Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2004 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2004

An Experiment in Collaborative Programming: How Distributed Cognition Impacts Performance Outcomes

Madeline Domino
University of South Florida

Tim Klaus
University of South Florida

Rosann Collins
University of South Florida

Follow this and additional works at: http://aisel.aisnet.org/amcis2004

Recommended Citation
Domino, Madeline; Klaus, Tim; and Collins, Rosann, "An Experiment in Collaborative Programming: How Distributed Cognition Impacts Performance Outcomes" (2004). *AMCIS 2004 Proceedings*. 194.
http://aisel.aisnet.org/amcis2004/194

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 2004 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

An Experiment in Collaborative Programming: How Distributed Cognition Impacts Performance Outcomes

Madeline Domino
University of South Florida
mdomino@coba.usf.edu

Tim Klaus
University of South Florida
tklaus@coba.usf.edu

Rosann Webb Collins
University of South Florida
rcollins@coba.usf.edu

ABSTRACT

Agile methods are innovative systems development methods that strive for greater speed and flexibility in development and focus on developer collaboration. Pair programming is an agile method in which two programmers work together to write test cases and program code. Anecdotal evidence and some research on pair programming find that this approach produces better quality software, in reduced time, with higher levels of developer satisfaction. To date, little explanation is offered about *how* developers should work together in pairs to realize these improved performance outcomes. In this paper we report on an intensive study of developers using pair programming on several tasks. We use the theory of distributed cognition to analyze the programming dyads' communications while working. We anticipate that this study will help illuminate why collaborative development methods can improve programming outcomes, and identify specific work processes that can be taught as best practices to those using such methods.

Keywords

Agile Methods, Pair Programming, Distributed Cognition, Collaborative Software Development

OVERVIEW

The software industry continues to struggle with producing quality software efficiently in shorter periods of time and it is widely recognized that the early detection of software errors in development enhances quality (McConnell, 1996). Many of the development methodologies in use today are derived from practices relevant to very different organizational and business environments. Accordingly, there is a need to consider newer development practices (Fitzgerald, 1997) while understanding that new practices require developers to change how they perform their tasks.

Collaborative programming is used in a small team setting. Anecdotally it is suggested that the newer, collaborative development methods produce better quality software (Beck, 2000; Cockburn, 2000); however, little explanation has been offered as to *how* developers should work together in pairs to realize these improved performance outcomes. We use the theory of distributed cognition to analyze the programming dyads' communications while working. According to distributed cognition theory, developer dyads that (1) use the test cases to make program requirements more concrete and visible, and (2) share and react to each others' views and ideas of the task (*perspective making* and *perspective taking*), will produce better programs. We are currently analyzing the programming pairs' work sessions using a pre-established coding scheme, and will then investigate the relationship between dyads' distributed cognition and their programming performance. We anticipate that this study will help illuminate why collaborative development methods can improve programming outcomes.

Research Approach and Question

This study is part of a larger research effort that investigates collaborative development methods in several settings. Collaborative methods seem to produce better software but there is little explanation offered as to how these improvements may occur. In this study we explore how developers work together in pairs on programming tasks by using the theory of

distributed cognition to analyze the programming dyads' communications while working. Past research has offered this theory as a viable explanation for improved performance in programming (Flor and Hutchins, 1991).

The specific research question addressed in this study is: Do developer dyads produce better programs when they (a) make program requirements concrete and visible and (b) communicate via positive *perspective making* and *perspective taking*?

LITERATURE REVIEW

Collaborative Software Development (Pair Programming)

Collaborative programming (pair programming) is an agile method that has received much interest because there is both anecdotal and some research evidence that this method results in higher quality code, in less time, with higher programmer satisfaction (Beck, 2000; Cockburn, 2000; Williams et al., 2000). This method involves two developers, working together in intense collaboration, producing one artifact. One developer takes the role of the driver, writing the code and using the keyboard, while the other developer functions as the navigator, monitoring results, looking for specific details and strategic defects. Periodically partners switch roles, resulting in a highly interactive development process (Beck, 2000). Another differentiating feature of collaborative programming is that the developers create test cases first and then write the associated code. This somewhat unconventional sequence of formulating test cases before writing code is believed to be the reason for the reduced defect rate in code developed with pair programming (McBreen, 2003).

