TASK COMPLEXITY AND EFFECTIVENESS OF PAIR PROGRAMMING: AN EXPERIMENTAL STUDY

by

VENUGOPAL BALIJEPALLY

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2006

ACKNOWLEDGEMENTS

Dr. Radha Mahapatra was not just my dissertation chairperson, but was also my mentor and guide during the Ph.D. program. His encouragement, advice and patient approach were invaluable particularly during the dissertation process. He was always accessible for any intellectual discussions and provided constructive feedback. Dr. Sridhar Nerur's eagerness and involvement during different stages of my model development and beyond helped me explore some new techniques and constructs. He had an open door policy for discussing any creative ideas or even for casual conversations. Dr. Kenneth Price was quite instrumental in steering me towards the latest research approaches in small group research. His insights and valuable guidance helped shape my research model and hypotheses and improve the theoretical and methodological rigor. Dr. Craig Slinkman with his passion for this research area has helped highlight the practical relevance of my research. Dr. Mark Eakin's guidance was invaluable in helping improve the rigor of my data analysis. Dr. James T. Teng lifted my spirits and enlightened me to new realms of conceptual thinking. The collective guidance of these committee members made my dissertation work much so much more interesting and thought provoking. I thank them all. Special appreciations are due to Dr. Riyaz Sikora for sparing his services as a software 'expert'.

I am grateful to several of my friends who have provided valuable help and support. In particular, I would like to thank Dr. Bharath Rayasam for his inspirational support and editorial help, George Managalaraj for logistical help through out the dissertation work, and Aakash Taneja for help in grading experimental tasks.

Most importantly, I would like to thank my mother Lalitha, who always trusted me to do the right things. She was a constant source of inspiration and support. I dedicate this dissertation to the memory of my late father Vidya Sagar, who would have been proud of his son's accomplishment. My wife Ramani was the source of love and support particularly during stressful times. She never once complained about the inattention that results from working on a project of this size. I am truly grateful to her.

December 5, 2005

ABSTRACT

TASK COMPLEXITY AND EFFECTIVENESS OF PAIR PROGRAMMING: AN EXPERIMENTAL STUDY

Publication No. ______

Venugopal Balijepally, PhD.

The University of Texas at Arlington, 2006

Supervising Professor: Dr. Radha Mahapatra

Extreme Programming, which is recently gaining popularity as an alternate software development methodology, involves two programmers working collaboratively to develop software. This study examined the efficacy of pair programming by comparing the effectiveness of collaborating pairs with those of programmers working individually. Student subjects participated in a controlled laboratory experiment. Two factors were manipulated in the experiment: programming task complexity (high vs. low) and programmers working individually vs. in pairs. The performance of programmer pairs was compared with those of the best performer and the second best performer from among nominal pairs.

An important finding of the study is that programmer pairs outperform second best programmers in nominal groups, but perform at comparable levels as the best programmers in nominal groups. The best programmers among collaborating pairs also develop significantly better understanding of the problem domain, reflected in their task mental model, compared to the second-best individuals working individually in nominal pairs. Their mental models were however comparable to that of the best programmers in the nominal groups. These two relationships were found to be consistent across different levels of task complexity. In terms of perceptual outcomes, the best programmers among the collaborating pairs have comparable levels of overall satisfaction as the best and second-best individuals in the nominal groups, while second-best programmers among collaborating pairs have higher satisfaction than the best and second-best individuals in the nominal pairs. An additional finding was that best programmers among the collaborating pairs have higher confidence in their solution than best programmers in nominal pair when task complexity is low, but not when it is high.

TABLE OF CONTENTS

Appendix

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

In the present knowledge economy driven by technological innovation, the ubiquity and growing importance of software products and services are all too evident. In the United States, software industry's contribution to the national economy in terms of value added was expected to exceed that of auto industry in the year 2000 and also overtake for the first time the contribution of all manufacturing industry groups (MacCormack, 2001). The average percent of IT spending on new packaged software by US firms which went down from 48.1% in 1999 to 34.8% in 2000 bounced back to 41.4% in 2002 (Rubin *et al.*, 2002). These figures suggest that despite some fluctuations due to business cycles, organizations are continuing to invest in software products and services to stay afloat in the present hyper competitive business environment.

Along with continuing improvements in software methodologies the IT project management in organizations has been improving over the years. According to the Standish group's "Extreme Chaos" Report (StandishGroup, 2001), the projects that meet the criteria of success have increased from 16% in 1994 to 28% in 2000. In their survey, projects that were completed on time, on budget and with all features and functions originally specified were categorized as success. During the same period the failed projects (projects canceled before implementation or never implemented) have decreased from 31% to 23%. One of the important reasons for this improvement is attributed to reduction in scope of projects as reflected in the reduction in average cost of projects by more than half during this period. Other reasons cited were use of better project management tools, better skilled project managers and better management processes being used. There is however a big gray area between successful and failed projects that were classified as challenged projects. These were projects completed and operational, but over-budget, over the time-estimate, and with fewer features and functions than originally specified. The challenged projects have decreased only marginally from 53% in 1994 to 49% in 2000. Cost overruns over the original estimate recorded a drastic decline from 189% in 1998 to 45% in 2000. Time overruns during the same period fell more drastically from 222% to 63%. Another positive trend noticed was that required features delivered on challenged projects increased from 61% to 67%. With the average cost of projects going down, the number of projects is expected to double for the year 2002 implying tight scoping of projects. In addition, minimized scope of projects is found to be the fifth most important factor contributing to project success (StandishGroup, 2001).

Though was improvement in the percentage of successful projects over the years, the nearly two-thirds of the projects that fall into the failed or challenged category underline the efforts needed to improve software development methodologies and practices apart from project management and other organizational processes. Use of a formal methodology is found to be one of the top ten critical success factors for software project success (StandishGroup, 2001). The Software community is acutely aware of the efforts needed in this regard, as reflected in the advent of new methodologies over the years.

Concurrent to the improvements in the software project performance, the nature of software development has undergone dramatic changes during the last two decades. While a silver bullet still eludes the software community, the need to build quality systems that are flexible, scalable, and resilient to change has engendered a host of approaches that challenge the wisdom that previously guided software development. The motivation for such methodological changes stems not only from turbulent business environments and the rapid strides made in technology (for example, new programming languages and tools), but also from a growing realization that existing approaches are inadequate and that there is an enormous scope for improvement.

A set of methods, collectively called Agile Development Methodologies, is gaining increasing popularity among software community. Extreme Programming (XP), Feature-Driven Development, Crystal Methods, Scrum, Dynamic Systems Development, and Adaptive Software Development are some of the more popular agile methods. What makes these methods philosophically and pragmatically different from traditional software development methods is their emphasis on: 1) People rather than processes and tools; 2) Adaptation rather than optimization; 3) High quality working software rather than extensive documentation; 4) Customer involvement and collaboration in the development cycle; 5) Embracing change rather than making futile attempts to eliminate it; and 6) Adopting an approach that has short iterations of planning, organizing, and coding along with continuous integration rather than following rigid plans that attempt to anticipate exceptional conditions and changes that might arise (Beck, 2000; Cockburn, 2002; Highsmith & Cockburn, 2001b). In the words of (Cockburn & Highsmith, 2001), "Agility requires that teams have a common focus, mutual trust and respect; a collaborative, but speedy, decision making process; and the ability to deal with ambiguity." In addition to increased agility, the vaunted benefits of agile methods are increased productivity, higher quality, and greater satisfaction/enjoyment of developers.

Extreme programming (XP) developed by Kent Beck (Beck, 1999) is one of the more popular amongst the agile methodologies (Charette, 2001). In a recent survey on XP projects in organizations more than 90% of respondents indicated that they would use XP again in future projects and 100% of the respondents indicated that they would actively advocate XP in the future (Rumpe & Schroder, 2002). At the heart of Extreme Programming is the notion of pair programming, a technique in which two programmers jointly plan, strategize, code, and inspect the software they develop. Pair programming also ensures joint ownership of the code, which is another core concept of XP. There is some anecdotal evidence to indicate that collaboration improves both the performance and enjoyment of the whole problem solving process for programmers (Cockburn & Williams, 2001; Nosek, 1998; Williams & Kessler, 2000; Williams, Kessler, Cunningham, & Jeffries, 2000). There are also studies reporting inconclusive findings on its efficacy (Nawrocki & Wojciechowski, 2001). However, pair programming has been the most controversial and difficult to implement among the XP practices . This could be a result of the traditional view of programming as a solitary activity (Weinberg, 1971) and the ensuing mindset of programmers. There is also some apprehension among IT managers about deploying two programmers on programming tasks that were being handled earlier by one programmer. Unfortunately, very little research was conducted to understand this phenomenon of two programmers working together as a team. The inconclusive empirical evidence is also suggestive of the need to examine contingent factors and the mediating processes and states affecting the outcomes of pair programming.

1.1 Research Questions

The following research questions concerning individual and pair programming are proposed in this study:

1. Whether programming is done individually or in pairs has an effect on the programming outcomes?
2. Whether task complexity moderates the effect of programming method (individual versus pair) on the programming outcomes?

1.2 Importance of Research

With the advent of graphical user interfaces, object-oriented designs and other innovations geared towards meeting the needs of the user in terms of required features and enhanced usability, the sophistication and complexity of software increased manifold. Despite organizations insisting on internal software inspections, rigorous testing, and quality control procedures, the software released and shipped often contains defects in terms of software bugs and security vulnerabilities. Though post release defects per thousand lines of code have been steadily decreasing over the years especially for US companies (Table 1.1), any efforts and improvements in methodologies that could address this issue and reduce defects further could result in enormous savings for businesses.

Table 1.1 - Post-Release Defects per Thousand Lines of Code
Adapted from (Rubin et al., 2002)

	1997	1998	1999	2000	2001
US Companies	0.62	0.57	0.51	0.54	0.37
Non-US Companies	0.44	0.46	0.48	0.47	0.47
All Companies	0.46	0.48	0.47	0.47	0.36

High incidence of computer glitches in mission critical business systems causing disruption or breakdown of operations not only affect the bottom line in terms of costly fixes and lost revenues, but also impact customer and investor confidence. In a recent incident, a software glitch in the flight operations system hosted by EDS Corp affected the operations of two airlines for several hours. For US Airways, the glitch affected 100 flights, while for American Airlines, the glitch affected nearly all of the airline’s 2200 daily flights due to the “ripple effect” (Rosencrance, 2004). In another incident, a software glitch caused payment chaos for Royal Bank of Canada due to payroll delays for thousands of Canadian workers (Bruce, 2004). In the present era of computer frauds, virus attacks, hacker intrusions, and cyber-terrorism, such defects and software bugs could also end up as openings for unwelcome predators, thus seriously compromising the security of the systems and the data with disastrous consequences for organizations. Given the magnitude of impact the software glitches could potentially cause during operations, any effort to improve software quality through reduction, or elimination of these bugs in the first place offers potentially huge payoffs to the organizations.

While software consultants and practitioners are trying to address the issue of software quality through new software practices and new methodologies, the software community is left perplexed about the efficacy of these suggested practices. They are often left wondering whether migrating to these new practices and methodologies is worth the trouble of giving up the security of familiar practices and habits.

In Extreme Programming, pair programming is advocated as a core practice to be followed for all tasks and all aspects of systems development. On the face of it, pair programming appears as a wasteful practice consuming the talents and resources of two people for the job of one. But XP proponents insist that benefits of XP cannot be realized without pairing up as pair programming takes care of dispensing with the big upfront design (BUFD) of traditional methodologies. Hence there is value in establishing the efficacy of pair programming empirically along with any contingent factors that potentially impact its efficacy.

Programming is traditionally conceived as an individual activity, and the profession attracted people of certain dispositions who are comfortable with the thinking and problem solving it involves, and the pleasure derived out of working with systems. With the advent of agile methodologies, software development, which has traditionally been a technical activity, is becoming more a socio-technical activity. There is definitely some hesitation and initial resistance among software developers to pair up. It is therefore important that they are convinced based on personal experience and based on evidence from well-conducted empirical studies about the efficacy of this practice. This leads them to understand the implications and make a more informed transition.

Though there is some anecdotal evidence suggestive of the efficacy of pair programming, there is a need for rigorously conducted empirical studies. Even the evidence available from existing studies is mixed with certain studies reporting positive findings (Nosek, 1998; Williams, 2000) and others reporting inconclusive results (Domino, 2004; Nawrocki & Wojciechowski, 2001).

Though pair programming for all types of software development tasks may prove to be valuable, there is a need to examine whether the benefits of pairing are any different for simple vs. complex tasks. Some practitioners have articulated that they see merit in pairing up for complex tasks as against simple tasks. This needs empirical confirmation to inform practice so that during periods of “death march” (critical phases of project completion especially while trying to beat difficult deadlines) (Yourdon, 1997), whether they could avoid pairing up for certain kinds of tasks with less impact on the performance outcomes.

Even without the context of XP and agile software development, understanding the efficacy of pair programming is a highly topical and pertinent question given the general scarcity of competent and brilliant programmers among IT developers. The software community knew the importance and value of high caliber programmers for a long time. Highly competent programmers are considered valuable resources in software projects that help reduce project risk. However, they are often scarce and are in great demand. Given this reality, it is interesting to study whether programming pairs could achieve software quality which could not be achieved with either of them working alone.

This dissertation addresses the issues discussed above to inform software practice of the efficacy of pair programming and the contingent effect of task complexity on pair performance.

1.3 Overview of the Dissertation

The rest of the dissertation is organized as follows: Chapter 2 provides a review of literature on both technical IS domain and the theoretical landscape from reference disciplines. From the IS technical domain, review is provided on agile methodologies, Extreme Programming and pair programming. For the theoretical perspectives, review is provided on group task typologies, task complexity, social motivation, mental model theory, distributed cognition theory, communication action theory, and individual versus group effectiveness literatures. Chapter 3 focuses on the theoretical development of the model and hypotheses. The experimental design, data collection methods are discussed in Chapter 4. In this chapter, a discussion on the research setting and sampling frame is first provided. Then, construct measures as well as method of analysis used to empirically test the hypotheses are summarized. The analysis and results are presented in Chapter 5, while the implications of the findings, relevance to practitioners and IS academics are discussed in Chapter 6. Limitations of the study and possible areas of future research are also presented here.

CHAPTER 2

LITERATURE REVIEW

This chapter reviews literature relevant to the study of the influence of mode of participation on the effectiveness of software development and the moderating role of task complexity on this relationship. The chapter consists of four sections. The first section examines agile software development and contrasts it with traditional software development approaches. The next section reviews literature on Extreme Programming (XP), one of the more popular agile methodologies and explores its practices. The third section reviews pair programming, an important practice in XP, and discusses empirical studies that investigated its effectiveness. Theoretical perspectives underpinning the present research are reviewed in the fourth section. In this section research on problem space theory, distributed cognition, mental model theories are reviewed along with literatures on task typologies, task complexity, social motivation, and individual versus group effectiveness. The last section introduces the proposed research model of the study.

2.1 Agile Software Development

The nature of software development has undergone dramatic changes in the last two decades. New Object Oriented programming languages, analysis and design methodologies are gaining prominence over traditional languages and structured analysis and design approaches. A growing need has emerged to build quality systems that are flexible, scalable, and resilient to change. The need for more control and ability to respond to changes in the environment at any stage of software development has engendered a host of approaches that challenge the conventional wisdom of software development. Amidst these changes, software development continues to be an important activity in various organizations with new systems being developed to meet the changing business requirements.

A set of methods, collectively called Agile Development Methodologies, is now beginning to capture the attention of software developers. These methods stress cooperative software development as against individuals developing code to predetermined specifications. The focus is more on the people and the dynamics of their interactions rather than on elaborate requirements planning and rigid software processes. Customers are now part of agile software development teams as full time members, who bring users' perspectives directly into the software development process. Short iterations of planning, organizing, and coding are adopted along with continuous integration to accommodate change throughout the system life cycle. Software developers have to frequently iterate between these different phases of systems development and alter their roles accordingly. Thus, there is a fundamental change in the roles and expectations of software developers involved in these agile teams.

The agile methodologies generated great enthusiasm in the recent past among both practitioners and researchers with several consultants championing different agile methodologies. Extreme Programming (XP), Feature-Driven Development, Crystal

Methods, Scrum, Dynamic Systems Development, and Adaptive Software Development are some of the more popular agile methods. However XP is possibly the best documented and most popular of these agile methodologies (Orr, 2002).

Traditional software development is based on the premise that changes in software requirements at later stages of software development process are more costly to implement and should be avoided through detailed upfront planning. The stress is on extensive requirements gathering, detailed analysis and design of the system followed by disciplined implementation and testing. Traditional software development process is thus conceived as a ‘waterfall’ model involving sequential progress through these various stages. Extensive planning, codified processes, and rigorous reuse are expected to increase predictability and gradual maturing of the process towards perfection (Boehm, 2002). However it became apparent over the years that despite elaborate planning, changes are inevitable during software development. Hyper-competitive environments, changing business requirements and rapid technological changes all call for changes in requirements during various stages of the software development process.

Agile software development has evolved in this setting from ‘exploratory projects’ that involved frontier technology, mission and time critical business applications, characterized by constantly changing requirements. Agility entails the ability to both create and respond to change (Highsmith, 2003; Highsmith & Cockburn, 2001b). In agile methodologies there is a conscious attempt to embrace change rather than make futile attempts to eliminate it. Perceived inadequacies of existing software practices prompted several like-minded consultants to come together and draft the Agile

Manifesto (AgileManifesto, 2001), the principles of which are embedded in these methodologies in varying degrees. These methods are philosophically and pragmatically different from traditional software development methods in important ways. In agile methodologies, there is an emphasis on people and their interactions rather than on processes and tools. Collective teamwork is emphasized over individual creativity. Producing high quality working software rather than extensive documentation is considered paramount. Continuous customer involvement and collaboration throughout the development cycle is another crucial requirement (Cockburn, 2002; Highsmith & Cockburn, 2001a). Table 2.1 summarizes important differences between traditional and agile software development methodologies.

Process Characteristics: The process characteristics of agile software development differ substantially from traditional development. Agile software development engenders short iterative cycles, feature based planning, constant feedback, change tolerance, team proximity, and customer intimacy with focus on overall team ecology (Highsmith & Cockburn, 2001a). User involvement and rapid prototyping are some important tenets shared across agile methodologies. The users prioritize the features and decide the scope and timing of product releases, while the IS developers deal with the estimates, development process and detailed scheduling (Orr, 2002). Developers build software in short iterative cycles involving analysis, design, and implementation, and testing phases to deliver incrementally the functionality desired by the users. Honest working code is considered paramount and hence in each of the iterations the individual stages are handled more in parallel.

Table 2.1 – Important Differences Between Traditional and Agile Methodologies

	Traditional	**Agile**
Focus	Individual creativity Processes and tools Technical	Creative teamwork People and interactions Socio-technical
Process characteristics		
Process focus	Optimization	Adaptation
Change Readiness	Low	High
Process flow	Sequential	Parallel and iterative.
User Involvement	Partial	Complete
Coding standards	Low emphasis	High emphasis
Dominant mode of communication	Documentation	Barrier-free interpersonal interaction
Management Issues		
Decision makers	Systems analysts and project managers	Collocated team members
Level of Management	Micro	Macro
Management control	Command and control	Trust based collaboration and self organization
People Issues		
Roles	Defined and remain constant	Change, often at developer's discretion
Code ownership	Individual accountability	More of team based collective ownership
Rewards	Individual	Team based

Figure 2.1 contrasts the iterative nature of agile software development with the sequential nature of traditional approaches. Open and honest interaction among users and other members of agile teams and not extensive documentation is the dominant mode of communication. Collective team ownership of code as against individual ownership in traditional software development is another hallmark of agile software development. This requires developers to adhere to coding standards so as to make the code understandable.

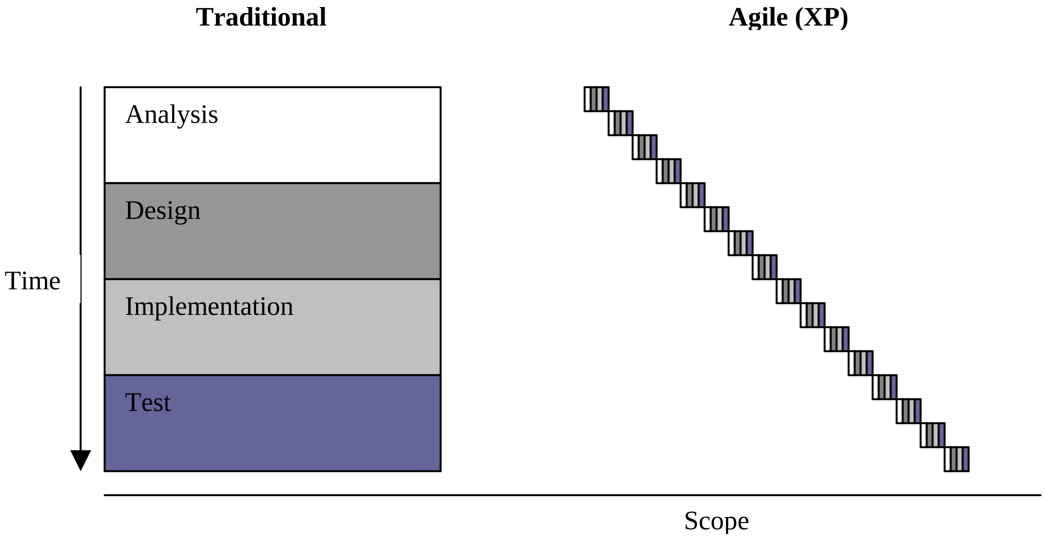


Figure 2.1 – Traditional Vs. Agile Software Development
(Adapted from (Beck, 1999)

Management Issues: Agile development calls for changes in management and leadership styles. In agile methodologies software development has moved away from the traditional command and control style management to more trust based collaborative decision-making style. While traditional software development methodologies place decision-making responsibilities with the systems analysts and the project managers, agile methodologies empower and trust agile teams with decision-making. Agile managers set goals and constraints and believe in macro management and provide agile teams with boundaries within which innovation could flourish. Teams themselves set priorities and schedules, design, test and deliver the software (Orr, 2002).

People Issues: Agile manifesto epitomizes the idea that people with all their abilities, skills, experiences, idiosyncrasies, and personalities constitute first order factors impacting project success. Agility demands valuing people, trusting them and supporting them in overcoming barriers of communication and collaboration (Highsmith, 2002a). There is emphasis on creative teamwork with intense focus on process maneuverability and effectiveness of software development (Highsmith & Cockburn, 2001a). The software team and not the individual developer is the key success factor as reflected in the collective ownership of the code. This entails huge change for software developers and IS managers who are trained to be ‘hardy individualists’ in traditional software development (Orr, 2002).

In agile software development, which involves collaborative teamwork, there is however substantial scope for conflict. Joint ownership of code and collocation of team members with stress on direct one to one interactions could cause conflict that could be both task and relationship based. Task conflicts are the conflicts that arise over substantive issues such as differences over ideas, opinions and ways of approaching a task, while relationship conflicts refer to interpersonal or socio-emotional disagreements generally related to feelings of animosity or annoyance (Bono, Boles, Judge, & Lauver, 2002). Relationship conflict is considered detrimental to the individuals involved and for the groups while task conflict within limits could be beneficial. In agile methodologies the opportunities for conflict are potentially higher than in traditional development. Aspects of agile methodologies such as pluralistic decision-making involving multiple stakeholders, self-organization, joint code ownership etc. could trigger task related conflicts.

2.2 Extreme Programming

Extreme programming (XP) is an agile software development methodology developed and advocated by Smalltalk code developer and consultant Kent Beck with colleagues Ward Cunningham and Ron Jeffries (Williams & Kessler, 2000). Among agile methodologies, XP is well documented, found to be effective, and yet controversial. XP stresses on small teams of 3 to 10 programmers, and requires presence of one or more customers with the development team for providing ongoing expertise. Development is done in iterations of about three-week durations. The deliverable at the end of every iteration is a running and tested code that is ready for customer use. Deliverables from two to five iterations are integrated and given as a ‘release’ to the customer.

A user visible functionality that can be developed in a single iteration, also called “user story”, is the unit for requirements gathering. These user stories are written by customers onto simple index cards and are used by programmers to estimate the time required to complete the functionality given in the card. Customers and programmers negotiate the scope and the time of releases. Customers prioritize, de-scope or alter the stories as needed so that the most important functionality gets done during the agreed time frame.

2.2.1 Core Practices of XP

XP advocates four values: communication, simplicity, feedback and courage. There are also twelve core practices in XP. XP’s practices closely mirror the quintessence of the agile manifesto. These practices emphasize communication and collaboration between the various participants that includes the users. Consistent with the agile manifesto extensive documentation is not done in XP. The core practices of XP are outlined below:

2.2.1.1 The Planning Game

In XP the planning game starts the development process encompassing the requirements determination and project planning. User representatives and developers actively participate in the planning game. Requirements of the systems are captured through user stories. User stories are similar to use cases and help in documenting the various requirements of the users. User representatives select and prioritize the stories to be included in each iteration. Subsequently planning for a particular iteration and release planning are done. Developers implement only the functionality demanded by the stories in each iteration (Beck, 1999).

2.2.1.2 Small Releases

XP follows an iterative and incremental development approach. Frequent releases of the system are delivered to users incorporating the most important functional requirements prioritized by the users. Typically these releases are made after every two to three weeks.

2.2.1.3 Metaphor

Metaphor in XP is equivalent to the system architecture. Metaphor helps the people involved in a project to understand the basic elements in a system. Metaphor concept is not new as highlighted by Beck. For example, a pension calculator application should be like a spreadsheet and here the spreadsheet is a metaphor of the system (Beck, 1999). This spreadsheet analogy helps in visualizing the application.

2.2.1.4 Simple Design

XP emphasizes simple software design to address the immediate needs of the customers. The designer is encouraged to focus on the requirements specified in user stories as opposed to future requirements. Simple design also translates into minimum numbers of classes and methods and no duplication of code (Beck, 1999).

2.2.1.5 Testing

Two types of testing procedures are emphasized in XP - unit tests and functional tests. XP implements Test Driven Development. Here the test cases are developed prior to the actual code with even the possibility of automating these tests. Customers write the functional tests to test the features of the system.

2.2.1.6 Refactoring

Refactoring of software code is the practice of improving the actual code without altering the observable behavior of a delivered system. Refactoring improves program design, makes it easy to comprehend, eliminates bugs and improves productivity. This is a time-tested concept that predates XP. Refactoring facilitates incorporating changes in design at any stage of the project thereby increasing the adaptability of the system.

2.2.1.7 Pair programming

Pair programming is one of the core requirements of XP. It involves two programmers working side by side at one computer, collaborating on the design, coding or testing on a continuous basis. The programmer sitting in front of the computer is the driver doing active coding. The partner acts as navigator overseeing the coding, looking for errors in syntax and logic, and deviations from programming practices and norms.

2.2.1.8 Collective Ownership

In XP everyone in the development team is responsible for modifying the code irrespective of owner authorship. Some common concerns regarding collective ownership are that some programmers would not be happy about changes made to their code by others. Managers also 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 that much harder to produce high quality designs and code (Highsmith, 2002b).

2.2.1.9 Continuous Integration

As soon as a piece of code is ready it is integrated into the code-base. Before accepting changes in the code all tests have to be run and passed. Typically the system is integrated several times in a day. The software community knew the problems with integration defects for a long time now. However, lack of tools and practices earlier prevented this knowledge from being put to good use. XP provides a revised perspective on practices and tools to focus on continuous integration.

2.2.1.10 Forty-hour Week

A maximum of 40-hour week is allowed and overtime weeks in succession are discouraged. Any such occurrence is treated as a problem to be solved. This practice has been enshrined as a rule to avoid burn out of the programmers. However what is important is whether there is a voluntary commitment on the part of the people and whether they anticipate each day at the work with great relish.

2.2.1.11 On-site Customer

Customer presence is required full time for the team. This is based on one of the oldest principles of software development i.e., user involvement. Customers write user stories, prioritize the stories/features for immediate development, and test the functionality of the systems during continuous integration process.

2.2.1.12 Coding Standards

Coding rules are laid down and developers are required to follow them. As coding is done in pairs with community ownership of the code, coding rules become all the more important. Communication through the code is encouraged. Peer pressure during pair programming and the need to make the code comprehensible to the partner is expected to lead to better compliance with coding standards (Beck, 1999).

Many of the practices outlined in XP are not new. However XP seeks to integrate these proven and not-so-proven practices into a coherent methodology (Beck, 1999). XP advocates insist on implementing all the practices together as they form a coherent system compensating for the lack of big upfront design and documentation. There is some anecdotal evidence of XP’s success reported such as in Chrysler Compensation system and Ford Motor Company’s Vehicle Cost and Profit System (VCAPS) which were successfully completed using XP after initial development using other methodologies ran into problems (Williams & Kessler, 2000).

2.3 Pair Programming

Pair programming is a core practice of Extreme Programming. Pairing is recommended to be dynamic with people pairing up with different individuals even in the course of a day. People could enlist a partner with particular competencies for a task unfamiliar to him/her or pair up with anyone in the team. 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 periodically switch roles doing coding and code inspection alternately. The code generated by pair programming is expected to be of much higher quality, as it has to pass the active scrutiny of two programmers who try to identify any possible sources of errors, both syntactical and logical. The code is expected to be much more readable. The programmers are also likely to be more confident of their solution when coded jointly as against working alone. Pair programming encourages each programmer to push the other partner a little harder to excel (Beck, 1999).

The software community knows the superiority of inspections by two programmers for a long time. However, what was little realized earlier was that 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. 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 (Domino, 2004), thus leading to a higher quality product (Williams & Kessler, 2000).

2.4 Theoretical Perspectives

This section reviews some of the critical literature on the theoretical perspectives informing this research.

