Programmer Ranker Algorithm (PRA) for Evaluating Programmer Effort in the Context of Pair Programming

Manisha Giri, Meeta Dewangan

Abstract— Pair programming is a style of programming in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test. In industry, the practice of pair programming has been shown to improve product quality, improve team spirit, aid in knowledge management, and reduce product risk. In education, pair programming also improves student morale, helps students to be more successful, and improves student retention in an information technology major. Project efficiency of pairs in program design tasks is identified by using pair programming concept. Pair programming involves two developers simultaneously collaborating with each other on the same programming task to design and code a solution. Programming aptitude tests (PATs) have been shown to correlate with programming performance. In this paper we will measure time productivity using pair programming, in two important ways: One is elapsed time to complete the task and the other is the total effort/time of the programmers completing the task. Using Programmer Ranker Algorithm (PRA) we will generate pair and Rank will be provided to each pair of Junior, Senior of industry. After providing rank the best pair is allocated to Embedded Software project type, Semi detached Software project type and Organic Software project type respectively.

Index Terms: Pair programming, PAT, Collaborative programming, Team building, PRA.

1. INTRODUCTION

Software applications grow larger and more complicated; these applications are then used in an infinite myriad of user systems. Perhaps, then, it is best for the complexity of these applications to be tackled by two humans at a time. The idea of pair-programming, two programmers working collaboratively on the same design, algorithm, code, or test, has independently emerged several times over the last decade. The practice of pair-programming is gaining popularity, primarily with the rise in the extreme Programming methodology [12]. The concepts underlying Pair Programming (PP) are not new [21], but PP itself has only recently attracted significant attention and interest within the software industry and academics.

Pair programming is a software practice that involves a pair of programmers simultaneously collaborating with each other on the same programming effort [12], [9], [16]. One programmer controls the keyboard and implements the program. The other programmer watches, identifies defects, and considers the direction of the work. Sitting side by side at one computer, two colleagues collaborate on solving the problem, designing the algorithm, and coding.

Pairs regularly switch the driver and navigator roles and rotate their partners with other teams: This practice is thought to facilitate skills transfer and job rotation [15].

Some take the view that pair Programming is neither as economical nor as productive as individual programming [4], [7]. Others argue that more studies of pair programming productivity are needed [9], [7], [1], [18]. Some further explore pair programming such as side-by-side programming [8] and a mixed software practice of pair programming and individual programming [13], while others propose more traditional alternatives to pair programming suh as reviews [17] and mutual programming [5], [4].

Several previous controlled experiments have validated the following quantitative benefits of pair programming over individual programming.

1. Significant improvements in functional correctness.
2. Various other measures of quality of the programs being developed.
3. Reduced duration (a measure of time to market), with only minor additional overhead in terms of total programmer hours (a measure of cost or effort)
4. Reduced the elapsed time and produced better software quality.

• *Manisha Giri is currently pursuing masters degree program in computer science and engineering from Chhatrapati Shivaji Institute of Technology, Dueg (C.G.),India E-mail: manishagiri1@gmail.com*
• *Meeta Dewangan is working as an assistant professor in Deapartment of Computer Science Engineering Chhatrapati Shivaji Institute of Technology, Durg India, E-mail:meetadewangan@csitdurg.in*

One exception is an experiment that showed no positive effects of PP with respect to time taken no improved functional correctness of the software product compared with individual development [7], which essentially doubled the cost of development. However, the results of that experiment also suggested that the standard deviation of the development times and program sizes of the PP group was lower, suggesting that PP might be more predictable than individual programming.

Therefore, in controlled experiments where design related tasks were intermingled with coding, elapsed time was less and the quality was better for pairs. Another unique aspect of PP included the first ever assessment of the moderating effects of system complexity and programmer expertise.

Earlier studies reported that pair programming took more man hours than individual programming. One limitation of the previous experiment that was addressed to some degree in this experiment was that the task was much more complex/novel for the subjects.

In this project we will measure time productivity using pair programming, in two important ways: One is lapsed time to complete the task and the other is the total effort/time of the programmers completing the task. Using Programmer Ranker Algorithm (PRA) we will generate pair and Rank will be provided to each pair of Junior, Senior of industry. After providing rank the best pair is allocated to Embedded Software project type, Semi detached Software project type and Organic Software project type respectively.