Distributed Cognition

The traditional view of cognition maintains that problem solving is exclusively an internal phenomenon (Salomon, 1993) that is best explained in terms of information processing at the individual level (Rogers, 1997). One alternative view of cognition that has gained interest over the last decade is distributed cognition. Originally conceptualized by Flor and Hutchins (1991), distributed cognition represents a new paradigm for rethinking all domains of cognition (Greenberg and Dickleman, 2000). Distributed cognition refers to the knowledge representation both inside the head of an individual and in the world, and the propagation of knowledge between individuals and artifacts (Greenberg and Dickleman, 2000). Central ideas of this theory are that collaborative work is more effective when individuals represent their task knowledge in a concrete, visible form (Nardi, 1996) and when knowledge is transmitted between individuals is a truly collaborative way. True collaboration is evidenced when individuals offer their knowledge and expertise (termed *perspective making*) that is received and appropriated by the other individual(s) (termed *perspective taking*) (Brown et al., 1993, Flor and Hutchins, 1991, Greenberg and Dickleman, 2000).

In a study of developer dyads working on a software maintenance task, Flor and Hutchins (1991) found a relationship between performance and communication between developers that demonstrated key distributed cognition dimensions: sharing goals, sharing memories, expansion of search alternatives. Distributing work across groups of agents (as in the programming dyad) requires co-ordinate activity through some form of communication, such as language or the transmission of artifacts (Hutchins, 1995; Perry, 1997). In the case of collaborative programming, these cognitive artifacts are represented by test cases and code.

TASK OUTCOMES

Since an important goal of collaborative programming is higher quality code, the *quality* of task outcomes is the dependent variable. In studies of collaborative programming the quality of performance has typically been viewed as *accuracy*, measured by fewer errors in the code produced (e.g. Domino et al., 2003). There are two primary reasons, according to distributed cognition, why pair programming should be an effective development technique. First, test cases are concrete, visual representations of how a program should process data and as such they are more easily shared than abstractions. Each test case represents a kind of *narrative* of a single operation of the program (an event, a situation). According to Perry (1997), narrative is a fundamental mode of human cognition that is as powerful as more abstract information processing modes. Therefore we hypothesize that:

H1: Developer dyads that create more test cases will create more accurate programs than dyads that create few or no test cases.

Second, the pair programming method, when faithfully employed, means that each developer shares his/her knowledge about the task (*perspective making*) and that his/her partner then reacts appropriately (*perspective taking*). These reactions may be statements of agreement, encouragement or appreciation of the perspective, elaboration on the idea, an expression of

appreciation of the pair's mutual dependency, or disagreement. Negative communications during collaboration include statements that express domination or control over the other person regarding the rightness of one's own perspective and failure to react to a perspective taken by the partner. Therefore we hypothesize that:

H2: Developer dyads who communicate while working on the task with more sequences of positive perspective making and perspective taking will create more accurate programs than dyads who either have more negative communications or do not communicate (one person does the work while the other watches).

Research Method

This research was performed at a university located in the southeastern United States. All scripts, instruments and experimental tasks were pre-tested, and based on these results, changes were incorporated as appropriate. A quasi-experiment using MIS majors was conducted with twelve pairs (24 subjects). The participants were part-time undergraduate and graduate students who had both knowledge of multiple programming languages and industry experience in programming.

Pairs of programmers were audio and videotaped in the laboratory while they completed three experimental programming tasks. Each session took place in one day in a four-hour session. The session began with a team building activity and an introduction to the study. Participants read and signed an Informed Consent Form (all study procedures and materials have been reviewed by the University Institutional Research Review Board). Training in the collaborative programming technique (pair programming) followed.

Pairs of subjects were then assigned to a computer lab. We allowed subjects to self-select where possible; otherwise, pairs were assigned at random. Subjects were randomly assigned to the role (driver or navigator) that they would assume during the experimental tasks. These roles remained constant for the first two tasks; partners switched roles for the last collaborative exercise.

Subjects were given the experimental tasks in both hard copy and electronic form, but were asked to save all final work on a disk. Three tasks were used to give the developer dyads time to become accustomed to the pair programming setting and to "jell" with their partners, as well as 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 were asked to follow the test-first, code-later sequence in completing the programming exercises.

Twenty minutes were given to complete Task I, which was designed to be a warm up exercise, while one hour was allotted for the completion of each of the two remaining programming assignments. Following the completion of Tasks II and III, subjects were instructed to save all work and complete a questionnaire. Subjects were debriefed at the end of the session. Due to space limitations, please contact the authors for the experimental tasks.

Measurement

In order to measure the communication processes associated with distributed cognition, the audiotapes were transcribed and analyzed using a pre-established coding scheme (Table 1). Coding of the work session interactions is currently underway. Pair performance on task was based on the correct test cases and code produced for in each programming task. A scoring template was developed by the researchers to rate the programming outcomes. A score of 1 – 10 was possible on both test cases and code for each programming task. The more complete and accurate the code, the higher the level of performance of the pair. Two independent raters evaluated all test cases and code for accuracy (inter-rater reliability = 90%). The rating of task outcomes is complete.

