December 2003

Three Studies of Collaborative Programming

Madeline Domino
University of South Florida

Three Studies of Collaborative Programming

Madeline Ann Domino
University of South Florida
MDomino@coba.usf.edu

Introduction

Despite an ever-increasing sophistication in software development tools and specialization of information technology (IT) personnel, the failure rate in software development continues to remain high. A recent U. S. Department of Commerce study concluded that software bugs, or errors, cost the U. S. economy an estimated $59.5 billion dollars annually. Although not all software errors are likely to be removed (Glass 2003), more than a third of these costs could be eliminated by an improved testing infrastructure that enables earlier and more effective identification of software defects (Trembly 2002). It is widely recognized that the early detection of software errors in development enhances quality, since it reduces the risks and costs associated with development processes (McConnell 1996).

The Problem and Its Importance

Producing quality software, in an acceptable time frame, is not a new challenge. Since the early 1980s, it has been estimated that the information technology (IT) industry has an 85% failure rate in the development of large-scale, mission-critical software (Ambler 2000). Despite efforts of the industry to remedy these shortcomings, the problem persists.

The quest for quality in software development has been underscored by the Software Engineering Institute's (SEI) ongoing efforts to assist organizations and individuals in improving their software engineering management practices. Specific to the goals of the SEI are "higher quality and productivity, faster delivery, lower costs and better morale." Capability Maturity Models (CMMs) assist organizations in maturing people, process and technology assets towards improving long-term business performance (SEI 2002).

Views of why there is such a high failure rate are varied. Some maintain that the traditional code-and-fix models are inadequate to handle the complexities of large-scale software development (Ghezzi 1991) common in today's turbulent business environment. Others contend that software development is a human endeavor and that traditional methods do not place enough emphasis on associated personnel issues (Cockburn 2000, Jordan 1994).

According to Fowler (2000) traditional processes are often viewed as rigid and change-resistant. As such, these methods may not always be the most appropriate for today's business climate and chaordic organizational structures. As a result, newer software development methodologies, such as collaborative programming have emerged.

A potential solution to the problems of producing higher quality software, in reduced time, may be found by using the newer, innovative development methods. While collaboration during development has always been used, these techniques emphasize high levels of interpersonal collaboration during the entire development process (Fowler 2000). For example, an instance of collaborative programming, which is gaining interest, is pair programming (Beck 2000, Cockburn 2000, Williams et al. 2000).

Anecdotally it is suggested that these development methods produce better quality software in reduced time with higher levels of developer satisfaction (Beck 2000, Cockburn 2000). The limited empirical work to date on pair programming shows mixed results. Nosek (1998) and Williams et. al (2000) found a positive relationship between the use of pair programming and performance outcomes, such as software quality and developer satisfaction. However, Nawrocki and Wojciechowsk's (2001) research does not show these same positive results. Additionally, little explanation has been offered to explain collaborative programming outcomes (Domino et. al 2002).

As companies strive to produce better quality software, more practitioners are beginning to experiment with and use the newer innovative development methods (Biggs 2000). Current practices suggest that some managers are using variations of pure pair programming. These practitioners contend that adaptations of the method produce equally good or better performance outcomes, with greater efficiency (Manzo 2002). While there continues to be growing interest in and use of collaborative programming, many questions remain to be answered.

Does collaborative programming produce higher performance outcomes? If so, what are the underlying factors that contribute to this success? What is the impact of individual developer differences on collaborative programming success? What is the impact of the developmental setting on performance results? What impact, if any, does the collaborative method have on successful performance outcomes? How do the processes used during development contribute to success? Given the continuing need to produce higher quality software, today's current development climate offers an unprecedented opportunity to examine collaborative methods

Purpose of the Study

The purpose of this study is to examine the individual developer characteristics, developmental settings, collaborative methods and the process during development that impact collaborative programming performance outcomes, i.e., task performance and satisfaction. The underlying premise of this study is that successful collaborative outcomes, especially fewer defects, are driven by these factors.

Understanding differences in performance and productivity between individual programmers is important, as it may help us understand how we may raise the lowest level of performance to much higher levels, as well as select individuals for the collaborative development setting. The current work environment often calls for virtual software development. Therefore, exploring the impact of the development setting on collaborative development processes is important as it may help us improve performance outcomes in different work settings. Investigating how adaptations of pure pair programming method impact collaborative processes may assist in implementing changes to the method that enhance productivity, efficiency and individual satisfaction.