The remainder of this paper is organized as follows: Section 2 provides a brief history of the use of pair programming. Section 3 identifies the problem in the existing system. Section 4 explains our approach of pair programming that is Programmer Ranker Algorithm(PRA). Section 5 provides the pair programming results. The final section provides concluding remarks and points some possible directions for future research.

2. Background

Since as early as 1991, cognitive researchers have been interested in how two programmers collaborate on the same task [19]. They reported that two programmers in a pair could generate more diverse plans and explore a larger number of alternatives than an individual programmer. In a faithful reenactment of a pair programming episode by two pair programming practitioners reported in [20], the pair spent more time talking, casually reasoning about requirements realization, data modeling, data structures, and semantic analysis, than discussing lexical analysis, syntax analysis, libraries of a computer language, and the integrated development environment (IDE). This suggests that pair programming may have benefits in situations such as design-related tasks, where alternative exploration can improve the solution. Some studies on pair programming attempted to simulate complex real situations so as to provide a rich picture of the behaviors inherent in pair programming. However, these programming tasks could not be done at a single time and the experimental task had to be split [16], [6]. The length of time between two experimental sessions can variably affect results [16], [6]. Oftentimes, like the real world requirements, descriptive programming tasks, instead of symbolic ones, were given to participants. As a result, some individuals misunderstood the problem, even at the beginning of the experiment [16], [6]. These studies have provided valuable information related to pair programming. However, because they were not strictly controlled experiments, it is difficult to sort out influences on the results.

3. Problem Identification

Programming teams [3] in industry in which pair programming was practices report significantly improved team work among the members. If the pair can work together, then they learn ways to communicate more easily and they communicate more often. In many cases, these industrial teams continually rotate partners; two people do not work together for more than a short increment. This increases the overall information flow and team jelling farther.

Analysis related to a multivariate model[23] that expresses Individual Performance as a predictor of Pair Performance. Moreover, Personality was included as a predictor of Individual Performance as shown in fig 1.

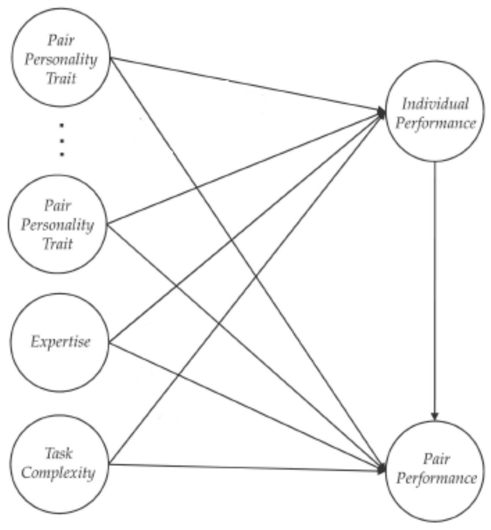


Fig. 1. Multivariate conceptual model(Individual Performance vs Pair Performance).

Earlier studies reported that pair programming took more man hours than individual programming. One limitation of the previous experiment that was addressed to some degree in this experiment was that the task was much more complex/novel for the subjects. As we know there are three types of Software projects, Embedded Software project type, Semi detached Software project type and Organic Software project required different knowledge and skill set so Using Programmer Ranker Algorithm (PRA) we will generate pair and Rank will be provided to each pair of Junior, Senior of industry. After providing rank the best pair is allocated to Embedded Software project type, Semi detached Software project type and Organic Software project type respectively.

4. Our approach

In pair programming, time productivity [24] can be measured in two important ways: One is elapsed time to complete the task and the other is the total effort/time of the programmers completing the task.
The effort equation is as follows :-

$$E = a\_b * (KLOC)\ b\_b$$

$$D = c\_b * (E)\ d\_b$$

Where E – effort applied by per person per month,
D – Development time in consecutive months,
KLOC – estimated thousands of lines of code delivered for the project.
The coefficients a_b, c_b, and the coefficients b_b, d_b are given in the Table:

Table 1: Coefficients & exponents used in the Basic COCOMO Model

Software Project Type	a_b	b_b	c_b	d_b
Organic	2.4	1.1	2.5	0.4
Semi-detached	3	1.1	2.5	0.4
Embedded	3.6	1.2	2.5	0.3

From above table we can say that Embedded Software project type require more effort as compare to Semi-detached and Organic. Hence Embedded Software project should be allotted to high ranked pair.

