A Framework for Understanding the Factors Influencing Pair Programming Success

Mustafa Ally, Fiona Darroch, and Mark Toleman

Department of Information Systems, University of Southern Queensland
Toowoomba Qld 4350 Australia
{Mustafa.Ally,Fiona.Darroch,Mark.Toleman}@usq.edu.au

Abstract. Pair programming is one of the more controversial aspects of several Agile system development methods, in particular eXtreme Programming (XP). Various studies have assessed factors that either drive the success or suggest advantages (and disadvantages) of pair programming. In this exploratory study the literature on pair programming is examined and factors distilled. These factors are then compared and contrasted with those discovered in our recent Delphi study of pair programming. Gallis et al. (2003) have proposed an initial framework aimed at providing a comprehensive identification of the major factors impacting team programming situations including pair programming. However, this study demonstrates that the framework should be extended to include an additional category of factors that relate to organizational matters. These factors will be further refined, and used to develop and empirically evaluate a conceptual model of pair programming (success).[1]

1 Introduction

Pair programming is a core (some would say mandatory) practice of eXtreme Programming (XP) [2], and commonly applied and or recommended for use in conjunction with many other Agile software development methods including Feature Driven Development, Scrum, Lean Software Development, Crystal, and Dynamic Systems Development Method.

Various definitions of pair programming have been proposed [12, 20, 23]. Jensen [12] describes it as 'two programmers working together, side by side, at one computer collaborating on the same analysis, design, implementation, and test'. Compared to traditional programming where typically one programmer is responsible for developing and testing their own code, in pair programming every code fragment is developed by a team of two programmers working at the same workstation. There are two roles, viz, a driver controlling the mouse, keyboard or other input device to write the code and unit tests, and a navigator, observing and quality assuring the code, asking questions, considering alternative approaches, identifying defects, and thinking strategically. The partners are considered equals and will regularly swap roles and partners [22].

[1] Accepted for XP 2005, http://www.xp2005.org, Sheffield, UK, 18–23 June

The need for this study arises from the lack of an all-encompassing theory about the factors that influence pair programming success and the extent of this influence. Pair programming may have a basis in theories of group problem solving and decision making [8, 14], but any explicit reference to such theories for its use seems difficult to locate. Most research has tended to be fragmented and restricted to specific issues. A useful framework for research on team programming situations is found in [8]. This paper builds on and extends their preliminary work by adopting a holistic approach and analyzing and synthesizing the literature findings and empirical data collected for this study, with the future aim of developing a conceptual model of pair programming success.

This paper reports an analysis of the literature involving empirical research on pair programming, to discover theoretical concepts and factors relevant to the practice. Since pair programming is one of the more contentious aspects of Agile system development (particularly eXtreme Programming), it has attracted much attention and consequently become the focus of numerous studies, both in the field with real-life examples and professionals, and in an educational context with students as subjects. These studies have employed a range of research methods to investigate this phenomenon including: case studies, experience reports, surveys and experiments. Space precludes all the literature being presented and represented here but that mentioned typifies the studies and views taken.

The paper also reports a comparison of the factors arising from the literature analysis and our recent Delphi study of pair programming [19]. Briefly, the Delphi technique emerged from work at the US Department of Defence and the RAND Corporation in the 1950s. It is a qualitative, structured group interaction technique and its use is well documented [5, 17]. The objective of the technique is to allow the researcher to obtain a reliable consensus from a panel of experts where the phenomenon or situation under study is political or emotional and where the decisions affect strong factional interests. The technique then collates, synthesizes and categorizes those opinions until general consensus is reached. In the Delphi study, 20 participants engaged in three (3) rounds to reach consensus on issues about pair programming from both an organizational and an individual's perspective. The Delphi participants, comprising academics, software developers and managers, were selected on the basis of their pair programming experience, software development expertise, and industry reputation. Participants were drawn from a range of different types and sizes of organizations to give a broad perspective to the study. In the text that follows, selected representative quotations from Delphi participants are shown in 'quotes'. This study reports partial findings from the Delphi study. Further in-depth analysis of the Delphi study data and the development of models to relate the factors identified with measures of pair programming success are yet to occur.