2.4.1 Problem Space Theory

According to Newell and Simon’s problem-space theory, for any given problem, there is an objective structure involving an initial state and a goal state. There are always several alternate paths available for traversing from the initial state to the goal state. At each stage there are certain legal operators available to navigate along with possibly some restricted operators. An individual’s problem solving behavior involves moving from an initial knowledge state to a goal state by the application of mental operators. Mental operators encode the legal moves and restrictions that are applicable at each state. For the given problem, basic problem space is conceived as the set of all possible states as generated by the legal operators (Eysenck & Keane, 2000; Newell & Simon, 1972).

Programming is considered a complex problem-solving activity involving search in multiple problem spaces – rule space, instance space and representation space. It is analogous to scientific discovery with programmers generating hypotheses in rule space and testing them in instance space. When difficulties are encountered in rule development or when alternate representations are available, programmers also change representations by searching in the representation space (Kim & Lerch, 1997). Similar to hypothesis testing in scientific discovery, programmers are known to retrieve existing schema from their long term memories and apply to the problem at hand with suitable modifications during problem solving (Adelson & Soloway, 1985; Rist, 1989). Programmers also traverse the solution instance for getting better insights into the problem as well as the rules for problem solving. Thus they evaluate and refine the existing rules based on test results from the instance space. When faced with the situation where existing rules are not adequate to offer any clues, programmers are known to engage in ‘exploratory mental simulation’ as the main operator for generation of new rules (Kim & Lerch, 1997). “A representation is a mental model that encodes the programmer’s current understanding of the target system” (Letovsky, 1986). Construct and Map are the two operators used in representation space that manipulate components of the current representation by either filling in empty slots or by changing the existing ones. Filling in empty slot is akin to programmers developing an initial representation before beginning problem solving. Any impasse in the rule or instance space triggers search in the representation space with Construct operator. When programmers face an impasse with the current representation, they invoke Map operator to change representation (Kim & Lerch, 1997). At the end of the problem-solving task, the final representation denotes the understanding of the programmer of the problem space. This mental model of the programmer drives the program solution created and its resultant effectiveness.

2.4.2 Mental Model Theory

The concept of mental model is attributed to Scottish psychologist Kenneth Craik as its originator. It is however Johnson-Laird (Johnson-Laird, 1981) who articulated the theory of mental models. Mental models are the internal representations of objects, people, situations, or actions. Johnson-Laird conceptualized mental model as "a state of affairs and accordingly its structure… plays a direct representational or analogical role. Its structure mirrors the relevant aspects of the corresponding state of affairs in the world." [(Johnson-Laird, 1981) p174]. Mental models are built based on experience and observation, of the particular entity of interest or of the world in general (Wilson, 2000). Even parsing of verbal propositions using procedural semantics creates a mental model that is structurally congruent with the represented world (Johnson-Laird, 1983). Depending upon the circumstances individuals may construct and use several different mental models.

The theory of mental models is based on a functionalist perspective. Functionalism allows defining mental states in terms of their causal effect on behaviors or other mental states (Stubbart, 1989). The mental model theory competes with the view that deduction is based on formal rules of inference. According to Johnson-Laird, creation of a mental model and its manipulation underlies most human cognition. He rejected the notion that cognition is based on formal logic and inference rules. He argued that formal logic arises from the construction and manipulation of certain mental models. Mental models enable individuals to understand the phenomena and make inferences and predictions, thus helping to experience events by proxy. They help decide what action to take and how to control its execution (Johnson-Laird, 1983). Mental models vary in the level of abstraction depending upon the context of investigation. The representational capability and power of mental models stems from the recursive operations on a set of tokens and from a different set of processes directing the creation of "model examples and counter-examples and their evaluations" (Wilson & Rutherford, 1989).

Mental models are distinct from related memory structures. Frames are data structures for representing stereotypical entity. Schemata are active data structures and are more specifically oriented towards psychological explanations. They emphasize more on the control aspects of the operation of a system. Similar to the structure inherent in a set of instructions or processes, schemas can be said to have a structure. Mental model may be considered as the aggregate set of schemata instantiated at any particular time. Alternatively, schemata may be conceptualized as providing procedures from which mental models are constructed. While mental models are considered temporary data structures created at the moment for understanding, it is schemata that are stored and activated. They represent the background knowledge of the mental models (Wilson & Rutherford, 1989).

Scripts and scenarios are typically used in language comprehension and refer to 'extended' activities and social situations. One similarity between different theoretical cognitive constructs such as schemata, frames, scripts and scenarios is that they provide declarative and procedural information in terms of a set of processing operations. They all provide generic or prototypical information and represent knowledge by virtue of typical criteria and not based on a set of necessary and sufficient conditions. Frames and particularly schemas are applied in a more varied manner such as in studies of perception, story comprehension, memory, and sensory motor actions. Scripts, scenarios, and even schemata may be considered as particular implementations of the notion of frame (Wilson & Rutherford, 1989).

Mental models facilitate efficient information processing by making it unnecessary to understand from scratch each time a novel situation is encountered. They help organize knowledge in robust parsimonious ways and reduce complexity. They facilitate learning by filling in gaps in information and memory and updating the models. Mental models direct the perception and processing of stimuli, which in turn help shape or change mental models (Vandenbosch & Higgins, 1996). Analogies help people build a structural map that simulates the way the components of a system interact. People use analogies to construct generative mental models by mapping the rules of transition and interaction from a known domain to an unknown domain (Collins & Gentner, 1987).

The notion of mental models is used in several disciplines. While the focus of mental models in cognitive psychology literature is on explaining mental processes, in other disciplines such as human factors the interest is on the product of such processes. The mental model concept has also been extended into the team domain. Team mental model refers to multiple levels of shared knowledge or aggregate of individual knowledge as well as to "a synergistic functional aggregation of the team's mental functioning similarity, overlap, and complementarity" [(Langan-Fox, Anglim, &

Wilson, 2004) p.335]. Team mental model embraces two team related concepts – mental models and teamwork. Teamwork encompasses concepts such as common valued goals, internal interdependence and coordination etc. Effective group work is facilitated if the members perceive, encode, store, and retrieve information in similar ways, that is, they have accurate and similar mental models and develop common knowledge. One limitation is that mental model and its representation is a function of the experimental paradigm and does not mean a true translation of the mental model. It is however considered a 'useful heuristic' for interpreting the complexity of team functioning as it spans across both knowledge and belief structures (Langan-Fox et al., 2004). In this paper the individual mental models and not team mental models are the focus.

In IS research, it is shown that executive support systems (ESS) help mainly in the maintenance of mental model of the particular domain for the users through focused search. To a lesser extent ESSs also help build mental models if users scan through them to help formulate problems (Vandenbosch & Higgins, 1996). In IS training literature it is shown that subjects who form conceptual mental models during training performed significantly higher than people who formed procedural mental models (Santhanam & Sein, 1994). In one study concerning code evaluation in programming, conceptual models helped improve conceptual understanding reflected in mental models of programmers. Also quality of mental models is found to be positively related to the transfer ability of procedural skills from code evaluation to code generation (Shih & Alessi, 1993).

2.4.3 Distributed Cognition

According to the traditional view of cognition, problem solving is exclusively internal phenomenon involving information processing at the individual level. Theory of distributed cognition extends cognition beyond the individual to interaction of individuals with the materials and resources in the environment (Hollan, Hutchins, & Kirsh, 2000). According to theory of distributed cognition enunciated by Flor and Hutchins (Flor & Hutchins, 1991), the collection of the individuals and the artifacts that are involved in the performance of a task constitutes a complex cognitive system. The mental state of this complex cognitive system comprises the external structures exchanged by the agents of the system. Unlike the mental states of individual cognition that are inaccessible, the external structures of complex cognitive system representing its mental state are observable and hence available for direct analysis. Based on their study of a programming pair undertaking a software maintenance task, Flor and Hutchins identified seven properties of this complex cognitive system involving distributed cognition (Flor & Hutchins, 1991). These seven properties are:

1. Reuse of system knowledge
2. Sharing of goals and plans
3. Efficient communication
4. Searching through larger spaces of alternatives
5. Joint production of ambiguous plan segments
6. Shared memory for old plans
7. Division of labor and collaborative interaction system.

The above properties are considered important for the successful task performance of the system. Active communication processes involving perspective taking and perspective making are considered critical for the effectiveness of a complex cognitive system (Boland & Tenkasi, 1995).

2.4.4 Group Task Typologies

During the long course of small group research, several task typologies have been proposed to study the group task. Steiner's task typology (Steiner, 1972), McGrath's task circumplex (McGrath, 1984) and Laughlin's group task categorization (Laughlin & Ellis, 1986) are the more popular ones used in the context of problem solving research in groups.

Steiner proposed a typology based on the premise that a team performs a variety of tasks that can be combined in different ways. It is based on three questions: 1) can the task be divided? 2) Is quantity more important than quality? 3) How are the individual's inputs related to the group product? Based on divisibility, the tasks could be considered as divisible (subtasks exist) vs. unitary (no subtasks exist). Unitary tasks yield a single outcome and must be performed by the group as a whole. Based on the importance of quantity vs. quality the tasks are divided as maximizing (quantity) vs. optimizing (quality) (Steiner, 1972).

The unitary tasks are further classified based on how members' efforts are combined to yield the group product. Disjunctive tasks involve selecting from individual judgments. They are typically unitary and optimizing, that is they are not divisible into subtasks and quality of output is emphasized. The group must get to a single solution and the group discusses till its members agree on a solution such as in juries and problem solving technical work teams. In Conjunctive tasks all group members must contribute. Such tasks are usually divisible and maximizing. They are not completed unless each of the group members has completed his/her parts as in assembly line. The groups' performance in such tasks is limited by the worst performing member (Steiner, 1972). A group can control this factor by providing support and motivation to this member to work harder, or by assigning the weakest member to the easiest task (Levi, 2001). Additive tasks combine the group members' contributions together such as when a group paints a house or sells a product. They are usually divisible and maximizing. The productivity of a group exceeds that of an individual, but is often less than the sum of individuals working alone due to motivational losses (Levi, 2001). Compensatory tasks involve averaging the input of the group members in creating a solution such as when a group leader seeks opinions of the individual members and then forms a single recommendation from the responses. The average score is typically better than most of the individual members' scores. (Levi, 2001). In Discretionary tasks group decides how to organize such as in self-managed teams. They could be divisible or unitary as well as maximizing or optimizing (Steiner, 1972). The performance is contingent upon whether a group selected an appropriate method to perform the task and hence highly variable (Levi, 2001). The decisions of the group could make it disjunctive (by assigning higher weight to the contributions of its most capable members), conjunctive (by making everybody to complete their tasks), additive (by assigning equal weight to the contributions of each group member), or some unique type (Shaw, 1981).

Laughlin articulated a group task continuum anchored by intellective and judgmental tasks. Intellective tasks have a demonstrably correct answer, while judgmental tasks are evaluative, behavioral, or aesthetic judgments with no correct answers (Laughlin & Ellis, 1986). These tasks also figure in the more comprehensive McGrath's task typology discussed below.

McGrath proposed a task circumplex in which the task categories are mutually exclusive, collectively exhaustive and logically related to one another (McGrath, 1984). The vertical axis denotes the degree to which the task involves collaboration and coordination or conflict resolution. The horizontal axis indicates the degree to which the task entails cognitive or behavioral performance. Figure 2 provides a brief description of the tasks in the circumplex. Intellectual and judgmental tasks of McGrath's typology that involve cognitive task performance are described below.

Intellective tasks or problem solving tasks have demonstrably correct answers, and require choosing correct answers. Consensus is required but reaching solution is straightforward and once the answer is recognized there is often little to debate. If anyone in the group does solve the problem, then the group has solved it. The need to coordinate may be limited. Laughlin's intellective tasks with correct and compelling answers, logic problems and other problem solving tasks with correct but not compelling answers, tasks where expert consensus defines answers, all fall under this category. The key notion is the correct answer.

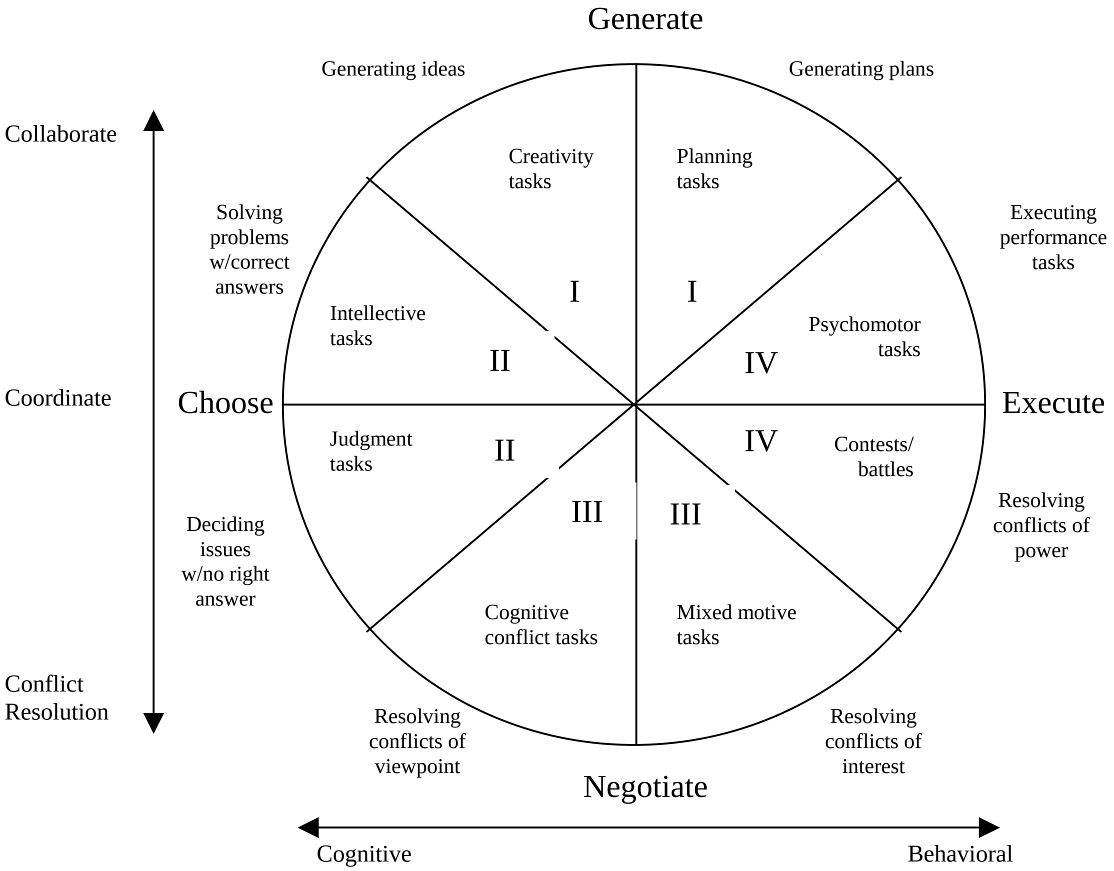


Figure 2.2 – McGrath's Group Task Circumplex
Adapted from (McGrath, 1984)

Decision-making tasks or judgment tasks involve reaching consensus on a preferred answer. They do not have a correct answer. Attaining consensus requires communicating not just facts, but also values, beliefs, and attitudes about the merits of alternate solutions. The effort to establish group choice involves considerable coordination activities. Jury tasks and tasks used in risky shift, choice shift, and polarization studies fall under this category. The key notion for these tasks is to have a preferred answer.

2.4.5 Task Complexity

The task has been central to small group research during the past several decades. A general finding of problem solving research is that the group performance is affected by the task involved (Hackman & Morris, 1976) and task characteristics moderate the relationship between group inputs, processes and outcomes (Goodman, 1986; McGrath, 1984). Among the task characteristics, task difficulty and related concepts figured prominently in the group research. The early task typologies categorized group task as simple versus complex (Shaw, 1954), and easy versus difficult (Bass, Pryer, Gaier, & Flint, 1958). Task difficulty is variedly defined in terms of the amount of effort required (Shaw, 1981) or the amount of thinking time required (Hackman, 1968) to solve a problem task. Tasks could be easy (requiring few operations, skills, and knowledge, and/or having a clear goal) or difficult (requiring many operations, skills, and knowledge, and/or having no clear goal). The perceived task difficulty of a group task is however related to several situational variables such as members' involvement, optimism, and quality of presentation (Hackman, 1968), and process factors such as coordination patterns within the group (Dailey, 1978).

Campbell explicated the construct of task complexity and distinguished it from task difficulty. He articulated the objective characteristics contributing to task difficulty and highlighted the differences between objective task complexity and experienced task complexity (Campbell, 1988). Complex tasks are by nature difficult, but not all difficult tasks are complex. Tasks could be difficult, that is requiring high effort, without being complex due to certain associated characteristics not intrinsic to the task such as communication failure in the group. The notion of difficulty represents a person x task interaction. A task could be difficult for certain people though not for others, where as a complex task is inherently complex without any connotation of person x task interaction. Thus task complexity is conceptualized as emanating only from objective task criteria based on task content but not based on task context (Campbell, 1988).

The review of complexity research by Campbell showed that task complexity was variedly treated as a psychological experience (arousal, challenge, simulation etc. as in (Taylor, 1981), as task-person interaction (familiarity, experience etc. as in (Shaw, 1981) and as a function of objective characteristics (inexact means and ends connections, uncertain/unknown alternatives or outcomes etc. as in (March & Simon, 1958). The experienced complexity is clearly related to objective task complexity. However other factors such as familiarity with the task, short-term memory, attention span, computational efficiency, time constraints, tool availability, and so forth affect this relationship. Other non-task related factors such as task context and anxiety, and individual difference variables such as cognitive complexity also could affect experienced complexity. For example, cognitively complex individuals are known to be able to   sustain their task performance on objectively complex tasks to a greater extent than cognitively simple individuals (Campbell, 1988).

Campbell identified several task attributes that contribute to its complexity. Any objective task characteristic that contributes to increase in information load, information diversity, or rate of information change is considered as contributing to complexity (Campbell, 1988). The four basic task characteristics meeting these criteria are:

Path multiplicity - The presence of multiple potential ways to arrive at a desired end state

Outcome multiplicity - The presence of multiple desired outcomes to be attained

Conflict interdependencies among outcomes - the presence of conflicting interdependence among paths to multiple outcomes

Uncertainty or probabilistic linkages - the presence of uncertain or probabilistic links among paths and outcomes.

Multiple paths – Multiple paths of arriving at desired outcomes lead to information load. This increases complexity when only one path leads to goal attainment among the seemingly possible multiple paths and there is an efficiency criteria embedded in the task against which the multiple paths are evaluated.

Multiple outcomes – Each outcome can be visualized as task dimension requiring a separate information-processing stream. Thus as the number of streams increase, the information processing requirements increase except when the desired outcomes are positively related. In such cases the degree of complexity is reduced due to redundancy.

Conflicting interdependence among paths – Negative relationships among desired outcomes could lead to increase in complexity, i.e., if achieving one desired outcome conflicts with achieving another desired outcome. E.g. quality vs. quantity of outcomes.

Uncertain or probabilistic linkages – Information-processing requirements increase substantially if the connection between potential path activities and desired outcomes cannot be established with certainty. If probabilistic linkages exist, potential paths cannot be eliminated quickly as different outcome contingencies need to be evaluated. This increases the information load for the problem solver.

As a combination of the four levels of complexity Campbell detailed 16 task categories. Further he grouped these 16 task categories under the four broad task types (Table 2.2). For example the decision tasks are distinguished by the presence of outcome multiplicity. Conflict interdependence and uncertainty when present could further enhance complexity. Path multiplicity is not an issue in decision tasks though. Problem tasks are distinguished by multiplicity of paths to a well-specified and desired outcome. Problem tasks could differ in terms of path’s relationship to each other and to the desired end. The task involves finding the best way to achieve the outcome. These are analogous to the intellective tasks of McGrath’s task typology (McGrath, 1984). Judgment tasks are complex from having conflicting interdependencies among outcomes, while fuzzy tasks derive complexity from all four dimensions, though path and outcome multiplicity are the primary drivers of complexity. Problem solving requires analytical skills and convergent processes. Decision-making on the other hand involves creativity and divergent processes (Campbell, 1988).

Table 2.2 – Different Task types and Task Complexity Dimensions
Adapted from (Campbell, 1988)

Task Type		Source of Complexity due to the presence of				Example
		Path multiplicity	Outcome multiplicity	Conflict interdependencies among outcomes	Uncertainty or probabilistic linkages	
A	Decision tasks		X	x	x	Employee selection Choosing a house Selecting a building site
B	Judgment tasks			X	x	Intelligence analysis Stock market analysis Multiple cue probability learning
C	Problem tasks	X		x	x	Chess problems Personnel scheduling Personal placement
D	Fuzzy tasks	X	X	x	x	Business ventures

The effect of task complexity on human cognitive processing and on group processes and outcomes may be explained through activation theory (Scott, 1966). According to this explanation, the level of individual activation or arousal is related to the intensity, meaningfulness, and variation of the cognitive stimulus. To the extent that an objectively complex task provides greater number of stimulus sources, the individual is expected to experience a heightened sense of arousal. This is in addition to the arousal attributed to other factors such as presence of another person in the workroom during task performance, performance-rewards expectations, and so forth (Campbell, 1988). While working on problem solving tasks individuals try applying their existing heuristics to see if they work. As the complexity increases then the individual sets out to create new programs specifically geared to the task (Newell & Simon, 1972). The information processing paradigm has demonstrated that as the objective task complexity increases, there is a corresponding increase in the performance, till the task demands match the cognitive capacity of the individual, beyond which the performance deteriorates (Schroder, Driver, & Streufert, 1967).

In certain studies, task complexity is conceptualized as a predictor variable affecting group process, simultaneously looking for other moderators of this relationship. For example, in one study group cohesiveness has been found to moderate the effect of task difficulty and task variability on group performance (Dailey, 1978). However in majority of the studies task complexity is conceptualized as a moderator affecting the process and outcomes. For example, task characteristics are investigated for their effect on group information processing characteristics (Bystrom & Jarvelin, 1995; Ito & Peterson, 1986), group communication structures (Brown & Miller, 2000), group decision-making activities (Ito & Peterson, 1986), individual and group goal-setting effectiveness (Kernan, Bruning, & Miller-Guhde, 1994; Wood & Mento, 1987), and leader-member dyadic relationships (Dunegan, Uhl-Bien, & Duchon, 2002) and group performance (Jehn, Northcraft, & Neale, 1999).

Human information processing literature has demonstrated that as the complexity of task increases, there is a corresponding increase in the complexity of information needed, both domain information and problem solving information. Also there is increase in the share of information obtained from general purpose sources, and the number of sources accessed. However there is corresponding decrease in the share of fact-oriented sources, internality of channels, and success of information seeking (Bystrom & Jarvelin, 1995). In information processing systems, the task difficulty is found to be directly related to greater participation in decision making and higher boundary spanning activities by the members of the system (Ito & Peterson, 1986). According to a study by Brown and Miller, groups tend to adopt a more centralized communication network while working on simple tasks. As the task complexity increases, more decentralized group communication networks emerge. The effects of task complexity were expected to magnify under conditions of high pressure with decentralized communication networks being highest in high task complexity and high time pressure setting. This task complexity and time pressure interaction was however not supported (Brown & Miller, 2000).

Goal setting theory argues that specific and difficult goals are more motivating for individuals and contribute to higher performance than easy or do-your-best goals (Locke, 1968). However studies have shown that task complexity has a moderating effect on the relationship between goal attributes and performance. It is demonstrated that goal setting effects were stronger for easy tasks than for difficult tasks (Wood & Mento, 1987). Increased performance on simple tasks is attributed to the direct motivational effects of goal setting such as attention, effort and persistence. As task complexity increases, universal and well learned task-specific strategies prove to be progressively inadequate. The need for cognitive processes such as search, information processing and task strategy development required in complex tasks could hamper the goal setting effects in such tasks. Although difficult goals inspire strategy search processes in difficult tasks, this may not contribute to improved performance immediately as the strategy selected could be inappropriate or strategy execution misdirected (Kernan et al., 1994). Task complexity is also found to moderate the relationship between individual participation in the goal setting process and task performance. Individual participation is found to exert main effect on task performance in the case of complex tasks, but not for simple tasks (Campbell & Gingrich, 1986).

In goal setting research, groups were found to be less affected by the task complexity or availability of information on task strategy than individuals. This is attributed to the pooling of resources and effective coordination in groups even in the absence of information on task strategy. Independently working individuals benefited significantly from such information, especially in more complex tasks (Kernan et al., 1994). Group goals and task component complexity positively impact group performance by affecting the amount, quality, and timing of planning and effort within group (Weingart, 1992). Incidentally component complexity is a sub-dimension of task complexity and is indicative of the number of unique acts required to perform a task (Wood, 1986).

In Leader-Member exchange (LMX) literature task related characteristics such as role conflict, role ambiguity, and intrinsic task satisfaction were found to moderate the relationship between Leader-Member dyad exchange and subordinate performance (Dunegan et al., 2002).

In IS research, task complexity has been investigated for its effect on student programming task performance (Chang, 2005), effectiveness of different communication modes (Carey & Kacmar, 1997), and effectiveness of negotiation support systems (Delaney, Foroughi, & Perkins, 1997). In student programming projects, computer experience is found inversely related to computer anxiety of students, which in turn is found positively related to perception of task complexity (Chang, 2005). In software development projects task innovativeness has been found to moderate the relationship between team work quality and team efficiency relationship but not team work quality and team effectiveness relationship (Hoegl, Parboteeah, & Gemuenden, 2003). As the task complexity and the corresponding information load increases, face-to-face interactions are found to be superior to other communication modes such as teleconferencing, in terms of reduced errors of communication and improved satisfaction (Carey & Kacmar, 1997). In negotiation tasks it is shown that negotiation support systems (NSS) enhance outcomes and overcome some of the limitations of conventional face-to-face meetings (Delaney et al., 1997). However in complex negotiation tasks, electronic communication component is found to add little value over the decision support system (DSS) components with the result face-to-face groups outperform electronic meeting system groups (Jain & Solomon, 2000).

2.4.6 Individual versus Group Effectiveness

In several problem solving situations, the individual could be as effective as the group, and hence individual versus group comparisons of productivity are of interest in such tasks (Hare, 1995). A general finding of research on group versus individual effectiveness in problem-solving tasks is that groups are better than the average individual, but rarely better than the best individual (Hill, 1982). Researchers were constantly looking for tasks and situations where groups outperform individuals. Based on information processing view of groups (Hinsz, Tindale, & Vollrath, 1997), it is reasoned that groups outperform individuals in highly intellectual problem solving tasks with large information processing requirements. In such tasks groups typically benefit from the pooling of information and perspectives brought in by the various members of the group (Laughlin, Zander, Knievel, & Tan, 2003). Such process gains could result if there is "cognitive stimulation" or the group members have the capacity to learn (Hill, 1982). Group effect emanates from having large number of people to generate ideas, identify objects and remember facts. Hence groups could typically outperform individuals in tasks of low creativity but involving large information processing component (Kanekar, 1982). Some of the problem solving tasks where groups found to outperform individuals are letters to number problems (Laughlin, Bonner, & Miner, 2002; Laughlin *et al.*, 2003). In the case of spatial problems groups solved more problems than best members of statistical aggregate. In the case of anagrams, the evidence has been mixed (Faust, 1959)

Productivity of group may be conceived as determined by the most competent member, plus process gains due to "assembly bonus effects" (resulting from efficient group interaction) minus process losses (Collins & Guetzkow, 1964). Failing to identify and use the resources of capable group members is one source of group process loss (Kerr & Tindale, 2004). Assembly bonus effect is realized when group performance is better than the performance of any individual or any combination of individual member efforts. Such effects are generally modest. Any such claims need to be critically evaluated as they may underestimate group potential or overstate group achievement (Kerr & Tindale, 2004; Tindale & Larson, 1992).

The behaviors essential for realizing assembly bonus effects are pooling of information and integrating it to create solution. The information pooling effect becomes more prominent for difficult tasks. In easy tasks, one competent member often determines the performance. As the group task becomes more complex, the groups benefit from members correcting each other, and the more competent member could draw on the resources of other members in completing the task (Hill, 1982). For example when solving easy crossword puzzles, the number of successful groups is found to be proportional to the number of competent individuals in the group who could potentially solve the puzzle. However when solving more complex crossword puzzles, the number of successful groups is found to be greater than the proportion of successful individuals suggesting information pooling effect in groups (Shaw & Ashton, 1976). For problems involving multiple stages, there is a higher probability for the groups to have at least one member competent in solving each stage (Hill, 1982).

Unit of Analysis - Groups typically benefit from the aggregation of members, thus increasing the probability that there is at least one exceptionally competent member to be available within the group. Hence, groups should be compared with a statistical pooling of equal number of individuals or with the best member of such statistical aggregates (Hill, 1982).

Group vs. Average Individual Performance - In solving spatial and verbal problems, groups solved more problems than co-acting individuals (Faust, 1959). Groups performance is qualitatively superior in complex problem-solving (Schoner, Rose, & Hoyt, 1974) and when group learning is transferred to individual performance (Johnson, Johnson, & Scott, 1978).

Some factors found to facilitate group problem solving include having someone to talk to during problem solving (Durling & Schick, 1976). In general working with others may be more enjoyable (Garibaldi, 1979). In problem solving tasks requiring higher levels of creativity as in computer programming, groups found to be more efficient with difficult and complex tasks as groups are likely to have at least one excellent problem solver (Laughlin & Barth, 1981). Individuals with low motivation may become more highly motivated in situations facilitating group motivation for success (Zander, 1974) and when others provide high performance role models (Hare, 1995). Groups generally do better after participating in a group problem solving due to the experience with the task and learning from more highly skilled members (Goldman & Goldman, 1981; Laughlin & Adamopoulos, 1980). Groups make fewer errors as compared to nominal groups (Kanekar, Libby, Engels, & Jahn, 1978) possibly due to someone in the group catching the errors. There is some research suggestive of higher risk taking behavior in groups (Chlewinski, 1975; Forgas, 1981; Hashiguchi, 1974; Yinon, Jaffe, & Feshbach, 1975), while others report no such differences (Felsental, 1979).

