceptual knowledge, design oriented skill, programming skills and other skills for working in software projects. Based on the skill test result and the previous project performance, 36 employees were termed as Experts and 30 were termed as Novice.

Entire 66 employees were split into two groups comprising 20 and 46 employees. First group comprising 20 employees (10 experts & 10 novices) were made to work individually. The other group comprising 46 employees were made into 23 pairs. These 23 pairs were made in such a way that 8 Pairs are formed as Expert-Expert Pair, 10 pairs are formed as Expert- Novice Pair, and 5 pairs are formed as Novice-Novice Pair.

One programming problem was given to every pair and solo programmer to work upon to implement it. Analysis shown here is based on the code done by the pair and solo programmer and the effectiveness of pair programming were measured. Similar analysis is done in all phases.

V. Quantitative Results

The quantitative results analysis focuses on each and every stage of development done by the employees and facts obtained from the experimental study are given in the form parametric values taken for each phase of the software development process.

Parameter 1: Duration

Duration was defined as the total time taken to complete the analysis of the system. To end with a correct measurement, Time Log sheets were used to record the current time, total time (in minutes) for that task.

For the duration measure to be meaningful, we considered duration only employees with correct solutions. The table 1 gives the tabulated values of duration parameter.

In pair programming, duration can be measured in two ways: One is elapsed time to complete the task and the other is the total effort/time of the programmers completing the task.

Table 1: Tabulating Duration Parameter

PAIRS				
Subjects	Problem 1(DD)		Problem 2(DD)	
	Pair No.	Defects Density	Pair No	Defect Density
Expert - Expert	1	38	5	37
	2	43	6	41
	3	46	7	34
	4	35	8	45
Expert - Novice	9	38	14	53
	10	47	15	49
	11	48	16	55
	12	54	17	52
	13	49	18	57
Novice-Novice	19	56	22	65
	20	67	23	69
	21	63	-	-
INDIVIDUAL				
Expert	1	48	6	49
	2	51	7	52
	3	56	8	46
	4	45	9	54
	5	53	10	58
Novice	11	67	16	78
	12	72	17	67
	13	76	18	74
	14	78	19	69
	15	89	20	81

In this, we incorporate both important measurements of time in a single measurement, that is, the Relative Effort Afforded by Pairs (REAP). REAP is used to measure for non programming related tasks, for example analysis related tasks.

$$REAP = \frac{(finish\_time\_of\_pair)\times 2-(finish\_time\_of\_individual)}{(finish\_time\_of\_individual)}\times 100 \quad (1)$$

There are five cases for us to consider with REAP:

- REAP < 0,
- REAP = 0,
- REAP is between 0 and 100,
- REAP = 100, and
- REAP > 100.

When REAP is negative, the total time of pair programmers is less than the time of the individual programmer, that is, pairs are actually more efficient than a single pair programmer and it is less costly to use pair programmers than individual programmers. The table 2 gives the calculated REAP values for various pairs.

If REAP is zero, this is a break-even point, where the total time of pair programming is the same as individual

programming, but pair programming halves the elapsed time required for individual programming.

When REAP is greater than zero but is less than 100 percent, pairs require more total man hours to complete the task but are faster than individual programmers, that is, the elapsed time to complete is less for pairs than for individual programmers.

If REAP is around 100 percent, the elapsed time for pair programmers is almost the same time as in the individual programmer; therefore, pair programming doubles the total man hours as compared to individual programming. When REAP is greater than 100 percent, then the elapsed time for pair programming is longer than the time for an individual programmer. The table 2 gives the REAP values for different level of pairs.

