Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2006 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2006

A cognitive perspective on pair programming

Radhika Jain
The University of Memphis

Jaime Muro
Zicklin School of Business

Kannan Mohan
Zicklin School of Business

A cognitive perspective on pair programming

Radhika Jain
Department of Management Information Systems
Fogelman College of Business and Economics
The University of Memphis
Memphis, TN 38152
Email: r.jain@memphis.edu

Jaime Muro Flomenbaum
Department of Computer Information Systems
Zicklin School of Business
Baruch College
New York City, NY 10010
Email: jaime_muroflomenbaum@baruch.cuny.edu

Kannan Mohan
Department of Computer Information Systems
Zicklin School of Business
Baruch College
New York City, NY 10010
Email: kannan_mohan@baruch.cuny.edu

ABSTRACT

Pair programming has gained widespread popularity with the advent of agile methodologies like Extreme Programming. Studies examining productivity improvement resulting from pair programming have produced mixed results. One of the important factors contributing to successful pair programming is pair composition. Developers' styles need to be complimentary to each other, in order for each developer to derive benefits from other's strengths. In this research, we focus on empirically investigating the impact of pairing experts and novices in different ways on their productivity.

Keywords

Pair programming, pair composition, cognitive biases, expert-novice characteristics, systems design performance, developer productivity

INTRODUCTION

Pair programming (PP) is a practice in which two developers work collaboratively on one computer on the same design, algorithm, code, or test. The pair is made up of a driver, who actively works with the computer and a navigator, who attentively watches the work of the driver, identifies problems, and makes suggestions (Williams and Kessler 2000). The two developers switch roles frequently. Proponents of PP claim that it brings significant benefits in terms of higher team productivity, improved quality, reduced time-to-market, lower costs, improved knowledge building and tacit knowledge transfer, and strengthened trust and morale (Canfora, Cimitile, and Visaggio 2003). PP allows developers to share their ideas immediately and to conduct ongoing review of the program code to reduce the defect density of the code (Padberg and Muller 2003). Differences in the performance of these approaches is attributed to the quality of developers' mental models of the task to be performed (Lim, Ward, and Benbasat 1997). Lim et al (1997) argue that subjects working together are likely to have better understanding of three aspects of development viz. the surface structure of a program, its deep structure, and execution of a program by machine (Lui and Chan 2004), and thus form higher quality mental models.

Despite the expected advantages of PP, results of past empirical studies on PP, report contradictory findings in terms of improvement in software quality and effort spent (For examples see (Hulkko and Abrahamsson 2005; Nawrocki and Wojciechowski 2001; Nosek 1998)). We argue that one of the important factors contributing to such contradictory findings of pair programming is pair composition. Pair composition focuses on the "characteristics (or development styles) of two developers paired together to work on a development task". Developers' styles need to be complimentary to each other, in

order for the pair to derive benefits by leveraging their knowledge and increase productivity. When working together on a problem solving task, these developers may exhibit different levels of cognitive biases as a result of their expertise and this in turn may affect the performance of the pair. Past research on PP has primarily focused on costs and benefits, largely ignoring how cognitive characteristics of paired developers affect the outcome. It has been recognized in the past literature that software developers are affected by cognitive biases when they perform various tasks. However, little attention has been focused on examining the differences in cognitive biases exhibited by experts and novices, and especially their collective cognition when paired together. Motivated by the dearth of research in this area, our research objective is *to examine the relationship between pair composition, cognitive biases, and productivity.*

In next section, we review common cognitive biases that impact software developers. We then review the literature on cognitive differences between experts and novices. We then present the details of research methodology used, preliminary findings, followed by conclusion.

COGNITIVE BIASES

Cognitive biases are mental behaviors that have influence on people's problem solving approaches (Arnott 2006). People employ cognitive biases to reduce uncertainty and complexity when processing information during problem solving (Tversky and Kahneman 1974). These biases are often employed to simplify complex inference tasks to more manageable proportions (Parsons and Saunders 2004). While there have been numerous studies done on the effects of cognitive biases, there is no one universal strategy that is prescribed to mitigate their effects. Suggested strategies in the literature are often problem-specific. While the effects of these biases have been well studied in general problem solving, their impact on pair composition in software development has gained limited attention. The most common cognitive biases experienced during problem solving include anchoring and adjustment, and confidence biases.