This paper is structured as follows. The next section identifies concepts found in both the literature and from the Delphi study and where the views relevant to pair programming coincide. In the third section concepts related to pair programming, where the views are opposing, are reviewed. The fourth section reviews concepts arising in only one source: either the literature or the Delphi study but

not both. The fifth section aligns the factors identified in this study to the initial framework proposed by [8]. The final section offers conclusions and expectations for future work.

2 Concepts in Common: Literature & Delphi

This section identifies concepts found in both the literature and in the Delphi study, where the views relevant to pair programming coincide. Literature relevant to the concept is reported, as is representative mention of the concept in the Delphi study providing further validation of the practical implications of the concept. Fifteen concepts are identified.

Quality. There are many references in the literature that support the concept of improved quality arising from pair programming situations. Poole and Huisman [15] suggest that pair programming results in improved engineering practices and quality, as evidenced by low error rates. Improved quality was manifested in earlier bug detection/prevention [4]. Experiments with 'industrial' programmers showed error rates were reduced by two-thirds and needed less iterations to fix them when pairing [12, 13]. Equally, in the Delphi study the issue of improved quality was raised many times, both from organizational and individual perspectives. The general consensus was that code quality improved through the pairing process resulting in fewer bugs and better designs. Also on the issue of code maintenance 'usually if someone is programming with you, better choices are made for variable names, better structure, (and) programmers aren't as lazy keeping coding standards'.

Team building and pair management. Two aspects of team management raised in the literature were that pair programming engenders a team spirit; and that there is a need for training in team building [12, 18]. The Delphi study affirmed the import of these team management concepts, emphasizing that pairing is a social activity where 'one has to learn how to work closely with others, (to) work effectively as a member of a team'. The need to develop these skills was highlighted in the Delphi study where 'traditionally, IT study/training alone does not equip individuals with the interpersonal skills required for effective pairing'. The Delphi also raised issues related to managing the paired programming process, including pragmatic issues such as pair rotation and 'what do you do when there are odd numbers of people on a team?'; resolving personality conflicts, for example where 'an obsessively neat person (is required to) work with a messy person' and 'people that no-one wants to work with'; and logistical issues such as when 'pairs need to start/end work at the same time'.

Pair personality. Dick and Zarnett [6] identified personality traits as playing a vital role in the success, or otherwise, of pair programming. The Delphi study identified two specific issues that need to be addressed: personality conflicts 'when two people have different ideas, or different styles of dealing (with) problems' and 'some people just don't like accepting other people's suggestions/ideas'; and divergent personal styles where 'some programmers like to

discuss with others, while some do like to work on issues by themselves' and '(pairing) strong personalities together with weak personalities'.

Threatening Environment. Both [6, 18] highlight the problem for individuals in the pair of the fear of feeling and/or appearing ignorant on some programming or system development aspect to one's partner. The Delphi Study supported this indicating that working in pairs may expose an individual's weaknesses and competencies. Comments included: 'Most people new to pairing find the prospect frightening/threatening – will I appear stupid/ignorant?' and 'Sharing knowledge (and ignorance) on a daily basis can be threatening'.

Project management. Pair programming raises issues for project management. On the positive side, it may act as a backup for absent or departing developers [7, 23]. This was also reflected in the Delphi study with 'no one person has a monopoly on any one section of the code, which should remove organizational dependencies on particular resources and mitigate risk to the business'. On a less positive note, there are challenges for project management in terms of planning and estimation. This was highlighted in the Delphi study by 'Methods of planning/estimating need to change when (a) team is pair-programming rather than tackling tasks as individuals'.

Design and problem solving. There is ample evidence that pair programming results in improved design and problem solving through the removal of 'tunnel vision' and the exchange of ideas [12, 16, 23]. It is especially suited to dealing with very complex problems that are too difficult for one person to solve [4, 21, 23]. Experiments have provided supporting empirical evidence about improved design [13]. This sentiment was affirmed in the Delphi study where it was felt that design decisions and difficult problem resolutions would be superior, and that 'it very often solves complex problems much more effectively than a single person would', as well as the potential to 'find problems in advance'.

