Pair Programming: Effects of Trust on Software Quality

George Mangalaraj  
Western Illinois University

Vikram Bhadauria  
The University of Texas Arlington

Follow this and additional works at: http://aisel.aisnet.org/amcis2007

Recommended Citation  
http://aisel.aisnet.org/amcis2007/117

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 2007 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
PAIR PROGRAMMING: EFFECTS OF TRUST ON SOFTWARE QUALITY

George A Mangalaraj
Western Illinois University
G-Mangalaraj@wiu.edu

Vikram S Bhadauria
University of Texas Arlington
vikram@uta.edu

ABSTRACT
Trust is increasingly becoming an important area of research in software development with the proliferation of agile methodologies. In this study, we propose to investigate the effect of trust on the performance of a pair programming team. High levels of trust in the competence of the fellow member are hypothesized to increase the perceived effort applied in software development. This relationship is moderated by the characteristics of the software development task. Perceived effort applied at the task is hypothesized to mediate the relationship between trust and the team performance which is measured in terms of the solution quality. We propose to use a laboratory experiment to test our model. Limitations are discussed and future research directions are suggested.

Keywords: Pair Programming, Trust, Software Development, Agile Methodologies, Software Quality

INTRODUCTION

Often highlighted high failure rates in software development have prompted companies to adopt radically different software development methodologies. These development methodologies are collectively called as Agile Software Development Methodologies (ASDM). These methodologies are increasingly becoming popular (Ambler 2006). Extreme programming is one of the most popular ASDM and it uses pairs in programming. Proponents of pair-programming highly acclaim the quality of software applications developed using pair programming. Traditionally, programming has been considered to be an individual task in which there was hardly any interaction with other team members while actually writing the code. However, in pair programming, two programmers work together while writing code for a particular application. The code thus developed is collectively owned by the members of the project team. Much of the earlier studies in pair-programming have tried to ascertain the benefits of such cooperative work. There exists some empirical (Williams et al. 2000a) as well as anecdotal evidence to support the claimed benefits of pair-programming.

The efficacy of a technique such as pair programming will depend upon mutual sharing of ideas and the effectiveness in building a symbiotic relationship with the peer programmer. Trust will be a key antecedent in the evolution of such a relationship. An atmosphere with high trust and mutual respect is often stressed upon as an important factor in the pair programming environment (Nerur et al. 2005). Though the role of trust had been studied with regard to different aspects in the Information Systems field, there is no systematic study in the literature to consider this aspect of trust in the area of extreme programming. This study strives to fill a crucial void by systematically considering the role of trust in pair-programming and its impact on software quality.
Higher failure rates among the software development projects have been often highlighted (The Standish Group 1995). The late 1990s saw the introduction of many innovative software development processes that were purported to address various issues that were plaguing software development projects. These processes are collectively called ‘Agile Software Development Methodologies’ (ASDM). They question the fundamental assumptions/processes adopted in the Traditional Software Development Methodologies (TSDM). ASDM promises to increase the prospect of successful completion of software development projects (Highsmith et al. 2001). Practitioners have shown considerable interest in the agile approach to software development, as evidenced by the findings of a recent survey (Ambler 2006; Charette 2003).

Extreme Programming (XP) is one of the ASDM approaches that is well documented, and found to be effective, but still controversial among some practitioners. XP stresses on small teams of 3 to 10 programmers, requires presence of one or more customers with the development team for providing ongoing expertise.

Elements of Extreme Programming

Adoption of XP revolves around the adherence to the twelve core practices. These practices are not rules and the adherence to a particular practice is contingent on the situation. Many of the practices outlined in XP are not new and they have existed well before the advent of XP (Beck 2001). The practices adopted by XP, emphasize on communication and collaboration between the various participants that includes the users. Extensive documentation is not done in XP and this remains to be one of the main differences between XP and traditional software development methodologies. XP short release cycles combined with other practices such as refactoring helps in responding to change in the user requirements. There are 12 core practices followed in using XP and pair-programming is one of the important practices.