Table 2: Tabulation of Reap Values

Expert – Expert pair with Expert Individual			
Problem 1		**Problem 2**	
Pair No	**REAP value**	**Pair No**	**REAP value**
1	58.33	5	51.02
2	68.52	6	57.69
3	64.28	7	47.82
4	55.55	8	66.66
Expert – Novice pair with Novice Individual			
9	13.43	14	35.89
10	30.55	15	58.20
11	26.31	16	48.64
12	38.46	17	50.72
13	10.11	18	40.74
Novice – Novice pair with Novice Individual			
19	67.16	22	66.66
20	86.11	23	105.9
21	65.78	-	-

The REAP values obtained for this pair is greater than zero but less than hundred. This implies pair requires more man hours but the pairers are faster than Individuals.

The results show that many of the pair programmers were done faster than that of the Individual programmers. Only few of the pairs results does not show significant difference between pair programmers and Individual programmers

The results show that, some of the pairs are really faster than individual. Similar manner duration and effort analysis is done for all the phases of software development process and found the results were favor to pair work

Parameter 2: Correctness of the Solution

Correctness of the solution in the analysis phase can be defined in the following way a) All the needs of the customer as transferred to requirements of the software. b) Use cases identifications are done correctly. The measurement of the correctness of the solution is done by means of Binary functional correctness score. The binary score is with value 1, when both the tasks t1 and t2 are done correctly. The binary score is 0, if any one of the tasks t1 and t2 contains errors. the pair and solo group for the given. The correctness of the solution parameter can be taken in all phases and the results are given. The tabulated values of correctness of the solution are given table 3.

The mean values are taken according to subject for the result analysis in the comparative way. The table 4 shows the mean values.

Table 3: Correctness of the Solution

PAIR				
Subjects	**Problem 1**		**Problem 2**	
	Pair No.	**Binary score**	**Pair No**	**Binary score**
Expert - Expert	1	1	5	1
	2	1	6	0
	3	1	7	1
	4	1	8	1
Expert – Novice	9	1	14	1
	10	0	15	1
	11	1	16	1
	12	0	17	0
	13	1	18	1
Novice-Novice	19	1	22	1
	20	1	23	0
	21	0	-	-
INDIVIDUAL				
Expert	1	1	6	1
	2	1	7	1
	3	0	8	1
	4	1	9	0
	5	0	10	0
Novice	11	1	16	0
	12	1	17	0
	13	0	18	1
	14	0	19	1
	15	0	20	0

Table 4: Mean Values of Correctness of the Solution

Subject	Program 1	Program 2
	Mean value	Mean value
Expert - Expert	1	0.75
Expert - Novice	0.6	0.8
Novice - Novice	0.66	0.5
Expert	0.6	0.6
Novice	0.4	0.4

The graphical representation of the percentage of correctness of the solution of solo versus pair programmers are figured in figure.3. The graphs show pair programmers have more correctness in their solutions.

Parameter 3: Defect Density Rate

We used the ratio of defects per lines of code to measure the quality of code. The benefit of using the defect density, instead of the absolute number of defects, is that it is normalized and comparable among methods of different size. It is also due to the empirical evidence that a total number of defects have a pattern of relationship with lines of code. Defect density, calculated as

$$DD=D_T/LOC \quad (10)$$

Where D_T is Total number of defects and LOC is Lines of code. LOC can be measured either Function point Method (COSMIC) or Software Size.

Defect density has been calculated both for traditional pair and conglomeration pair. After removal of defects the modification rate can be calculated as

$$MR= (A_{LOC}+D_{LOC}) / (LOC+ A_{LOC}) \quad (11)$$

Where A_{LOC} Total Added Lines of code and D_{LOC} Total Deleted Lines of Code

VI. Formal Tests of Hypotheses

In this research, the paired sample t- test(Easton *et al.*, 1997) has been used as the statistical technique for validating the experimental results. Generally the paired sample t-test is used to determine whether there is a significant difference between the average values of the same set of quantifications made under two different conditions. The usual null hypothesis is that the difference in the mean values is zero. This null hypothesis may be tested against any of the appropriate alternative hypotheses, depending on the question posed. The paired sample t-test is a more powerful alternative for the two sample t-test.