Programmer resistance. An important issue for management consideration is that many programmers resist (at least initially) pair programming. There are many facets to this issue including: a reluctance to share ideas; ego problems where some people think they are always right; and lack of trust where comments may be taken as personal criticism [18]. Therefore, there is a need for strategies to introduce pair programming 'softly', recognizing that even after coaching some programmers resist working in pairs [18]. In the Delphi study the resistance to pairing was also raised, especially among the 'old-school programmers who find it difficult to change habits'.

Communication. The literature cites communication as integral to pair programming [7], and that it helps to get people to work better [4]. The Delphi study also supported that pair programming requires and 'fosters communication skills in the team', and 'improves interactions between team members'.

Knowledge sharing. The literature proposes that pair programming presents opportunities for improved knowledge transfer [7]. Pairs learn a great deal from one another [4] including changed behaviour, habits, ideas and attitudes [18]. The Delphi study also cited improved knowledge transfer in a variety of contexts including: that the programmers' knowledge became more broad-based;

that it enabled a concurrent understanding rather than a post-explanation; that it resulted in more thorough domain knowledge; and that it could act as a backup/contingency plan in cases of illness or resignation. In contrast, the Delphi study also raised some negative aspects of knowledge sharing viz. that pairing may result in programmers having a broader, but shallower understanding of the system; and that some may find knowledge sharing to be threatening.

Mentoring. The literature found that pair programming provides an ideal environment that greatly facilitates mentoring [6, 15]. Positive outcomes were enjoyed by senior staff [15], as well as less experienced programmers, who learned from their more experienced partners [12, 13]. Several responses in the Delphi study attested to the mentoring benefits arising out of their pair programming experiences: 'There is a fast tracking of skills development with careful choice of pairs (for example) mentor/ junior role' and 'it can really help with the development of new programmers'. However, 'it helps if you have a good programmer who is able to explain, or teach, an inexperienced programmer'. The point where mentoring becomes training was raised in the Delphi study as evidenced by 'that each (developer) has a say in how the task is to go ahead (that is) it is a team not a mentor/junior process. In this case I don't think it is pair programming but more like training'.

Environment requirements. An unsuitable physical environment may act as a barrier to pair programming success [12]. This was reflected in the Delphi study. The physical environment should facilitate two programmers working at a single workstation because 'most single person desks are not comfortable for two people to sit at' and 'our desks are L-shaped, and as such do not allow two developers to sit side-by-side comfortably'. This work environment may be more disruptive as 'good pairs interact constantly'. Of course individual environmental preferences may vary, for example a liking for differing styles of keyboards.

Effective pairs. While there is agreement that the dynamics of the pairs needs to be carefully considered, there is no agreement as to what constitutes the most effective pair combinations. For instance, [12] suggests that it is counterproductive to pair two programmers of equal skill. This sentiment was also reflected in the Delphi study: 'for two equally competent programmers I see this as a waste of resource'. The contrary view was also expressed that 'pair programming between experienced programmers is often more useful when it comes to making design decisions'. Two instances where effective pairing may produce beneficial results are (1) where a new developer is placed in a pairing situation and can start being productive immediately, and (2) where a junior programmer might need mentoring. However, the Delphi study revealed that the dynamics of the pairs needs to be considered carefully. Many combinations would not work well: a novice programmer could slow down (and potentially annoy) a skilled programmer, while lowering the self-esteem of the former; two skilled programmers working together could have the effect of negating productivity benefits, for instance when the navigator becomes increasingly frustrated at the lack of involvement, or when there is contant 'clashing of the minds'; two novice pro-

grammers could benefit from pair programming, but they run the risk of 'the blind leading the blind'.