Decision tasks – group may be no better than the best individual member (Miner, 1984)

Judgment tasks – groups report fewer but more accurate facts. In tasks requiring no division of labor, group productivity is less than that of individuals (Hill, 1982). A more statistical group effect explaining the higher accuracy of groups is that the average of a number of judgments is usually more accurate than that of one individual (Laughlin & Barth, 1981).

Brainstorming Tasks – A robust finding of this stream of research is that nominal groups with individuals working alone produce more ideas than interacting brainstorming groups (Mullen & Salas, 1991). The process losses in brainstorming groups are attributed to production blocking (inability of more than one person to talk or even think at the same time), evaluation apprehension, and convergence on a relatively low standard of performance due to social comparison effect in face-to-face groups (Larey & Paulus, 1995).

Complex tasks - Complex problems, similar to brainstorming tasks involve more than one acceptable or correct answer. Group performance generally tends to be superior to individual performance but not with respect to statistical pooling of responses. The quality of group solution is found to be superior across situations (Hill, 1982). In such tasks, group interaction tends to be dysfunctional during the idea generation phase, but is beneficial for integration of complementary information (Howell, Gettys, Martin, Nawrocki, & Johnston, 1970). In the evaluation phase group interaction is useful for clarification and justification. Discussion also improved solution accuracy compared to written feedback (Hill, 1982). Groups performance is found to be qualitatively superior to individual performance in complex problem-solving (Schoner et al., 1974) and when group learning is transferred to individual performance (Johnson et al., 1978). When the problem is more complex or subject to interpretation, the correct member could sometime be overruled by incorrect members (Faust, 1959), especially in groups of low ability. In case of complicated tasks, the group productivity could suffer due to difficulty in learning from group experience (Allison & Messick, 1985) or if other people interfere with the activity (Hare, 1995).

Group vs. the most competent member of a statistical aggregate – This stream of research produced mixed findings. Performance is found to be quantitatively similar in number completing crossword puzzles (Shaw & Ashton, 1976), and qualitatively similar in an executive decision making task (Schoner et al., 1974). Groups found to be better than the best member in terms of number of correct solutions in an anagram task (Faust, 1959), and worse in weather station morale problem (Fox & Lorge, 1962).

Group superiority over individuals in problem solving hinges on the demonstrability of the strategies, operations, and procedures that lead to the problem solution (Laughlin et al., 2003). This is consistent with the review findings that performance superiority of groups over individuals is highest in problem solving, intermediate in vocabulary tasks and lowest in world knowledge tasks (Hastie, 1986). These tasks have decreasing levels of solution demonstrability. Four conditions of demonstrability identified in literature are: a) availability of sufficient information; b) group consensus on a conceptual system; c) incorrect members being able to correct response when proposed; d) correct members have the sufficient time, ability and motivation to demonstrate the correct response to other members. With increasing demonstrability, problem-solving groups show distinctly superior performance over individuals. Research comparing group performance with equal number of individuals showed that groups perform at the level of the best individual member or best group member on problem solving tasks with highly demonstrable solutions as in mathematical, insight, and rule induction problems. On vocabulary tasks, analogies, and ranking tasks, groups perform at the level of the second best individual or group member. On tasks involving weakly demonstrable estimations of quantities, groups perform at the level of the average individual (Laughlin et al., 2002).

2.4.7 Social Facilitation

Social facilitation researchers have shown that in the presence of another individual, the performance on well-learned tasks is facilitated, while performance on novel or more complex tasks is hampered. The simple tasks where social facilitation effect was found include negotiating simple mazes (Hunt & Hillery, 1973), dressing in familiar clothes (Markus, 1978), fishing reel winding (Triplett, 1898) and copying simple material (Sanders & Baron, 1975). Some tasks where presence of others was found to hamper performance include solving difficult anagrams (Geen, 1977), recognition of novel stimuli (Cottrell, Wack, Sekerak, & Rittle, 1968), dressing in unfamiliar clothes (Markus, 1978), and negotiating difficult mazes (Hunt & Hillery, 1973). Based on these results it is argued that individuals perform better when working in groups on simple tasks, but perform worse on difficult tasks than do individuals working alone. Bond and Titus's meta-analysis of social facilitation literature suggested that the presence of others impairs both quantity and quality of performance in complex tasks (Bond & Titus, 1983).

The social facilitation effects are explained in terms of drive, evaluation apprehension, cognitive processes and other theories (Aiello & Douthitt, 2001). According to drive theory, working in groups increases drive. Increased drive facilitates performance in simple tasks as more dominant and well-learned responses are facilitated by the enhanced drive. When the dominant responses in situations are likely to lead to poor performance as in difficult tasks, then individuals perform poorly in drive state (Zajonc, 1965). Zajonc argued that though the significance of the presence of another individual may vary depending upon the situation and the behavior of the other, even mere presence is necessary and sufficient for social facilitation (Zajonc, 1980).

Based on social comparison theory Cottrell proposed that it is not the mere presence, but when individuals are concerned about how others may evaluate them that their drive levels could get elevated. Also prior evaluation experiences could help develop ‘learned drive’ among individuals as a drive reaction. Cottrell’s theory tries to account for social facilitation effects in animals through the ‘learned drive’ mechanism (Cottrell, 1972).

Based on self-presentation theory Baumeister proposed that in the presence of others, people with a desire to please others are motivated to present a certain public image of them. Using drive as a potential mechanism influencing performance, Baumeister suggested that presence of other individuals perceived as evaluative could trigger more drive than the presence of others who could not evaluate performance (Baumeister, 1982). Bond’s self-presentation theory could account for social facilitation effect without any recourse to the concept of drive. According to this explanation people make an attempt to appear competent to others. When working on simple tasks the impression management efforts of the individual could facilitate performance. While working on difficult tasks, the embarrassment from committing mistakes could impair performance (Bond, 1982).

Baron proposed a cognitive theory to explain social facilitation and argued that attention conflict in the presence of others could produce drive-like effect on performance, facilitating simple task performance, while impairing complex ones. The conflict is likely to be triggered when a) the distraction is hard to ignore or is very interesting, b) there is pressure to complete the task accurately and in time, c) it is hard or impossible to simultaneously attend to the task and the distracter. The conflict itself could result from both internal and external factors. The internal distractions could stem from ruminations about task performance. The theory therefore also accounts for the evaluation apprehension and self-presentation concerns. Performance may be facilitated up to a point by the distraction, beyond which it starts to deteriorate (Baron, 1986). Paulus’s Cognitive-motivational model (Paulus, 1983) and Sanna’s expectancy model (Sanna, 1992) are the other theoretical explanations offered for social facilitation. Figure 2.3 summarizes the main causes identified for the social facilitation effect.

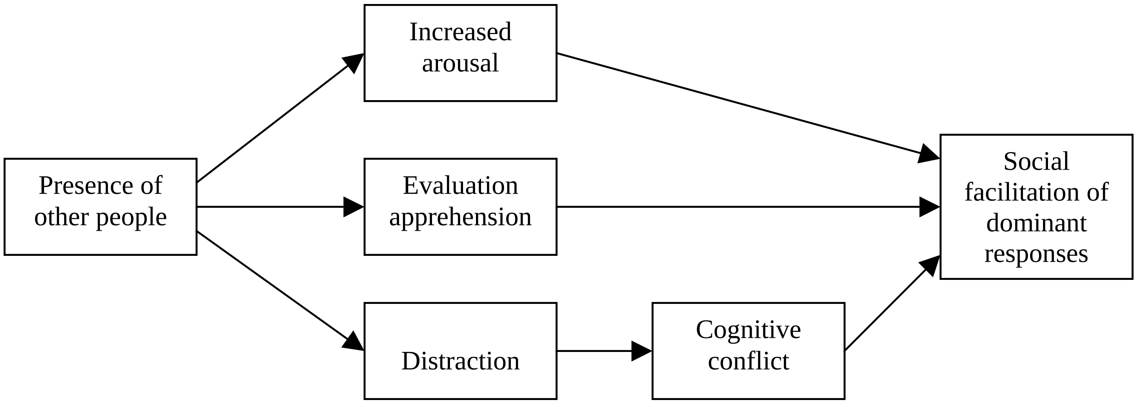


Figure 2.3 - Causes of Social Facilitation
Adapted from (Forsyth, 1999)

Jackson and Williams articulated that the condition of individual working alone in social loafing studies is equivalent to group condition in social facilitation studies due to the presence of other participants and the performance being monitored individually. Social facilitation groups are constructed to have higher drive levels to facilitate performance on simple tasks. Jackson and Williams reasoned that in social loafing condition, there is opposite effect in play with groups designed to have lower drive, resulting in poor performance on simple tasks. They argued that the tasks generally used in social loafing studies such as shouting, clapping, pumping air, rope pulling and the like are simple well learned tasks, in which the dominant response is to perform well. Hence, individual working alone condition of social loafing (resembling group condition of social facilitation) produces higher drive levels facilitating performance on easy well learned tasks generally used in such studies. Based on previous research, they argued that in uncertain or fearful situations, “working collectively is calming”

[(Jackson & Williams, 1985) p.938]. Thus working on difficult tasks in social loafing group condition, people might experience reduced drive levels than coworker individual condition. If one could relax working on a difficult task, then it should help enhance performance on these difficult tasks. In support of their assertions they found that participants in collaborative-difficult task condition, performed significantly better than coworker-difficult task condition (Figure 2.4)(Jackson & Williams, 1985).

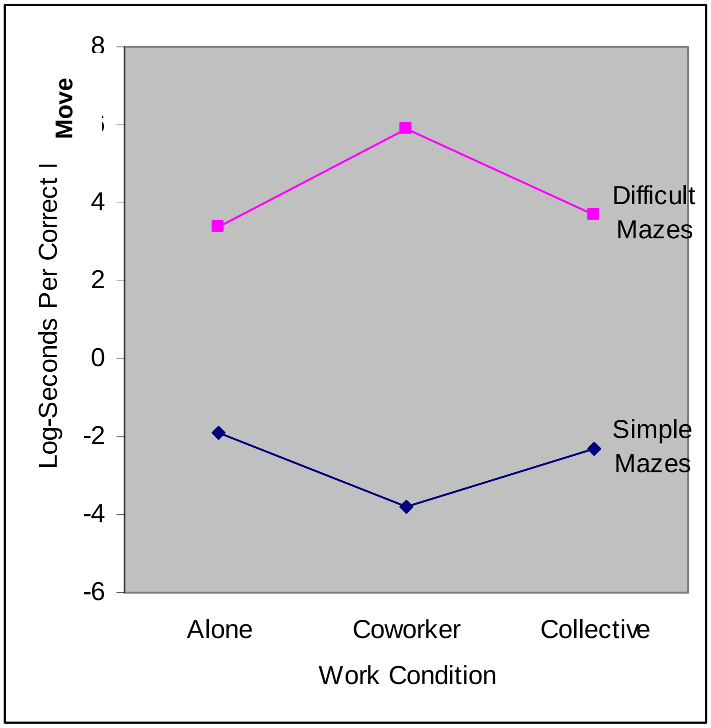


Figure 2.4 – Effects of Task Difficulty and Group Work Condition on Maze Performance Adapted from (Jackson & Williams, 1985)

While social loafing is typically viewed as a negative phenomenon, the reduced drive that contributes to social loafing also facilitates complex task performance and possibly reduces stress when working in collaboration condition. The presence of others acts to arouse drive when they are the sources of drive, but contributes to drive reduction if the others are also co targets (Jackson & Williams, 1985).

While social facilitation research has made enormous contributions to the understanding of the effects of social presence on individuals, there are also problems identified with the social facilitation theories. The boundary conditions for the various theories are not clearly delineated. Issues that require attention in this regard are: the kinds of people and relationships between them for which social facilitation predictions could hold; conceptual distinction between social facilitation and competition; explicit focus on both physical and virtual proximity of people; temporal variations in the effects of social presence; the need to look beyond output/quality performance measures such as citizenship or contextual performance. Assumptions in social facilitation theories concerning how drive leads to performance, how information processing and impression management are done need elucidation for empirical testing. The theoretical constructs such as drive/arousal, task difficulty, performance and other mediators also need better definitions and measures (Aiello & Douthitt, 2001).

2.4.8 Monitoring

The social facilitation effect is known to occur even when the observer is not physically present or visible as in electronic performance monitoring (Aiello & Kolb, 1995). Computer monitoring has been associated with some impairment in complex task performance, higher stress and less satisfaction among monitored individuals. The ability to control monitoring is however associated with no impairment in performance (Aiello & Svec, 1993; Stanton & Barnes-Farrell, 1996). Some factors found to affect the satisfaction levels of monitored employees are the way monitoring is used, feedback is provided, and the consideration shown by the supervisors (Chalykoff & Kochan, 1989). Satisfaction of monitored individuals is found related to perceived control over monitoring but not actual control (Stanton & Barnes-Farrell, 1996). Some individual difference variables such as negative affectivity (Douthitt & Aiello, 2000), and locus of control (Kolb & Aiello, 1996) were found to moderate the relationship between computer monitoring and outcomes such as stress and satisfaction. In one study low negative affectivity individuals reported no difference in satisfaction between monitoring and no monitoring conditions. Monitoring also found to affect higher ability participants unless they have personal control over it (Douthitt & Aiello, 2000). In another study internal locus of control individuals felt more stress when monitored while external locus of control individuals experienced more stress when not monitored (Kolb & Aiello, 1996). It is also argued that national culture dimensions such as individualism/collectivism, uncertainty avoidance and others moderate the relationship between electronic monitoring and outcomes (Panina & Aiello, 2005 (In press)).

2.4.9 Social Loafing

Studies in small group performance have found that when individuals work in groups collectively on relatively simple tasks, they exert less effort in comparison to the situations when they work individually. Reduced risks of evaluation, opportunity to free ride on others' efforts, and unwillingness to shoulder the work of a capable, free-riding member of the group are some of the psychological mechanisms underlying social loafing (Kerr & Tindale, 2004).

Social loafing is found to occur when there is a higher possibility of redundant effort (Harkins & Petty, 1982), lack of cohesiveness within group (Williams, Harkins, & Latane, 1981) and when there is reduced responsibility for the final outcome (Petty, Harkins, Williams, & Latane, 1977). Social loafing can be reduced where individuals are made aware that their output in the group work is identifiable (Williams et al., 1981), when task difficulty or challenge involved is high which results in the perception that they could make unique contribution to the group effort (Harkins & Petty, 1982; Jackson & Williams, 1985) or by giving each subject a different task to perform (Harkins & Petty, 1982). While working on a collective task reduces drive to exert effort, in difficult tasks where increased drive is not conducive to performance, working collectively improves performance. For example, while working on maze problems individuals were found to perform better working alone on simple tasks, but working collaboratively on complex tasks. In situations involving working with a partner but independently, performance was found to be between these two limits (Jackson & Williams, 1985).

The social loafing research has adopted the approach of isolating individual conditions under which social loafing could be minimized, rather than looking for an overall theory to explain the phenomenon. The theoretical perspectives and concepts used to explain social loafing are: social impact (Latane, 1981), arousal reduction (Jackson & Williams, 1985), evaluation potential (Harkins, 1987; Williams *et al.*, 1981), dispensability of effort (Kerr & Brunn, 1983), matching of effort (Jackson & Harkins, 1985b), and self-attention (Mullen, 1983).

According to social impact theory, individuals in a social situation may be viewed as either as source or targets of social impact. The extent of social impact experienced by an individual is a function of the strength, immediacy, and number of sources and targets of social impact (Latane, 1981). In the group condition of social loafing studies, the experimenter urging the subjects to try as hard as possible may be considered as the source of social impact with the subjects being the targets. The impact of the experimenter is divided among the several target subjects thus resulting in reduced effort with increase in the size of group (Karau & Williams, 1993a).

Jackson and Williams indicated that working in a group is drive reducing when other individuals are not sources of social impact but co-targets of social impact. In social facilitation studies the presence of others is considered as drive inducing, as others are the sources of social impact. Thus reduced drive while working in groups contributes to social loafing. However, it is shown that reduced drive experienced in group working, while contributing to social loafing on simple tasks may in fact facilitate performance in novel and difficult tasks (Jackson & Williams, 1985).

One explanation for social loafing in groups is lack of evaluation of individual output so that members can “hide in the crowd” (Davis, 1969). People realize that they may not get fair share of credit or blame for group performance. In many situations making individuals’ collective inputs verifiable to anyone including oneself may be sufficient to eliminate social loafing. However two requirements need to be satisfied for evaluation to be possible from any source – the individual’s output should be known or identifiable and there should be a standard - objective, social or personal, available for comparison (Harkins, 1987; Harkins & Jackson, 1985; Harkins & Szymanski, 1989).

Social loafing may also be attributed to members' feeling that their contributions are not essential to the group performance. In threshold tasks using disjunctive rule where group succeeds if any of the group members reaches some performance criteria, group members tend to reduce collective effort. Even when the individual contributions of members is made available to themselves, other members, and the experimenter, this reduction in effort is not affected (Karau & Williams, 1993a; Kerr & Brunn, 1983).

In groups people expect others to slack off and hence reduce their efforts to maintain equity (Jackson & Harkins, 1985a). The job attitudes research also suggests that workers' perceptions and motivations towards task are influenced by the task assessments of their peers. (Zalesny & Ford, 1990). When individuals expect their co-workers to perform poorly on a meaningful task, they may even increase their effort in what is called as social compensation effect (Williams & Karau, 1991).

Another explanation for social loafing is in terms of self-attention (Mullen, 1983). According to this perspective, individuals experience reduction in self-awareness while working in groups. Thus some of the self-regulatory processes involving salient performance standards may be disregarded in group condition. Individuals when working alone are more attentive to such task demands and performance standards. There is however not enough empirical support for this explanation (Karau & Williams, 1993a).

Some of the factors moderating social loafing effect are evaluation potential, expectation of co-worker performance, task meaningfulness, and culture. (Williams & Karau, 1991), personal involvement (Brickner, Harkins, & Ostrom, 1986), and increasing the instrumentality of individual members' contribution (Shepperd & Taylor, 1999). Need for cognition is one individual differences variables found to moderate the effect of social loafing (Smith, Kerr, Markus, & Stasson, 2001).

Social compensation research has demonstrated that individuals may increase effort and work harder collectively than individually to compensate for the expected poor performance of other group members. When paired with a group member who is believed to exert low effort, group members work harder when the partner has low abilities, but typically loaf when the partner has high abilities (Hart, Bridgett, & Karau, 2001).

2.4.10 Group Motivational Gains

People intuitively expect some motivational gains to occur in group work, though a vast majority of studies have reported of motivational losses as in social loafing. Some studies have however demonstrated motivational gains in certain conditions of collective working as in social compensation (Williams & Karau, 1991) and Koehler effect (Witte, 1989).

When individuals expect their co-workers to perform poorly on a meaningful task, they may increase their effort in what is called as social compensation effect (Williams & Karau, 1991). When paired with a group member who is believed to exert low effort, group members work harder when the partner has low abilities, but typically loaf when the partner has high abilities (Hart et al., 2001).

Kohler demonstrated motivational gains in certain conjunctive tasks where its weakest member drives performance of the group. Using a physical endurance task, Kohler showed that when members of the dyad have moderate difference in ability (not too similar or dissimilar), they performed better as a pair compared to their expected individual performance (Witte, 1989). The motivational gains were found to result mainly from the weaker member (Stroebe, Diehl, & Abakoumkin, 1996).

While social compensation effect is attributed to greater effort put in by the more able partner, Kohler effect is credited to the motivational gains of the less able partner. When there is ability discrepancy within a dyad, which of these two effects is likely to result is contingent on the perceived instrumentality of individual effort to the group performance. Social compensation effect is likely to result in an additive task, when the higher ability partner is likely to work harder to compensate for the low ability partner. Kohler effect is likely to occur in a conjunctive task where the contribution of the low ability member holds the key to the group performance (Williams, Harkins, & Karau, 2003).

2.4.11 Satisfaction

Satisfaction is defined as “the difference between the amount of rewards workers receive and the amount they believe they should receive” (Robbins, 1998). Locke conceptualizes job satisfaction as “the positive emotional response to a job resulting from attaining what the employee wants and values from the job” (Locke, 1976). Locke argued that satisfaction and dissatisfaction are value responses resulting from an individual’s appraisal of an object or a situation against certain standard that is considered desirable or beneficial. Values however differ in terms of the content (what is valued) and intensity (how much is desired). Task related values include task activity, and task success or achievement. Some individuals derive value in doing a task just for the sake of the task, even in the absence of extrinsic rewards. Such tasks would be perceived as ‘interesting’ and carry intrinsic value for them. Also when individuals set a particular goal on a task and are successful in achieving it, it is experienced as a pleasurable feeling. Conversely, failing to achieve the goal could be experienced as unpleasant (Locke, 1970).

Organizational research has identified several factors that enhance job satisfaction of individuals in work settings. Proposing an integrated model of work motivation, Locke and Latham have categorized factors that contribute to satisfaction/dissatisfaction in terms of values and personality, work characteristics, organizational policies, and performance outcomes. Work characteristics such as mental challenge are positively related to individual satisfaction. Also organizational policies based on procedural justice, perceived fairness of resulting outcomes, and distributive justice contribute positively to individual satisfaction (Locke & Latham, 2004). Job characteristics theory identified five task characteristics, if when present could lead to higher satisfaction. These factors are: personal significance, variety, responsibility and autonomy, feedback, and identity (that is having a complete part of the work) (Hackman, Oldham, Janson, & Purdy, 1975).

Satisfaction and performance are different but somewhat correlated concepts. The job satisfaction-performance relationship has been one of the most ‘venerable’ research areas in industrial-organizational psychology (Judge, Thoresen, Bono, & Patton, 2001). Locke suggested that satisfaction should primarily be considered as a consequence of performance and only indirectly as a predictor of performance (Locke, 1970). Despite a strong general belief among lay people that happy employees are more productive employees (Fisher, 2003), the findings of several meta analytic studies are suggestive of only a modest correlation between job satisfaction and job performance (Iaffaldano & Muchinsky, 1985; Judge *et al.*, 2001; Petty, McGee, & Cavender, 1984). One previous meta-analysis based on a 16 studies by Petty and his colleagues has shown a correlation of 0.31 between satisfaction and performance (Petty et al., 1984). Another meta-analysis based on 217 correlations from 74 studies found a substantial range in the correlation between the facets of satisfaction and performance (Iaffaldano & Muchinsky, 1985). Judge and his colleagues based on a comprehensive review of literature have argued that job performance and satisfaction have a reciprocal relationship. They proposed an integrative model suggesting several mediators and moderators of this relationship. The proposed mediating variables include success and achievement, task specific self-efficacy, goal progress and positive mood. The moderator variables suggested are need for achievement, performance-rewards contingency, job characteristics, work centrality, and aggregation (Judge et al., 2001). Based on value theory (Locke, 1970) task complexity is another potential factor that could moderate the relationship between performance and satisfaction, as performance in complex tasks may satisfy many of the individuals' values for intrinsic fulfillment.

Organizational researchers have noted that groups could hold the key to employee satisfaction. In general group work is expected to lead to higher satisfaction among the individuals and contribute to higher quality work. This is attributed to reduced competition and enhanced cooperation among coworkers in group work situations (Campion, Medsker, & Higgs, 1993). Thus organizations that use teams and work groups are expected to generate higher satisfaction for their members (Forsyth, 1999).

Small group research has identified several factors contributing to satisfaction of members of groups. Shaw and his colleagues have demonstrated that the satisfaction and performance of group members is dependent on task interdependence, reward interdependence, and individual's preference for group work (Shaw, Duffy, & Stark, 2000). Task interdependence indicates the degree to which group members interact and rely on each other to accomplish work and is found related to satisfaction (Campion et al., 1993). Reward interdependence denotes the extent to which an individual's reward is dependent upon the performance of other group members. Higher levels of reward interdependence denoting 'community of fate' (Besser, 1995) perspective of reward structure is expected to contribute to higher group satisfaction (Wageman, 1995). Group based reward structures are expected to reduce competition and increase cooperative group effort (DeMatteo, Eby, & Sundstrom, 1998b). For example in dyads, reward interdependence is found to be related to higher motivation and higher performance

(Hom & Berger, 1994). Preference for group work denotes the extent to which an individual prefers group work to autonomous work. This is a sub dimension of the broader construct of individualism-collectivism (Wagner & Moch, 1986). Individuals with higher preference for group work are likely to be more satisfied, while individuals preferring autonomous working are likely to be dissatisfied in group working (Wagner & Moch, 1986).

Goal setting research also has shed some light on the conditions that contribute to individual satisfaction in work settings. According to goal setting theory, difficult goals are more motivating and lead to superior performance than easy or ‘do best’ goals through their effect on direction and persistence of effort. Goals serve as the benchmarks for satisfaction/dissatisfaction. In any given attempt, exceeding the goal enhances satisfaction to the extent of the positive discrepancy. Conversely, the negative discrepancy resulting from not achieving the goal leads to dissatisfaction Also the effect of goals on performance is moderated by goal commitment and feedback (Locke & Latham, 1990). Goal orientation and task difficulty are known to interact in determining performance satisfaction of individuals. When working with learning goal orientation, individuals found to express higher performance satisfaction in complex tasks than in simple tasks. However, while working with performance goal orientation, individuals are found to express higher performance satisfaction in simple tasks than in complex tasks (Steele-Johnson, Beauregard, Hoover, & Schmidt, 2000).

Some meta-analytic studies (O'Leary-Kelly, Martocchio, & Frink, 1994) have demonstrated the existence of goal setting effect in groups as well. Compared to autonomous individuals, group members display higher goal commitment and more positive attitudes towards goal attainment and report higher satisfaction with their performance (Hinsz & Nickell, 2004). Also, group members tend to be more satisfied with their performance even when their performance does not differ from that of individuals (Hinsz, 1995).

There is a general tendency in group literature suggesting that group members react positively to working in a group compared to working alone (Hinsz & Nickell, 2004). One reason articulated for this is that groups fulfill some of the social and emotional needs of individuals (Levine & Moreland, 1998). Individuals may expect group work to be an enjoyable experience though this may not always be the case (Barker, 1993). Social identity theory also suggests that if group members feel attached to the group, then they start to identify themselves with the group and derive some self-esteem from group activities. Brainstorming research has shown that members may have more positive attitude towards group work as they may have a perception of group superiority in productivity, though this could be illusory (Paulus, Dzindolet, Poletes, & Camacho, 1993).

2.4.12 Confidence

Confidence is the "strength of a person’s belief that a specific statement is the best or most accurate response” (Peterson & Pitz, 1988). Confidence reflects the fit and coherence of the rationale that people construct for their beliefs (Koehler, 1991). Groups typically report higher confidence in their performance than do individuals (Sniezek, 1992; Stephenson & Wagner, 1989). This is explained through ‘rational construction view'. According to this view "as people organize their thoughts and articulate a coherent rationale for their choices during interaction, the increased organization and coherence of their case makes them more confident in their decisions." (Heath & Gonzalez, 1995).

Interactive decision-making is "the procedure where individuals consult with others but make their final decision alone" (Heath & Gonzalez, 1995). A consistent finding in sports prediction and risky shift dilemmas is that interaction increases people's confidence in their decisions. Group interaction provides this opportunity to develop a more coherent rationale for their choices (Heath & Gonzalez, 1995).

The correlation between confidence and accuracy is found to be highly dependent upon the task. Confidence and accuracy (performance) tend to be highly correlated for intellective tasks, while being weakly correlated for judgmental tasks. It is reasoned that in judgment tasks, it is difficult to be sure of the accuracy of a judgmental response and to convince others of it (Zarnoth & Sniezek, 1997). Whether a task is judgmental or intellective is contingent upon the demonstrability of the solution. When the solution demonstrability is high the tasks fall into the intellectual spectrum and when it is weak, the tasks fit into the judgmental domain. Demonstrability of the task is a function of not only the task but also of the group members and the decision environment. The same task could be judgmental to low-ability members under time pressure while it could be intellective for high ability members with infinite time.

Realism is another construct used in judging the confidence ratings of groups and individuals. Realism is considered good when the probability assigned to a set of answers being correct is same as the proportion of correct answers. Realism measured as over confidence is found to be less in pairs than in autonomously working individuals (Allwood & Granhag, 1996).

CHAPTER 3

HYPOTHESES DEVELOPMENT

This chapter focuses on hypotheses development and the conceptual models of the study.

3.1 Research Questions

This study examines the following research questions relating to the effect of mode of participation in software development and task complexity on programming outcomes:

Whether programming done individually or in pairs has an effect on programming outcomes?

Whether programming done individually or in pairs has an effect on the programmers' mental model developed during task performance

Whether task complexity moderates the effect of programming method (individual versus pair) on task mental model and programming outcomes?

Figure 3.1 showcases the research model.

3.2 Nature of Programming Tasks

Software development typically involves problem analysis, design and implementation, debugging and testing. However an implementation task in general involves comprehending design, generating alternatives, choosing appropriate logic of implementation, and developing code to the correct semantics and syntax of the programming language. Compiling, debugging and testing help make the program work as per design specifications. However during actual implementation all these stages are tackled more iteratively than sequentially.