Anchoring and adjustment bias

Anchoring and adjustment bias results from people forming initial estimates about a problem and adjusting their initial estimates to arrive at more appropriate final solutions (Parsons and Saunders 2004; Stacy and MacMillan 1995). This initial estimate, known as an anchor, is 'a familiar/known position or a reference point' (Kahneman and Tversky 1973). Different starting points may yield different estimates that are biased towards these initial starting points. Anchoring and adjustment have been known to impact several tasks like artifact reuse and decision making (Parsons and Saunders 2004). Anchors are typically inaccurate and developers may not be aware of these inaccuracies and thereby not realize the need for adjustment or adjust insufficiently. Insufficient adjustments are made when there is an uncertainty about the correct solution. This may lead to premature termination of the task due to reluctance to spend more effort in adjusting sufficiently.

For example when working together in a pair, requests for clarification from the navigator may often bring out alternate strategies, shortcomings, or flaws in the approach or solution. In this process of clarification, the driver revises his/her own assumptions and adjusts the solution. Without such brain-storming, the driver may unwittingly settle for less than optimal solutions. This exercise of providing clarifications may solidify and deepen driver's own understanding and improve mental representation of the problem for both the developers, leading to improved *inference potential* (Lim et al. 1997). Lim et al (1997) define interference potential as "potential [of mental model] for generating proper inferences about and predictions of a system's behavior".

Confidence biases

Confidence biases such as confirmation and availability bias as the name indicates "*act to increase individual's confidence in his or her prowess as a decision-maker (Arnott 2006)*". As a result, individuals may discontinue their search for new evidences to help accomplish problem-solving task and may make less informed decisions. For example, confirmation bias suggests that people tend to focus on information that is consistent with their preconceived notions while they disregard information that is inconsistent with their past conceptualizations (Fisher and Statman 2000). However when working together, people are likely to observe actions or feedback of the other person, and incorporate it in their decision-making process.

For example when working together in a pair, brainstorming can bring up better alternate strategies or complement the existing knowledge-base of the pair. Creation of such co-owned understanding can help reduce the impact of such confidence biases. This reduces the emphasis that developers may otherwise place on their potentially incomplete pre-conceived notions or past conceptualizations.

In the next section, we review the literature on expert-novice comparison to gain insights into how experts and novices approach problem-solving tasks.

EXPERT NOVICE RESEARCH SUMMARY

Various dimensions on which experts and novices differ are summarized in **Table 1**. This summary on the cognitive abilities of the expert and novices suggests that major differences lie in

1. The nature of knowledge and its generalizability,
2. How knowledge is acquired, organized, retrieved, and applied,
3. Characteristics of their problem solving strategies,
4. The nature of post completion activities performed, and
5. Use of comprehension aids.

When working together developers may be de-biased as a result of the interactions and brainstorming that take place. Such de-biasing may work in their favor improving their solutions. We argue that the extent to which de-biasing occurs and its effectiveness depends on the nature of pair composition. Given important differences between experts and novices, it is likely that experts and novices will be affected differently by cognitive biases and this in turn will affect collective cognition in development activities. For example, it is possible that when expert and novice are paired together, clarifications requested by the novices can bring up many hidden assumptions made by the expert. This interaction as a whole will then improve collective cognition of this pair, improving their mental model representations and thus their performance.

There has been little attention on pair composition based on cognitive characteristics of developers. Lui and Chan (2004) propose a staged cognitive programming model (definition, representation, model, schema, algorithm, and code) to explain when and why a pair may outperform two individuals when working on a problem. With an empirical evaluation of their model, they conclude that pair programming strategy is more effective when pairs work together on un-familiar problems and least effective when they work on familiar problems. Muller and Padberg (2004) suggest that the level of comfort that developers feel with pair programming has impact on pair productivity and not so much upon how pairs are composed. Cao and Xu (2005)'s examination of activity patterns in PP indicates that different pair composition yields differences in activity patterns of a given pair. For example, they found that when two highly competent student subjects were paired together, their interactions were richer and involved deep-thinking activities. These contradictory findings require further examination.

Based on the above discussion, our research objective leads to the following questions:

1. How does pair composition affect cognitive biases exhibited by a pair?
2. How does the collective cognition of a pair impact their pair performance?

RESEARCH METHODOLOGY

We conduct verbal protocol analyses to gain insights into how cognitive biases may be impacted by different pair compositions. This methodology is commonly used in research on process tracing, knowledge acquisition, model formulation, and decision making behavior (Todd and Benbasat 1987). Here, we examine the cognitive processes followed by subjects as they work alone and in pairs. We focus on identifying the common cognitive biases that impact the developers' performance.