Shared responsibility. Both [1, 18] argued that by spreading responsibility and decision-making load, pairs effectively 'halve' the problem solving. Individuals feel more confident about the decisions made, and less overwhelmed by decision-making responsibilities. The Delphi study respondents agreed, noting that 'new developers can feel more confident about attacking complex code because there is someone else there with them', and further that they are 'helping someone else with their assigned duties'.

Human resource management. Pair programming has implications for the recruitment process of hiring programmers [6]. It also challenges traditional human resource ideas of individual-based performance evaluation and remuneration. These team-based approaches need new management strategies to be considered that are significantly different from those traditionally used for software developers [18]. Many of these human resource issues were raised in the Delphi study generally: 'traditional performance measures focus on the individual – how do you map entrenched HR practices/requirements of a large organization to a collaborative team structure'; from an organizational perspective: 'emphasis moves to team success rather than individual success'; and from an individual perspective: 'a programmer cannot look at a subsystem and say "I did that"; success is now team-based, not individual-based'.

Attitude. A stereotypes of programmers is the 'lone-hacker'. Pair programming has been shown to change programmers' attitude from withdrawn, introverted and worried, to outgoing, gregarious and confident [1]. Delphi study participants agreed, noting that 'some people have entered the industry because it is one where they can be alone for long periods' but that 'one has to learn how to work closely with others' and 'work effectively as a member of a team'.

3 Opposing Perspectives: Literature & Delphi

A significant finding of this study is that some of the issues raised in the literature were also raised in the Delphi study, but from opposing viewpoints. It is notable that for these factors the literature is consistently positive about the concept in contrast to the Delphi study in which the same issues appear as barriers or hindrances. Thus the Delphi study effectively contradicts the practical implications of the concept as it is presented in the literature.

Morale. The literature suggests that morale can be improved by using pair programming, especially when working on a difficult or complex system. This morale 'boost' may be in the form of positive reinforcement by peers [15] but also because 'there's someone there to celebrate with when things go right' [1]. In the Delphi study, the impact of pair programming on morale was raised in a negative light for example when it came to a mismatch of skills: 'there's a high probability one member of the pair will resent the other one and lower their morale whilst working with this person'.

Productivity. The literature cites many examples of improved productivity arising from pair programming [12, 15, 18]. This includes experiments with 'industrial' programmers [12, 13]. This was in part attributed to a shared conscience where pairs are less likely to indulge in time-wasting activities [23]. Pairs wasted less time trying to solve problems compared to working alone [12]. However in the main, the Delphi study suggested lower productivity or at least perceptions of lower productivity. In particular, management is yet to be convinced of the productivity benefits of pair programming: 'corporate viewed pairing as being ... twice as slow as traditional development'. In certain pairing scenarios pairing was seen to be less productive: 'two top programmers would (have) lower productivity'. It was even suggested that the quantity of code 'usually goes down per hour when taking into account the number of people working on it'. In contrast, it was suggested by one participant that the definition of code generation needed to be considered in developing any measure of productivity: 'if design reviews are accepted as being part ... then productivity gains are higher'.

Development costs. Clearly, development costs are an important issue for software construction. The literature suggests code costs are slightly higher ([21] suggests 15%) with pair programming, but that it is offset by improved code quality, and minimization of and the earlier detection of bugs [4, 13]. Delphi study participants were far from convinced. They noted the problem for management of an apparent doubling of cost for development of the same feature: 'development throughput is reduced, not only by halving the number of people actively coding simultaneously, but also because there is additional collaboration on the design of the code'. An interesting take on the situation was that 'a pair wasting time costs twice as much as a single developer wasting time'.

Enjoyment of work. Students and professional programmers report finding their work to be more enjoyable when pairing [23]. However, this is an opposing viewpoint to the Delphi study where it was described as an unpopular activity that resulted in lowered personal satisfaction.

4 One Source Concepts: Literature & Delphi

Some factors were raised in either the literature or the Delphi, but not both, which may suggest that saturation of all the issues involved has not yet occured.