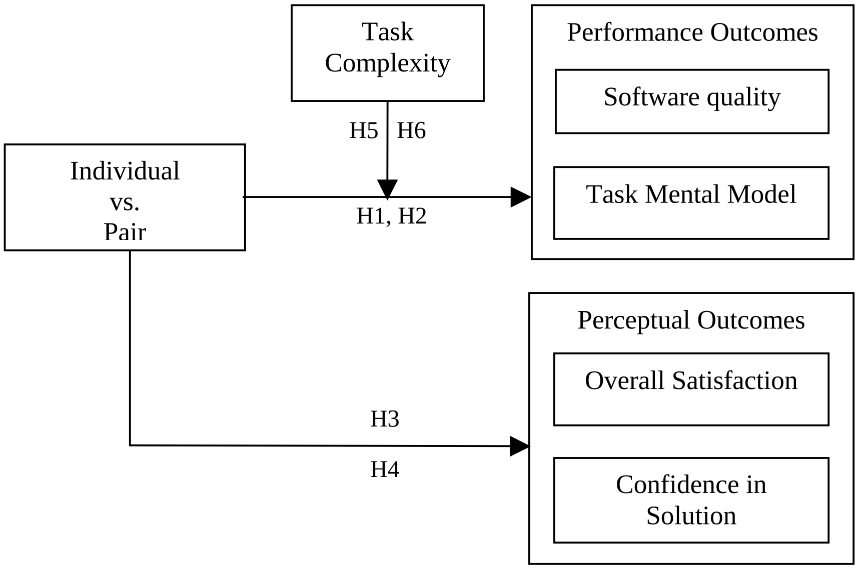


Figure 3.1 – Research Model

An implementation task in collaborative programming may be considered as an intellective problem-solving task of the McGrath's task typology. Programming tasks are intellective tasks to the extent that they have a correct solution, though they may not be very compelling. Some cognitive conflict as in McGrath's cognitive conflict tasks (McGrath, 1984) may also result while generating and choosing the program implementation logic, debugging, and error correction. In terms of Steiner's typology (Steiner, 1972) programming tasks are unitary (not divisible and result in a single outcome or solution), disjunctive (involve choosing from different individual judgments and approaches, but members need to agree on a single solution), and optimizing (emphasis on quality over quantity). When programming is done by pairs, once a partner figures out the logic and syntax of solving the problem and implements it, then the group has solved the problem. In terms of creativity, systems analysis and design tasks are highly creative, while coding and implementation tasks involve less creativity in comparison. However during debugging, novel thinking is required to understand and fix bugs.

Programming tasks are highly intellective tasks requiring high information processing. For example, problem solving during coding task in any object-oriented programming domain involves identifying the attributes and behaviors of various objects, the relationships between them, data and control flow between different objects for accomplishing required behaviors. The developer has to remember and simultaneously keep track of program logic, data and control flow, and the signatures of various methods in terms of the input and output parameters, and their data types.

In the next few sections various hypotheses concerning the effectiveness of pair versus individual programming are derived.

3.3 Effect of Individual versus Pair Programming on Software Quality

Performance effectiveness of individuals versus groups on problem solving tasks was investigated in social loafing, social facilitation, and individual versus group effectiveness research. Social loafing literature argues that there are motivational losses involved when individuals work in groups on relatively simple tasks. This results in individuals exerting less effort in groups as against working individually. When programming is done collaboratively by pairs, there could be social loafing due to higher possibility of redundant effort (Harkins & Petty, 1982), reduced responsibility for the final outcome (Petty et al., 1977), especially when low coworker effort is expected (Kerr, 1983). However social loafing effects in general are found to be moderate in magnitude though generalizable across tasks and subject populations (Karau & Williams, 1993b). Social loafing is likely to be less or eliminated when tasks are perceived as high in meaningfulness or personal involvement (Brickner et al., 1986), and when group size is small (Kerr & Brunn, 1983). Programming task in general could be considered as meaningful as it is a cognitive problem solving activity and programmers typically work on tasks that are part of software projects. In the experimental setting involving student subjects, task may be considered as meaningful if the problem domain is intrinsically interesting and related to the course content of the students. There is some evaluation potential for the task performance in the pair context as the partner is continually aware of the effort put in by the individual. According to social impact theory (Latane, 1981), social loafing is directly proportional to the size of the group. The group size of two in the pair-programming context also suggests that social loafing even when present could be modest. Hence we expect social loafing effects to be minimal or present to a modest degree.

Social facilitation literature argues that when an individual is working on a task, mere presence of another individual causes arousal and facilitates dominant responses while hampering less dominant ones. Thus there is facilitation effect in simple task performance, while there is impairment in complex task performance (Bond & Titus, 1983).

We argue that programming task by its very nature is much more complex than the simple tasks where social facilitation effects have been reported in small group research such as dressing in familiar clothes (Markus, 1978), fishing reel winding, and copying simple material (Sanders & Baron, 1975). There may be simple subtasks involved during programming that are comparable to the simple tasks used in social facilitation studies. For an experienced programmer some well learned tasks could be compiling the program, basic debugging involving correction of typographical errors or syntax errors, reusing code from an existing implementation, reusing objects, creating some routine methods such as set and get methods that initialize or set values for attributes or return stored values of attributes. The dominant response while performing such well-learned tasks is to do well. However most code implementation tasks involve some aspects where even experienced programmers have to think through the problem, decide on a particular logic path and carry out implementation. They may however take less time to think through and implement code as against novices or students learning programming. But, in any other meaningful programming task performance, the number of such simple subtasks is few and far between. Even the simplest of programming problems have some path multiplicity so that it could be implemented in multiple ways. On most meaningful programming tasks, programmers cannot go by the most dominant responses, but need to reflect and do some exploration to implement the tasks. The difficulty inherent in even simple programming tasks is therefore comparable to that of difficult anagrams (Geen, 1977), recognition of novel stimuli (Cottrell et al., 1968), and negotiating difficult mazes (Hunt & Hillery, 1973), though they may have a few simple subtasks. So, it is argued that except for some trivial tasks, it is reasonable to expect programmers to encounter some novel situations. In such complex tasks, social facilitation studies have shown that the mere presence of others impairs performance due to higher drive levels (Zajonc, 1980).

Jackson and Williams have shown that the presence of other individuals could be drive reducing if they are not perceived as sources of social impact (evaluators of their performance), but as co-targets of social impact (Jackson & Williams, 1985). The presence of other individual in the pair-programming context is not evaluative presence as in social facilitation studies, but as a co-target of the social impact. Also the pair frequently switches roles during the programming task performance increasing the level of collaboration and the likelihood of perceiving the partner as a co-target rather than cause of social impact. Hence based on Jackson and Williams (Jackson & Williams, 1985) we argue that presence of others is drive reducing and facilitates task performance in the context of pair programming, where the task is intrinsically complex.

A robust finding of small group research is that performance in groups is better than the average individual performance but rarely better than the best individual performance (Hill, 1982). Group effect in general is attributed to having large number of people to generate ideas, identify objects and remember facts. Hence groups could typically outperform individuals in tasks of low creativity but involving large information processing component (Kanekar, 1982). Based on information processing view of groups (Hinsz et al., 1997) groups are expected to outperform individuals in highly intellectual problem solving tasks with large information processing requirements. In such tasks groups typically benefit from the pooling of information and perspectives brought in by the various members of the group (Laughlin et al., 2003). Such process gains could result if there is “cognitive stimulation” or the group members have the capacity to learn (Hill, 1982).

As brought out in the previous section, programming is an intellective problem-solving task involving huge information processing component. Two programmers working collaboratively on the programming task could benefit from pooling of ideas, information, and perspectives brought in by each of them. Pair programming in agile development involves one programmer actively doing coding at the keyboard as driver, while the other programmer actively inspects the code and acts as the navigator. We expect the ‘assembly bonus effect’ (Collins & Guetzkow, 1964) in the pair programming condition due to the nature of programming task and the process of collaborative working stipulated in Extreme Programming. Programming task has a demonstrably correct answer, though may not be a very compelling one. We argue that the four conditions of solution demonstrability articulated in the literature as underpinning superiority of groups over individuals (Laughlin et al., 2003) are available to a sufficient degree in the pair programming context. Specifically, the pairs working collaboratively on the programming task are expected to have sufficient information concerning the problem. The partners could typically arrive at a consensus on a conceptual system leading to the solution. Also we expect the member to have abilities, sufficient time, and motivation to demonstrate the correct solution to their partners. The wrong members also should be able to identify the correct approach when identified by their partners. As program is compiled and debugged several times during the development process, it contributes immensely to the demonstrability of strategies, operations, and procedures leading to the problem solution. Hence programming tasks have high demonstrability for the groups to be able to outperform best individuals. Previous IS studies in collaborative programming provide anecdotal evidence suggestive of software quality of pairs to be higher than individual programming (Williams, 2000). Thus we expect programming pairs to outperform best and second best individuals in the individual condition in terms of software quality.

H1a - While working on a programming task, performance in terms of software quality of a collaborating pair is higher than the performance of the best programmer in a nominal pair

H1b - While working on a programming task, performance in terms of software quality of a collaborating pair is higher than the second-best programmer in a nominal pair

3.4 Effect of Individual versus Pair Programming on Programmers' Task Mental Model

Mental models are internal representations people use while dealing with the environment (Van der Veer & Melguizo, 2003). They are an amalgamation of the given information that can be acquired from a situation or a context along with the prior knowledge of the person retrieved from the long term memory (Schraw & Nietfeld, 2003). For example mental models may be used during learning to understand the working of a computer system or software. They may also be used in problem solving as in performing a novel task or debugging an error (Carroll & Olson, 1988). Stern (Stern, 1993) articulated episodic situation model (Reusser, 1990) and problem model (Riley & Greeno, 1988) as two situation models that help problem solvers understand and represent the problem in the context of word problem solving. While problem situation model enables understanding of specific problem context, the problem model includes only the structural and relational information directly relevant to the problem. It is represented by abstracting data or elements such as names, objects, actions or intentions of actors from the problem situation. The problem model drives the search and identification of appropriate mathematical model in the case of word problem solving tasks (Stern, 1993).

Similar to the problem model (Riley & Greeno, 1988), we conceptualize task mental model as including the structural and relational information relevant to the programming task. It represents the programmer's understanding of the relationships between various objects, attributes and behaviors (methods) of the problem task. The task mental model drives the search for appropriate programming solution, which is reflected in the software quality. While the software quality represents the final implementation based on the particular language semantics and syntax, task mental model represents the instantiated knowledge structures (Wilson & Rutherford, 1989) facilitating such a solution. While a programmer could have a correct task mental model, this may result in higher software quality only if the programmer is able to translate it into the particular language domain.

Based on information processing we expect more alternatives to be explored in the pair context. In learning tasks involving computer systems, a collaborating pair is found to develop mental models with higher inference potential than self-discovering individuals. While self-discovering individuals tend to focus on surface structures of the system, co-discovering individuals were found to have a better understanding of the link between physical actions and goals (Lim, Ward, & Benbasat, 1997). The notion of social construction of knowledge (Vygotsky, 1978) also argues that when individuals interact with their peers, they constantly encounter information that is not consistent with their existing beliefs or ideas. While working on a task collaboratively to generate a unified outcome, people will have to reconcile their differences in ideas and perspectives. This may involve adopting their own ideas, or abandoning them in favor of others' ideas or through an appropriate synthesis of multiple ideas and perspectives. These interactions involve reorganization of their knowledge structures through filling in gaps, adding details, and correcting misunderstandings (King, 1989). This facilitates enhancement of their mental models.

Based on the theory of distributed cognition, a collaborative pair of programmers and the artifacts involved in the task performance may be conceived as a complex cognitive system. Some observed characteristics of such a system include searching through larger space of alternatives, shared memory for old plans, and joint production of ambiguous plan segments (Flor & Hutchins, 1991). Active communication processes involving perspective taking and perspective making are considered crucial for the effectiveness of such a system (Boland & Tenkasi, 1995). When programming is done individually, such dynamics are missing. Hence we expect the ‘assembly bonus’ effects to occur for the pair condition in terms of superior task mental model over the individual condition. Thus we hypothesize that

H2a - While working on a programming task, the task mental model of the best programmer of a collaborating pair is better than the task mental model of the best programmer in a nominal pair

H2b – While working on a programming task, the task mental model of the best programmer of a collaborating pair is better than the task mental model of the second-best programmer in a nominal pair

H2c – While working on a programming task, the task mental model of the second-best programmer of a collaborating pair is better than the task mental model of the second-best programmer in a nominal pair

3.5 Effect of Individual versus Pair Programming on Programmers’ Overall Satisfaction

In programming tasks programmers often have to introspect and deliberate about the choices to be made at each stage. While programming with a partner, the partner could serve as a sounding board for the ideas and approaches to be used. In problem solving tasks talking to a partner is generally found to be helpful (Durling & Schick, 1976). Small group research suggests that group work in general is more enjoyable (Garibaldi, 1979) and is therefore expected to lead to higher satisfaction among the individuals. This is attributed to reduced competition and enhanced cooperation among coworkers facilitated by group work (Campion et al., 1993). It is reported that group members tend to be more satisfied with their performance even though their performance did not differ from that of individuals (Hinsz, 1995). Task and reward interdependence in group tasks is also found to lead to higher satisfaction (Shaw et al., 2000). Programming tasks when done collaboratively involve high task interdependency. In terms of Steiner's group task typology (Steiner, 1972), programming tasks are unitary, disjunctive and optimizing. Also programming in pairs involves high reward interdependence as Extreme Programming (XP) stipulates joint code ownership and team based rewards. Group based reward structures are expected to reduce competition and increase cooperative group effort (DeMatteo, Eby, & Sundstrom, 1998a) and lead to higher satisfaction (Shaw et al., 2000).

Preference for group work is one individual difference variable that could impact satisfaction/dissatisfaction of group members. Individuals with higher preference for group work are likely to be more satisfied, while individuals preferring autonomous working are likely to be dissatisfied in group working (Wagner & Moch, 1986). This is however not measured in the present study and could moderate the level of satisfaction reported by individuals in the pair condition. However there is a general tendency in group literature suggesting that group members react positively to working in a group compared to working alone (Hinsz & Nickell, 2004). One reason articulated for this is that groups fulfill some of the social and emotional needs of individuals (Levine & Moreland, 1998). Individuals may expect group work to be an enjoyable experience though this may not always be the case (Barker, 1993). Social identity theory also suggests that if group members feel attached to the group, then they start to identify themselves with the group and derive some self-esteem from group activities. Brainstorming research has shown that members may have more positive attitude towards group work as they may have a perception of group superiority in productivity, though this could be illusory (Paulus et al., 1993).

The higher satisfaction in the pair condition could also result from superior performance. Satisfaction and performance are known to have a reciprocal relationship (Judge et al., 2001). Several meta analytic studies have reported of a modest correlation between satisfaction and performance (Iaffaldano & Muchinsky, 1985; Judge *et al.*, 2001; Petty *et al.*, 1984). As argued in previous sections, we expect the pair performance to be higher than the best individual performance. So, we expect the higher performance also to contribute to higher satisfaction in the pair condition.

Group goal setting literature suggests that compared to autonomous individuals, group members tend to have higher goal commitment and more positive attitudes towards goal attainment and report higher satisfaction with their performance (Hinsz & Nickell, 2004). Previous pair programming studies in IS also report of higher satisfaction of programmers in the pair condition over the individual condition (Nosek, 1998; Williams, 2000). Consistent with the hypotheses derived earlier comparing the performance of best programmers in the collaborating pairs and nominal pairs, we expect best programmers in the collaborative pair to have higher satisfaction than best programmers in the nominal pairs. We also expect the second-best programmer in the collaborating pair to report higher overall satisfaction than the second-best individual programmer.

H3a – While working on a programming task, the overall satisfaction of the best programmer in a collaborating pair is higher than the overall satisfaction of the best programmer in a nominal pair

H3b – While working on a programming task, the overall satisfaction of the best programmer in a collaborating pair is higher than the overall satisfaction of the second-best programmer in a nominal pair

H3c – While working on a programming task, the overall satisfaction of the second-best programmer in a collaborating pair is higher than the overall satisfaction of the second-best programmer in a nominal pair

3.6 Effect of Individual versus Pair Programming on Programmers' Confidence in Solution

Based on Peterson and Pitz (Peterson & Pitz, 1988) confidence in solution is defined as the strength of a programmer's belief that the software solution produced is the best or most accurate. Groups are known to typically report higher confidence in their performance than do individuals (Sniezek, 1992; Stephenson & Wagner, 1989). A consistent finding in sports prediction and risky shift dilemmas is that interaction increases people's confidence in their decisions. Based on 'rational construction view' it is said that group interaction facilitates individuals to organize and articulate a coherent rationale for their thoughts resulting in higher confidence in their decisions (Heath & Gonzalez, 1995).

In the pair-programming context, we expect similar dynamics to work enhancing the confidence of collaborating pairs over that of programmers working individually. The interaction facilitated by pair programming is expected to help bring in new information relating to the problem. The interaction also helps programmer to better organize and articulate a coherent rationale for the various programming decisions they undertake during code development. Due to the better rationale programmers develop for their beliefs about the software solution due to interaction, we expect programmers to report higher confidence in their solution in the group context than when working individually. Consistent with the previous hypotheses that compare the best and second-best programmers in the collaborating pairs and nominal pairs, we expect that

H4a - While working on a programming task, the confidence in solution of the best programmer in a collaborating pair is higher than the confidence in solution of the best programmer in a nominal pair

H4b - While working on a programming task, the confidence in solution of the best programmer in a collaborating pair is higher than the confidence in solution of the second-best programmer in a nominal pair

H4c - While working on a programming task, the confidence in solution of the second-best programmer in a collaborating pair is higher than the confidence in solution of the second-best programmer in a nominal pair

3.7 Moderating Effect of Task Complexity on Programming Outcomes

When two programmers work on a programming task of low complexity, the perception of redundant effort among the programmers is likely to be higher as each individual may assume that other person should be able to finish the task without much help. Lack of identifiability of individual’s contribution, reduced responsibility for the final outcome, group based reward system as in XP, all contribute to some social loafing (Karau & Williams, 1993b). As the complexity of task increases, the perception of meaningfulness of task among group members increases. When the programming pair work on a complex task, social loafing is likely to be less due to the perception of members that their contributions are essential to the group performance. So, social loafing is likely to be less when the task is more complex than when it is less complex.

As brought out earlier, most non-trivial programming tasks have certain components that are inherently complex, especially due to the multiplicity of solution paths. In most practical programming situations, there are often some simple subtasks that are well learned, similar to the simple tasks of social facilitation studies. But, most other subtasks involve some level of task complexity, and involve some new learning similar to the complex tasks used in social facilitation studies. Based on Jackson and Williams (Jackson & Williams, 1985) we argue that working in the pair condition will be drive reducing. The presence of the partner would be perceived less as an evaluative presence, but more as a co-target of the social impact of the supervisor. Hence performance on complex tasks is facilitated but not hampered in the pair condition. As the complexity of task increases, the effect of reduction in drive in the pair condition is likely to get accentuated resulting in better understanding of the task domain and improved software quality compared to the individual condition.

Group effect emanates from pooling of information and ideas, group memory of things, and identification of objects and patterns by multiple members of the group (Hinsz et al., 1997). The behaviors essential for realizing assembly bonus effects are pooling of information and integrating it to create solution. The information pooling effect becomes more prominent for difficult tasks. In easy tasks one competent member often determines performance. As the group task becomes more complex, the groups benefit from members correcting each other, and the more competent member could draw on the resources of other members in completing the task (Hill, 1982). As the task complexity increases, there is a higher probability for the pair to have at least one programmer competent in solving each stage of the problem. As the complexity of task increases, programmers are also likely to experience higher 'cognitive stimulation.' In tasks involving higher 'cognitive stimulation' groups are expected to realize 'assembly bonus effect' and outperform individuals (Hill, 1982). When working on a complex task, programming pair would be able to use the information better, explore more alternatives, correct each other, and have higher probability of having the necessary competency to solve the problem within team. Hence we expect the pair superiority over best individual performance to get amplified with increasing task complexity.

H5 – While working on a programming task, the difference in performance in terms of software quality between a collaborating pair and the best programmer *in a nominal pair is higher for tasks of high complexity than for tasks of low complexity*

As the complexity of task increases, the information-processing requirements, in terms of both domain information and problem solving information, increase. Compared to independent programmers, programming pairs working on more complex tasks are able to benefit more from the higher information processing abilities available within the dyad. Research on individual information processing suggests that as task complexity increases, there is an increase in the number of sources accessed, and in the share of information gathered from general purpose sources by individuals (Bystrom & Jarvelin, 1995). In groups, task complexity is associated with greater participation in decision-making and higher boundary spanning activities by the members (Ito & Peterson, 1986).

Based on distributed cognition theory, the programming pair and the artifacts involved in task performance may be conceptualized as a complex cognitive system. Such systems are characterized by shared memory of old plans, ability to search through larger space of alternatives, and joint creation of ambiguous or more complex code segments (Flor & Hutchins, 1991). When two programmers work collaboratively on a more complex task as in XP, the programmer acting as the driver concentrates on coding, while the navigating programmer is able to look for errors, explore the problem domain more strategically and understand the relationships between the objects and methods involved in the programming task. Based on distribution cognition theory (Flor & Hutchins, 1991), mental model theory (Johnson-Laird, 2001) and information processing view (Hinsz et al., 1997), we expect the group effect in terms of superior task mental model of the best programmer in the collaborating pair over the best programmer in the individual condition, to get amplified in the difficult task condition. Thus, due to the reasons articulated above, we expect programming pairs to explore the task domain to a greater extent and develop much superior task mental model than programmers in individual condition while working on a task of high complexity than on a task of low complexity.

H6 – While working on a programming task, the difference between the task mental models of best programmer in a collaborating pair and the best programmer in a nominal pair is higher for tasks of high complexity than for tasks of low complexity.

In the next chapter research design, definition of constructs and their measures, experimental controls and related validity issues are discussed.

CHAPTER 4

RESEARCH METHODOLOGY

In this chapter the methodology, research setting, sample size, manipulations and measurement issues are discussed.

4.1 Methodology

A laboratory experiment was conducted to examine the effects of individual versus pair programming and task complexity on the programming outcomes. A laboratory experiment was chosen for the study so as to achieve a high degree of control over extraneous factors likely to affect the relationships of interest. Improved control leads to higher internal validity of the study, enabling more explicit causal attributions to be made with a higher degree of confidence. To understand the dynamics involved in pair programming, experimental study was considered as the most appropriate methodology in view of the controls afforded (Pedhazur & Pedhazur-Schmelkin, 1991).

This experiment attempted to simulate software developers undertaking programming tasks. Two methodological settings for programming were contemplated:

1. Individual programming
2. Pair programming

Two task complexity levels were also contemplated:

1. Low complexity
2. High complexity

Four dependent variables were proposed to be measured.

1. Software quality
2. Task mental model
3. Overall Satisfaction
4. Confidence in Solution

Computers loaded with Java JDK5 (current Java platform) and WordPad were provided to simulate the Java development environment to all the subject units. The computers were also provided with installed Java documentation. No Internet access was provided from these computers to prevent access to the online Java resources and program solutions. Along with the task description, some minimal documentation help on syntax for the Java classes relevant to the problem was provided as instructions. The dependent variable measurements, manipulation checks and demographic data were obtained, at the end of the experiment.

4.2 Subjects

The subjects were undergraduate and graduate students who were enrolled in Information Systems courses in the College of Business, University of Texas at Arlington. The subjects were required to have the knowledge of Java I, the first level course in object-oriented programming. The subjects earned class credit for their participation. The instructors provided alternative course assignments to the students who were not willing to participate in the experiment or had already participated in earlier semesters. Subjects were recruited from programming related courses with the approval of the instructors. Subjects were also asked to read and sign an Informed Consent Form upon sign-up. As the experiment involved human subjects, prior approval was obtained for the research protocol from the Institutional Review Board (IRB) through the Office of Research Compliance, University of Texas at Arlington.

4.3 Experimental Setting

The setting was quiet cubicles in the behavioral research lab in the college of business. All the cubicles were provided with stand alone laptop computers loaded with identical versions of the needed software. Subjects could not see other computer screens or other subjects, except their partners in the case of pair programming. The subjects in the individual condition were not allowed to talk or otherwise communicate with other subjects or outsiders. The subjects in the pair condition were not allowed to talk or communicate with other subjects or outsiders except with their own programming partners.

4.4 Planned Sample

The sample size for each of the four treatment conditions was planned to be about 30 subjects. In the two pair conditions, there would be 15 pairs or 30 subjects each. In the two individual programming conditions, there would be 30 subjects each. The approximate total sample size was expected to be 120 (4x30). Each experimental condition was run with a maximum of 9 individuals at a time and all treatment conditions were planned for every session. As the availability of students from different courses in each semester was not expected to be large, experimental sessions were planned across multiple semesters. A total of 33 sessions were planned across three semesters (Spring, Summer and Fall 2005). To the extent feasible, it was attempted to balance the subjects in different treatments for each semester.

4.5 Design

A two (methodology) by two (task complexity) fully factorial design was used for the study. The two methodological settings were (1) Individual programming and (2) Pair programming. The two conditions of task complexity were (1) Low complexity and (2) High complexity.

		Task Complexity	
		Low	High
Methodology	Individual	Condition One	Condition Two
	Pair	Condition Three	Condition Four

Figure 4.1 – Experimental Treatments

4.6 Experimental Task

Initially, subjects were told that they are participating in an experiment involving application development in Java. They were told that they would be working individually or collaboratively with a partner. Subjects who turned up for different experimental sessions were randomly assigned to one of the four treatment conditions. To the extent possible, it was attempted in every session to have subjects in all treatment conditions. All the treatment groups were informed that the experiment consisted of two sessions – first session of 15 minutes duration involved working on a warm up task; the second session of 2-hour duration involved working on the experimental task. They were then requested to go to the cubicle randomly assigned to them, close the door and read the instructions provided therein. For all treatment groups written instructions were provided for logging into the computer, and for compiling Java programs developed using Notepad.

For treatment groups three and four, involving programming in pairs, written instructions were provided on pair programming and how they should work collaboratively. Instructions also included how one of them would be working as driver typing with the keyboard, while the partner acted as navigator helping the driver. They were also instructed to transfer the keyboard to the partner and switch their roles of driver and navigator at frequent intervals. They were told to make the role switch every 10 to 15 minutes. The experimenter visited them every 15 minutes to remind them to switch and maintained a log of the visits.

For all the treatment subjects, similar instructions were provided for a small warm up programming task. They were asked to do the coding in Java. The subjects were told that it was a warm up task and the purpose of the trial was to familiarize them with the JDK environment and the Notepad application for doing coding. This would give them a chance to recollect some of the Java language syntax. They were also informed that the code developed would not be graded or used for experimental purposes. Additionally, in case of treatments involving paired subjects, they were appraised that the trial helps them to get familiarized with the partner and ways of collaboratively working on the problem task.

Stopwatches were used in all the cubicles to time the beginning and end of the warm up task and the main experimental task. The students requiring to pick up refreshments or go to restroom were allowed to do so with the consent of the experimenter. Care was taken to see that subjects did not come in contact with other subjects except with their own partner during such recess breaks. Stopwatches were paused during such breaks to provide full time of 2 hours for the experimental task. If they were able to complete the task ahead of time, the experimenter noted the completion time. If they were not able to complete the task during the assigned time, their work was still graded to the extent completed based on quality of the code.

Finally, at the end of task completion, they were provided with paper-based questionnaires to fill out individually. The total duration of the experiment was three hours.

4.7 Subject Compensation

Subjects earned class credit for their participation. To increase their motivation, they were informed that they would be considered for a lottery of three prizes each of $50 value to be drawn and announced after completion of the whole experiment. To motivate them further, they were informed that whatever grade (software quality) they make on the coding task, equal amount of money in terms of cents would be provided by the experimenter to a charity. After finishing filling in the questionnaire, all subjects were debriefed about the experiment and the research objectives. The program developed by different treatment subjects was independently graded by two doctoral GTAs who were not directly involved with the experiment. The resultant scores were normalized for different treatment conditions and the instructors were informed for giving course credit to the students. However, original scores were used for experimental purposes.

4.8 Response Variable Measurements

Measurements for various constructs are provided in Appendices. The response variables are discussed below.

4.8.1 Software Quality

Software quality represents the objective task performance of the individual or the pair on the programming task. Two doctoral GTAs who are experienced in grading student programming solutions in Java courses independently evaluated the software developed by the subjects in the experiment on a scale of 1-125. A common grading scheme was developed in consultation with a faculty expert who taught programming courses. While grading the pilot test tasks, the two graders grading approaches were calibrated with that of a faculty expert and the experimenter. Separate scoring sheets were created and refined while grading the pilot test tasks. The same grading sheets were later used by the two doctoral GTAs to grade experimental tasks of subjects. Where large discrepancies were noticed, they were requested to reconcile the scores to prevent any errors in grading. To reduce any possible sources of bias, the average of the scores of the two raters was used as a measure of the software quality. The grading sheets used for grading the two experimental tasks are provided as Appendix J.

4.8.2 Task Mental Model

Mental models are the internal representations of objects, people, situations, or actions. They are built based on experience and observation, of the particular entity of interest or of the world in general (Langan-Fox et al., 2004). The task mental model as defined here is the level of similarity of programmer's internal representation of the task domain, based on experience and observation during task performance, with that of an expert's mental model. Mental model is a hypothetical construct specific to the task. For the programming tasks involved in the present study, the various classes and the methods involved in the problem domain and the programmer's understanding of the relationship between these task components represents the programmer's mental model of the task. An expert's mental model of the task domain represents the benchmark for the understanding of the task domain. So, the level of similarity of programmer's task mental model with the expert's mental model objectively indicates the quality of task mental model of the programmer. This was measured by eliciting programmer's rating of subjective similarities/relationships between the various components of the task domain. This can be diagrammatically represented using visual modeling tools such as pathfinder (Schvaneveldt, 1990).

Pathfinder (PF) is a technique that generates suitable psychological scaling with regard to the underlying structure between concepts. The raw paired comparison ratings elicited from individuals is taken as input by the pathfinder algorithm and transformed into a network structure. In the network, the nodes represent the concepts, while the links indicate the relatedness of concepts. The NETSIM function in PF was used to compare the similarity of two networks. The advantage of PF over other representation techniques such as Multidimensional Scaling (MDS) is that PF provides index of similarity that could be used as a variable in further analysis. PF is a popular tool used to represent and analyze a wide range of cognitive structures in several research domains such as training, and in studies of expertise and human-computer interaction (Langan-Fox, Code, & Langfield-Smith, 2000; Mohammed, Klimoski, & Rentsch, 2000; Schvaneveldt, 1990).