By adopting the principle of the paired-sample t-test, here it is conducted to establish the effectiveness of the pair programming over individual. In this research, the paired sample t-test is being carried out in three steps as described below:

Step-1: Definition of Hypotheses: The null and alternate hypotheses are defined such that to factually explore the differences between the proposed and existing methodologies as given below:

i). Null Hypotheses (H0) : There is no difference between the pair programming and individual programming.

ii). Alternative Hypotheses (H1): There exists difference between the pair programming and individual programming.

Step-2: Calculation of t-statistic for the given data set: The theoretical estimate of the t-value can be found using the following Eqn. (7.10).

$$t_{calc} = \frac{M_x - M_y}{\sqrt{\frac{S_x^2}{n_x}+\frac{S_y^2}{n_y}}} \quad - (2)$$

where,

$$S^2 = \frac{\sum(x-M)^2}{n-1}.$$

M_x – Mean of sample x, here x refers to the pair programming approach.

M_y – Mean of sample y, here y refers to the Individual approach.

S_x^2 – Variance of sample x.

S_y^2 – Variance of sample y.

n_x – Sample size of x.

n_y – Sample size of y.

df = $n_x - 1$ or $n_y - 1$, when $n_x = n_y$ and $S_x^2 \neq S_y^2$.

Step-3: Comparison between the tabulated and calculated values: The term t_{calc} refers to the estimated value of *t*, whereas t_{tab} refers to the tabulated *t*-value for the appropriate significance level and degree of freedom. If $\boldsymbol{t_{calc} > t_{tab}}$, null hypothesis *H0*is rejected and thereby the alternate hypothesis*H1*is accepted. Otherwise, the *H0* should be accepted by rejecting the *H1*.

VII. Statistical Results

The paired sample t-test was conducted over the two different result sets of Expert and Novice for the estimate Time

taken (Tt) and are tabulated in Table 7. The findings of the statistical analyses from Table 7 are reported as follows:

- The t-test results of Tt(Time taken)shows that there is a significant difference between the scores for Expert (M=50.6, SD=4.277) and Novice (M=76.4, SD=8.20). Since the absolute value of t_{calc} (-7.17) is greater than t critical two-tail (2.77), Null Hypothesis (H0) is rejected. Moreover since the probability P(T<=t) two-tail (0.002) is less than α (0.05), Null Hypothesis is strongly rejected and hence the Alternate Hypothesis (H1) is accepted at the confidence level of 95%.

Table 5: Measures of Duration taken by different pairs with Statistical Result-Problem 1

Completion Time

	N	Mean	SD	SE	95% Confidence Interval for Mean		Min	Max
					LB	UB		
Expert-Expert	5	41.20	4.550	2.035	35.55	46.85	35	46
Expert-Novice	5	47.20	5.805	2.596	39.99	54.41	38	54
Novice-Novice	5	68.00	10.050	4.494	55.52	80.48	56	83
Total	15	52.13	13.627	3.518	44.59	59.68	35	83

S.No	Problem1(Defects)		Problem 2(defects)	
	Expert	Novice	Expert	Novice
1	48	67	49	57
2	51	72	52	59
3	56	76	46	51
4	45	78	54	69
5	53	89	58	72
Paired Sample T-Test Result				
Mean (M)	50.6	76.4	51.8	61.6
Std. Deviation (SD)	4.27785	8.203658	4.604346	8.70632
Variance	18.3	67.3	21.2	75.8
Observation	5		5	
Degree of Freedom	4		4	
t_{calc}	-7.1722029		-4.937168674	
P(T<=t) one tail	0.001000513		0.003916303	
T Critical one tail	2.131846786		2.131846786	
P(T<=t) one tail	0.002001027		0.007832605	
T Critical two tail (t_{tab})	2.776445105		2.776445105	
Confidence Level (α)	95% (0.05)		95% (0.05)	

Completion Time

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	1978.133	2	989.067	19.09	.000
Within Groups	621.600	12	51.800		
Total	2599.733	14			

If p < 0.5, so we reject the null hypothesis. The means of the three populations are not all equal. At least one of the means is different. So Expert-Expert pair is better then Expert individual. In the test case number of test pass of Expert-Novice is higher then the Expert individual. If F > F crit, we reject the null hypothesis. This is the case, 19.09395>3.885294. Therefore, we reject the null hypothesis.