Work to be Completed

The work sessions must still be coded for distributed cognition and inter-rater reliability assessed. Then the hypothesized relationships will be tested using non-parametric statistics. Non-parametric statistics are used because of the relatively small number of subjects typical in this kind of process study and since we have little reason to assume that the distributed cognition variables are normally distributed.

ANTICIPATED CONTRIBUTIONS

The goal of this study is to test distributed cognition theory in the program development context and to explore whether or not *how* developers work together during pair programming explains the improved performance outcomes reported for this agile method. We anticipate that the test of the theory of distributed cognition, which is a relatively new way to understand cognition in collaborative work, will contribute to our understanding of human cognition. We also anticipate that this study

The first level of coding categorizes the type of activity being done or discussed: • RI: Reading of instructions • TP: Task planning • TC: Working on test cases • PS: Working on pseudocode • IR: Interpersonal relationship • OT: Other
The second level of coding describes the nature of the distributed cognition. • PM: An individual is expressing his/her own understanding of what is to be done or how to do it or actually does the work (*perspective making*) o D: An individual is expressing domination or control over the other person regarding the rightness of his/her perspective or work • PT: An individual is reacting to the other person's expression of understanding or work (*perspective taking).* This reaction may take the form of: o A: Agreement o E: Encouragement or appreciation of the perspective o EL: Elaboration of the idea, may be agreement and mild disagreement o MD: Expression of appreciation of the pair's mutual dependency o D: Disagreement o I: Ignore (i.e., no reaction from the other person)

Table 1. Coding Scheme

will make a practical contribution to understanding how to improve program development using pair programming. If we understand in more detail what it is about pair programming that leads to performance gains, we can then teach individuals in those specific processes. While the experimental nature of the study offers a more controlled test of the theory, it also creates some limitations. The results are not generalizable to a known population, and the relatively short duration of work on the programming tasks (compared to a normal work setting) may result in weaker effects (the novelty of working together may make positive distributed cognition more difficult to achieve, but also mask individual's negative communications, such as domination).

REFERENCES

1. Beck, K. *Extreme Programming Explained* Addison-Wesley, Boston, 2000.
2. Brown, A. L., Ash, D. Rutherford, M. Nakagawa, K., Gordon, A., & Campione, J. (1993). "Distributed expertise in the classroom." In G. Salomon (Ed.), *Distributed cognitions* (pp. 188 – 228). New York: Cambridge Press.
3. Cockburn, A. "Just-In-Time Methodology Construction: Humans and Technology," 2000.
4. Domino, M.A., Collins, R.W., Hevner, A.R., and Cohen, C.F. "Conflict in Collaborative Software Development," Proceedings of the 2003 ACMSIGCPR Conference, Philadelphia, Pennsylvania, 2003.
5. Fitzgerald, B. (ed.) *System Development Methodologies -Time to Advance the Clock*. Plenum Press, New York, 1997.
6. Flor, N. & Hutchins, E. "Analyzing distributed cognition in software teams: A case study of team programming during perfective 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
7. Greenberg, J. D. and Dickleman, G., "Distributed Cognition: A Foundation for Performance Support", *Performance Improvement*, July 2000.
8. Hutchins, E. *Cognition in the Wild*. Cambridge MA: MIT Press, 1995.

9. Nardi, B. A. "Study Context: A comparison of activity theory, situated action models and distributed cognition". In A. A. Nardi (Ed.), *Context and consciousness: Activity through and human-computer interaction*, Cambridge: MIT Press (1996).
10. McBreen, P. *Questioning Extreme Programming* Addison-Wesley, 2003.
11. McConnell, S. *Rapid Development: Taming Wild Software Schedules* Microsoft Press, Redmond, Washington, 1996.
12. Perry, M. J., "Distributed Cognition and Computer Supported Collaborative Design: The Organization of Work in Construction Engineering", A Thesis submitted for the degree of Doctor of Philosophy, April 1997; Extracted from www.brunel.ac.uk/~cssrmjp/MP_thesis/Ch3.pdf - (2002), pp. 44 - 65.
13. Rogers, Y. A brief introduction to distributed cognition. Retrieved from: http://www.cogs.susx.ac.uk/useres/yvonner/dcog.html (1997)
14. Salomon, G. *Distributed cognitions: Psychological and educational considerations*. New York: Cambridge University Press (1993).
15. Williams, L., Kessler, R., Cunningham, W., and Jeffries, R. "Strengthening the Case for Pair Programming," *IEEE Software* (14), July -August 2000, pp 19-25.