In the present study, the task mental model score for a programmer was measured as the similarity index between the PF networks of the programmer and the expert. Two IS faculty members who teach programming and other technical courses were requested to be the experts. First the two experts were requested to independently provide what they considered as important objects and methods involved in the programming task. The two experts then reconciled minor differences between their lists and a consensual list was compiled separately for each of the experimental tasks. For each task (low task complexity, high task complexity), the experts were then requested to first independently rate the similarities between different objects and methods involved therein and then discuss and come up with single rating scores. The PF networks representing the consensual similarity rating of the two experts were considered as the expert task mental models for the two experimental tasks (tasks of high and low task complexity). The PF networks of individual subjects were compared with that of the expert using NETSIM function in the Pathfinder software. A task mental model score of 1 represented perfect similarity with the expert task mental model, while a score of 0 represented no similarity. Higher this score, higher the task mental model of the programmer, reflecting the programmer's understanding of the task domain. Task mental model was measured in both individual and pair conditions at the individual level of the programmer. For the simple and complex tasks used in the present study, the questionnaires used for eliciting similarity ratings are provided in Appendix H.

4.8.3 Satisfaction and Confidence in Solution

Satisfaction represents affective response of the individual to the overall task performance. The measure for this variable has been adapted from (Bhattacherjee, 2001). Confidence in solution is the strength of a programmer's belief that the software solution produced is the best or most accurate. The measure for this variable was designed based on existing literature.

4.8.4 Manipulation Checks

For the four treatments administered in the study, manipulation check were done using measures indicated in Appendix I.

4.8.5 General Questions

The questionnaire for eliciting general demographic information such as age, number of years of programming experience, number of programming languages known is provided in Appendix I.

4.8.6 Programming Ability

Apart from the two manipulations, programming ability is a very important variable likely to impact the performance of the subjects. To control for this effect, the programming ability was measured and used as a covariate in the ANCOVA/MANCOVA analysis. The programming ability was measured as the average GPA of each of the subjects in all the previous Information Systems (IS) courses taken at the UTA with the following weights: Grades in programming and analysis and design courses were given twice the weight as other IS courses. This information was obtained from the UTA records and consent of the subjects for using this information was obtained as a part of Informed Consent.

4.8.7 Course Credit

The subjects for the experiment were drawn from students enrolled in multiple IS courses during three semesters. All the instructors involved were requested to provide a uniform course credit of 5% of the grade to control for any changes in the motivation levels of subjects across semesters.

4.9 Debriefing

Debriefing was done for all the four treatment groups at the end of the experiment. Debriefing form used for the experimental subjects is provided as Appendix C.

4.10 Statistical Analysis

In planning this study, ANCOVA and MANCOVA were the anticipated statistical methods of analysis to be used. The rationale for conducting two separate statistical analyses is discussed in the next chapter. Hypotheses 1 and 5 that involved comparison between collaborating pair and the best, second-best programmers in nominal pairs were tested using ANCOVA analysis. Hypotheses 2-4, and 6 that involved comparison between the best, second-best programmers in collaborating pairs and nominal pairs were tested using MANCOVA analysis. After assumption check and testing the overall significance of the model using MANCOVA, individual ANOVAs were run to test hypotheses including the interaction effects.

4.11 Pilot Testing

A pilot test of the experiment was conducted prior to the main experiment where all four-treatment conditions were tested. A sample of 2 subjects in each of the treatments involving pairs and a sample of 1 subject for treatments involving individual programming (total 6 subjects) were used for the pilot study. Changes required in the scripts and logistics of the experiment were fixed at this stage.

CHAPTER 5

RESEARCH RESULTS

The results of preliminary analyses, hypotheses testing and posttest questions are presented in this chapter.

5.1 Preliminary Analyses

This section provides the data and analyses regarding sample characteristics, reliability of dependent measures, preliminary tests and methods of analysis.

5.1.1 Sample Characteristics

A total of 122 subjects participated in the experiments. Two subjects were dropped from the study to balance the subjects across the four treatments. These subjects either did not meaningfully complete the task or did not complete the dependent variable measurements. The mean (standard deviation) age of the experimental subjects was 27.50 (6.39) years and the gender composition was 89 (74%) males and 31 (26%) females. In terms of academic level, 99 (82.5%) subjects were undergraduate students, while 21 (17.5%) were graduate students. In terms of nationality, 78 (66.7%) were US citizens with 39 (33.3%) being citizens from other countries. Subjects' self-reported programming experience in any language was as follows: 0-1 years - 44 (37.9%) subjects, 1-2 years - 30 (25.9%) subjects, 2-4 years – 20 subjects (17.2%), and > 4 years – 22 (19.0%). In terms of self-reported programming experience in Java, the distribution was as follows: 0-1 years - 85 (73.3%) subjects, 1-2 years - 24 (20.7%) subjects, 2-4 years – 6 (5.2%) subjects, and > 4 years – 1 (0.9%) subject. The experiment was conducted over three semesters and the distribution of subjects across the three semesters was as follows: Spring – 49 (40.8%), Summer - 25 (20.8%), and Fall 46 (38.3%). Figure 5.1 shows the distribution of subjects across the different treatment conditions.

		Task Complexity	
		Low	High
Methodology	Individual	Condition (1,1) n = 30[a]	Condition (1,2) n = 30[b]
	Pair	Condition (2,1) n = 30 (15 pairs)	Condition (2,2) n = 30 (15 pairs)

Total Sample size (N) – 120

a – one subject dropped from the sample for not meaningfully completing the task

b – one subject dropped from the sample for not meaningfully completing the task and for not completing the dependent variable measurement

Figure 5.1 – Distribution of Subjects Across Different Treatment Conditions

Two sets of analyses were done for testing various hypotheses involving constructs defined at the team level and individual level. Software quality was conceived as a team level construct for the collaborating pairs. Hence a 3 x 2 ANCOVA design was used for analysis of the dependent measure of Software Quality with GPA as a covariate. Figure 5.2 shows the distribution of subjects for the ANCOVA design.

The best and second-best programmers in nominal pairs were determined based on the software quality scores achieved by them.

Individual or Group	Task Complexity Low (1)	 High (2)
Best in Nominal Pair (1)	Condition (1,1) n = 15	Condition (2,1) n = 15
Second-best in Nominal Pair (2)	Condition (2,1) n = 15	Condition (2,2) n = 15
Collaborating Pair (3)	Condition (3,1) n = 15 pairs	Condition (3,2) n = 15 pairs

Figure 5.2 – 3 x 2 ANCOVA Design for Analysis of Dependent Measure of Software Quality

Task Mental Model, Overall Satisfaction, and Confidence in Solution were conceived as individual level constructs for individuals working in both nominal pairs and collaborating pairs. Hence a 4 x 2 MANCOVA Design was used for analysis of these dependent measures with GPA as a covariate. The best, second-best programmers in nominal pairs and collaborating pairs for the MANCOVA analysis were determined based on the task mental model scores achieved by them. As software quality was measured at the team level for collaborating pairs, it could not be used for determining the best, second-best among the collaborating pairs. Figure 5.2 shows the distribution of subjects for the MANCOVA design.

		Task Complexity	
		Low (1)	High (2)
Person	Best in Nominal Pair (1)	Condition (1,1) n = 15	Condition (2,1) n = 15
	Second-best in Nominal Pair (2)	Condition (2,1) n = 15	Condition (2,2) n = 15
	Best in Collaborating Pair (3)	Condition (3,1) n = 15	Condition (3,2) n = 15
	Second-best in Collaborating Pair (4)	Condition (4,1) n = 15	Condition (4,2) n = 15

Figure 5.3 – 4 x 2 MANCOVA Design for Analysis of Dependent Measures at the Individual Level

5.1.2 Factor Structure

Three perceptual variables were examined in this study: (1) overall satisfaction, (2) performance assessment and (3) confidence in solution. Overall satisfaction was measured by four questionnaire items, performance assessment by one item and confidence in solution by two items. Table 1 is a correlation matrix of the items measuring the three perceptual variables along with their means and standard deviations. The distribution of items was as follows: items 1-4 measured overall satisfaction: items 5 measured performance assessment; items 6-7 measured confidence in solution. The questionnaire used is provided as Appendix I. In addition there was one performance measure for software quality. This was taken as the average of two scores independently assessed by two doctoral GTA graders. Another dependent variable task mental model was assessed as a questionnaire based measure. Software quality is a group level construct in the pair condition while other measures were individual measures in both individual and pair conditions.

Table 5.1 – Correlation Matrix for the Perceptual Measure Items

	1	2	3	4	5	6	7
1	4.08 (1.723)						
2	0.896	4.13 (1.695)					
3	0.717	0.747	3.79 (1.789)				
4	0.741	0.779	0.814	4.13 (1.573)			
5	0.595	0.529	0.521	0.610	5.59 (2.760)		
6	0.620	0.576	0.562	0.596	0.871	3.92 (1.903)	
7	0.620	0.572	0.589	0.592	0.865	0.972	3.87 (1.904)

Exploratory factor analyses on the perceptual measures with orthogonal (Varimax) rotation revealed that the questionnaire item measuring performance assessment (item 5) and the two items measuring confidence in solution were highly correlated and load into one factor (Table 5.2). The four items indicative of overall satisfaction load into a separate factor. As performance assessment was not found to be distinct from confidence in the solution, this item is combined with the two items measuring confidence in solution. However the scales for items measuring performance assessment and confidence in solution were different. Item for performance assessment (item 5) was measured on 1-10 Likert scale while the two items for Confidence in solution (items 6 and 7) were measured on 1-7 Likert scale. A summated scale was created for confidence in solution by adding the standardized individual items and then again standardizing the resultant summated variable. The resultant *confidence in solution* variable had a mean of 0 and SD of 1.

Table 5.2 – Varimax Rotated Orthogonal Factor Loadings

Questionnaire Item	Factor 1 Overall Satisfaction	Factor 2 Confidence in Solution	Communality Estimate h_i^2
1	0.843	0.362	0.843
2	0.897	0.280	0.883
3	0.843	0.300	0.801
4	0.839	0.351	0.827
5	0.323	0.886	0.889
6	0.341	0.919	0.961
7	0.349	0.914	0.956
Eigen Value λ_j	5.112	1.048	
Cum. Variance Explained	73.03%	88.00%	

Rotated factor scores better reflect the orthogonality of the rotated variables. Summated scales using factor scores may be created and used if correlation in the items is likely to adversely impact subsequent analysis. However summated scales using item scores may be preferred if generalizability of questionnaire items is of interest (Hair, Anderson, Tatham, & Black, 1998). The summated item scores for overall satisfaction and summated standardized item scores for confidence in solution were used in the present analysis in the interest of generalizability of questionnaire items.

5.1.3 Reliability of Dependent Measures

Table 5.3 shows the correlation matrix of the dependent measures at the individual level of analysis along with the means (standard deviation). Reliability of measures in terms of Cronbach's alpha was found to be 0.934 and 0.945 for overall satisfaction and confidence in solution respectively. Cronbach's value is an estimate of the internal consistency or homogeneity of the variable items in measuring a given construct (Kerlinger, 1986). According to Nunnally, reliabilities exceeding 0.70 are considered acceptable (Nunnally, 1978).

Table 5.3 – Correlation Matrix of Dependent Measures Measured at Individual Level

	Task Mental Model	Satisfaction	Confidence
Task Mental Model	.409 (0.128)		
Satisfaction	.05	4.03 (1.55)	
Confidence	.149	.657*	0.018 (0.980)

Task mental model was measured as the similarity score between the PF networks of the subject and the expert. This measure was computed using the Pathfinder software. Similar to software quality, this was computed as one score. While the reliability of attitudinal measures containing multiple items could be judged from Cronbach's alpha values, no such computational measures are available to judge the reliability of task mental model.

A power analysis was also conducted on the experimental results. The power/effect sizes of the statistical results at an alpha level of 0.05 are tabulated in Table 5.4. The effect size was estimated by the non-centrality parameter. An effect size of 1.0 is considered small, 2.0 as medium and 3.0 as large (Neter, Kutner, Nachtsheim, & Wasserman, 1996). Determining the adequacy of power levels for detecting differences in dependent variables is highly subjective with no clear guidelines. The power levels presented here appear to be sufficiently strong to detect important differences between treatment conditions where they exist. The observed power tabulated here is based on two-tailed tests provided by statistical packages. For a given sample size and effect size, power is typically higher for one-tailed tests over two-tailed tests. As all the hypotheses in the study were set up for one-tailed tests, the observed power levels reported here are quite conservative.

5.1.4 Test for the Effects of Demographic Factors

The effect of various demographic variables on the dependent measures was tested using individual one-way ANOVAs. This was done to identify factors that could be used as blocking variables to help reduce error variance in the dependent measures.

Table 5.4 – Power Analysis

Dependent Variable	Parameter	Effect Size (Non-centrality Parameter)	Power*
Software Quality	I1 – P	2.565	0.718
	I2 – P	1.979	0.382
	C1 – C2	2.969	0.835
	I1*C1 – I1*C2	1.862	0.452
	I2*C1 – I2*C2	0.847	0.133
Task Mental Model	I1 – P2	6.315	1.000
	I2 – P2	0.967	0.160
	P1 – P2	4.171	0.985
	C1 – C2	4.222	0.987
	I1*C1 – I1*C2	2.427	0.672
	I2*C1 – I2*C2	0.659	0.100
	P1*C1 – P1*C2	0.947	0.155
Overall Satisfaction	I1 – P2	0.713	0.109
	I2 – P2	2.065	0.534
	P1 – P2	0.333	0.063
	C1 – C2	1.677	0.383
	I1*C1 – I1*C2	2.155	0.569
	I2*C1 – I2*C2	0.238	0.056
	P1*C1 – P1*C2	1.075	0.187
Confidence in Solution	I1 – P2	0.229	0.056
	I2 – P2	1.998	0.221
	P1 – P2	0.721	0.110
	C1 – C2	2.443	0.677
	I1*C1 – I1*C2	3.286	0.902
	I2*C1 – I2*C2	0.303	0.060
	P1*C1 – P1*C2	1.255	0.237

Legend:
I1 – Best individual in Nominal Pair
I2 – Second best Individual in Nominal Pair
P – Pair Condition
P1 – Best Individual in Collaborating Pair
P2 – Second-best Individual in Collaborating Pair
C1 – Low Complexity task
C2 – High Complexity task

*computed at alpha = 0.05

As software quality was measured at the group level for collaborative pairs and demographic variables could not be aggregated to the group level, ANOVA testing was not done on this dependent measure. Table 5.5 summarizes the results of one-way

ANOVA tests for the various demographic factors on the dependent measures of task mental model, overall satisfaction, and confidence in solution.

It is evident from the ANOVA test results in Table 5.5 that subjects' gender, citizenship and experience in Java did not have any significant effect on the dependent measures. Also there were no significant differences noticed in the dependent measures across semesters. In terms of academic level of subjects, ANOVA analysis identified significant differences between undergraduate and graduate student subjects for satisfaction and confidence. However when academic level of subjects was used as a blocking factor in the MANCOVA analysis, the effect size (non-centrality parameter) and power of the tests for the main effects was adversely affected possibly due to reduction in the degrees of freedom. As graduate students constituted a small percentage (17.3%) of the subject pool, when academic level was used as a blocking factor, the sample size in the treatment blocks within the graduate student subjects was too small for any meaningful interpretation. Since there was no net benefit to be gained, academic level of subjects was not included as a blocking factor.

In terms of programming experience of subjects, ANOVA analysis identified significant differences on this factor for task mental model but not for overall satisfaction or confidence in solution. Self-reported programming experience was measured on four levels. Again, when used in the MANCOVA analysis as a blocking factor, the effect size and power for main effects were adversely affected due to reduction in the degrees of freedom. As there was no net-benefit to be gained, programming experience was not included as a blocking factor.

Table 5.5 – One-way ANOVA Results on Demographic Factors

Factor	N	Task Mental Model			Satisfaction			Confidence		
		Mean	SD	F	Mean	SD	F	Mean	SD	F
				(p-value)			(p-value)			(p-value)
Gender				0.315			2.871			0.481
Male	88	0.405	0.131	(0.576)	4.170	1.428	(0.093)	0.037	0.943	(0.489)
Female	31	0.420	0.120		3.630	1.820		-0.108	1.160	
Semester				0.330			0.913			0.415
Spring	49	0.404	0.137	(0.719)	4.220	1.656	(0.404)	-0.055	1.050	(0.661)
Summer	25	0.428	0.143		3.710	1.518		0.162	1.027	
Fall	46	0.404	0.112		4.010	1.452		-0.030	0.942	
Academic Level				2.269			6.597			7.205
Undergrad	99	0.401	0.128	(0.135)	3.870	1.479	(0.011*)	-0.110	0.961	(0.008*)
Grad	21	0.447	0.127		4.830	1.688		0.519	1.039	
Citizenship				1.178			3.210			3.881
US	78	-0.161	0.947	(0.280)	3.800	1.466	(0.076)	-0.161	0.946	(0.051)
Others	39	0.214	1.023		4.340	1.616		0.214	1.024	
Programming Experience				2.84			0.941			1.122
0 to 1 years	44	0.380	0.116	(0.041*)	3.790	1.540	(0.423)	-0.178	0.984	(0.343)
1 to 2 years	30	0.440	0.117		3.880	1.481		0.060	0.880	
2 to 4 years	20	0.453	0.146		4.140	1.390		-0.117	1.096	
> 4 years	22	0.371	0.138		4.420	1.696		0.260	0.975	
Experience in Java				1.237			2.082			1.352
0 to 1 years	85	0.401	0.128	(0.300)	3.910	1.470	(0.107)	-0.061	0.977	(0.261)
1 to 2 years	24	0.434	0.134		3.970	1.597		-0.097	0.946	
2 to 4 years	6	0.414	0.111		5.420	1.744		0.681	1.044	
> 4 years	1	0.207			2.750			0.766		

** significant at alpha =0.05*

5.1.5 Test for Interactions - ANCOVA and MANCOVA

As discussed earlier, in the pair condition, software quality was measured at the group level while other dependent measures (task mental model, overall satisfaction and confidence in solution) were all measured at the individual level. GPA of students on a continuous scale of 1-4 was used as a covariate.

The test for significance of effects was done using 3 (best individual, second-best individual, collaborating pair) x 2 (low task complexity – high task complexity) ANCOVA and 4 (person – best, second-best in nominal pair and collaborating pair) x 2 (low task complexity – high task complexity) MANCOVA. ANCOVA analysis was used to compare software quality of collaborating pair with the software quality of individuals in the nominal pairs, while MANCOVA analysis was used to compare other dependent measures between individuals in collaborating pair and nominal pair conditions.

The purpose of these initial tests was to check for significant interactions. If initial tests reveal significant interactions, cell means model should be used to examine the differences between treatment means (Neter et al., 1996). Table 5.6 summarizes the results. The table shows the F-ratios (p-values) for the main effects and interaction effects. The interaction term for software quality, task mental model, and overall satisfaction were not significant at alpha = 0.05. The interaction term for confidence in solution was however found to be highly significant (p = 0.006). Hence confidence in solution was separately examined later using ANCOVA cell means model with 8 levels representing the 8 treatment conditions in the research design matrix.

Table 5.6 – Results of Tests for Interactions Showing F (p) Values

Dependent Variable	Individual or Group	Task Complexity	Interaction
Software Quality	12.727 (0.000*)	8.548 (0.004)*	1.737 (0.182)

Dependent Variable	Person	Task Complexity	Interaction
Task Mental Model	20.765 (0.000*)	32.561 (0.000*)	2.118 (0.103)
Overall Satisfaction	4.577 (0.005*)	0.889 (0.348)	1.920 (0.131)
Confidence in Solution	4.081 (0.009*)	2.326 (0.130)	4.447 (0.006*)

*Significant at alpha = 0.05

5.1.6 Test of Significance- ANCOVA and MANCOVA

GPA of students, which was used as a covariate, was collected at the individual level. For ANCOVA analysis, average GPA for the pair was used as the covariate. For ANCOVA analysis, the best and second best individuals in nominal pairs were determined by the software quality of their solutions. Results of the 3 (best individual, second best individual, pair) by 2 (low task complexity, high task complexity) ANCOVA analysis on the software quality indicated a significant main effect for the best, second-best individuals, and pair (that is individual or pair condition) with p value of 0.000. The ANCOVA analysis also revealed a significant main effect for the task complexity dimension with a p value of 0.004. The interaction between the two main factors was however not found to be significant (p = 0.182). Table 5.7 summarizes the 2-way ANOVA results for software quality and other dependent measures.

Table 5.7– One-Way ANOVA Results on Software Quality

A. Individual or Pair

	Individual or Pair						F Value	Sig p value
	Best Individual in nominal pair		Second-best Individual in nominal pair		Collaborating Pair			
	Mean	SD	Mean	SD	Mean	SD		
Software Quality	75.000	22.927	42.45	23.371	61.483	31.644	12.727	0.000*

B. Task Complexity

	Task Complexity				F Value	Sig p value
	Low		High			
	Mean	SD	Mean	SD		
Software Quality	66.267	27.825	53.022	29.447	8.548	0.004*

C. Individual or Pair x Task Complexity

	Individual or Pair						F Value	Sig p value
	Best Individual in nominal pair		Second-best Individual in nominal pair		Collaborating Pair			
	Mean	SD	Mean	SD	Mean	SD		
Software Quality							1.737	0.182
Task Complexity Low	74.420	5.968	50.790	5.973	74.983	5.938		
Task Complexity High	71.626	5.936	35.969	5.939	50.080	5.935		

** significant at p=0.05*

To protect against inflated type I error, each of the significant p-values was less than 0.025 (the Bonferroni's alpha adjustment), which should alleviate concerns over the experiment-wise error rate.

A 4 (best, second-best in nominal pairs, best, second-best in collaborating pairs) by 2 (low task complexity, high task complexity) MANCOVA analysis was conducted on the three dependent variables measured individually in both collaborating pair and nominal pair conditions with GPA as covariate. The best, and second best individuals in both collaborating pairs and nominal pairs were determined by their task mental model values. Where their scores were equal, they were randomly designated as the best or the second-best. The MANCOVA model was found to be significant with Pillai's Trace F = 7.397, p = 0.000. The MANCOVA analysis also revealed a significant main effect for the task complexity dimension on the task mental model with Pillai's Trace F = 10.948, p = 0.000. Results of the 4 (best, second-best individual in nominal pairs and collaborating pairs) by 2 (low task complexity, high task complexity) MANCOVA analysis on the three dependent measures indicated a significant main effect for the person (best, second-best individuals in the collaborating pair and nominal pair conditions) factor. Table 5.8 summarizes the MANCOVA results for software quality on the two treatment factors.

Table 5.8 – MANCOVA Results for the Hypotheses

	Value	F Value	Degrees of Freedom		Sig p value
			Between Group	Within Group	
Condition (best, second-best in nominal pairs and collaborating pairs) vs. Dependent Measures					
Pillai's Trace	0.536	7.397	9	306	0.000*
Wilks' Lamda	0.519	8.377	9	244	0.000*
Hotelling-Lawley Trace	0.825	9.040	9	296	0.000*
Roy's Largest Root	0.682	23.185	3	102	0.000*
Task Complexity vs. Dependent Measures					
Pillai's Trace	0.247	10.948	3	100	0.000*
Wilks' Lamda	0.753	10.948	3	100	0.000*
Hotelling-Lawley Trace	0.328	10.948	3	100	0.000*
Roy's Largest Root	0.328	10.948	3	100	0.000*
GPA (Covariate) vs. Dependent Measures					
Pillai's Trace	0.046	1.625	3	100	0.188
Wilks' Lamda	0.954	1.625	3	100	0.188
Hotelling-Lawley Trace	0.049	1.625	3	100	0.188
Roy's Largest Root	0.049	1.625	3	100	0.188
Condition x Task Complexity vs. Dependent Measures					
Pillai's Trace	0.153	1.828	9	306	0.063
Wilks' Lamda	0.847	1.905	9	244	0.052
Hotelling-Lawley Trace	0.180	1.969	9	296	0.043*
Roy's Largest Root	0.177	6.002	3	102	0.001*

** significant at p=0.05*

With a significant MANCOVA model, individual ANOVA results were analyzed for the three dependent variables and the results summarized in Table 5.9. It can be seen that the main effect in the ANOVA model was significant for the person (best, second-best individuals in both collaborating pairs and nominal pairs) factor on the dependent measures of task mental model (p = 0.000), overall satisfaction (p = 0.005), and confidence in solution (p = 0.009).

Table 5.9 – One-Way ANOVA Results on Other Dependent Measures

A. Person

	Person									
	Best Individual in nominal pair		Second-best Individual in nominal pair		Best individual in collaborating pair		Second-best individual in collaborating pair		F Value	Sig p value
	Mean	SD	Mean	SD	Mean	SD	Mean	SD		
Task Mental Model	0.494	0.107	0.347	0.117	0.461	0.973	0.333	0.109	20.765	0.000*
Overall Satisfaction	3.590	1.473	3.590	1.424	4.190	1.559	4.790	1.565	4.577	0.005*
Confidence in Solution	-0.380	0.785	-0.156	0.939	0.263	1.106	0.314	0.948	4.081	0.009*

B. Task Complexity

	Task Complexity				F Value	Sig p value
	Low		High			
	Mean	SD	Mean	SD		
Task Mental Model	0.459	0.112	0.360	0.124	32.561	0.000 *
Overall Satisfaction	4.170	1.553	3.880	1.580	0.889	0.348
Confidence in Solution	1.341	1.004	-0.133	0.944	2.326	0.130

C. Person x Task Complexity

	Person								F Value	Sig p value
	Best Individual in nominal pair		Second-best Individual in nominal pair		Best individual in pair		Second-best individual in collaborating pair			
	Mean	SD	Mean	SD	Mean	SD	Mean	SD		
Task Mental Model									2.118	0.103
Task Complexity Low	0.510	0.105	0.406	0.119	0.511	0.101	0.407	0.807		
Task Complexity High	0.478	0.111	0.288	0.084	0.408	0.058	0.252	0.073		
Overall Satisfaction									1.920	0.131
Task Complexity Low	3.160	1.577	4.020	1.476	4.270	1.321	5.210	1.220		
Task Complexity High	3.980	1.297	3.160	1.281	4.100	1.833	4.330	1.804		
Confidence in Solution									4.447	0.006*
Task Complexity Low	-0.776	0.796	0.213	0.796	0.391	1.036	0.708	0.778		
Task Complexity High	-0.011	0.585	-0.525	0.950	0.126	1.203	-0.110	0.957		

** significant at p=0.05*

To protect against inflated type I error, it can be seen that each of these p-values was less than 0.025 (the Bonferroni's alpha adjustment), which should alleviate concerns over the experiment-wise error rate.

5.1.7 Tests of Assumptions

ANOVA models rely on the assumptions of normality, constant variance and independence of error terms (Neter et al., 1996). The omnibus normality test was conducted on the dependent measure of software quality across the two factors and the results are summarized in Table 5.10. The normality assumption could not be rejected for the first factor of 'individual or pair' (best, second-best individuals in nominal pair, pair). For the second factor of 'task complexity', the normality assumption was rejected suggesting the grades distribution across the two levels of task complexity may not be normal. However, Martinez and Iglewicz normality test for grades across individual treatment cells (3 levels of 'Individual or pair' x 2 levels of 'task complexity') could not be rejected at a significance level of alpha = 0.05. The Modified Levene test was conducted for testing the equality of error variance of grades across the two factors individually. The results tabulated in Table 8 suggest that the assumption of equal variance of error terms across the two factors could not be rejected at a significance level of alpha = 0.05 across both the factors. The assumption of equality of error variances across the groups (six treatment groups) also could not be rejected (p= .063).

Table 5.10 - Diagnostic Information for ANCOVA Assumptions

	Omnibus Normality Test		Modified Levene Test	
	Statistic	Sig	Statistic	Sig
Software Quality vs. Individual or Pair	0.6450	0.7243	3.098	0.0501
Software Quality vs. Task Complexity	12.2690	0.0022*	0.5549	0.4583

** significant at p=0.05*

MANCOVA analysis involved three dependent variables (task mental model, overall satisfaction and confidence in solution) across two factors (person and task complexity) with one covariate (GPA). MANCOVA may be viewed as MANOVA of the regression residuals, or variance in the dependent variable not explained by the covariates (Hair et al., 1998). The person factor involved 4 levels (best and second best individuals in nominal pair and collaborating pair conditions). The second factor of task complexity involved two levels (low and high). MANCOVA models also rely on the assumptions of multivariate normality, constant variance and equality of error variance (Hair et al., 1998). The omnibus normality test across the two factors on the three dependent measures suggested that the assumption of normality of error terms could not be rejected at a significance level of alpha = 0.05 for the dependent measures of task mental model and overall satisfaction. For the third dependent variable of ‘confidence in solution’, the normality assumption is rejected across both the factors suggesting the error term distribution of this factor across the two factors may not be normal and needed further investigation. The, Martinez and Iglewicz normality test for error terms of confidence in solution across individual treatment cells (4 levels of person x 2 levels of task complexity) could not be rejected at a significance level of alpha = 0.05. Thus, the assumption of normality of errors terms was satisfied for the MANCOVA analysis. The Modified Levene test was conducted for testing the equality of error variances of the three dependent variables across the two factors individually. The results tabulated in Table 5.11 suggested that the assumption of equal variance of error terms could not be rejected at a significance level of alpha = 0.05 across both the factors.

Equality of covariance matrices across the groups (eight cells) was tested by Box's M test. Table 9 shows the Box's M test results across the two factors separately. The Box's M statistic was significant with a p value of 0.108 suggesting that the assumption of equality of covariance matrices could not be rejected at a significance level of alpha = 0.05.

Table 5.11 - Diagnostic Information for MANCOVA Assumptions

	Omnibus Normality		Modified Levene Test		Box' M Test		Bartlett's Test of Sphericity	
	Statistic	Sig	Statistic	Sig	Statistic	Sig	Statistic	Sig
A. Dependent Measures vs. Individual or Pair								
Task Mental Model	3.7466	0.1536	0.4141	0.7432	13.0118	0.8268	67.502	0.000*
Overall Satisfaction	4.0712	0.1306	0.2003	0.8960				
Confidence in Solution	10.9760	0.0041*	0.6891	0.5605				
B. Dependent Measures vs. Task Complexity								
Task Mental Model	0.3310	0.8475	0.0244	0.8762	9.5171	0.1602		
Overall Satisfaction	0.8777	0.6448	0.1308	0.7183				
Confidence in Solution	20.4612	0.0000*	0.9043	0.3436				

** significant at p=0.05*

5.2 Hypotheses Testing

As most interactions were not found to be significant, ANCOVA/MANCOVA models were used to examine each dependent variable. Bonferroni’s multiple inference procedure was used to test the hypothesized family of inferences. Bonferroni’s procedure was used as it is considered superior to other multiple inference procedures when the family of inferences is finite (about the same as the number of factor levels or less) and could be specified in advance (Neter et al., 1996). In the ANCOVA analysis, the differences in the factor level means on the first factor (individual or pair) were of interest. The second factor (task complexity) was of interest mainly for its interaction effect with the main factor. In view of lack of significance found for the interactions in the ANCOVA model tested earlier, pair wise differences on the task complexity factor were not further investigated.