Factors that appeared only in the literature included: **project schedule** potential where project timelines can be realistically shortened through a change in workflow to a more speedy iteration of plan, code, test and release [1, 3, 9, 24]; **fit of pair programming to project type** where experiments have shown that pair programming is especially suited to situations characterized by changing requirements, and unfamiliar, challenging or time-consuming problems [13]; **code readability** where source code readability is greatly enhanced by using pair programming [10]; and **distributed pair programming** where appropriate tools can assist distributed pair programming where co-location is not possible [11].

Factors that arose only in the Delphi study included: **collective code ownership** through pair programming minimizes the introduction of coding flaws

and enhances concurrent understanding of the code base; **accountability** concerns the shift of responsibility from the individual to the pair through collective code ownership; **customer resistance** to pair programming through the perception of increased costs; **organizational culture** and its influence on the acceptance of pair programming, and the influence of pair programming on the organization; and **solitude and privacy opportunities** are reduced when pair programming, with the increased potential for stress and 'programmer burnout'.

These factors (and possibly others) will be more fully analyzed prior to incorporation into a conceptual model.

5 Extension of Gallis et al. (2003) Framework of Factors

An initial framework for research on pair programming has been proposed ([8] summarized in their Fig. 1). While their study was based on four different configurations of team programming, this study focuses specifically on pair programming. In addition, [8] considered a specific set of literature including their own pair programming studies in developing their research framework. This study extends their framework by considering additional literature, and individual and organizational issues identified in our Delphi study. They identified dependent variables (time, cost, quality, productivity, information and knowledge transfer, trust and morale, and risk) which were affirmed and context variables, which were affirmed but further elaborated in this study (see Table 1).

Also, our findings reveal an additional category of context variables, namely organizational factors, which were repeatedly raised by the Delphi study participants. The impact of the adoption of pair programming on organizations and the impact of organizational culture on the practice of pair programming are clearly important issues for further consideration. This extended framework will form the basis for the development of a model of pair programming success.

6 Conclusions

Pair programming is controversial: the diverse literature on the practice and discussion with practitioners confirms the variety of factors affecting its success as a practice, how it is viewed by practitioners, and its impact on software development success. While there is a great deal of evidence in the literature about pair programming success, there is much work to be done by an organization to properly prepare for its implementation (especially overcoming resistance), and it is clear that many of the 'people' issues require in-depth consideration.

This study suggests that further research is required particularly to examine the breakdowns, that is, where the literature and practitioners hold opposing views. In addition a more complete analysis is required of those factors that appear in only one of either the literature or the practitioner experience.

Dependent and independent variables have been identified in the framework, but further refinement is necessary. The next step is to formulate a conceptual model of pair programming (success), which can be quantitatively tested. This

Table 1. Extension and elaboration of the Gallis et al. (2003) framework context variables (sections where the variable appears in this paper are shown in parenthesis)

Gallis et al. (2003)	This study
Subject variables	
Education & experience	Mentoring (2)
Personality	Pair personality (2) Programmer resistance (2)
Roles	Shared responsibility (2)
Communications	Communication (2)
Switching partners	Project management (2) Effective pairs (2) Attitude (2) Enjoyment of work (3) Knowledge sharing (2) Threatening environment (2)
Task variables	
Type of development activity	Design & problem solving (2) Code readability (4)
Type of task	Fit of pair programming to project type (4)
Environment variables	
Software development process	Project schedule (4)
Software development tools	Distributed pair programming (4)
Workspace facilities	Environment requirements (2) Solitude & privacy (4)
Organizational variables	
	Team building & pair management (2) Human resource management (2) Accountability (4) Customer resistance (4) Organizational culture (4) Collective code ownership (4)

work is in progress. There is a need for multi-disciplinary and mixed-method research that will uncover behavioural strategies for a more complete understanding of the complexities of the human aspects of pair programming. Other research includes pair programming experiments with students and practitioners.

References

1. Baer, M.: The New X-Men, viewed 1/12/04, http://www.wired.com/wired/archive/11.09/xmen.html?pg=1&topic=&topic_s (2003).
2. Beck, K.: *Extreme Programming Explained: Embrace Change*, Addison Wesley, Boston (1999).
3. Brooks, F.P.: *The Mythical Man-Month Essays on Software Engineering*, Anniversary Edition, Addison-Wesley, Boston (1995).