Category	Expert	Novice	References
Nature of the Knowledge	• Experts draw more upon the abstract experiential knowledge (schematized knowledge).	• Novices draw more upon the concrete knowledge (case-driven knowledge).	(Ball, Ormerod, & Morley, 2004)
Knowledge Generalizability	• Schematized knowledge implies that abstractions of problem and problem solving knowledge are much more crystallized. • By making abstractions, experts' knowledge is applicable in other scenarios.	• Case driven knowledge is much more concrete and is less amenable to be applicable in wide range of scenarios • Novices can apply a problem solving technique to a specific problem, but can not abstract the concept and later use it in a different context.	(Ball, Ormerod, & Morley, 2004)
Knowledge Acquisition	• Abstractions are developed through the identification of patterns on multiple occasions. • Experts use patterns to store knowledge about which actions should be taken in similar situations. • Experts easily pick the right subset of knowledge required to solve a problem	• Novices tend to see superficial details and are unable to recognize patterns.	(Bransford, Brown, & Cocking, 1999; Chase & Simon, 1973; A. Ericsson & Kintsch, 1995)
Knowledge Organization	• Experts recognize features and meaningful patterns of information that are not noticed by novices. • Experts organize knowledge in memory around a relatively smaller number of "big ideas," such as fundamental concepts, principles, theories, or themes in ways that reflect the situation and problems they apply to. • Experts' memory is "tailored" to store information efficiently and effortlessly. • With time experts not only acquire content memory but also skills to better store and retrieve these concepts.	• Novices' knowledge is encoded using everyday concepts and does not reflect clustering around "big ideas". Concepts are individual and not associated with each other. • Knowledge in short term memory is stored in the same time sequence it is captured.	(Bransford et al., 1999; A. Ericsson & Kintsch, 1995; Niemi, 1997)

Category	Expert	Novice	References
Knowledge Retrieval skills	• Expert who may have performed similar tasks multiple times store knowledge in hierarchically organized long term memory, which provides faster retrieval. • Experts also acquire memory skills to allow for the efficient application of this knowledge	• Novices lack the organized long-term memory knowledge. • Knowledge retrieval from short-term memory is sequential and unorganized. • Concepts are piled with no regard to how they are related. This makes the retrieval of even their limited relevant knowledge difficult and unreliable	(A. Ericsson & Kintsch, 1995)
Knowledge Application	• Experts process task information in a more conceptually oriented fashion. • Experts contextualize knowledge with information about where it's applicable. Adding contextual information has been offered to account for the incremental accuracy of experts over novices.	• Novices employ a more data driven processing strategy • Novices resort to lower level definitions or equations to explain behavior.	(Hershey, Walsh, Read, & Chulef, 1990)
Problem-solving Approach Characteristics	• When analyzing problems, experts initially look for principle or laws that help solve the problem. Lower level details are grasped rapidly. • Experts use knowledge and experience to make inferences and connections, in order to "fill the gaps" in problems. • Select the relevant information and encode it in special representations in working memory that allow planning, evaluation and reasoning about alternative courses of action.	• Novices concentrate on decoding lower level details before they can solve a problem. Novices' knowledge is associated with unique or specific concepts, such as a chapter of a book or a particular problem. • Novices lack techniques or procedural methods to solve problems.	(Devine & Kozlowski, 1995)
Evaluative Skills	• Have strong self-monitoring skills and tend to evaluate their solutions for completeness and accuracy.	• Novices generally do not tend to evaluate the quality of their solutions once problem-solving tasks are complete.	(K. Ericsson & Lehmann, 1996; Glaser & Chi, 1988)
Comprehension aids used	• Domain knowledge and past design experience • Consider documentation as waste of time and use it (if at all) as a last resort	• Start with documentation to get initial understanding • Uses knowledge gained through learning, textbook application of concepts	Preliminary findings from this study

Table 1: Major Categories of Differences in Experts and Novices

Experimental design

The experimental design is as shown in Table 2, with five groups. Subjects in each group are provided with design documents and UML models of a library management system. Subjects are asked to perform four tasks that involve incorporating changes to the existing design to address new requirements. It should be noted that when paired together, developers typically switch their navigator and driver roles frequently.

Table 2: Experimental design

No pairing	Novices (G1)
	Experts (G2)
Paired	Two novices paired together (G3)
	Two experts paired together (G4)
	One expert and one novice paired together (G5)

Verbal Protocol Analysis

Ten developers participated in our study. Five of these subjects were experts (G2) (average experience in systems design was more than five years). The remaining five were considered as novices (G1) (average experience of about one year). Subjects were asked to perform four maintenance tasks (two of these were training tasks) at two different levels of complexity. Subjects were asked to verbalize their thoughts as they performed the tasks. Time taken by subjects to complete the tasks ranged from about one to two hours. The first two tasks were simple training tasks to get subjects used to verbalizing their thoughts aloud. Their verbal protocols were recorded, transcribed, and analyzed. Analysis of the verbal protocols is guided by the research model shown in Figure 1. Data collection is currently ongoing for the three remaining groups G3, G4, and G5.