Pair-Programming

XP methodology has pair programming as one of its core requirements. Pair programming involves two programmers working side by side at one computer, collaborating on the design, code or test on a continuous basis. The programmer sitting in front of the computer would be the driver and will be actively coding while the second programmer would oversee the coding, looking for errors in syntax, logic and deviations from programming practices and norms. The second programmer is expected to also think strategically the overall scenario looking at how the work fits with the rest and the further directions they should be taking. The partners are encouraged to switch roles of coding and code inspection frequently.

The code generated by pair programming is expected to be of much higher quality, as it has to have the approval of both the programmers. The programmers are likely to be more confident of their solution when coded jointly as against working alone. XP also promotes the concepts of collective code ownership. This means that any part of the code can be changed by anybody in the project team at any time. Pair-programming also relies on this practice so that ownership of the code produced does not lie on an individual developer. Some common concerns regarding collective ownership are that some programmers are not happy with another programmer changing their code while managers could be apprehensive about accountability in case of problems in the code. However collective ownership encourages the entire team to work together more cohesively with each pair striving a little harder to produce high quality designs and code (Highsmith 2000).

The software community has known the superiority of inspections by two programmers for a long time. However, what is little realized earlier is programming in pairs could be cost effective in not only uncovering defects, but also in preventing defects in the first place through learning and incorporation of better programming practices. According to one estimate cited by Highsmith (2000), the time required to isolate per defect was 15 hours with testing, 2-3 hours with inspections and only 15 minutes by finding the defect before it got to the inspection. The ongoing team learning engendered in inspections helps bring about this drastic reduction in the time required to isolate defects.

There is also some anecdotal evidence to indicate collaboration improved both the performance and enjoyment of the whole problem solving process for the programmers (Nosek 1998). It has been shown that when two programmers work together, work more than twice as fast and think of more than twice as many solutions to a problem as two working alone, while achieving higher defect prevention and removal and thus leading to a higher quality product (Williams et al. 2000b). Pair programming encourages each programmer to drive the other partner a little harder to excel. Pair-programming is also claimed to affect duration, effort, and correctness of solution (Arisholm et al. 2007).

Role of Trust in Pair-programming

The issue of trust is gaining increased interest in organization studies and Information Systems field is no exception to it. Trust has been defined in many different ways and Mayer, Davis, & Schoorman (1995) define trust as:
Willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to trustor, irrespective of the ability to monitor or control that other party.

Trust is a multidimensional construct and these dimensions include (a) Integrity, (b) competence, and (c) consistency, (d) benevolent motives, and (e) openness (Butler et al. 1984). Based on a review of literature, Mayer, Davis, & Schoorman (1995) proposed an integrated model of organizational trust encompassing (a) ability, (b) benevolence, and (c) Integrity. They highlighted that these factors work in combination to create the perceived trustworthiness.

Though there have been many studies that highlighted the role of trust in teams/work groups, but there is no systematic study to analyze this aspect in software development teams. Advent of XP with radical practices such as pair programming and collective code ownership has accentuated the need to study the role of trust in teams. Importance of trust in pair-programming have been highlighted in the literature (Nerur et al. 2005; Williams et al. 2003).

Practices such as pair-programming and collective code ownership require a trusting environment in an XP project. Williams et al.(2000b) highlight how ego among the pairs can cause mistrust in a pair-programming environment and the negative effects of it. Pairs that do not trust each other can become dysfunctional and this may lead to potential problems. For instance, pairs in which members do not trust the competence of the partner may not be willing to accept partner’s views on solving a programming problem. Though all the dimensions of trust is important in task performing group, the current study focuses on competence-based trust alone. Limiting the focus on a particular dimension of trust will help in gaining in depth of understanding of the phenomenon.

Competence/ability based trust is domain specific and it enables a party to have influence within specific domain (Mayer et al. 1995). Competence-based trust allows one to feel confident in listening and learning from another person (Abrams et al. 2003). Competence-based trust was defined by Olson et al. (2007) as the trust based on the cognitive process of choosing whom we would trust based on competence, responsibility, reliability, and dependability.