4.1 Programming Aptitude Tests(PATs)

Programming Aptitude Tests (PATs) can be correlated with programming performance. Aptitude tests in problem solving and algorithm design can be used to test the effect of pairs in these tasks. The results of PATs can be used to generate pairs. PAT score will calibrate as follows:-

Time productivity can be measured in two important ways: One is elapsed time to complete the task and the other is the total effort/time of the programmers completing the task. Both important measurements of time can be incorporated in a single measurement, that is, the Relative Effort Afforded by Pairs (REAP)[14]:-

$$REAP=\left\{\frac{finish\_time\_of\_pair\times 2(finish\_time\_of\_individual)}{Finish\_time\_of\_individual}\right\}$$

There are five cases to consider with REAP:

1. REAP < 0 When REAP is negative, the total time of pair programmers is less than the time of the individual programmer.
2. REAP $\approx$ 0 If REAP is zero, this is a break-even point, where the total time of pair programming is the same as individual programming.
3. REAP is between 0 and 100 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.

4. REAP $\approx$ 100 If REAP is around 100 percent, the elapsed time for pair programmers is almost the same time as in the individual programmer.

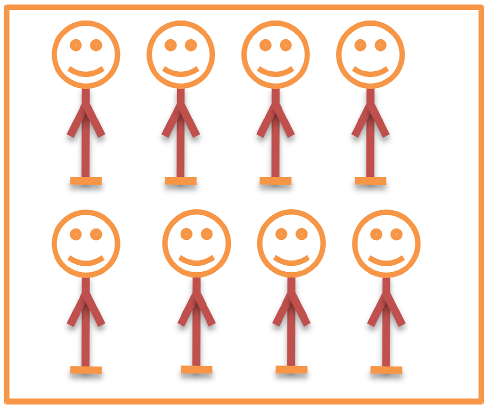
5. REAP > 100 When REAP is greater than 100 percent, then the elapsed time for pair programming is longer than the time for an individual programmer.

4.2 Programmer Ranker Algorithm(PRA)

```
Procedure Gen_Pair()
//indiTime -> Finish Time of Individual
        //p1Time -> Pair-I Finish Time
//p2Time -> Pair-II Finish Time
//p3Time = -> Pair-III Finish Time

    REAP1 = (((p1Time * 2) - indiTime) / indiTime) * 100;
    REAP2 = (((p2Time * 2) - indiTime) / indiTime) * 100;
    REAP3 = (((p3Time * 2) - indiTime) / indiTime) * 100;

     if (REAP1 < REAP2)
     {
        if (REAP1 < REAP3)
        {
           "The Pair One is Best compare to the others";
        }
        else
        {
           "The Pair three is Best compare to the others";
        }
    }
    else if (REAP1 > REAP2)
    {
        if (REAP2 < REAP3)
        {
            "The Pair two is Best compare to the others";
        }
        else
        {
            "The Pair three is Best compare to the others";
        }
    }

End Gen_Pair
```

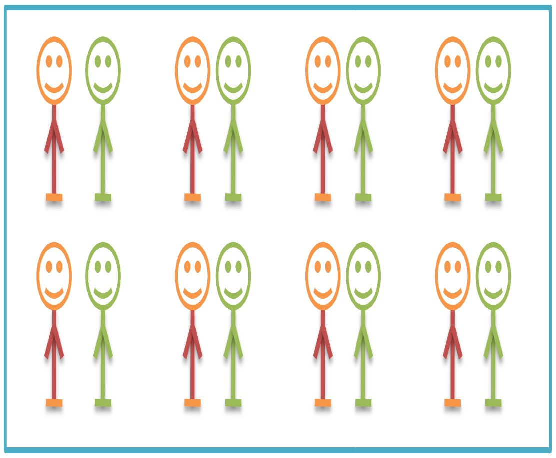
Pair will be generated among Junior and Senior Staffs of industry. Now our next procedure will evaluate correct pair among different pairs generated using Gen_Pair procedure.