5.2.1 Hypotheses Concerning Software Quality

In this section, Hypotheses 1 and 5 are examined which involve only ANCOVA models and the resultant hypothesized comparisons on the dependent variable of software quality. Table 5.12 shows the results of Bonferroni’s custom contrast report for software quality means. Software quality was estimated as the average score provided by two examiners who graded the software solutions independently. Software quality was tested separately from other dependent variables as it was measured at the group level for collaborating pairs, while other dependent measures were measured at the individual level even within collaborating pairs.

Table 5.12 – Bonferroni's Custom Contrast Report of Software Quality at Alpha = 0.05

Treatment Condition	Mean	Standard Deviation		1	2	3
1	75.000	22.927	1			
2	42.450	23.371	2	--		
3	61.483	31.644	3	p = 0.083	p = 0.000*	

1 – Best programmer in nominal pair
2 – Second-best programmer in nominal pair
3 – collaborating pair

** significant at alpha = 0.05*

5.2.1.1. Hypothesis H1a

H1a - While working on a programming task, performance in terms of software quality of a collaborating pair is higher than the performance of best programmer in a nominal pair

Table 5.13 shows the relevant means.

Table 5.13 – Means for Software Quality (A)

Factor Level	Means
Best Programmer in Nominal Pair	75.000
Collaborating Pair	61.483

** significant at p=0.05*

The performance in terms of software quality of collaborating pair was not found to be higher than the performance of best programmer in nominal group (p = 0.959 for one-tailed test). It was found to be somewhat less than the performance of best programmer in nominal group, though the difference was not significant. Hypothesis1a is therefore not supported. It can therefore be said that performance of collaborating programming pair was found to be comparable to that of the best individual programmer in nominal pair.

5.2.1.2 Hypothesis1b

H1b - While working on a programming task, performance in terms of software quality of a collaborating pair is higher than the performance of second-best programmer in a nominal pair

Table 5.14 shows the relevant means.

Table 5.14 – Means for Software Quality (B)

Factor Level	Means
Second-best Programmer in Nominal Pair	42.450*
Collaborating Pair	61.483

** significant at p=0.05*

The performance in terms of software quality of collaborating pair was found to be significantly higher than the performance of second-best programmer in nominal pair (p=0.000 for one-tailed test). Hypothesis 1b is therefore fully supported. It can therefore be said that performance of collaborating programming pair is higher than that of the second-best programmer in nominal pair and comparable to that of the best programmer in nominal pair. Figure 1 shows a plot of the marginal means of software quality.

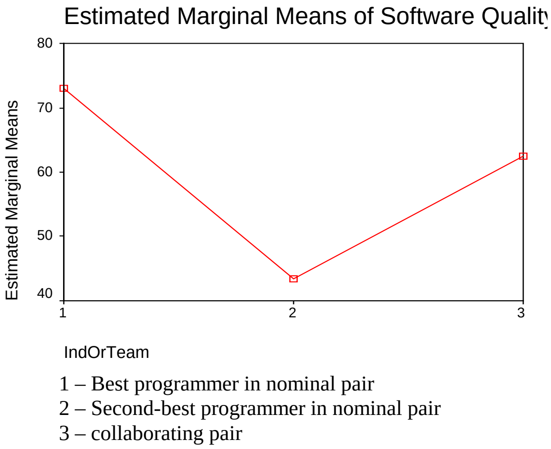


Figure 5.4 – Marginal Means of Software Quality in the Three Conditions

5.2.1.2 Hypothesis 5

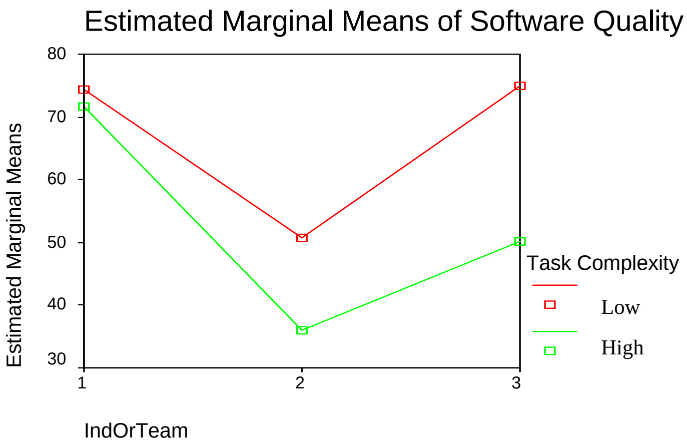
H5 - While working on a programming task, the difference in performance in terms of software quality between a collaborating pair and the best programmer in a nominal pair is higher for tasks of high complexity than for tasks of low complexity.

Table 5.15 shows the relevant means.

Table 5.15 – Means for Software Quality (C)

Treatment Level	Means
Best Programmer in Nominal Pair	
Task Complexity Low	77.400
Task Complexity High	72.600
Second-best Programmer in Nominal Pair	
Task Complexity Low	47.633
Task Complexity High	37.267
Collaborating Pair	
Task Complexity Low	73.767
Task Complexity High	49.200

In the ANCOVA model tested for significance earlier, interactions between the two factors (individual or group and task complexity) were found to be not significant (p=0.182). In ANCOVA models when there are no significant interactions, factor means could be compared based on significance of the main factors. When the interactions were not significant, no meaningful information could be obtained by comparing the cell means. Hence, hypothesis 4 is not supported. It can therefore be said that the difference in performance in terms of software quality between the collaborating pair and the best programmer in nominal group is not significantly different between tasks of different complexity. Figure 5.5 shows a plot of the marginal means of software quality for tasks of low and high complexity.



1 – Best programmer in nominal pair
2 – Second-best programmer in nominal pair
3 – Collaborating pair

Figure 5.5 – Marginal Means of Software Quality in the Three Conditions for tasks of Low and High Complexity

5.2.2 Hypotheses Concerning Task Mental Model

In this section, hypotheses 2 and 6 are examined which involve only MANCOVA models and the resultant comparisons of means on one of the dependent variables, namely task mental model. Table 5.16 shows the results of Bonferroni's custom contrast report of task mental model means. Bonferroni's procedure was used as it is considered superior to other multiple inference procedures when the family of inferences is finite (about the same as the number of factor levels or less) and could be specified in advance (Neter et al., 1996). Task mental model was estimated as the similarity between the mental models of individual subjects and the expert mental model of the problem solution.

Table 5.16 – Bonferroni's Custom Contrast Report of Task Mental Model at Alpha = 0.05

Factor Level	Mean	Standard Deviation					
				1	2	3	4
1	0.493	0.107	1				
2	0.347	0.117	2				
3	0.461	0.973	3	p=0.172	p=0.000*		
4	0.333	0.109	4		p=0.476		

1 – Best programmer in nominal pair
2 – Second-best programmer in nominal pair
3 – Best programmer in collaborating pair
4 – Second-best programmer in collaborating pair

** significant at alpha = 0.05*

5.2.2.1 Hypothesis 2a

H2a - While working on a programming task, task mental model of the best programmer of a collaborating pair is better than the task mental model of the best programmer in a nominal pair

Table 5.17 shows the relevant means.

Table 5.17 – Means for Task Mental Model (A)

Factor Level	Means
Best Programmer in Nominal Pair	0.493
Best Programmer in Collaborating Pair	0.461

** significant at alpha = 0.05*

The task mental model of the best programmer of the collaborating pair was not found to be better than the task mental model of the best programmer in the nominal pair at alpha = 0.05 (p = 0.914 in a one-tailed test). Hypothesis 2a is therefore not supported. Hence it can be inferred that the task mental model of the best programmer in the collaborating pair is comparable to that of the best programmer in nominal pair.

5.2.2.2 Hypothesis 2b

H2b - While working on a programming task, task mental model of the best programmer of a collaborating pair is better than the task mental model of the second-best programmer in a nominal pair

Table 17 shows the relevant means and pairwise comparisons

Table 5.18 – Means for Task Mental Model (B)

Factor Level	Means
Second-best Programmer in Nominal Pair	0.347
Best Programmer in Collaborating Pair	0.461*

** significant at alpha = 0.05*

The task mental model of the best programmer of the collaborating pair was found to better than the task mental model of the second-best programmer in the nominal pair (p = 0.000 in a one-tailed test). Hypothesis 2b is therefore fully supported.

It can therefore be said that the task mental model of the best programmer in the collaborating group is better than that of the second-best programmer in the nominal pair and is comparable to that of the best programmer in nominal pair.

5.2.2.3 Hypothesis 2c

H2c - While working on a programming task, task mental model of the second-best programmer of a collaborating pair is better than the task mental model of the second-best programmer in a nominal pair

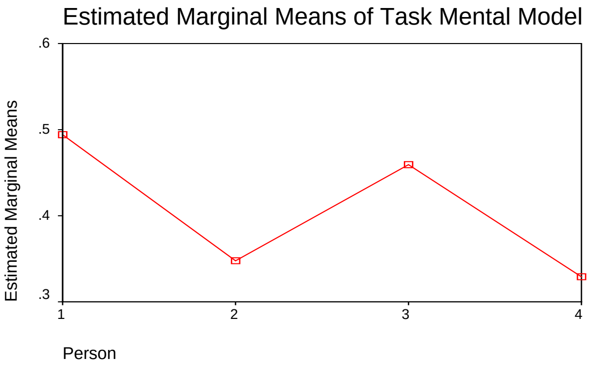
Table 5.19 shows the relevant means.

Table 5.19 – Means for Task Mental Model (C)

Factor Level	Means
Second-best Programmer in Nominal Pair	0.347
Second-Best Programmer in Collaborating Pair	0.333

** significant at p=0.05*

The task mental model of the second-best programmer of the collaborating pair was not found to be better than the task mental model of the second-best programmer in the nominal pair at alpha = 0.05 (p = 0.762 in a one-tailed test). Hypothesis 2c is therefore not supported. It can therefore be said that the task mental model of the second-best programmer in the collaborating group is comparable to that of the second-best programmer in nominal pair. Figure 5.6 shows a plot of the marginal means of task mental model.



1 – Best programmer in nominal pair
2 – Second-best programmer in nominal pair
3 – Best programmer in collaborating pair
4 – Second-best programmer in collaborating pair

Figure 5.6 – Marginal Means of Task Mental Model

5.2.2.4 Hypothesis 6

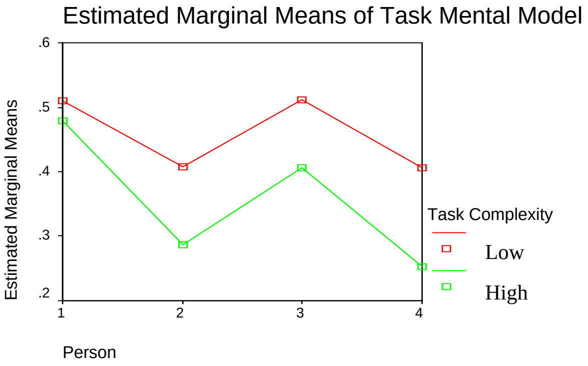
H6 - While working on a programming task, the difference between the task mental models of best programmer in the collaborating pair and the best programmer in the nominal pair is higher for tasks of high complexity than for tasks of low complexity.

Table 5.20 shows the relevant means.

Table 5.20 – Means for Task Mental Model (D)

Treatment Level	Means
Best Programmer in Nominal Pair	
Task Complexity Low	0.510
Task Complexity High	0.478
Second-best Programmer in Nominal Pair	
Task Complexity Low	0.406
Task Complexity High	0.288
Best Programmer in Collaborating Pair	
Task Complexity Low	0.511
Task Complexity High	0.408
Second-best Programmer in Collaborating Pair	
Task Complexity Low	0.407
Task Complexity High	0.252

In the MANCOVA model tested for significance earlier, interactions between the two factors (person and task complexity) were found to be not significant (p = 0.103). Hence, hypothesis 4 is not supported. In ANCOVA models when there are no significant interactions, factor means could be compared depending upon significance of the main factors. When the interactions are not significant, no additional information could be obtained by comparing the cell means. It can therefore be said that the difference in task mental model between the best programmer in the collaborating pair and the best programmer in nominal pair is not significantly different between tasks of differing complexities. Figure 5.7 shows a plot of the marginal means of task mental model for tasks of low and high complexity.



1 – Best programmer in nominal pair
2 – Second-best programmer in nominal pair
3 – Best programmer in collaborating pair
4 – Second-best programmer in collaborating pair

Figure 5.7 – Marginal Means of Task Mental Model for Tasks of Low and High Complexity

5.2.3 Hypotheses Concerning Overall Satisfaction

In this section, Hypotheses 3 is examined which involves MANCOVA models and the resultant comparisons on one of the dependent variables, namely overall satisfaction. Table 5.21 shows the results of Bonferroni's custom contrast report of the overall satisfaction means. Overall satisfaction was estimated as the summated score on four items.

Table 5.21 – Bonferroni's Custom Comparison Report of Overall Satisfaction at Alpha = 0.05

Factor Level	Mean	SD		1	2	3	4
1	3.590	1.473	1				
2	3.590	1.424	2				
3	4.190	1.559	3	p=0.117	p=0.110		
4	4.790	1.565	4		p=0.002*		

1 – Best programmer in nominal pair
2 – Second-best programmer in nominal pair
3 – Best programmer in collaborating pair
4 – Second-best programmer in collaborating pair

** significant at p=0.05*

5.2.3.1 Hypothesis 3a

H3a - While working on a programming task, the overall satisfaction of the best programmer in a collaborating pair is higher than the overall satisfaction of the best programmer in a nominal pair

Table 5.22 shows the relevant means.

Table 5.22 – Means for Overall Satisfaction (A)

Factor Level	Means
Best Programmer in Nominal Pair	3.59
Best Programmer in Collaborating Pair	4.19

** significant at p=0.05*

The overall satisfaction of the best programmer of the collaborating pair was not found to be higher than the overall satisfaction of the best programmer in the nominal pair at alpha = 0.05 (p = 0.059 in a one-tailed test). Hypothesis 3a is therefore not supported. It can therefore be inferred that the overall satisfaction of the best programmer in the collaborating pair is comparable to that of the best programmer in the nominal pair.

5.2.3.2 Hypothesis 3b

H3b - While working on a programming task, the overall satisfaction of the best programmer in a collaborating pair is higher than the overall satisfaction of the second-best programmer in a nominal pair

Table 5.23 shows the relevant means.

Table 5.23 – Means for Overall Satisfaction (B)

Factor Level	Means
Second-best Programmer in Nominal Pair	3.59
Best Programmer in Collaborating Pair	4.19

** significant at p=0.05*

The overall satisfaction of the best programmer in the collaborating pair was not found to be higher than the overall satisfaction of the second-best programmer in the nominal pair at alpha = 0.05 (p = 0.055 in a one-tailed test). Hypothesis 3b is therefore not supported. It can therefore be concluded that the overall satisfaction of the best programmer in the collaborating group is comparable to that of the second-best programmer in the nominal pair.

5.2.3.3 Hypothesis 3c

H3b - While working on a programming task, the overall satisfaction of the second-best programmer in a collaborating pair is higher than the overall satisfaction of the second-best programmer in a nominal pair

Table 5.24 shows the relevant means.

Table 5.24 – Means for Overall Satisfaction (C)

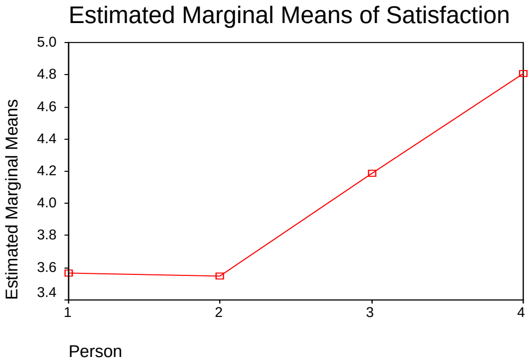
Factor Level	Means
Second-best Programmer in Nominal Pair	3.59
Second-best Programmer in Collaborating Pair	4.79*

** significant at p=0.05*

The overall satisfaction of the second-best programmer in the collaborating pair was found to be higher than the overall satisfaction of the second-best programmer in the nominal pair at alpha = 0.05 (p = 0.001 in a one-tailed test). Hypothesis 3c is therefore fully supported. It can therefore be said that the overall satisfaction of the second-best programmer in the collaborating group is higher than that of the second-best programmer in the nominal pair (Figure 5.8).

5.2.4 Hypotheses Concerning Confidence in Solution

In this section, Hypothesis 4 is examined which involves MANCOVA model and the resultant comparisons on one of the dependent variables, namely confidence in solution.



1 – Best programmer in nominal pair
2 – Second-best programmer in nominal pair
3 – Best programmer in collaborating pair
4 – Second-best programmer in collaborating pair

Figure 5.8 – Marginal Means of Overall Satisfaction

5.2.4.1 Hypothesis 4a

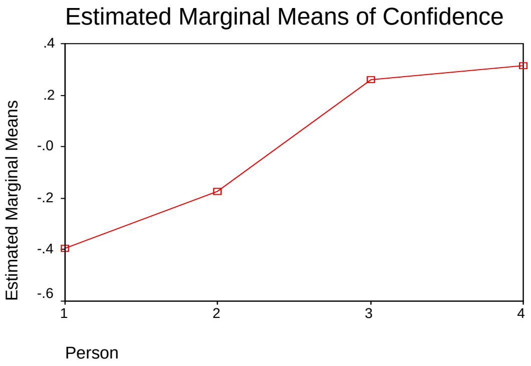
H4a - While working on a programming task, the confidence in solution of the best programmer in a collaborating pair is higher than the confidence in solution of the best programmer in a nominal pair

Table 5.25 shows the results of tests for significance in the ANCOVA model for the main effect and the interaction effect for the dependent variable of confidence in solution. As can be seen from the table, both the main effect and the interaction effect were significant. When interactions are significant and important, then it is only meaningful to interpret cell means and not the factor means (Neter et al., 1996). Hence the hypotheses for the main effect of confidence in solution were tested in terms of the sub-hypotheses involving differences in the cell means. That is the hypotheses were tested separately for the two levels of Task complexity.

Table 5.25 – Result of Test for Interactions

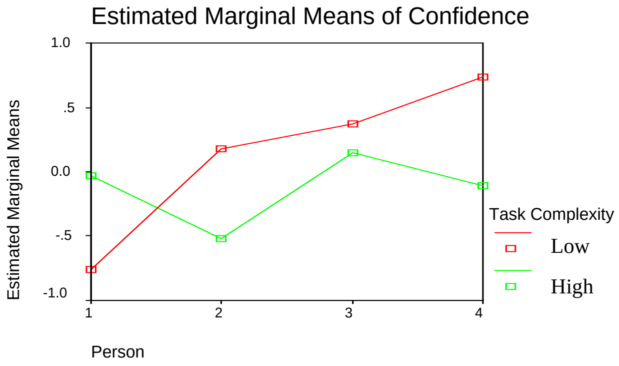
Dependent Variable	Person	Task Complexity	Interaction
Confidence in Solution	4.081 (0.009*)	2.326 (0.130)	4.447 (0.006*)

Figure 5.9 shows the plot of marginal means of confidence in solution (main effect) and Figure 5.10 shows the plot of marginal means of confidence for tasks of low and high complexity. It can be seen that the interaction between person and confidence is disordinal with mean difference between confidence levels for complex and simple tasks being positive or negative for different types of person.



1 – Best programmer in nominal pair
2 – Second-best programmer in nominal pair
3 – Best programmer in collaborating pair
4 – Second-best programmer in collaborating pair

Figure 5.9 – Marginal Means of Confidence in Solution



1 – Best programmer in nominal pair
2 – Second-best programmer in nominal pair
3 – Best programmer in collaborating pair
4 – Second-best programmer in collaborating pair

Figure 5.10 – Marginal Means of Confidence in Solution for Tasks of Low and High Complexity

Table 5.26 shows the post-hoc Bonferroni's multiple comparison test results of cell means for the dependent measure of confidence in solution. A multiple comparison technique (e.g. Bonferroni's) was chosen to help control for the confidence coefficient across the family of tests (Neter et al., 1996).

Hypotheses 4a was divided into two sub hypotheses 4a-1 and 4a-2 and tested for significance as given below:

Hypothesis 4a-1 - While working on a programming task of low complexity, the confidence in solution of the best programmer in a collaborating pair is higher than the confidence in solution of the best programmer in a nominal pair

Table 5.26 – Bonferroni’s Multiple Comparison Test of Cell Means for Confidence in solution

Treatment Cell	Mean	SD	Cell	“ * ” Significant,				“ -- " Insignificant			
				11	21	31	41	12	22	32	42
11	-0.776	0.796	11								
21	0.213	0.796	21	--							
31	0.391	1.036	31	*	--						
41	0.708	0.778	41	*	--	--					
12	-0.011	0.585	12	--	--	--	--				
22	-0.525	0.950	22	--	--	--	*	--			
32	0.126	1.203	32	--	--	--	--	--	--		
42	-0.110	0.957	42	--	--	--	--	--	--	--	

Legend

11 – Best programmer in nominal pair	Low Task Complexity
21 – Second-best programmer in nominal pair	Low Task Complexity
31 – Best programmer in collaborating pair	Low Task Complexity
41 – Second-best programmer in collaborating pair	Low Task Complexity
12 – Best programmer in nominal pair	High Task Complexity
22 – Second-best programmer in nominal pair	High Task Complexity
32 – Best programmer in collaborating pair	High Task Complexity
42 – Second-best programmer in collaborating pair	High Task Complexity

** significant at alpha = 0.05*

Table 5.27 shows the relevant treatment means.

Table 5.27 – Treatment Means for Confidence in Solution for Cells 11 and 31

Person	Task Complexity	Means
1 - Best Programmer in Nominal Pair	1- Low	-0.776
3 - Best Programmer in Collaborating Pair	1- Low	0.391*

** significant at alpha=0.05*

While working on tasks of low complexity, the confidence in solution of the best programmer in the collaborating pair was found to be higher than the confidence in solution of the best programmer in the nominal pair at alpha = 0.05 (p < 0.025 in a one-tailed test). Hypothesis 4a-1 is therefore fully supported.

H4a-2 - While working on a programming task of high complexity, the confidence in solution of the best programmer in a collaborating pair is higher than the confidence in solution of the best programmer in a nominal pair

Table 28 shows the relevant treatment means and pairwise comparisons

Table 5.28 – Treatment Means for Confidence in Solution for Cells 12 and 32

Person	Task Complexity	Means
1 - Best Programmer in Nominal Pair	2 - High	-0.011
3 - Best Programmer in Collaborating Pair	2 - High	0.126

** significant at p=0.05*

While working on tasks of high complexity, the confidence in solution of the best programmer in the collaborating pair was found to be not higher than the confidence in solution of the best programmer in the nominal pair at alpha = 0.05. Hypothesis 4a-1 is therefore not supported.

In view of the mixed results for the two sub-hypotheses involving treatment means, the main hypothesis of 4a involving comparison of factor means is considered as not supported.

5.2.4.2 Hypothesis 4b

H4b - While working on a programming task, the confidence in solution of the best programmer in a collaborating pair is higher than the confidence in solution of the second-best programmer in a nominal pair

In view of significant interactions between the two factors, Hypotheses 4b is divided into two sub hypotheses 4b-1 and 4b-2 and tested for significance as given below:

H4b-1 - While working on a programming task of low complexity, the confidence in solution of the best programmer in a collaborating pair is higher than the confidence in solution of the second-best programmer in a nominal pair

Table 5.29 shows the relevant treatment means.

Table 5.29 – Treatment Means for Confidence in Solution for Cells = 21 and 31

Person	Task Complexity	Means
2 - Second-best Programmer in Nominal Pair	1 - Low	0.213
3 - Best Programmer in Collaborating Pair	1 - Low	0.391

** significant at p=0.05*

While working on tasks of low complexity, the confidence in solution of the best programmer in the collaborating pair was not found to be higher than the confidence in solution of the second-best programmer in the nominal pair at alpha = 0.05. Hypothesis 4b-1 is therefore not supported.

H4b-2 - While working on a programming task of high complexity, the confidence in solution of the best programmer in a collaborating pair is higher than the confidence in solution of the second-best programmer in a nominal pair

Table 5.30 shows the relevant treatment means.

Table 5.30 - Treatment Means for Confidence in Solution for Cells 22 and 32

Person	Task Complexity	Means
2 - Second-best Programmer in Nominal Pair	2 - High	-0.525
3 - Best Programmer in Collaborating Pair	2 - High	0.126

** significant at p=0.05*

While working on tasks of high complexity, the confidence in solution of the best programmer in the collaborating pair was not found to be higher than the confidence in solution of the second-best programmer in the nominal pair at alpha = 0.05. Hypothesis 4b-2 is therefore not supported.

As the two sub-hypotheses involving treatment means were not supported, the main hypothesis of 4b involving comparison of factor means is considered not supported.

Therefore while working on a programming task, the confidence in solution of the best programmer in collaborating pair is not higher than the confidence in solution of the second-best programmer in the nominal pair

5.2.4.3 Hypothesis 4c

Hypothesis 4c - While working on a programming task, the confidence in solution of the second-best programmer in a collaborating pair is higher than the confidence in solution of the second-best programmer in a nominal pair

In view of significant interactions between the two factors, Hypotheses 4c is divided into two sub hypotheses 4c-1 and 4c-2 and tested for significance as given below:

H4c-1 - While working on a programming task of low complexity, the confidence in solution of the second-best programmer in a collaborating pair is higher than the confidence in solution of the second-best programmer in a nominal pair

Table 5.31 shows the relevant treatment means and pairwise comparisons

Table 5.31 – Treatment Means for Confidence in Solution for Cells 21 and 41

Person	Task Complexity	Means
2 - Second-best Programmer in Nominal Pair	1 - Low	0.213
4 – Second best Programmer in Collaborating Pair	1 - Low	0.708

** significant at p=0.05*

While working on tasks of low complexity, the confidence in solution of the second-best programmer in the collaborating pair was not found to be higher than the confidence in solution of the second-best programmer in the nominal pair at alpha = 0.05. Hypothesis 4c-1 is therefore not supported.

H4c-2 - While working on a programming task of high complexity, the confidence in solution of the second-best programmer in a collaborating pair is higher than the confidence in solution of the second-best programmer in a nominal pair

Table 5.32 shows the relevant treatment means and pairwise comparisons

Table 5.32 - Treatment Means for Confidence in Solution for Cells 22 and 42

Person	Task Complexity	Means
2 - Second-best Programmer in Nominal Pair	2 - High	-0.525
4 – Second-best Programmer in Collaborating Pair	2 - High	-0.110

** significant at p=0.05*

While working on tasks of high complexity, the confidence in solution of the second-best programmer in the collaborating pair was not found to be higher than the confidence in solution of the second-best programmer in the nominal pair at alpha = 0.05. Hypothesis 4c-2 is therefore not supported.

As the two sub-hypotheses involving treatment means were not supported, the main hypothesis of 4c involving comparison of factor means is considered not supported.

Therefore while working on a programming task, the confidence in solution of the second-best programmer in collaborating pair is not higher than the confidence in solution of the second-best programmer in the nominal pair.

5.2.5 Summary of Hypotheses Testing

Three hypotheses in all were fully supported. For one hypothesis (Hypotheses 4a) there was mixed support with the hypothesis supported for tasks of low complexity but not for tasks of high complexity. The results of hypotheses testing are summarized in Table 5.33. A complete discussion on the reasons for exceptions is presented in chapter 6.

Table 5.33 - Results of Hypotheses Testing

Hypothesis	Description	Result
1a	*While working on a programming task, performance in terms of software quality of a collaborating pair is higher than the performance of best programmer in a nominal pair*	Not Supported
1b	*While working on a programming task, performance in terms of software quality of a collaborating pair is higher than the second-best programmer in a nominal pair*	Supported
2a	*While working on a programming task, the task mental model of the best programmer of a collaborating pair is better than the task mental model of the best programmer in a nominal pair*	Not Supported
2b	*While working on a programming task, the task mental model of the best programmer of a collaborating pair is better than the task mental model of the second-best programmer in a nominal pair*	Supported
2c	*While working on a programming task, the task mental model of the second-best programmer of a collaborating pair is better than the task mental model of the second-best programmer in a nominal pair*	Not Supported
3a	*While working on a programming task, the overall satisfaction of the best programmer in a collaborating pair is higher than the overall satisfaction of the best programmer in a nominal pair*	Not Supported
3b	*While working on a programming task, the overall satisfaction of the best programmer in a collaborating pair is higher than the overall satisfaction of the second-best programmer in a nominal pair*	Not Supported
3c	*While working on a programming task, the overall satisfaction of the second-best programmer in a collaborating pair is higher than the overall satisfaction of the second-best programmer in the nominal pair*	Supported
4a	*While working on a programming task, the confidence in solution of the best programmer in a collaborating pair is higher than the confidence in solution of the best programmer in a nominal pair*	Not Supported
4b	*While working on a programming task, the confidence in solution of the best programmer in a collaborating pair is higher than the confidence in solution of the second-best programmer in a nominal pair*	Not Supported
4c	*While working on a programming task, the confidence in solution of the second-best programmer in a collaborating pair is higher than the confidence in solution of the second-best programmer in a nominal pair*	Not Supported
5	*While working on a programming task, the difference in performance in terms of software quality between a collaborating pair and the best programmer in a nominal pair is higher for tasks of high complexity than for tasks of low complexity.*	Not Supported
6	*While working on a programming task, the difference between the task mental models of best programmer in a collaborating pair and the best programmer in a nominal pair is higher for tasks of high complexity than for tasks of low complexity.*	Not Supported

5.3 Manipulation Checks

Manipulation check questions were asked to determine whether subjects correctly perceived the treatment condition they were assigned randomly to. One question was asked to check whether the subjects worked individually (nominal pair) or with a partner (collaborating pair). This was to verify after the experiment that subjects' responses were consistent with their allotted conditions, and correct questionnaires pertaining to their treatments were administered to them. A cursory check revealed that there were no discrepancies between the condition allotted to the subjects and their responses regarding the condition they were in.