Research Questions

A multi-phase methodology is proposed, consisting of an intensive process study and two laboratory experiments. The results of these studies will facilitate our understanding of collaborative software development practices, with an eye towards improving these methods and related performance outcomes. Increased understanding of these innovative software development techniques is of importance to researchers and practitioners alike.

The major research questions proposed for the study are:

> *"Within the context of the collaborative programming technique, how do individual developer characteristics and the processes used during collaborative programming impact performance outcomes?"*
>
> *"Within the context of collaborative programming, does the developmental setting (face-to-face or virtual) impact both the processes used during collaborative programming, as well as related performance outcomes?"*
>
> *"Within the context of collaborative programming, do variations in the type of collaboration impact both the processes used during collaborative programming, and related performance outcomes?"*

Anticipated Contributions

This research addresses the need for more research on the newer software development methodologies. By understanding how, why and when collaborative programming techniques produce better performance outcomes, it is hoped that IT professionals may better address the quality issues that plague the industry today. Additionally, the study will extend our knowledge of important organizational issues related to collaborative techniques, personnel selection, training, and methodology interventions. And finally, the results of this study should add to the body of MIS knowledge, as researchers continue to examine the newer, innovative software development technologies.

Research Design and Methodology

In this multi-phase research design, three studies are conducted to explore the individual developer differences, developmental setting, collaborative methods and process differences that impact collaborative programming performance outcomes. The results of Study 1 are used to inform Studies 2 and 3. The triangulation of the study results will further our understanding of collaborative programming methods and related research questions.

The high-level research model (including the related constructs used in each of the three studies) is shown in Figure 1.

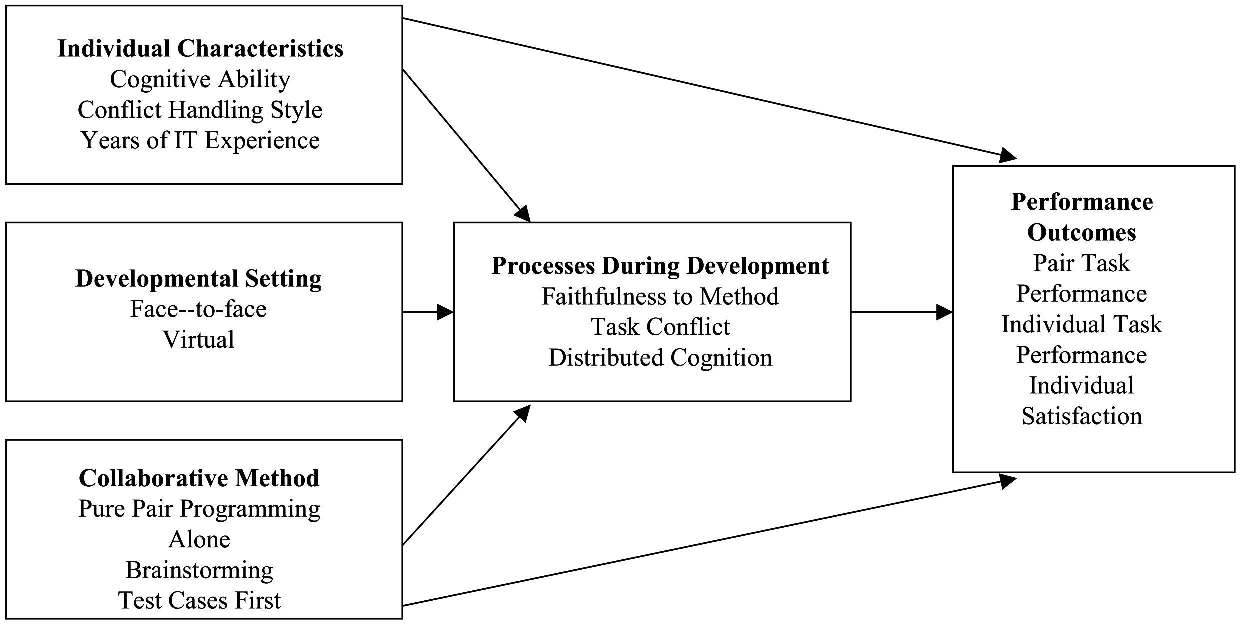

Figure 1. High Level Research Model

The model borrows from Jex's (2002) summary of the most important individual characteristics that impact performance outcomes and Hackman's & Oldman's (1976, 1980) complete integrated job characteristics model. Table 1, shown on the next page, summarizes the constructs used in each phase of the study and details their measurement.