Kim, Ferrin, Cooper, & Dirks (2004), in their experiment, used competence/integrity based trust and the consequences of violation of them. Olson et al. (2007) found competence-based trust to affect task conflict in top management teams.

Influence of trust and the resultant impact on the software quality is presented through the proposed model for software development pair effectiveness. The proposed model is based on Yeatts et al.(1998)’s integrated model for team effectiveness.

**Model for Software Development Pair Effectiveness**

Yeatts et al.(1998)’s model for team effectiveness considered interpersonal process and its impact on work process and the resultant team performance. Our proposed model utilizes part of the original model but considers explicitly few factors in each of the processes. A moderating variable namely task complexity is introduced in the adapted model as shown in Figure 1.

![Figure 1 Proposed Research Model](image)

**Perceived Effort Applied**

Trust increases the perceived effort applied by the subjects to the task. When trust exists, the team members of pair programming team will be willing to accept the ideas and opinions of the other person in the team. High trust creates an atmosphere where team members will be able to obtain information they needed quickly and also reveal errors that may have occurred (Yeatts et al. 1998).

Dirks (1999) used expectancy theory and explained for the variation in the effort applied with different levels of trust. Trust influences the member of pair programming team’s expectations about the extent to which the effort can be
converted to team performance. In teams with high level of trust, individuals could feel comfortable directing their effort toward the group task (Dirks 1999). Klimoski et al. (1976) considered trust in creative problem solving groups and its impact on the perceived effort applied. They found high trust groups outperformed the low trust groups while solving problems.

McEvily et al. (2003) in their work on trust as an organizing principle, explain the role of trust in mobilizing resources. They highlight how trust helps in motivating actors to contribute resources such as time, effort, attention and knowledge toward the organizational goal. Hoegl et al. (2001) in the work on software development team performance studied the effects of effort applied on the task on the quality of the work produced by the team. Trusting relationships would help members of a team to openly challenge others’ perspectives without the fear of ridicule or retribution (Olson et al. 2007). In the absence of trust, team members reduce their efforts and withhold co-operation with others (Furst et al. 1999).

This study hypothesizes that higher level of trust among the members of a pair programming team will lead to higher levels of perceived efforts being applied to task and this argument is consistent with past studies (Jones et al. 1998; Yeatts et al. 1998).

**Hypothesis 1:** A higher level of trust is positively associated with increased level of perceived effort applied on the task.

**Task Characteristics as a Moderator of Perceived Effort Applied**

The characteristics of tasks are often considered as an influential factor in studying the performance of software development project teams’ (Aladwani 2002; Andres et al. 2001; Lakhanpal 1992). Task characteristics have been studied in various ways and one of the frequently used characteristics is the task interdependence. Task interdependence refers to the level of independence between various participants in completing the tasks and the most commonly used classification of task interdependence is of Thompson (1967).

Task difficulty/complexity is also used as task characteristics in prior studies. Aladwani (2002) in their model of IS project performance model used problem goal clarity as one of the factors in influencing the performance of the projects. In the context of pair-programming, Arisholm et al. (2007) utilized complexity of the task as a moderating variable and found pair-programming to produce better results in tasks with high complexity. Al-Kilidar et al. (2005) considered complexity of software design tasks in their study and found pairs to perform better in simple design tasks when compared to complex tasks.

In this study task characteristic in terms of complexity is used as a moderating variable. It is argued programming problems with high/low complexity will have varying levels of effect on the perceived effort applied by the programmers in differing trusting environments. Kidwell Jr. et al. (1993) in their work on employee’s propensity to withhold effort considered task characteristics as one of the factors.

Based on this argument, it is hypothesized that the higher level of trust will increase the perceived effort applied on the task at varying levels depending on the complexity of the task performed.

**Hypothesis 2:** Task complexity will moderate the relationship between competence-based trust and perceived effort applied.