The perceived difficulty of the experimental task was elicited from the subjects in terms of two sets of questions. In the first set of two questions subjects were asked to judge the difficulty of the experimental task in an absolute sense on a scale of 1-7. In the second set of two questions subjects were asked to judge the difficulty of the experimental task with that of the warm-up task. Warm up task used was common across treatments. As difficulty is perceived as a subject x complexity interaction, the second set of questions involving comparison with the warm up task was expected to better serve the purpose of manipulation check. Table 5.34 provides results of one-way ANOVA tests for the difference in the means of perceived task difficulty across the treatments involving tasks of low and high complexity. Both the manipulation check measures were significantly different between factor levels of low and high task complexity. Hence, there is no reason to doubt the effectiveness of the manipulation of task complexity across treatment conditions.

Table 5.34 - One-way ANOVA results for Manipulation Check

Manipulation Check Measure	Reliability	Task complexity Low		Task complexity High		F (p-value)
		Mean	SD	Mean	SD	
Perceived task difficulty						
Absolute terms	0.822	3.856	1.339	4.644	1.207	11.273 (0.001*)
Compared to warm up task	0.868	4.283	1.240	5.195	1.178	16.895 (0.000*)

*significant at alpha = 0.05

The next chapter discusses the summary of the research findings along with significance of the findings. The limitations of the study and possible future research extensions are also presented.

CHAPTER 6

SUMMARY, LIMITATIONS AND FUTURE RESEARCH

Pair programming is an important practice in agile methodologies such as XP. The growing adoption of pair programming in software projects is evident from a recent global survey, which indicated of its use in 35% of 104 projects sampled. The adoption has been highest in India (58.3% of projects sampled) followed by US (35.5%), Europe (27.2%) and Japan (22.2%) (Cusumano, MacCormack, Kemerer, & Crandall, 2003).

Proponents of XP believe in the effectiveness of pairing and underscore its importance as a core practice in XP. The continuous inspection of code inherent in pair programming helps facilitate other XP practices such as simple design, collective code-ownership, minimal documentation, small releases, and continuous integration.

Anecdotal evidence and evidence from some empirical studies highlighted the benefits of pair programming such as improved code readability, reduction in errors in the code, higher satisfaction and higher reported confidence in the generated solution by the programmers. After some jelling period, the additional time taken for completion by pair programmers to arrive at the software solution was also found to be not significantly different from the time taken by the individual programmers (Nosek, 1998; Williams, 2000). These studies are extensively cited as evidence for the effectiveness of pair programming. There are other studies conducted using student projects in programming courses of computer science departments that reported benefits of pair programming such as improved retention of students in introductory programming courses, and reduction in the burden on the lab instructors as pairs helped each other (Nagappan et al., 2003).

A subsequent experimental study suggested that pair programming as in XP might not be as efficient as claimed by the previous studies. It was argued that XP is an expensive practice as the time taken is significantly higher than that of individual programming with no significant benefits in terms of reduction in defect rates (Nawrocki & Wojciechowski, 2001). The initial positive evidence for effectiveness of pair programming however needed to be empirical confirmed and extended through rigorously conducted experimental studies. This is considered particularly important in view of some mixed results reported in previous research studies.

Small group research literature has grappled for a long time with the issue of effectiveness of individuals and groups on various tasks including problem-solving tasks. A robust finding of this stream of research was that in intellective problem solving tasks, groups typically outperform average individuals but rarely exceed the performance of best individuals (Hill, 1982). Most research designs used for comparison of group effectiveness over individuals both in small group research and in IS research made comparison between groups and equal number of independently working individuals. Such designs allow comparison of group performance with that of an average individual. Group superiority over average individuals is expected based on probabilities alone, as groups are likely to have at least one good problem solver who could lead the group to the solution (Hill, 1982). A more illuminating research question would be to compare group performance with the best individual or the second-best individual in nominal groups. Using nominal group designs some recent small group research studies have shown that on certain problem solving tasks involving high information processing requirements, groups could potentially outperform best individuals [e.g. Letters to numbers tasks used in Laughlin et. al (Laughlin *et al.*, 2002; Laughlin *et al.*, 2003) studies].

We consider that programming task involves high information processing requirements. Similar to Laughlin's letters to numbers tasks, there could be 'assembly-bonus' effect when pairs collaboratively develop software code. We expected that the conditions of demonstrability required for groups to generate the 'assembly-bonus' effect would be available to a sufficient extent when programmers collaboratively work in pairs to develop software code.

It is also expected that when pairs work on the task, they may explore the problem domain better, which should lead to a better solution. Mental model is a construct that could capture programmers understanding of the programming task domain. The understanding reflected in the mental model is expected to drive code implementation and the resulting quality of solution. Mental models are used in cognitive psychology and human computer interaction studies to capture the understanding of individuals' knowledge of the systems and problem situations to predict performance. Task mental model could help capture whether pairs have better understanding of the situation over individual programmers.

There is some evidence in small group research suggesting that benefits of group work increase with increase in complexity of group task. Proponents of XP and agile methodologies have also speculated that benefits of pair programming should accentuate with increase in the complexity of programming task.

In light of the above, this research endeavored to examine whether programming pairs outperform independently working best and second-best programmers of nominal pairs in terms of software quality achieved. A related question was whether pairs develop superior task mental models compared to their independently working counterparts. The research hypotheses in this study were built on theoretical and empirical underpinnings from small group research and cognitive psychology domains. The findings of the study suggested that pairs approach the performance of best individuals but do not outperform them.

6.1 Summary of Research Findings

This section discusses the experimental results on the effect of pair programming on software quality, task mental model, overall satisfaction, and confidence in solution.

6.1.1 Software Quality

The findings of this study suggest that while working on a programming task, performance in terms of software quality of a collaborating pair is higher than the performance of second-best programmer in nominal pair, but is comparable to that of the best programmer in nominal pair. The best individual’s performance is in fact marginally higher than that of the pair, though the difference was not statistically significant. To the best of our knowledge, no previous IS study has attempted such comparisons using nominal groups. Hence we cannot comment on the plausibility of this finding from previous IS research. This finding is however consistent with the general finding in small group research that suggests groups to be rarely better than the best individuals. Research comparing group performance with equal number of individuals showed that groups perform at the level of the best individual member or best group member on problem solving tasks with highly demonstrable solutions as in mathematical, insight, and rule induction problems. On tasks involving weakly demonstrable estimations of quantities, groups perform at the level of the average individual (Laughlin et al., 2002). Groups could potentially outperform best individuals in intellective tasks with high information processing requirements. Four conditions of demonstrability identified in literature were: a) availability of sufficient information; b) group consensus on a conceptual system; c) incorrect members being able to correct response when proposed; d) correct members having sufficient time, ability and motivation to demonstrate the correct response to other members. With increasing demonstrability, problem-solving groups are expected to show distinctly superior performance over individuals (Laughlin, VanderStoep, & Hollingshead, 1991).

In the present study, we could speculate on some potential reasons for pairs not outperforming the best individuals. First, the effect even when present could be small and realized only in conditions of high solution demonstrability. Second, the conditions of solution demonstrability may not be available to the desired extent in the experimental setting: availability of sufficient information and best member in the group having sufficient abilities to demonstrate the solution to the partner.

Demonstrability of the task is a function of not only the task but also of the group members and the decision environment. The same task could be judgmental to low-ability members under time pressure while it could be intellective for high ability members with infinite time (Zarnoth & Sniezek, 1997). Student programmers typically have problem remembering the syntax of the programming language. Though access to online documentation is provided, they may lack familiarity and expertise in getting to the required help resource or correctly interpreting it. Though subjects working independently in nominal groups also experience similar problem, this particularly reduces the level of solution demonstrability that is so crucial for realizing group superiority over the best individuals. It is again speculated that certain minimum competency levels may be required of the programmers to realize this effect. Third, anecdotal evidence from previous IS studies that used multiple student projects during a semester suggests the importance of initial ‘jelling period’ for the pairs to click as a team. It is only after the ‘jelling period’ that pairs start to show performance improvements expected of them (Williams, 2000). In the present study, a warm up task of short duration (15 minutes) was used before the main task to allow the subjects to get familiarized with the procedures, setting, and with the partners when present. This would not be sufficient time for the pairs to jell. Longitudinal repeated measure designs would be required for the ‘pair jelling’ effect to be tested.

The relative performance differences between groups and individuals in nominal pairs were found to be consistent across tasks of different levels of complexity. That is task complexity was not found to have a moderating effect on the effect of individual/pair programming on the software quality. In fact it is noticed that software quality of pair suffered marginally compared to that of the individuals when working on complex task, though the difference was not significant. It is again speculated that process losses were higher when pairs worked on complex tasks. When the solution demonstrability was not high in the experimental condition due to the reasons articulated above, the process losses could outweigh process gains.

6.1.2 Task Mental Model

In addition to software quality, performance was also measured in terms of the task mental model. We conceptualized mental model as including the structural and relational information relevant to the programming task. For the purpose of this study we conceptualized task mental model as the similarity between programmer's and the experts' networks of the structural and relational information relevant to the programming task. It represented the programmer's level of understanding of the relationships between various objects, attributes, and behaviors (methods) of the problem task. Unlike software quality, which was evaluated at the group level, task mental model was measured at the individual level for subjects in both collaborating and nominal pairs.

The findings of this study suggest that while working on a programming task, the task mental model of the best programmer of the collaborating pair is better than the task mental model of the second-best programmer in the nominal pair, but is comparable to that of the best programmer in the nominal group. This relationship was found to be consistent across tasks of different levels of complexity. This finding is also consistent with the previous finding concerning software quality. The best programmer in the pair condition did not outperform the best individual in the nominal pair as hypothesized. We again speculate the reasons behind this finding to be very similar to the discussion in the previous section. That is, solution demonstrability may not be adequate in the pair condition for the pair superiority to be realized. Again in the absence of high levels of solution demonstrability, process losses could match or outweigh process gains when pairs work on complex tasks. Thus the hypothesized moderating effect of task complexity was also not found.

6.1.3 Overall Satisfaction

An interesting finding of the study is that best programmers in the collaborating pairs have comparable levels of overall satisfaction as the best and second-best individuals in the nominal pairs, while second-best programmers among collaborating pairs have higher satisfaction than second-best individuals in the nominal pairs. Prior IS research findings, and findings from small group research suggested that group work is generally more satisfying. A post-hoc comparison indicated that second-best individuals among collaborating pair were more satisfied than even the best individuals in the nominal pair. The best programmers in the collaborating pairs also reported marginally higher satisfaction than programmers in the nominal groups, though the difference was not statistically significant. It is plausible that second-best programmers among the collaborating pairs were more satisfied, as they were able to perform at significantly higher levels in the group than they would when performing individually.

6.1.4 Confidence in Solution

Task complexity had a significant moderating effect on the effect of individual/ pair programming on the programmers' confidence in their solutions. This was not hypothesized for lack of prior insights into this interaction effect. An interesting finding of the study based on post-hoc analysis was that the best programmers among collaborating pairs have higher confidence in their solution than best programmers in nominal pair when task complexity is low, but not when it is high. As discussed in the previous sections, with increasing task complexity, solution demonstrability levels could potentially go down especially if the abilities or information is lacking to tackle the problem, thus increasing the judgmental content (no right answers) of the task. An important finding in small group research is that confidence and accuracy (performance) are highly correlated for intellective tasks, but not for judgmental tasks. It is reasoned that in judgment tasks, it is difficult to be sure of the accuracy of a judgmental response and to convince others of it (Zarnoth & Sniezek, 1997). Coupled with the process losses inherent in group working that could affect pair performance, the best programmers in collaborating pairs have no extra means of bolstering their confidence over best programmers working alone. This could possibly explain the lack of significant differences in the confidence levels of best programmers in the collaborating pairs and nominal pairs for complex tasks.

6.2 Significance of Findings

This study makes significant contributions in understanding the effectiveness of pair programming. First, collaborating pairs were found to produce software solutions of significantly better quality than second-best programmers working individually in nominal group, but of comparable quality as the solutions of best programmers working individually.

Second, best programmers among collaborating pairs develop significantly better understanding of the problem domain reflected in their task mental model compared to the second-best individuals working individually in nominal pairs. Their mental models were however comparable to that of the best programmers in the nominal groups.

Third, best programmers in the collaborating pairs have comparable levels of overall satisfaction as the best and second-best individuals in the nominal groups, while second-best programmers among collaborating pairs have higher satisfaction than best and second-best individuals in the nominal pairs.

Fourth, best programmers among the collaborating pairs have higher confidence in their solution than best programmers in nominal pair when task complexity is low, but not when it is high.

Fifth, by adopting nominal group design, this study has compared performance and perceptual outcomes of best/second-best programmers among collaborating pairs with best/second-best individual programmers in nominal pairs. Previous studies on pair programming typically compared performance of pair with the performance of average individual. Superiority of groups over average individuals is expected based on probabilities alone, as groups are more likely than individuals to have at least one exceptional programmer who could solve the problem at hand. This study has raised the bar for evaluating the effectiveness of pairs by demonstrating through a controlled experiment that pair performance may not surpass that of the best individuals.

Sixth, this study drew theoretical perspectives from social psychology and cognitive psychology and established sound theoretical underpinnings for evaluating the performance effectiveness of pair programming.

For enhancing practice of pair programming in organizations, this study has made the following contributions. First, pair programming could be used to achieve reduced defect rate and higher software quality comparable to that of the solutions of the best programmers, even when they do not have such programmers in sufficient numbers. It is said that for effective software development, developers' experience of building systems is important in both agile and traditional software development teams. While in traditional teams, about 25% to 33% of developers should be 'experienced and competent' for project success, this figure could be much smaller in the range of about 10% in teams adopting pair programming, due to the mentoring involved (Lindvall et al., 2002). Given the general shortage of exceptional software developers, pair programming could be viewed as an insurance mechanism to achieve software quality comparable to that produced by best programmers.

Second, pair programming contributes to increased satisfaction especially for the low ability members. Though satisfaction does not contribute directly to the bottom line, it is a necessary though not sufficient condition for employee productivity and retention.

6.3 Limitations of Study

The types of software developers simulated in this study were programmers engaged in systems development in organizations. The student subjects used in this study may not be truly reflective of programmers in the organizational settings. It is reasonable to speculate that university students in general and undergraduate student subjects in particular might be considerably different from real world programmers on salient features such as age, education, intellect, and maturity. Also, many complex situational variables of the work place may be absent in the sterile laboratory environment. However we have no reason to doubt that the underlying cognitive and psychological processes reflected in these findings operate very similarly in the workplace. Social psychologists have examined the effectiveness of individuals and groups on problem-solving tasks (e.g. with real world groups) and have reported similar results.

Some previous studies of pair programming had suggested that pair performance improves after an initial 'jelling' period. This was however not simulated in the present study except for providing a little adjustment time while working on a warm up task of 15 minutes duration.

The time to completion of the problem task was not measured in the study. The experimental task involved working on the programming task for a maximum of two hours. In all, the experimental session required student participation for three hours. It is reasonable to speculate that time pressure could have impacted the performance outcomes. The time of 2 hours for the main experimental task was set based on pilot testing after seeking input from the subjects. There were some programmers who could finish the task within the time limit. Having a longer experimental session was not considered possible, as it would be difficult to maintain the motivation and interest of subjects for such extended time.

6.4 Future Research Directions

Agile methodologies in general and pair programming in particular have transformed programming from being a technical endeavor to a socio-technical phenomena. Establishing the effectiveness of pair programming over individual programming is the logical first step before seeking to understand the effect of other contingent variables on this phenomenon.

Social psychologists have used performance of best individual in nominal group as the benchmark for judging the performance effectiveness of groups over individuals. This is a logical performance standard to be adopted in IS research to judge the performance effectiveness of groups in general and programming pairs in particular. As per the findings of the present study, this is difficult to achieve in pair programming. However, examining the individual differences, contextual and process variables that will help realize the 'assembly-bonus' effect in pair programming would be an interesting and rewarding research program to pursue.

Among the different individual difference variables, programmer's ability holds the maximum potential in realizing the 'assembly bonus' effect. Based on findings of current research it is speculated that there may be certain minimum threshold of programming ability required of the programmers being paired up to realize this effect.

In general, the solution demonstrability for the problem task should be high for the pair to potentially outperform best individuals. However solution demonstrability involves not just task characteristics but also person characteristics. Among the conditions for solution demonstrability articulated by Laughlin et. al. (Laughlin et al., 1991), ability could hold the key to achieving higher demonstrability and thereby pair performance superior to that of the best individuals. This is an interesting avenue to pursue with potentially huge impact on software practice.

With the advent of agile methodologies, software development is undergoing ‘extreme makeover’. Software practice is in need of robust research findings to wade through the methodological quagmire facing them. An enlightened perspective is however needed that stresses rigorous and generalizable research to avoid such deleterious consequences as wasted developer productivity, software project failures, and operational disruptions caused by defective software.

APPENDIX A

PROFESSOR’S CONSENT FOR STUDENTS’ PARTICIPATION

Professor’s Consent for Students’ Participation

Dear Professor ________:

Venu Balijepally needs your help to complete his dissertation research. He wants the students in your classes to participate in systems design/ programming experiments that he will run this semester. You are requested to provide suitable course credit to encourage student participation. The participation of the students is entirely voluntary. If any student decides not to participate in the experiment, he/she may complete an alternative written class assignment in Java. I hope you will encourage your students to participate in these experiments and support Venu’s research efforts.
Thank you for your support.

Best regards,

RadhaKanta Mahapatra
Associate Professor
INSY Ph.D. Coordinator
Department of Information Systems &
Operations Management
The University of Texas at Arlington
Box 19437
Arlington, TX 76019-0437
Phone: 817 272-3590
Fax: 817 272-5801
Email: mahapatra@uta.edu

I consent to permit students in my section to take part in the research study.

Professor’s Signature __________________________ Date __________________

Professor’s Name __________________________

Course ___________________ Section _______________________ Semester ________

APPENDIX B

INFORMED CONSENT

INFORMED CONSENT

In this study you will be asked to work on a problem involving programming in Java. You may be working individually or with another partner. You will work initially on a practice task before proceeding to work on the experimental task. An outline of the problem and some java code will be provided to you. Using the given code you should do further coding to implement the tasks given in the problem statement. After completing the task you will be asked to complete a questionnaire about your reactions to working on the task. The total experimental session will last approximately three hours.

Since you may be working with other people you may experience some emotional discomfort, similar to what you could experience in the workplace when working on tasks of this nature. Those discomforts may include fatigue, boredom or frustration when you work with other people to solve problems.

The major benefits that you will receive from participation in this research are increased familiarity with behavioral science research methods and exposure to specific software development techniques. Additionally, you will be debriefed after the experiment and will receive credit for your research participation and performance. You will also be included for winning a lottery of $50, which will be drawn at the end of all the experiments. Also an amount in cents equal to your performance grade in the task will be donated to the charity by the investigator. The benefits to the investigator are increased understanding of performance implications and reactions to programming in Java.

Your participation in this experiment is voluntary. If you decide not to participate in the experiment, you may complete an alternative written class assignment in Java. If you find any procedures objectionable you can withdraw your informed consent at any time during the experiment without any penalty, but to fulfill an alternative written assignment in Java to be provided by your course instructor. Records of your participation and any data collected will be held in strict confidence. Only your teacher will be informed of your performance scores on the task in order to give course credit.

This research study has been reviewed and approved by the University of Texas at Arlington Institutional Review Board. In the event you are injured in the course of this study, you may go to the UTA Health Service Center and be treated in the usual way, provided that you are a student currently registered at UTA. Otherwise, you may be covered under optional medical insurance that you carry. UTA does not offer any other compensation for injury.

This research is under the supervision of Dr. Radha Mahapatra. Dr. Mahapatra's office is room 502 of the Business Building, College of Business. His phone number is (817) 272-3590. If you have any questions about your rights as a subject or about a research related injury, you may contact the office of Research Compliance at 817-272-3723.

I had a chance to ask all questions regarding this study. I hereby consent to participate in the experiment and understand the above procedures.

Signature _________________________ Print Name ___________________________

Date _______________

APPENDIX C

DEBRIEFING

DEBRIEFING

In this study we were interested in examining the effectiveness of programming individually versus programming in pairs. We were also interested to study the effect of task complexity on the effectiveness of pair versus individual programming. We were looking to see if working in pairs or individually affected the understanding of the task domain, quality of programming solution, and overall task experience.

To examine this question we randomly selected which individuals are allotted to the individual condition or the pair condition, and less complex task condition or more complex task condition. The quality of your program, your overall understanding of the task domain and overall satisfaction with the task performance will be used to judge the effect of various factors on the effectiveness of pair versus individual programming. Your grades based on the quality of software developed will be normalized with respect to your experimental condition so that you are not penalized in any manner for lower performance on a complex task.

We apologize if not meeting your expectations of the level of complexity of the task given to you. We will be conducting this study with more students in the next few months. It is vitally important to us and the success of this experiment that you keep the information that you have learned here in confidence. Please do not tell anyone about this experiment. We are confident that we can trust you. Thank you.

Please sign below to indicate that you understand this debriefing and that you promise to keep what you have learned in confidence. Again, thank you very much for your participation.

Signature ________________________ Print Name_________________________

Date ___________________

APPENDIX D

INSTRUCTIONS FOR SUBJECTS

INSTRUCTIONS FOR USING THE COMPUTER

Computer Login - You may log into the computer using “student login” option with the password “**user**”. The laptops are standalone computers and are not connected to the Internet.

Launching Notepad – If the Notepad is not already open on your computer, you may open the Notepad application by double clicking the shortcut provided on the desktop.

Opening Command Window – If the command window is not already open, you may launch it by clicking Windows Start button, then clicking “Run” option and typing ‘cmd’ in the Run window.

Opening Java API documentation – If the Java API documentation window is not already open, you may launch it by double clicking the shortcut “Java 2 Platform API Specification” provided on the desktop.

Opening Java class files of warm up task and subsequent experimental task – The files for java classes are available in the floppy drive kept on your table. You may insert the floppy in the floppy drive of the laptop computer and open these files. You may use these files to develop code further as indicated in the problem statement. **You are finally required to save all the files on the floppy and leave it on the table in your room. Please do not save any data onto the hard drive.**

Compiling .java files – the code written in say Student.java file can be compiled by typing the command “javac Student.java” at the command prompt.

Warm up Task – Warm up task is provided to give you a feel of the computer and the environment. You are not expected to finish the warm up task in the 15 minutes of time provided.

Main Experimental Task – In case you are not able to complete the experimental task in the time provided (120 minutes), your performance would be still be evaluated to the extent of features and coding completed. So, you may utilize the full time provided, unless you are able to complete the task ahead of time.

ADDITIONAL INSTRUCTIONS FOR NOMINAL PAIRS

Today you will work on the following: First, you will work on a practice task involving coding in Java. The duration of the practice task is for 15 minutes. Then you will work on an experimental problem involving developing code in Java to accomplish the functionality described in the problem statement. The duration of this task is 120 minutes (2 hours). Your computer is loaded with Notepad and JDK5 for developing and compiling your program. In both the warm up task and the main task, you are provided with a program template, which may be used for developing code further. Also some basic documentation on Java syntax is provided for some of the statements you may have to use during coding. The documentation is however not exhaustive. You may access Java API documentation that is available on the computer. There is a link provided on the desktop, which may be used to launch it if it is not already opened on your computer. While you may talk your ideas loud, you may not converse with any other person during task performance. You may use paper and pencil during task performance if you will. But the program coded in Notepad and saved as ".class" files and compiled with JDK5 will be used for evaluating the performance.

The template files for various classes for the two tasks (warm up task and main experimental task) are available in the floppy provided to you. You may modify these program files in Notepad to accomplish the desired functionality. You may save these files in the floppy finally after completion of the tasks. At the end of the experiment, make sure that your program files are saved on to the floppy and the floppy left on the table in your room.

ADDITIONAL INSTRUCTIONS FOR COLLABORATING PAIRS

Today you will work on the following: First, you will work collaboratively on a practice task involving coding in Java. The duration of the practice task is for 25 15 minutes. Then you will work on an experimental problem involving developing code in Java to accomplish the functionality described in the problem statement. The duration of this task is 120 minutes (2 hours). Your computer is loaded with Notepad and JDK5 for developing and compiling your program. In both the warm up task and the main task, you are provided with a program template, which may be used for developing code further. Also some basic documentation on Java syntax is provided for some of the statements you may have to use during coding. The documentation is however not exhaustive. You may access Java API documentation that is available on the computer. There is a link provided on the desktop, which may be used to launch it if it is not already opened on your computer. While you may talk your ideas loud, and converse with your partner, you may not converse with any other person other than your partner during task performance. You may use paper and pencil during task performance if you will. But the program coded in Notepad and saved as .class files and compiled with JDK5 will be used for evaluating the performance.

The template files for various classes for the two tasks (warm up task and main experimental task) are available in the floppy provided to you. You may modify these program files in Notepad to accomplish the desired functionality. You may save these files in the floppy finally after completion of the tasks. At the end of the experiment, make sure that your program files are saved on to the floppy and the floppy left on the table in your room.

When working collaboratively on the programming task, you are required to follow the following procedure. At any time, one of you will be acting as the driver doing coding at the keyboard. The other person will act as the navigator, and assist the partner by actively inspecting the code, and looking for errors of syntax or logic. You may actively converse with each other to discuss the task at hand. Frequently at logical points during coding, you should be switching roles. The driver programmer will take on the role of the navigator and vice versa. Switching of roles can be done as often as desired, but should certainly be done once every 15 minutes. I will be coming to you to at about 15 minute intervals to remind you of switching. It is highly essential that you frequently switch your roles and take turns with the keyboard.

APPENDIX E

WARMUP TASK

BANK APPLICATION

A banking application with two classes is provided to you. The application has an Account class and an AccountTest class. The application creates two bank accounts and implements methods for doing banking transactions such as money deposit and withdrawal.

You are required to modify the classes suitably to accomplish the following:

a. A minimum balance of $200.00 for each bank account should be specified so that money cannot be withdrawn beyond the minimum balance limit.
b. The account number should be generated automatically by the program for each new account. In the present implementation the account numbers are provided manually for each new account.

```
//class that tests the Account class
//has only one method - main(..)

public class AccountTest {
  public static void main(String[] args)
  {
      //First, test the default constructor
      Account firstAccount = new Account();
      //display it - balance and account number should be 0
      System.out.println(firstAccount);  //account's toString() is
                                   //being tested by this

      //test the second constructor
      Account secondAccount = new Account(1, 500.00);
      //account number is 1 and initial balance is 500
      //display it
      System.out.println(secondAccount);

      //do some transactions on both
      firstAccount.deposit();
      firstAccount.withdraw();
      secondAccount.deposit();
      secondAccount.withdraw();

      //display new values
      System.out.println(firstAccount);
      System.out.println(secondAccount);
  } //end of main
} //end of class
```

```
import java.util.*;

public class Account
{
   //This section defines the attributes. The following attributes are
   //typical of an account
   //1. accountNumber - this is declared as an integer. This establishes
   //an account object's identity because it is a unique identifier. This
   //should ideally be set automatically
   //2. balance - this is declared as a double. Each account object has its own
   //copy of balance. Therefore, balance is an instance variable.

   private int accountNumber;
   private double balance;

   //This section defines the methods for the class. We will have the following
   //methods:
   //Constructors to create an account object. There will be two of them:
   //First constructor: public Account() { ..} is the default constructor
   //Second constructor: public Account(int accountNumber, double balance) {,….}
   //The second constructor takes an account number and a balance as
   //parameters
   //withdraw() - method prompts the user for amount and deducts it from balance
   //deposit() - method prompts the user for amount and adds it to balance
   //getAmount(String prompt) a helper method used by withdraw() and deposit()
   //to get an amount to be withdrawn or deposited
   //toString() - method that converts an account to a string

   //default constructor
   public Account()
   {
      accountNumber = 0;
      balance = 0.0;
   }

   //second constructor

   public Account(int accountNumber, double balance)
   {
      this.accountNumber = accountNumber;
      this.balance = balance;
   }
   //transactions to be performed by account
   public void withdraw()
   {
      double withdrawAmount;
      withdrawAmount = getAmount("Enter amount to withdraw: ");
      balance = balance - withdrawAmount;
      System.out.println("New balance is: " + balance);
   }

   public void deposit()
   {
      double depositAmount;
      depositAmount = getAmount("Enter amount to be deposited: ");
      balance = balance + depositAmount;
      System.out.println("New balance is: " + balance);
   }

   public double getAmount(String prompt)
   {
      System.out.print(prompt);
      Scanner input = new Scanner(System.in);
      return input.nextDouble();
   }

   //toString() to convert an account object to a string
   public String toString()
   {
     return "Account# " + accountNumber + "\n" +
          "Balance: " + balance + "\n";
   }
                                                }
```

APPENDIX F

EXPERIMENTAL TASK - LOW COMPLEXITY

STUDENT GRADES

You are to complete an application that involves a Student class. An initial effort at creating the Student class resulted in Student.java. Complete the methods in the Student class. A brief description of these methods is shown below:

***Student(String id, String name)* –** constructor for the Student class. The two parameters are for initializing the student's id and name

***getScores()* –** A student has two exam scores. This method prompts the user for the score on each of the exams.

***computeAverage()* –** This method computes the average of the two exam scores and returns the value

***computeGrade()* –** The purpose of this method is to compute a letter grade based on the average of the two exam scores. The grade calculation is as follows:
average score >= 90.0 is an 'A'
average score between 80.0 and 89.99 is a 'B'
average score between 70.0 and 79.99 is a 'C'
average score between 60.0 and 69.99 is a 'D'
average score < 60.0 is an 'F'
Notice that the method returns a character.