VIII. Conclusion

The contributions of this work are twofold. The first part summarizes the empirical knowledge on the effects of pair programming on the quality of produced code, as compared to solo programming. The second part presents the relationship between pair programming practices and the defect rate in different circumstances, based on the data collected over 14 months from an industrial XP team. The summary of current knowledge shows that the effects of pair programming are inconsistent across different situations. Its effectiveness depends, to some extent, upon the features of subjects, the features of performed tasks, and the choices of adopted quality measures. In our exploratory analysis, the usage of pair programming is investigated in the context of the defect corrections and the implementations of user stories. The results shows that the defect rate in the code appears to be lower for the parts of code involved during pair programming practices than other parts. On the basis of this observation, we formulate a hypothesis that pair programming helps to reduce the introduction of new defects when existing code is modified.

References

[1] Arisholm, E., Gallis, H., Dyba, T., Sjoberg, D.I.K.: Evaluating pair programming with respect to system complexity and programmer expertise. IEEE Trans. Softw. Eng. 33(2), 65—86 (2007)

[2] Basili, V.: Applying the goal question metric paradigm in the experience factory. In Proceedings of the Tenth Annual Conference of Software Metrics and Quality Assurance in Industry (1993)

[3] Begel, A., Nagappan, N.: Pair programming: what's in it for me? In ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, pp. 120–128. ACM, New York, NY, USA (2008)

[4] . Canfora, G., Cimitile, A., Garcia, F., Piattini, M., Visaggio, C. A.: Evaluating performances of pair designing in industry. J. Syst. Softw., 80(8), 1317–1327 (2007)

[5] Chong, J., Hurlbutt , T.: The social dynamics of pair programming. In ICSE'07: Proceedings of the 29th international conference on Software Engineering, pp. 354–363. IEEE Computer Society, Washington, DC, USA (2007)

[6] Cockburn, A., Williams, L.: The costs and benefits of pair programming. pp. 223–243, (2001)

[7] Coman, I. D., Sillitti, A., Succi, G.: A case-study on using an automated in-process software engineering measurement and analysis system in an industrial environment. In ICSE'09: Proceedings of the 31st International Conference on Software Engineering, pp. 89–99. IEEE Computer Society, Washington, DC, USA (2009)

[8] Dyba, T., Arisholm, E., Sjøberg, D. I. K., Hannay, J. E., Shull, F.: Are two heads better than one? on the effectiveness of pair programming. IEEE Softw., 24(6), 12–15 (2007)

[9] Fronza, I., Sillitti, A., Succi, G.: An interpretation of the results of the analysis of pair programming during novices integration in a team. In ESEM '09: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, pp. 225–235. IEEE Computer Society, Washington, DC, USA (2009)

[10] Hulkko, H., Abrahamsson, P.: A multiple case study on the impact of pair programming on product quality. In ICSE '05: Proceedings of the 27th international conference on Software engineering, pp. 495–504. ACM, New York, NY, USA (2005)

[11] Lui, K. M., Chan, K. C. C., Nosek, J.: The effect of pairs in program design tasks. IEEE Trans. Softw. Eng., 34(2), 197–211 (2008)

[12] Madeyski, L.: The impact of pair programming and test driven development on package dependencies in object oriented design – an experiment. In Product-Focused Software Process Improvement, LNCS, vol. 4034, pp. 278–289 (2006)