Junior Staff (Individual)



Senior Staff (Individual)

Pair of Junior and Senior Staff of Industry(Pairs)



Procedure Pair_Rank()

```
// p1Time -> Pair-I Finish Time
// p2Time -> Pair-II Finish Time
// p3Time -> Pair-III Finish Time
.
.
.
.
// pnTime -> Pair-n Finish Time

//T=                /n
//n -> Total Number of Pair

for(i=1; i<=n ; i++)
{
        T=T+piTime
}
T=T/n
//pi[n] -> To Store piTime

//Sorting pi
SORT_PAIR(A, p, r)
        if p < r
         then q ← PARTITION(A, p, r)
         SORT_PAIR(A, p, q - 1)
         SORT_PAIR(A, q + 1, r)
End SORT_PAIR

PARTITION(A, p, r)
         x ← A[r]
         i ← p - 1
         for j ← p to r - 1
                 do if A[j] ≤ x
                 then i ← i + 1
                 exchange A[i] ↔ A[j]
                 exchange A[i + 1] ↔ A[r]
         return i + 1
end PARTITION
End Pair_Rank
```

5. Pair Programming Results

Anecdotal and empirical evidence reported in the literature suggest several organizational and personal benefits of PP over individual programming, such as reduced time to market,reduced development costs,improved quality of the software,reduced costs of training new personnel, and enhanced trust,motivation, and information and knowledge transfer among developers.

Fig 2 provides the comparison of pair programmers and individuals. It shows that the effort spent to develop the project can be reduced by pair programming. Programmer Ranker Algorithm(PRA) is used to generate pairs and the pairs generated by PRA can significantly reduce the Project development time and cost.

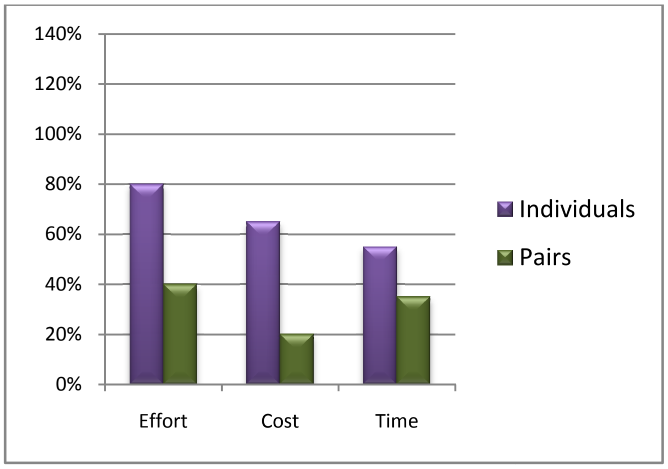


Fig 2. Comparison of pair programmers and individuals

6. CONCLUSION

The primary contribution of this study is to provide an overview of Pair Programming and to demonstrate the use of Programming Aptitude Test in the aspect of pair generation or team building that facilitates to make pair of newly hired programmers in an industry.

In our study, we have pointed out the use of PAT as a measurement of productivity and to evaluate the performance of individuals and pairs in order to generate the correct pairs. Our study showed that junior individuals may lack the necessary skills to perform tasks with acceptable quality, in particular, on more complex systems. Junior pair programmers achieved a significant increase in correctness compared with the individuals and achieved approximately the same degree of correctness as senior individuals. Software testing is often viewed as requiring less skill than initial system development and is thus often allocated to the more junior staff. Our study concludes that, if juniors are assigned to complex tasks, they should perform the tasks in pairs.

Programmer Ranker Algorithm (PRA) will generate pair and Rank will be provided to each pair of Junior, Senior of industry. After providing rank the best pair is allocated to Embedded Software project type, Semi detached Software project type and Organic Software project type respectively. This will reduce the time and effort requires developing the Embedded Software project which will eventually reduce overall cost of software.

References

[1] A. Parrish, R. Smith, D. Hale, and J. Hale, "A Field Study of Developer Pairs: Productivity Impacts and Implications," IEEE Software, vol. 21, no. 5, pp. 76-79, Sept./Oct. 2004.