4. Cockburn, A. and Williams, L.: The Costs and Benefits of Pair Programming, *Proceedings of XP2000*, Sardinia, Italy, June 21–23 (2000).
5. Day, L.: Delphi Research in the Corporate Environment, in Linstone and Turoff (Eds) *The Delphi Method: Techniques and Applications*, Addison-Wesley, London (1975).
6. Dick, A.J. and Zarnett, B.: Paired Programming and Personality Traits, *Proceedings of XP2002*, Sardinia, Italy, May 26–29 (2002) 82–85.
7. Flies, D.D.: Is Pair Programming a Valuable Practice?, viewed 6/12/04, `http://csci.mrs.umn.edu/UMMCSciWiki/pub/CSci3903s03/StudentPaperMaterials/flies-pairprogramming03.pdf` (2003).
8. Gallis, H., Arisholm, E. and Dybå, T.: An Initial Framework for Research on Pair Programming, *Proceedings of the 2003 International Symposium on Empirical Software Engineering-ISESE 2003* (2003) 132–142.
9. Glass, R.L.: *Software Runaways*, Prentice Hall, New Jersey (1998).
10. Grenning, J.: Launching Extreme Programming at a Process-Intensive Company, *IEEE Software*, **18**(6) (2001) 27–33.
11. Hanks, B.: Empirical Studies of Pair Programming, *2nd International Workshop on Empirical Evaluation of Agile Processes*, New Orleans, Louisiana (2003).
12. Jensen, R.W.: A Pair Programming Experience, *Journal of Defense Software Engineering*, March, viewed 1/12/04, `http://www.stsc.hill.af.mil/crosstalk/2003/03/jensen.html` (2003).
13. Lui, K.M. and Chan, K.C.C.: When Does a Pair Outperform Two Individuals? in M. Marchesi and G. Succi (Eds) *Extreme Programming and Aile Processes in Software Engineering—4th International Conference, XP 2003* Lecture Notes in Computer Science **2675** (2003) 225–233.
14. Napier, R.W. and Gershenfeld, M.: *Groups: Theory and experience*, Sixth Edition. Boston, Houghton Mifflin Company (1999).
15. Poole, C. and Huisman, J.W.: Using Extreme Programming in a Maintenance Environment, *IEEE Software*, **18**(6) (2001) 42–50.
16. Pulugurtha, S., Neveu, J. and Lynch, F.: Extreme Programming in a Customer Services Organization, *Proceedings of XP2002*, Sardinia, Italy, May 26–29 (2002) 193–194.
17. Rowe, G., Wright, G. and Bolger, F.: Delphi: A Reevaluation of Research and Theory, *Technological Forecasting and Social Change*, **39**(3) (1991) 235–251.
18. Sharifabdi, K. and Grot, C.: Team Development and Pair Programming—Tasks and Challenges of the XP Coach, *Proceedings of XP2002*, Sardinia, Italy, May 26–29 (2002) 166-169.
19. Toleman, M., Ally, M. and Darroch, F.: A Delphi Study of Pair Programming, Working paper, Department of Information Systems, University of Southern Queensland (2005).
20. Wiki: viewed 1/12/04, `http://c2.com/cgi/wiki?PairProgramming` (2004).
21. Williams, L.: The Collaborative Software Process, unpublished PhD dissertation, Department of Computer Science, University of Utah (2000).
22. Williams, L. and Kessler, R.: *Pair Programming Illuminated*, Addison-Wesley, Boston (2002).
23. Williams, L., Kessler, R.R., Cunningham, W. and Jeffries, R.: Strengthening the Case for Pair Programming, *IEEE Software*, **17**(4) (2000) 19–25.
24. Yourdon, E.: *Death March: The Complete Developer's Guide to Surviving 'Mission Impossible' Projects*, Prentice Hall, New Jersey (1999).