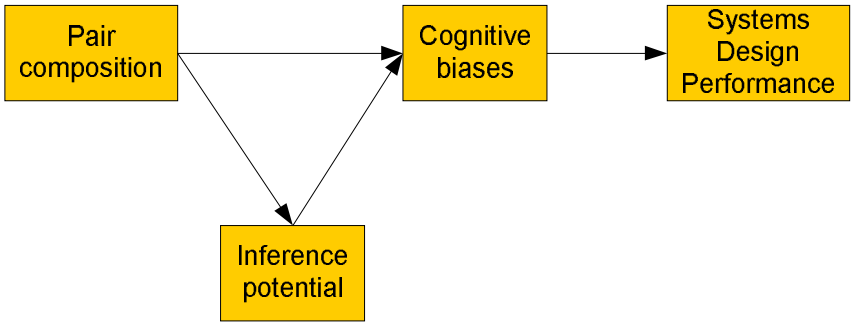

Figure 1: Research Model

Initial findings reveal that software developers are indeed affected by cognitive biases. Key preliminary findings are summarized below:

1. Experts exhibit confidence biases as they were reluctant to examine documentation. Novices were less susceptible to this bias.
2. Experts were more willing to explore wide range of solutions rather than anchoring to just one solution. Novices, due to their inability to identify such wider range of solutions, tended to anchor to their initial solution.
3. Also, propensity of experts to self-evaluate led them to make recurring adjustments to their solutions, unlike the novices who rarely evaluated quality of their solutions, resulting in little or no adjustments to their solutions.

CONCLUSION

This research emphasizes the importance of the relationship between pair composition and cognitive biases exhibited by software developers and how this affects performance. Investigation of this phenomenon on a wider scale by collecting quantitative data to measure the variables shown in the research model (Figure 1) is a subject of ongoing research.

REFERENCES

1. Arnott, D. (2006) Cognitive biases and decision support systems development: a design science approach, *Information Systems Journal*, 16, 1, 55-78.
2. Canfora, G., Cimitile, A. and Visaggio, C.A. (2003) Lessons learned about distributed pair programming: what are the knowledge needs to address?, *Proceedings of Twelfth IEEE WETICE'03*,
3. Cao, L. and Xu, P. (2005) Activity Patterns of Pair Programming, *Proceedings of 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 3*, Big Island, HI, 88a.
4. Fisher, K. and Statman, M. (2000) Cognitive Biases in Market Forecasts: The frailty of forecasting., *The Journal of Portfolio Management*, 1-10.
5. Hulkko, H. and Abrahamsson, P. (2005) A Multiple Case Study on the Impact of Pair Programming on Product Quality, *Proceedings of 27th ICSE'05*, St. Louis, Missouri, USA,
6. Kahneman, D. and Tversky, A. (1973) On the psychology of prediction, *Psychology Review*, 80, 237-251.
7. Lim, K., Ward, L. and Benbasat, I. (1997) An Empirical Study of Computer System Learning: Comparison of Co-Discovery and Self-Discovery Methods., *Information Systems Research*, 8, 3, 254-272.
8. Lui, K. and Chan, K. (2004) A cognitive model for solo programming and pair programming, *Proceedings of Third IEEE ICCI'04*, 94-102.
9. Muller, M. and Padberg, F. (2004) An Empirical Study about the Feelgood Factor in Pair Programming, *Proceedings of Tenth METRICS'04*,
10. Nawrocki, J. and Wojciechowski, A. (2001) Experimental Evaluation of Pair Programming, *Proceedings of 12th European Software Control and Metrics Conference*, 269–276.
11. Nosek, J. (1998) The Cast for Collaborative Programming, *Communications of the ACM*, 41, 3, 105–108.
12. Padberg, F. and Muller, M. (2003) Analyzing the cost and benefit of pair programming, *Proceedings of Ninth International Software Metrics Symposium (METRICS 2003)*, Sydney, Australia, 166 - 177.
13. Parsons, J. and Saunders, C. (2004) Cognitive Heuristics in Software Engineering: Applying and Extending Anchoring and Adjustment to Artifact Reuse, *IEEE Transactions On Software Engineering*,, 30, 12, 873-888.
14. Stacy, W. and MacMillan, J. (1995) Cognitive bias in software engineering, *Communications of the ACM*, 38, 6, 57 - 63.
15. Todd, P. and Benbasat, I. (1987) Process Tracing Methods in Decision-Support Systems, *MIS Quarterly*, 11, 4, 493-512.
16. Tversky, A. and Kahneman, D. (1974) Judgment under Uncertainty: Heuristics and Biases, *Science*, 185, 4157, 1124-1131.
17. Williams, L. and Kessler, R. (2000) All I Really Need to Know About Pair Programming I Learned in Kindergarten, *Communications of the ACM*, 43, 5, 108-114.