**Software Quality**

Software development team performance can be evaluated using various dimensions. In the context of XP, the claimed benefits are increased productivity, quality, developer satisfaction, and adhering to project deadlines (Beck 2001). In the past there have been studies that tried to empirically validate these claimed benefits of XP (George et al. 2004; Nosek 1998).

In this study, quality of the resultant program is considered to be the dependent variable. Quality of the programs thus created can be evaluated using varying methods. Some of the common ways to evaluate the developed programs is to use, (a) subjective evaluations of experts on the quality of the solution (Prechelt et al. 2002), (b) objective measure of software quality such as the number of bugs present and other measures of problems in the software (George et al. 2004).

Extreme programming XP adopts a strategy called *Test Driven Development.* Here the test cases are written first before actual implementation code is written (Beck 2001). This gives an objective way to determine the quality of the programs developed. Test cases are run using automated framework called JUnit and can be carried out with limited human intervention (Williams 2003). There is already a precedence in utilizing this testing methodology in evaluating the quality of the developed programs in the literature (George et al. 2004). In this experiment these automated test procedures are planned to be used to arrive at the values for software quality.

Quality of the software produced is related to the level of perceived effort applied. Rasch et al. (1992) in their model on software developers’ performance considered developer efforts to be positively related to the performance. In the same way, it is hypothesized that the perceived effort applied on the task to be positively related to the quality of the produced software.
**Hypothesis 3:** A higher level of perceived effort applied on the task is positively associated with increased quality of the produced software.

**Perceived Effort Applied: Mediator**

Recent literature suggests Trust as an important success factor in performance of teams working on IT projects (Peslak et al. 2007). As discussed earlier, higher effort applied leads to better performance. Teams with higher trust allow for open discussions (Olson et al, 2007), which we believe is critical for better project outcomes. Based on the arguments put forward for hypothesis 1 and 3, we argue here that the perceived effort applied will mediate the relationship between trust and the quality of the software produced. Using Baron and Kenny (1986) approach, we posit that the relationship between Trust and Software Quality will weaken when perceived effort applied is considered in model.

**Hypothesis 4:** Perceived effort applied will mediate the effects of competence-based trust on software quality.

**RESEARCH METHODOLOGY**

**Overview of Procedures**

In order to validate the various hypotheses mentioned in the earlier section, an experiment was proposed to be conducted. This section outlines the experimental setting, procedures in manipulating the independent variables and the measurement of various variables used in the study.

**Design**

Completely randomized experiment is planned to be used in this study. There are two treatments: trust and task complexity with two levels in each. We plan to utilize students as subjects for this study and this is consistent with the past research on pair-programming (Hanks 2005; Williams et al. 2000a). The potential subject pool for this study comes from students in various programming languages courses. Subjects will be randomly assigned to the various treatment conditions. The planned sample size for this study is eighty. Resultant forty pair programming teams will be assigned to the various treatment conditions as shown in the following table.

<table>
<thead>
<tr>
<th>Experimental Condition</th>
<th>Task Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>High complexity</td>
<td>Low complexity</td>
</tr>
<tr>
<td>High Trust</td>
<td>10 pairs</td>
</tr>
<tr>
<td>Low Trust</td>
<td>10 pairs</td>
</tr>
</tbody>
</table>

**Threats to Internal / External Validity**

This study proposes to utilize posttest-only experimental group design. Both the groups were experimental groups with appropriate treatments for low and high trust environments. Nature of this study, which is oriented towards answering the theoretical question, the effect of trust in pair programming teams gives emphasis to internal validity. The design adopted in this study overcomes the various threats to internal validity. External validity that is generalizability of results to larger real world programming teams should be done with caution.

**Manipulation of the Independent Variables**

Levels of trust will be manipulated and there will be two levels of it. Competence/ability based trust will be manipulated. In the past there have been instance in manipulating and studying individual dimensions of trust (Kim et al. 2004; Levin et al. 2002).