Experimental Tasks

An overview of the experimental tasks is now presented. These tasks have been used in prior research. Three tasks were used in Studies 1 and 2 (i.e. Tasks I, II, and III) to give the pairs time to become accustomed to the collaborative programming setting and to "jell" with their partners, as well as to vary the difficulty of the tasks. Jelling is not part of Study 3, so only Tasks II and III are included. Pseudocode was used in each task (to deal with unknown differences within pairs on specific programming languages), and participants were asked to follow the test-first, code-later sequence in completing all programming exercises.

Task I was designed to be a warm up task. For Task I, subjects were given the pseudocode and test data sets and asked to check the module for accuracy. This required completion of the test data and additional coding.

For Task II, subjects were asked to compute discounts for an invoice. They were given the specifications and asked to create the test data sets and write pseudocode. Task complexity is derived from the interaction from the two discounts.

For Task III, subjects were asked to create a sales report. They were given the specifications and asked to create the test data sets and write the pseudocode. Task complexity is derived from the need to sort and calculate data prior to output.

Table 1. Study Constructs and Measures

Construct	**Measurement & Prior Research**	**Study** 1	2	3
Cognitive Ability	Wonderlic Personnel Test 50 questions, timed 12 minute test Prior research: strong positive link to performance	X	X	X
Conflict Handling Style	Rahim Organizational Conflict Inventory 35 items, 5 point Likert scale 5 subscales: integrating, obliging, domination, compromising Prior research: integrative style has positive link to problem solving and performance	X	X	X
IT Experience	Questionnaire item: number of years of IT experience Prior research: link to performance	X	X	X
Collaborative Method	"Pure" pair programming (all work done in pairs, with test cases written first)	X	X	
	Work done alone (control group) and collaboratively			X
Faithfulness to Method	Type and amount of interaction between partners; Study 1 -- observation utilizing 5-point Likert scale Study 3 – adapted Likert scale on consensus of appropriation (Salisbury and Chin 2002) Prior research*: how* methods are appropriated impacts performance	X		X
Task Conflict	Number of task conflict episodes on each task.. Study 1 -- observation, utilizing a 5-point Likert scale Prior research: low to moderate amounts of task conflict favorably impacts on performance	X		
Distributed Cognition	Study 1 -- observation Prior Research: Enhances performance as individuals work collaboratively	X		
Pair Task Performance	Study 1 - Number of test case and code errors, i.e. more errors the lower the performance. One rater evaluated pair task performance. Study 2 – Completed correct test cases and puesdocode produced (content and sequence), i.e. the greater the number of correct test cases and code, the higher the level of performance. Two raters were used to evaluate performance. Prior research: evaluated the quality of outcomes on the amount of code errors.	X	X	
Individual Task Performance	Study 3 -- Completed correct puesdocode produced (content and sequence), i.e. the greater the amount of correct code, the higher the level of performance. Two raters will be used to evaluate individual performance. Prior research: evaluated the quality of programming outcomes on the amount of code errors.			X
Individual Satisfaction	Adapted 7-point Likert scale Venkatesh and Vitalari (1992) and Watson-Fritz, et al (1996) Prior research: developers working collaboratively have higher levels of satisfaction	X	X	X

Research Methodology

Study 1

Study 1 is an in-depth process analysis with small numbers of developers programming collaboratively (pair programming). This qualitative research focuses on *how* individual developer differences, and specifically task conflict, impact the collaborative software development process and related outcomes, i.e., errors and individual satisfaction.

Subjects completed a series of instruments designed to measure individual differences and receive training in the collaborative programming technique. Subjects were assigned to pairs and asked to complete three experimental tasks. Three tasks were used to give the pairs time to become accustomed to the pair programming setting, to "jell" with their partners, and to vary the difficulty of the tasks.

Pseudocode was used in each task (to deal with unknown differences within pairs on specific programming languages), and participants are asked to follow the test-first, code-later sequence in completing the programming experimental task. Subjects completed a series of instruments designed to measure process differences and satisfaction, after completing experimental Tasks II and III. Subjects were also audio and video taped while performing all experimental programming exercises.