[13] Muller, M. M.: Are reviews an alternative to pair programming? Empirical Softw. Engg., 9(4), 335–351 (2004)

[14] Muller, M. M.: A preliminary study on the impact of a pair design phase on pair programming and solo programming. Inf. Softw. Technol., vol. 48, pp. 335–344, (2006)

[15] Nawrocki, J., Wojciechowski, A.: Experimental evaluation of pair programming. In Proc. European Software Control and Metrics Conf. (ESCOM) (2001)

[16] Phongpaibul, M., Boehm, B.: An empirical comparison between pair development and software inspection in thailand. In ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, pp. 85–94. ACM, New York, NY, USA (2006)

[17] Sillitti, A., Janes, A., Succi, G., Vernazza, T.: Collecting, integrating and analyzing software metrics and personal software process data. In EUROMICRO '03: Proceedings of the 29th Conference on EUROMICRO, pp. 336. IEEE Computer Society, Washington, DC, USA (2003)

[18] Sison, R.: Investigating pair programming in a software engineering course in an asian setting. In APSEC '08: Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference, pp. 325–331. IEEE Computer Society, Washington, DC, USA (2008)

[19] Sison R.: Investigating the effect of pair programming and software size on software quality and programmer productivity. In APSEC '09: Proceedings of the 2009 16th Asia-Pacific Software Engineering Conference, pp. 187–193. IEEE Computer Society, Washington, DC, USA (2009)

[20] Succi, G., Pedrycz, W., Marchesi, M., Williams, L.: Preliminary analysis of the effects of pair programming on job satisfaction. In: Proceedings of the 3rd International Conference on Extreme Programming (XP), pp. 212–215 (2002)

[21] Vanhanen, J., Abrahamsson, P.: Perceived effects of pair programming in an industrial context. In EUROMICRO '07: Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 211–218. IEEE Computer Society, Washington, DC, USA (2007)

[22] Vanhanen, J., Korpi, H.: Experiences of using pair programming in an agile project. In HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences, pp. 274b. IEEE Computer Society, Washington, DC, USA (2007)

[23] Vanhanen, J., Lassenius, C.: Effects of pair programming at the development team level: an experiment. In International Symposium on Empirical Software Engineering. 2005, pp. 336-345 (2005)

[24] Williams, L., Kessler, R. R., Cunningham, W., Jeffries, R.: Strengthening the case for pair programming. In IEEE Softw., 17(4), 19–25 (2000)

[25] Williams, L., Shukla, A., Anton, A. I.: An initial exploration of the relationship between pair programming and brooks' law. In: Proceedings of the Agile Development Conference, pp. 11–20. IEEE Computer Society, Washington, DC, USA (2004)

[26] Phongpaibul, M., Boehm, B.: A replicate empirical comparison between pair development and software development with inspection. In ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, pp. 265–274. IEEE Computer Society, Washington, DC, USA (2007)

[27]] Nosek, J. T.: The case for collaborative programming. Commun. ACM, 41(3), 105–108 (1998)

[28] Xu, S., Chen, X.: Pair programming in software evolution. In: Canadian Conference on Electrical and Computer Engineering, 2005, pp. 1846 –1849 (2005)

Mr. Prabu is currently working as an Assistant Professor in the Department of Computer Application at Sri Krishna College of Engineering & Technology, Coimbatore, Tamil Nadu, India. He has more than 7 years of teaching and industrial experience. His research interests include Software Engineering, Webservice, Open Source Tools and Mobile Application.

Dr.S. Duraisamy has over 16 years of experience in teaching and is currently heading the Department of Computer Applications at Sri Krishna College of Engineering and Technology, Coimbatore, India. He received his Doctorate in Computer Science and Engineering from Alagappa University, Karaikudi, India. He has published many papers in various reputed international journals and guiding dozens of research scholars. His areas of interest include software engineering, object oriented concepts and wireless sensor networks.