The critical issue in manipulating competence based trust of one member of the team on another one is creation of a suitable scenario. Subjects will be randomly assigned to various teams and this eliminates the prior contact with the other team members. This experiment manipulates the competence based trust and it is done in the following fashion:

**High trust environment:** In this environment, subjects in a pair programming team will be informed individually that the other member of the team is highly competent in the assigned task. This creates a positive perception about each other members of the team. This creates an atmosphere in which mutual respect for the views of one another is high. It is argued that this manipulation will result in an environment of high trust and this is consistent with prior studies in creating competence based trust (Kim et al. 2004).

**Low trust environment:** Here the team members will be informed individually that other member of the team is not good at the assigned task. This creates an atmosphere were there will distrust between the members while performing the experimental task.
Appropriate manipulation checks will be carried out to make sure the treatment really produces the intended effects on the subjects. This study proposes to use the procedures adopted by Kim et al. (2004) for checking the manipulation for trust. Participants will be asked about the task complexity and this will serve as an adequate manipulation check.

Moderator

Task Complexity:

Two types of tasks will be assigned to the subjects. These tasks were classified as low complexity or high complexity based on the programming requirements. Based on the opinion of subject matter experts, these tasks will be formulated. Complexity of the tasks also relies on the type of subjects utilized in this study.

Dependent Measures

Perceived Effort Applied:

This will be perceptual measure to evaluate the perceived effort applied by the subjects on the task. In the past researches have used single item measure for this construct (Hoegl et al. 2001). There have been instances of using instruments based on expectancy theory to measure perceived effort applied in tasks (Rasch et al. 1992). In this study a four item scale was developed and some of the questions were based on the prior studies mentioned earlier.

Software Quality:

The dependent measure namely software quality will be measured using the test cases developed by the investigator. These test cases would be developed to meet the requirements of the assigned programming problems. This study proposes to utilize objective measures for the software quality and this would eliminate any bias due to subjective evaluation.

Consideration of Common Method Bias

Podsakoff et al.(2003) highlight the problem of common method biases in experiments with a mediating variable. In this current study only the mediating variable will be measured through the subjects’ perceptual measure. The dependent variable, Software Quality, will be measured objectively using the programs developed by team in passing the test cases for each of the two types of programming tasks. This eliminates the problems due to common method biases.

DISCUSSIONS

Increased acceptance of the concept of extreme programming which emphasize on pair programming has given rise to many interesting issues. Though there have been calls for considering the importance of trust in such pair programming teams there is no systematic study to explore this aspect. This study considered one of the dimensions of trust namely, competence-based trust and its effects on the performance of such pair programming teams. In the past IS researchers mostly have explored the relationship between interpersonal variables such as coordination, conflict, cohesion, communication, etc on the direct performance measures such as productivity, software quality, developer satisfaction etc. This study has identified a mediating variable, perceived effort applied in the relationship between trust and the performance of the pair programming team. This is one of the contributions of this study. Ignoring the trust between the members of pair programming team may create a detrimental effect on the quality of the software produced and this goes against the very necessity of using this methodology.

Limitations of this Study and Suggestions for Future Research

This study explicitly considered only one of the dimensions of trust namely; competence-based trust and this could be a potential limitation. Other dimensions such as integrity and benevolence based trust may limit the effect of competence based trust in some conditions and this calls for a study with more comprehensive consideration for the trust construct. Trust and trustworthiness vary with the progression of time and interaction with the parties involved. In organizations often times the interactions will be between familiar actors (Bigley et al. 1998) and this study did not consider this aspect. Future studies in a field setting, with interaction between familiar actors will give a better understanding on the effects of trust on the performance of the teams.

Increasing popularity of extreme programming with radical practices such as pair programming and collective code ownership has transformed the way in which software is developed. An experiment based on this study is proposed to be conducted. Results from this study will unravel the critical role of trust in pair-programming.

REFERENCES


The Standish Group "Chaos Report," The Standish Group, West Yarmouth, MA.