In order to measure the collaborative process, the researchers viewed the audio and videotapes of the developers as they worked together on each programming task. The process results are based on independent analyses of the developer interactions scored by using a pre-established rating form, which measured faithfulness to the method and type and amount of conflict. Performance on task was based on the number of test case and code errors in each programming exercise. The more errors noted, the lower the performance of the pair. One rater evaluated all tasks for errors.

Analysis will also explore the effect of distributed cognition. According to Flor and Hutchins (1991) and Greenberg and Dickleman (2000), distributed cognition enhances or enables performance. In this context, cognition is distributed within the work environment creating a complex, interactive system as two individuals work together on a problem.

Study 2

Forty-two pairs participated in a laboratory study of collaborative software development (pair programming), in which the impact of developmental setting on collaborative programming results is manipulated. It also represents a continued exploration of the impact of individual developer differences and process differences impact on collaborative programming outcomes, i.e., task performance and developer satisfaction.

The researchers randomly assigned classes of students to one of two treatment groups: face-to-face or virtual. Subjects completed a series of instruments designed to measure individual differences and receive appropriate training in the collaborative programming technique. Pairs who were assigned to the virtual treatment group received additional training needed to work in this developmental setting. Within each treatment group, the researcher randomly assigned participants to work together in pairs on three experimental programming tasks.

The same experimental tasks, instruments and protocols used in Study 1 (described above) are utilized in this experiment. Performance on task is based on both the number of correct test cases and the completed correct code produced (content and sequence) for in each programming task. The greater the number of correct test cases and code, the higher the level of performance of the pair. Two raters were used to evaluate task performance.

Study 3

The primary focus of this study is to focus upon the impact of variations in collaborative method on performance outcomes. The underlying premise of this study is that developers, who work collaboratively, will have higher performance outcomes than developers working alone. In this study, solo programmers use a structured task method, i.e. performing test cases before writing code. We further suggest that developers who work collaboratively utilizing a structured task method (test cases) will have higher performance outcomes than developers working collaboratively who use an unstructured task method (brainstorming).

The theory of distributed cognition suggests that problem solving ability is enhanced as individuals work together in a collaborative manner. Newell and Simon (1972) contend that human problem solving is an educated trial and error process, oriented towards explaining behaviors seen in protocols or transcriptions of verbal behavior as subjects "talk aloud". Brainstorming is one such problem solving technique.

Successful outcomes in collaborative software development are also driven by individual developer characteristics and process differences unique to the collaborative method.

Little work to date has explored how the collaborative method impacts collaborative programming outcomes. In this study, we continue to explore the impact of how individual differences and collaborative process differences impact performance outcomes.

A controlled experiment is proposed utilizing experienced software developers. A one by three experimental design, with repeated measures, is proposed for the study as outlined in Figure 2. This type of design is chosen to allow for testing the impact of the covariates and the collaborative processes that may impact outcomes.

Write Test cases first alone, then Write code Alone
Brainstorm first Collaboratively, then Write code Alone
Write Test cases first Collaboratively, then Write code Alone

Figure 2. Experimental Manipulation for Study 3

Research Questions

The specific research questions addressed in Study 3 are:

RQ1. Will developers working collaboratively using a structured task method (writing test case first, before writing code alone) have higher levels of performance outcomes than developers working collaboratively using an unstructured task method (brainstorming first, before writing code alone)?

RQ2. Will developers working collaboratively using a structured task method (writing test cases first, before writing code alone) have higher levels of performance outcomes than developers working alone using a structured task method (writing test cases first, before writing code)?

RQ3. Will developers working collaboratively using an unstructured task method (writing test cases first, before writing code alone) have higher levels of performance outcomes than developers working alone using a structured task method?

Repeated measures experimental designs are often referred to as within-subjects design and offer researchers opportunity to study research effects while "controlling" for subjects. Experimental designs called "repeated measures' designs are characterized by having more than one measurement of at least one given variable for each subjects and offer greater statistical power relative to sample size.

Schedule for Completion

The data has been collected for Studies 1 & 2 and data analysis is now in process. A pretest of Study 3 will be conducted during the spring of 2003 and the full experiment will then follow in the Fall of 2003.

References

Ambler, S. *Adopting Extreme Programming*. Computing Canada, Willowdale; April 2000.
Beck, K. *Extreme Programming Explained.* Boston: Addison-Wesley, 2000.
Biggs, M. "Pair programming: Development times two," *InfoWorld*; Framingham; July 24, 2000.