[2] D.B. Mayer and A.W. Stalnaker, "Selection and Evaluation of Computer Personnel: The Research History of SIG/CPR," Proc. 23rd ACM Nat'l Conf., pp. 657-670, 1968.

[3] F. P. J. Brooks, The Mythical Man-Moth: Addison-Wesley Publishing Company,1975.

[4] G. Keefer, "Extreme Programming Considered Harmful for Reliable Software," Proc. Sixth Conf. Quality Eng. in Software Technology, pp. 129-141, 2002.

[5] G. Keefer, "Mutual Programming: A Practice to Improve Software Development Productivity," Proc. Int'l Conf. Practical Software Quality and Testing '03, 2003.

[6] H. Hulkko and P. Abrahamsson, "A Multiple Case Study on the Impact of Pair Programming on Product Quality," Proc. 27th Int'l Conf. Software Eng., pp. 495-504, 2005.

[7] J. Nawrocki and A. Wojciechowski, "Experimental Evaluation of Pair Programming," Proc. 12th European Software Control and Metrics Conf., pp. 269-276, Apr. 2001.

[8] J. Nawrocki, M. Jasin˜ ski, L. Olek, and B. Lange, "Pair Programming versus Side-by-Side Programming," Proc. 12th European Conf. Software Process Improvement, pp. 28-38, Nov. 2005.

[9] J. Nosek, "The Case for Collaborative Programming," Comm. ACM, vol. 41, no. 3, pp. 105-108, 1998.

[10] J.M. Wolfe, "A New Look at Programming Aptitudes," Business Automation, vol. 17, pp. 36-45, 1970.

[11] J.M. Wolfe, "Perspectives on Testing for Programming Aptitude," Proc. 25th ACM/CSC-ER Ann. Conf., pp. 268-277, 1971.

[12] K. Beck, Extreme Programming Explained: Embrace Change. Reading, Massachusetts: Addison-Wesley, 2000.

[13] K.M. Lui and K.C.C. Chan, "Software Process Fusion: Uniting Pair Programming and Individual Programming Processes," Proc. Int'l Software Process Workshop and Int'l Workshop Software Process Simulation and Modeling, pp. 115-123, 2006.

[14] Kim Man Lui, Keith C.C. Chan, and John Teofil Nosek "The Effect of Pairs in Program Design Tasks" IEEE transactions on software engineering, VOL. 34, NO. 2, march/april 2008

[15] L. Williams and R.R. Kessler, Pair Programming

Illuminated. Addison-Wesley, 2003.

[16] L. Williams, R.R. Kessler, W. Cunningham, and R. Jeffries, “Strengthening the Case for Pair Programming,” IEEE Software, vol. 17, no. 4, pp. 19-25, July/Aug. 2000.

[17] M. Ciolkowski and M. Schlemmer, “Experiences with a Case Study on Pair Programming,” Proc. First Int’l Workshop Empirical Studies in Software Eng., 2002.

[18] M.M. Mu¨ ller, “Two Controlled Experiments Concerning the Comparison of Pair Programming to Peer Review,” J. Systems and Software, vol. 78, no. 2, pp. 166-179, 2005.

[19] N. Flor and E. Hutcheins, “Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance,” Proc. Fourth Ann. Workshop Empirical Studies of Programmers, 1991.

[20] R.M. Martin, Agile Software Development: Principles, Patterns and Process of Software Development. Prentice Hall, 2001.

[21] T.D. Cook and D.T. Campbell, Quasi-Experimentation—Design & Analysis Issues for Field Settings. Houghton Mifflin, 1979.

[22] W.J. MeNamara and J.L. Hughes, “A Review of Research on the Selection of Computer Programmers,” Personnel Psychology, vol. 14, pp. 39-51, 1961.

[23] Jo E. Hannay, Erik Arisholm,Harald Engvik, and Dag I.K. Sjøberg,"Effects of Personality on Pair Programming" IEEE Transactions on software engineering, vol. 36, no. 1, january/february 2010.

[24] Nancy Merlo – Schett "COCOMO Model" Seminar on Software Cost Estimation Requirements Engineering Research Group Department of Computer Science University of Zurich, Switzerland, 2002 / 2003