Cockburn, A . "Just-In-Time Methodology Construction: Humans and Technology," Technical Report, TR 2000.01, Retrieved from **http://crystalmethodologies.org/articles/jmc/justintimemethodologyconstuction.html**, 2000.

Domino, M. A., Collins, R. W., Hevner, A. R. and Cohen, C. F., "Conflict in Collaborative Software Development," *Proceedings of the 2002 ACMSIGCPR Conference*, Philadelphia, Pennsylvania, April 2003, forthcoming.

Flor, N. & Hutchins, E. "Analyzing distributed cognition in software teams: A case study of team programming during software maintenance," In J. Koenemann-Belleveau et al. (Eds.), Proceedings of the fourth annual workshop on empirical studies of programmers, 1991, pp. 36 – 59, Norwood, NJ: Ablex Publishing.

Fowler, M. "Extreme Programming: What is a Lightweight Mythology?" Retrieved from **http://www.extremeprogramming.org/light2.html**, November 2000.

Glass, R. L., "Error-Free Software Remains Extremely Elusive," *IEEE Software*, January / February 2003.

Ghezzi, C. Jazayeri, M, and Mandrioli, D, *Fundamentals of Software Engineering*. Englewood Cliffs, NJ: Prentice Hall, 1991.

Greenberg, J. D. and Dickleman, G., "Distributed Cognition: A Foundation for Performance Support," *Performance Improvement*, July 2000.

Hackman, J. R. & Oldham, G. R. (1976). "Motivation through the design of work: Test of a theory". *Organizational Behavior and Human Performance*, 16, pp. 250 - 279.

Hackman, J. R. & Oldham, G. R. (1980) *Work Redesign*, Reading, MA: Addison-Wesley.

Jex, S. M. *Organizational Psychology, A Scientist-Practioner Approach*, John Wiley & Sons, Inc. 2002.

Jordan, D. *Information Systems Development: an Investigation of Leader Behaviors, Communication competence and Communicator Style as Predictors of Project Leader Effectiveness,* dissertation, May 1994.

McConnell, S. *Rapid Development, Taming Wild software Schedules*, Redmond, Washington, Microsoft Press, 1996.

Manzo, J, "The Odyssey and Other Code Science Success Stories," *CrossTalk The Journal of Defense Software Engineering*, Vol. 15, No. 10, October 2002, pp. 22 – 24.

Newell A. and Simon, H. A. *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall (1972).

Nosek, J. "The Case for Collaborative Programming," *Communications of the ACM*, (41:3), March 1998, pp. 105 – 108.

Nardi, B. A., "Concepts of cognition and consciousness: Four voices," *Journal of Computer Documentation*, 22, 31 – 48. (1998)

Nawrocki, J. and Wojciechowski, A. Experimental Evaluation of Pair Programming, in K. Maxwell, S.Oligny, R. Kusters, E. van Veenendaal (eds.), *Project Control: Satisfying the Customer*, Shaker Publishing 2001 (Proceedings of the 12th European Software Control and Metrics Conference, ESCOM 2001, 2-4 April 2001, London), 269-276.

Rahim, M. *Rahim Organizational Conflict Inventory-II*. Palo Alto, CA: Consulting Psychologist Press, 1983.

Salisbury, D., Chin, W, Gopal, A., Newsted, P. "Research Report: Better theory through measurement—developing a scale to capture consensus on appropriation," *Information Systems Research*, Linthicum, March 2002.

Software Engineering Institute, Retrieved from **http://www.sei.cum.edu/cmmi/presentations/euro-sepg-tutorial/sld014.htm**, CMMI, 2002.

Trembly, A. C. "Software bugs cost billions, study says," *National Underwriter:* Erlanger; July 29, 2002.

Venkatesh, A. and Vitalari, N. "An emerging distributed work arrangement: an investigation of computer-based supplemental work at home," *Management Science*, 38, 12, (1992), pp. 1687 – 1706.

Watson-Fritz, M., Narasimham, S. and Rhee, H. "The impact of remote work on information organizational communication," *Proceedings of the telecommuting '96 Conference*, April 25 – 26, 1996, Jacksonville, FL

Williams, L., Kessler, R., Cunningham, W. and Jeffries, R. "Strengthening the Case for Pair Programming," *IEEE Software* (14), July/August 2000, pp. 19-25.

Wondelic, Inc. (1999) *Wonderlic Personnel Test & Scholastic Level Exam User's Manual.* Libertyville, IL: Author.