***toString()* –** This method is used to convert a student object to a String. It returns a Student record as a string that contains the id, name, average score, and letter grade.

After you implement these methods, complete the StudentTest.java application. The application creates three students objects and adds them to the array. It then gets the exam scores for each of the students. Finally, display the three students in the following format:

```
ID      Name  Average       Grade
```

Example:
Assume that we have the following student objects:
Student 1: studentID = "111", name = "Doug Walters", examOneScore = 90.0, examTwoScore = 80.0
Student 2: studentID = "222", name = "Garry Sobers", examOneScore = 92.0, examTwoScore = 100.0
Student 3: studentID = "333", name = "Viv Richards", examOneScore = 80.0, examTwoScore = 78.0

The output is shown below:

```
111   Doug Walters      85.00      B
222   Garry Sobers      96.00      A
333   Viv Richards      79.00      C
```

```
public class Student {
   private String studentID;
   private String name;
   private double examOneScore;
   private double examTwoScore;

   //constructor - takes two parameters, the first is the id of the
   //student and the second is the name of the student
   //to be written


   //method to read in two scores. Get the two exam scores and assign them
   //to examOneScore and examTwoScore
   public void getScores()
   {

   }

   //compute the average of the scores
   public double computeAverage()
   {
    //return the average of examOneScore and examTwoScore
   }

   //method to compute grade - assume the following:
   //An average >= 90.0 is an A
   //average between 80 and 89 is a B, between 70 and 79 is a C
   //60-69 is a D and anything below 60 is an F
   //The method returns the grade as a character
   public char computeGrade()
   {

   }

   //override the toString method to display a student object as a String
   //The returned string contains the id, name, average score, and the
   //letter grade followed by a newline character
   public String toString()
   {
   }
}

public class StudentTest {
   public static void main(String[] args)
   {
       //The application creates three student objects and
       //stores them in an array called students. The declaration
       //of the array is given below
       Student[] students = new Student[3]; //can store 3 student objects

       //create 3 student objects and add them to the array


       //use a for loop to iterate through the array to get the
       //exam scores for each of the students



       //display the student objects in the array
   }
}
```

APPENDIX G

EXPERIMENTAL TASK – HIGH COMPLEXITY

MOVIE RENTAL APPLICATION

Summary of task: Provide a way to keep a list of movies and a way to display a movie, given its title. Assume that movie titles are unique.

An incomplete application is available to fulfill this task. The application displays three options: 1) Add Movie; 2) Display Movie; and 3) Exit. These options have not been implemented. It is your responsibility to implement them. A brief description of each option follows.
Add Movie: This option should prompt the user for a title, create a movie object using the title, and then add the movie to a list of movies maintained in another class.
Display Movie: This option asks for a title, retrieves the movie and displays its title. If the given title is not found, it should display a message to that effect.
***Exit* –** exits the application.

Modify the Movie class to include the following instance variables:
***Category* –** which tells you what type of movie it is. This could be “Comedy”, “Mystery”, “Western”, “Classic”, “Action”, etc.
***Rating* –** indicates the rating of the movie (e.g. PG, PG-13, R, NC-17, etc.)
Write appropriate constructor(s) and methods to handle these changes.

Modify the addMovie method in the user interface to reflect these changes. In other words, the method should now get title, category, and rating, and then create a movie object with those values.
Modify the displayMovie method in the user interface to display title, category, and rating of the retrieved movie.

Add a menu option called “***About***”. Make it the 3rd option and make ***Exit*** the fourth option. The ***About*** option displays a message that reads as follows:

> Movie list application programmed by ***your name***

Hashtable description

A hashtable may be used to store key-value pairs. You may think of the key as the primary key of a relational table. A value may be retrieved using the key. The following example shows how a hashtable can be used.

```
Hashtable employees = new Hashtable(); //creates a hashtable called employees

//in this table, employee id will be used as the key, and the corresponding employee
//object will be the value associated with that key
Employee e = new Employee("111-11-1111", "Doug Walters", "102 Oak St");

//here e is an employee object with id "111-11-1111", name of "Doug Walters", and the
//last parameter is the address
employees.put("111-11-1111", e); //inserts the employee object in the hashtable

Employee a = new Employee("999-99-9999", "Garry Sobers", "1111 Hemlock St");
Employees.put("999-99-9999", a); //inserts a second employee object in the hashtable
```

To retrieve an employee object, we must provide the id. So, to retrieve and display an employee object, you might do something like this….

```
String id = JOptionPane.showInputDialog("Enter employee id: ");
Employee temp = (Employee) employees.get(id); //use the id to retrieve the employee
//object
//Notice that the hashtable returns an Object which must be cast to what you need – in
//this case an Employee
```

The two main methods are:
put – takes two arguments, the key and the value, both of which should be objects
get – takes one argument, the key that it uses to retrieve the Object

Other Notes:
JOptionPane.showMessageDialog is used to display a message in a window. A sample is shown below:

```
String display = "Hello there!!";
JOptionPane.showMessageDialog(null, display); //notice that the first argument here is
//null   and the second is the string to be displayed
```

JOptionPane.showInputDialog is used to retrieve data as a String from the keyboard. Here is how it is used:

```
String name = JOptionPane.showInputDialog(“Enter your name: “); //this pops up a
//window with “Enter your name:” as the prompt and displays a text box in which the
//user can input a response. When the user selects the OK button, the response is
returned //as a String

public class Movie {
   private String title;

   public Movie(String title)
   {
         this.title = title;
   }

   public String getTitle()
   {
         return title;
   }
}

public class Application {
   public static void main(String[] args)
   {
      UserInterface u = new UserInterface();
      u.run();
      System.exit(0);
   }
}

import java.util.*;

public class MovieCollection {
   private Hashtable movies;

   public MovieCollection()
   {
         movies = new Hashtable();
   }

   public void add(Movie m)
   {
      movies.put(m.getTitle(), m);
   }
}
```

```
import java.util.*;
import javax.swing.*;

public class UserInterface {

   public int menu()
   {
      String display = "1. Add Movie\n" +
                   "2. Display Movie\n" +
                   "3. Exit\n" +
                   "Enter selection: ";
      return Integer.parseInt(JOptionPane.showInputDialog(display));
   }

   public void run()
   {
      int choice = menu();
      while ( choice != 3 )
      {
         switch (choice)
         {
            case 1: addMovie();
                  break;
            case 2: displayMovie();
                  break;
            default: JOptionPane.showMessageDialog(null, "Inavlid choice");
         }

         choice = menu();
      }
   }

   public void addMovie()
   {
      JOptionPane.showMessageDialog(null,
               "You have to implement this");
   }

   public void displayMovie()
   {
      JOptionPane.showMessageDialog(null,
               "You have to implement this");
   }
}
```

APPENDIX H

TASK MENTAL MODEL CONSTRUCT

GUIDELINES FOR ASSESSING STRENGTHS OF RELATIONSHIPS BETWEEN CONCEPTS

(Guidelines that may be used to answer questions related to task mental model)

Relationships between classes

In general, a class A has a relatively high strength of relationship with class B if they are involved in an

- *association*,
- an *aggregation* -whole-part relationship,
- a *composition* - a stronger whole-part relationship in which the part:
  - can belong to only one whole and
  - cannot exist independent of the whole,
- or a *dependency* (e.g. class A uses an object of type/class B either as a parameter or as a local reference in a method.

Of these, composition is semantically stronger than aggregation which, in turn, is stronger than association.

Relationships between a method and a class

In deciding on the strength of the relationship between a method, say foo(), and a class, say A, one may ask the following questions:

- Is foo() within class A?
- Does foo() rely ONLY on data/attributes/references and/or methods in class A in order to accomplish its task?
- Does foo() rely on data and/or methods in other classes to get its job done?
- The criticality of the method foo() in class A?

Relationships between methods

A method, say foo(), is strongly related to another, say do(),

- if foo() *includes* the behavior of do() (that is, foo() always calls do()).
- If foo() calls do() under certain conditions, then the strength of the relationship is slightly lower. In this case, we say that do() extends the behavior of foo().

In addition to *include* and *extend*, two methods may have a somewhat strong relationship because of the fact that they both belong to the same class.

Relationships between methods and attributes/references

In assessing the strength of the relationship between a method (foo()) and an attribute/reference (say, a), ask yourselves the following questions:

- Does foo() need the attribute or reference a?
- Is the attribute in the parameter list of the method?
- Is it a local variable in the method?
- Is it a class or instance variable belonging to the class in which foo() is defined?
- Is the attribute or reference that foo() uses from a different class?

Does foo() access the attribute or reference indirectly (that is, through a public method)?

TASK MENTAL MODEL QUESTIONS FOR STUDENT GRADES APPLICATION (LOW COMPLEXITY)

Please answer the following questions based on your understanding of the programming task that you had just completed. Please indicate your perception of how closely related are the following classes and methods of the programming task.
Use a rating scale from 1 – Not at all related to 7 – Highly related

Student class	StudentTest class	1	2	3	4	5	6	7
Student class	getScores ( )	1	2	3	4	5	6	7
Student class	computeAverage ( )	1	2	3	4	5	6	7
Student class	computeGrade ( )	1	2	3	4	5	6	7
Student class	toString ( )	1	2	3	4	5	6	7
Student class	main ( )	1	2	3	4	5	6	7
StudentTest class	getScores ( )	1	2	3	4	5	6	7
StudentTest class	computeAverage ( )	1	2	3	4	5	6	7
StudentTest class	computeGrade ( )	1	2	3	4	5	6	7
StudentTest class	toString ( )	1	2	3	4	5	6	7
StudentTest class	main ( )	1	2	3	4	5	6	7
getScores ( )	computeAverage ( )	1	2	3	4	5	6	7
getScores ( )	computeGrade ( )	1	2	3	4	5	6	7
getScores ( )	toString ( )	1	2	3	4	5	6	7
getScores ( )	main ( )	1	2	3	4	5	6	7
computeAverage ( )	computeGrade ( )	1	2	3	4	5	6	7
computeAverage ( )	toString ( )	1	2	3	4	5	6	7
computeAverage ( )	main ( )	1	2	3	4	5	6	7
computeGrade ( )	toString ( )	1	2	3	4	5	6	7
computeGrade ( )	main ( )	1	2	3	4	5	6	7
toString ( )	main ( )	1	2	3	4	5	6	7

The above questions capture the perceived relationships between the following seven classes/methods of the student grades application:

1. Student class
2. StudentTest class
3. getScores()
4. computeAverage()
5. computeGrade()
6. toString()
7. main()

Two experts jointly identified these classes/methods as the most important concepts of the problem task. The perceived strength of relationships between these concepts reflects the subject's understanding of the programming task. Path Finder technique was used to create network representing the mental model for the subject. Task mental model was calculated as the similarity of the Path Finder network of the subject with that of the expert, using Netsim function of the Path Finder software.

TASK MENTAL MODEL QUESTIONS FOR MOVIE RENTAL APPLICATION (HIGH COMPLEXITY)

Please answer the following questions based on your understanding of the programming task that you had just completed. Please indicate your perception of how closely related are the following classes and methods of the programming task.
Use a rating scale from 1 – Not at all related to 7 – Highly related

Application	Movie	1	2	3	4	5	6	7
Application	MovieCollection	1	2	3	4	5	6	7
Application	UserInterface	1	2	3	4	5	6	7
Application	getTitle ( )	1	2	3	4	5	6	7
Application	add ( )	1	2	3	4	5	6	7
Application	run ( )	1	2	3	4	5	6	7
Application	addMovie ( )	1	2	3	4	5	6	7
Application	displayMovie ( )	1	2	3	4	5	6	7
Application	menu ( )	1	2	3	4	5	6	7
Movie	MovieCollection	1	2	3	4	5	6	7
Movie	UserInterface	1	2	3	4	5	6	7
Movie	getTitle ( )	1	2	3	4	5	6	7
Movie	add ( )	1	2	3	4	5	6	7
Movie	run ( )	1	2	3	4	5	6	7
Movie	addMovie ( )	1	2	3	4	5	6	7
Movie	displayMovie ( )	1	2	3	4	5	6	7
Movie	menu ( )	1	2	3	4	5	6	7
MovieCollection	UserInterface	1	2	3	4	5	6	7
MovieCollection	getTitle ( )	1	2	3	4	5	6	7
MovieCollection	add ( )	1	2	3	4	5	6	7
MovieCollection	run ( )	1	2	3	4	5	6	7
MovieCollection	addMovie ( )	1	2	3	4	5	6	7
MovieCollection	displayMovie ( )	1	2	3	4	5	6	7

MovieCollection	menu ( )	1	2	3	4	5	6	7
UserInterface	getTitle ( )	1	2	3	4	5	6	7
UserInterface	add ( )	1	2	3	4	5	6	7
UserInterface	run ( )	1	2	3	4	5	6	7
UserInterface	addMovie ( )	1	2	3	4	5	6	7
UserInterface	displayMovie ( )	1	2	3	4	5	6	7
UserInterface	menu ( )	1	2	3	4	5	6	7
getTitle ( )	add ( )	1	2	3	4	5	6	7
getTitle ( )	run ( )	1	2	3	4	5	6	7
getTitle ( )	addMovie ( )	1	2	3	4	5	6	7
getTitle ( )	displayMovie ( )	1	2	3	4	5	6	7
getTitle ( )	menu ( )	1	2	3	4	5	6	7
add ( )	run ( )	1	2	3	4	5	6	7
add ( )	addMovie ( )	1	2	3	4	5	6	7
add ( )	displayMovie ( )	1	2	3	4	5	6	7
add ( )	menu ( )	1	2	3	4	5	6	7
run ( )	addMovie ( )	1	2	3	4	5	6	7
run ( )	displayMovie ( )	1	2	3	4	5	6	7
run ( )	menu ( )	1	2	3	4	5	6	7
addMovie ( )	displayMovie ( )	1	2	3	4	5	6	7
addMovie ( )	menu ( )	1	2	3	4	5	6	7
displayMovie ( )	menu ( )	1	2	3	4	5	6	7

The above questions capture the perceived relationships between the following ten classes/methods of the movie rental application:

1. Application class
2. Movie class
3. MovieCollection class
4. UserInterface class
5. getTitle ( )
6. add ( )
7. run ( )

8. addMovie ( )
9. displayMovie ( )
10. menu ( )

Two experts jointly identified these classes/methods as the most important concepts of the problem task. The perceived strength of relationships between these concepts reflects the subject's understanding of the programming task. Path Finder technique was used to create network representing the mental model for the subject. Task mental model was calculated as the similarity of the Path Finder network of the subject with that of the expert, using Netsim function of the Path Finder software.

APPENDIX I

QUESTIONNAIRE FOR MEASURING PERCEPTUAL CONSTRUCTS AND DEMOGRAPHIC VARIABLES

NOMINAL PAIRS

1. Please indicate how you worked on the programming task today

a. Individually b. With a partner

2. How do you feel about the main programming task you performed today?

Very Easy	1	2	3	4	5	6	7	Very Difficult
Very Simple	1	2	3	4	5	6	7	Very
Complex								

3. How do you feel about the main programming task you performed, as compared to the warm up task?

Very Easy	1	2	3	4	5	6	7	Very Difficult
Very Simple	1	2	3	4	5	6	7	Very
Complex								

4. How do you feel about your overall experience of working on the programming task today?

Very Dissatisfied	1	2	3	4	5	6	7	Very Satisfied
Very Displeased	1	2	3	4	5	6	7	Very Pleased
Very Frustrated	1	2	3	4	5	6	7	Very Contended
Absolutely Terrible	1	2	3	4	5	6	7	Absolutely
Delighted								

5. Imagine that we selected ten results at random from those who participated in this task. How would your performance rank among these ten results?

Worst results out of Ten.	1	2	3	4	5	6	7	8	9	10	Best results out of Ten

6. How do you feel about the quality of your programming solution?

Not at all Confident	1	2	3	4	5	6	7	Very Confident
Not at all Certain	1	2	3	4	5	6	7	Very Certain

7. How do you feel about your own performance on the programming task today?

Very Dissatisfied	1	2	3	4	5	6	7	Very Satisfied
Very Displeased	1	2	3	4	5	6	7	Very Pleased
Very Frustrated	1	2	3	4	5	6	7	Very Contended
Absolutely Terrible	1	2	3	4	5	6	7	Absolutely
Delighted								

COLLABORATING PAIRS

1. Please indicate how you worked on the programming task today

a. Individually b. With a partner

2. How do you feel about the main programming task you performed today?

Very Easy	1	2	3	4	5	6	7	Very Difficult
Very Simple	1	2	3	4	5	6	7	Very Complex

3. How do you feel about the main programming task you performed, as compared to the warm up task?

Very Easy	1	2	3	4	5	6	7	Very Difficult
Very Simple	1	2	3	4	5	6	7	Very Complex

4. How do you feel about your overall experience of working on the programming task today?

Very Dissatisfied	1	2	3	4	5	6	7	Very Satisfied
Very Displeased	1	2	3	4	5	6	7	Very Pleased
Very Frustrated	1	2	3	4	5	6	7	Very Contended
Absolutely Terrible	1	2	3	4	5	6	7	Absolutely Delighted

5. Imagine that we selected ten results at random from those who participated in this task. How would your group performance rank among these ten results?

Worst results out of Ten.	1	2	3	4	5	6	7	8	9	10	Best results out of Ten

6. How do you feel about the quality of your programming solution?

Not at all Confident	1	2	3	4	5	6	7	Very Confident
Not at all Certain	1	2	3	4	5	6	7	Very Certain

7. How do you feel about <u>your own performance</u> on the programming task today?

Very Dissatisfied	1	2	3	4	5	6	7	Very Satisfied
Very Displeased	1	2	3	4	5	6	7	Very Pleased
Very Frustrated	1	2	3	4	5	6	7	Very Contended
Absolutely Terrible Delighted	1	2	3	4	5	6	7	Absolutely

8. How do you feel about the performance of your group on the programming task today?

Very Dissatisfied	1	2	3	4	5	6	7	Very Satisfied
Very Displeased	1	2	3	4	5	6	7	Very Pleased
Very Frustrated	1	2	3	4	5	6	7	Very Contended
Absolutely Terrible Delighted	1	2	3	4	5	6	7	Absolutely

DEMOGRAPHIC AND BACKGROUND VARIABLES

1. Please circle your gender: Male Female

2. Please indicate your age on your last birthday __________

3. Please indicate your country of citizenship __________

4. Is English your first language? Yes No

5. What are the programming languages you are familiar with other than Java? (Tick all that apply)
   a) C b) C++ c) Dot Net d) C# e) Visual Basic
   f) HTML g) Cobol h) None i) Others (Name others)
   ____

6. Indicate number of years of your programming experience in any programming language?
   A. 0 – 1 B. 1 - 2 C. 2-4 D. > 4

7. Indicate number of years of your programming experience in Java?
   A. 0 – 1 B. 1 - 2 C. 2-4 D. > 4

APPENDIX J

GRADING SHEETS

GRADING SHEET FOR SIMPLE TASK

Session _____ Room ________

SNo	Description	Max Points	Points Scored	Remarks
I	**Student class evaluation**			
A	***Student(String id, String name)*** • Proper parameters • Visibility: public • No return type • Method name: same as class name (i.e. Student) • Should only initialize id and name of student • For name, this.name = name should be used	**10**		
B	***getScores()*** • visibility: public • return type: void • assignments	**20**		
	Give 5 additional points if the getScores() method is generic. That is, it asks the user for the number of courses and then gets that many exam scores.	5		
C	***computeAverage()***	**10**		
D	***computeGrade()***	**12**		
E	***toString()***	**10**		
II	**Student Test Class Evaluation:**			
A	***Creating three student objects*** (to be put in the array called students):			
A1	Option 1: Solution is hard coded. E.g. students[0] = new Student("111", "Viv Richards");	**9**		

A2	Option 2:
	Generic solution using loop: Example..
	for(int i = 0; i < students.length; i++)
	{
	//code for initializing
	}
		14		

B	***Getting the exam scores for each***			
B1	Option 1: No loop	**9**		
B2	Option 2: With loop (generalizable):	14		
C	***Displaying the three students:***			
C1	Option 1: Hard coded solution (no loop, don't use toString() concept, etc.): 20 Points Use the following criteria for evaluation: • Output format • Accuracy of results	**20**		
C2	Option2: Generic solution. They use a loop and System.out.println(studentObject).	25		
III	**Going beyond requirements:** Maintainability considerations – • appropriate comments • indentation.	5		
	Total	**125**		

GRADING SHEET FOR COMPLEX TASK

Session _____ Room ________

SNo	Description	Max Points	Points Scored	Remarks
I	**Movie class evaluation**			
A	Add 2 String variables for **category** and **rating** If data type is wrong, deduct 3 points	**5**		

B	Add a constructor:
	public Movie(String title, String category, String rating)
	{
	this.title = title;
	this.category = category;
	this.rating = rating;
	}
	Check for: return type (none for constructor), variable names – deduct some points if they are incorrect.
		10		

C	Add gettor/accessor methods for category and rating	**5**		
D	Additional points if toString() is implemented	5		
II	**Display of menu** – if correct	**5**		
III	**AddMovie class evaluation**			
A	Get title, category, rating (5 points each)	**15**		
B	Create a movie object:			
B1	Make sure they have a reference to MovieCollection in the UserInterface class **OR** They could handle this by making the add and get methods static in the MovieCollection class	**5**		
B2	Movie object created with proper parameters	**10**		
C	Adding the movie – proper call to MovieCollection's "add" method	**5**		
!V	**Implementation of Display Movie Option**			
A	Adding a "get" method to the MovieCollection that takes the title as a parameter and return a movie object	**10**		
A1	if they display "Title not found" when movie is not found.	**5**		
A2	Additional points if they use exceptions	10		
B	Getting/reading the title	**5**		
C	Displaying the movie	**10**		
C1	Additional points if toString() is implicitly used	5		
V	**"About" option implementation**	**5**		
VI	**Exit – if correct**	**5**		
VII	**Going beyond requirements:** Maintainability considerations – • appropriate comments • indentation.	5		
	Total	**125**		

REFERENCES

Adelson, B., & Soloway, E. (1985). The Role of Domain Experience in Software Design. *IEEE Transactions on Software Engineering, 11*, 1351-1360.

AgileManifesto. (2001). *Agile Manifesto*. Retrieved 12/20/04 Website: http://www.agilemanifesto.org/

Aiello, J. R., & Douthitt, E. A. (2001). Social Facilitation from Triplett to Electronic Performance Monitoring. *Group Dynamics, 5*(3), 163-180.

Aiello, J. R., & Kolb, K. J. (1995). Electronic Performance Monitoring and Social Context: Impact on Productivity and Stress. *Journal of Applied Psychology, 80*(3), 339-353.

Aiello, J. R., & Svec, C. M. (1993). Computering Monitoring of Work Performance: Extending the Social Facilitation Framework to Electronic Presence. *Journal of Applied Social Psychology, 23*, 537-548.

Allison, S., T., & Messick, D. M. (1985). The Group Attibution Error. *Journal of Experimental Social Psychology, 21*, 563-579.

Allwood, C. M., & Granhag, P. A. (1996). Realism in Confidence Judgments as a Function of Working in Dyads or Alone. *Organizational Behavior & Human Decision Processes, 66*(3), 277-289.

Barker, J. R. (1993). Tightening the Iron Cage: Concertive Control in Self-Managing Teams. *Administrative Science Quarterly, 38*(3), 408-437.

Baron, R. S. (1986). Distraction-Conflict Theory: Progress and Problems. *Advances in Experimental Social Psychology, 19*, 1-36.

Bass, B. M., Pryer, M. W., Gaier, E. L., & Flint, A. W. (1958). Interacting Effects of Control Motivation, Group Practice and Problem Difficulty on Attempted Leadership. *Journal of Abnormal and Social Psychology, 56*, 352-358.

Baumeister, R. F. (1982). A Self Presentational View of Social Phenomena. *Psychological Bulletin, 91*, 3-26.

Beck, K. (1999). Embracing Change with Extreme Programming. *IEEE Computer, 32*(10), 70-77.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison Wesley Longman Inc.

Besser, T. L. (1995). Rewards and Organizational Goal Achievement: A Case Study of Toyota Motor Manufacturing in Kentucky. *Journal of Management Studies, 32*(3), 383.

Bhattacherjee, A. (2001). Understanding Information Systems Continuance: An Expectation-Confirmation Model. *MIS Quarterly, 25*(3), 351-370.

Boehm, B. (2002). Get Ready for Agile Methods with Care. *IEEE Computer, 35*(1), 64-69.

Boland, R. J., & Tenkasi, R. V. (1995). Perspective Making and Perspective Taking in Communities of Knowing. *Organization Science, 6*(4), 350-372.

Bond, C. F. (1982). Social Facilitation: A Self-Presentational View. *Journal of Personality and Social Psychology, 42*, 1042-1050.

Bond, C. F., & Titus, L. J. (1983). Social Facilitation: A Meta-Analysis of 241 Studies. *Psychological Bulletin, 94,* 265-292.

Bono, J. E., Boles, T. L., Judge, T. A., & Lauver, K. J. (2002). The Role of Personality in Task and Relationship Conflict. *Journal of Personality, 70*(2), 311-344.

Brickner, M. A., Harkins, S. G., & Ostrom, T. M. (1986). Effects of Personal Involvement: Thought-Provoking Implications for Social Loafing. *Journal of Personality and Social Psychology, 51*, 763-770.

Brown, T. M., & Miller, C. E. (2000). Communication Networks in Task-Performing Groups: Effects of Task Complexity, Time Pressure, and Interpersonal Dominance. *Small Group Research, 31*(2), 131-157.

Bruce, L. (2004, Jun 4). Canadian Bank Tackles Processing Glitch. *Computer World*.

Bystrom, K., & Jarvelin, K. (1995). Task Complexity Affects Information-Seeking and Use. *Information Processing & Management, 31*(2), 191-213.

Campbell, D. J. (1988). Task Complexity: A Review and Analysis. *Academy of Management Review, 13*(1), 40-52.

Campbell, D. J., & Gingrich, K. F. (1986). The Interactive Effects of Task Complexity and Participation on Task-Performance - a Field Experiment. *Organizational Behavior and Human Decision Processes, 38*(2), 162-180.

Campion, M. A., Medsker, G. J., & Higgs, C. A. (1993). Relations Between Work Group Characteristics and Effectiveness: Implications for Designing Effective Work Groups. *Personnel Psychology, 46*(4), 823-850.

Carey, J. M., & Kacmar, C. J. (1997). The Impact of Communication Mode and Task Complexity on Small Group Performance and Member Satisfaction. *Computers in Human Behavior, 13*(1), 23-49.

Carroll, J. M., & Olson, J. R. (1988). Mental Models in Human-Computer Interaction. In Halender, M. (Ed.), *Handbook of Human-Computer Interaction* (pp. 45-65). North Holland: Elsevier Science Publishers.

Chalykoff, J., & Kochan, T. A. (1989). Computer-Aided Monitoring: Its Influence on Employee Job Satisfaction and Turnover. *Personnel Psychology, 42*(4), 807-834.

Chang, S. E. (2005). Computer Anxiety and Perception of Task Complexity in Learning Programming-Related Skills. *Computers in Human Behavior, 21*(5), 713-728.

Charette, R. (2001). The Decision is In: Agile versus Heavy Methodologies. *Cutter Consortium e-Project Management Advisory Service, 2*(19).

Chlewinski, Z. (1975). Cognitive Conservatism and Radicalism in Individual and Group Decisions. *Polish Psycholgical Bulletin, 6*, 139-146.

Cockburn, A. (2002). Agile Software Development Joins the "Would-Be" Crowd. *Cutter IT Journal, 15*(1), 6-12.

Cockburn, A., & Highsmith, J. (2001). Agile Software Development 2: The People Factor. *IEEE Computer, 34*(11), 131-133.

Cockburn, A., & Williams, L. (2001). The Costs and Benefits of Pair Programming. In Succi, G. & Marchesi, M. (Eds.), *Extreme Programming Explained* (pp. 223-248): Addison Wesley.

Collins, A., & Gentner, D. (1987). How People Construct Mental Models. In Holland, D. & Quinn, N. (Eds.), *Cultural Models in Language and Thought* (pp. 243-265). Cambridge: Cambridge University Press.

Collins, B. E., & Guetzkow, H. G. (1964). *Social Psychology of Group Processes for Decision-Making*. New York, NY: Wiley.

Cottrell, N. B. (1972). Social Facilitation. In McClintock, C. G. (Ed.), *Experimental Social Psychology* (pp. 185-236). New York: Holt.

Cottrell, N. B., Wack, D. L., Sekerak, G. J., & Rittle, R. H. (1968). Social Facilitation of the Dominant Responses by the Presence of an Audience and the Mere Presence of Others. *Journal of Personality and Social Psychology, 9*, 245-250.

Cusumano, M., MacCormack, A., Kemerer, C. F., & Crandall, B. (2003). Software Development Worldwide: The State of the Practice. *IEEE Software, 20*(6), 28-34.

Dailey, R. C. (1978). Perceived Group Variables as Moderators of the Task Characteristics - Individual Performance Relationship. *Journal of Management, 4*(2), 69.

Davis, J. H. (1969). *Group Performance*. Reading, MA: Addison-Wesley.

Delaney, M. M., Foroughi, A., & Perkins, W. (1997). Am Empirical Study of the Efficacy of a Computerized Negotiation Support System. *Decision Support Systems, 20*, 185-197.

DeMatteo, J. S., Eby, L. T., & Sundstrom, E. (1998a). Team-based rewards: Current empirical evidence and directions for future research. In *Research in Organizational Behavior, Vol 20, 1998* (Vol. 20, pp. 141-183).

DeMatteo, J. S., Eby, L. T., & Sundstrom, E. (1998b). Team-Based Rewards: Current Empirical Evidence and Directions for Future Research. In Staw, B. M. & Cummings, L. L. (Eds.), *Research in Organizational Behavior* (Vol. 20, pp. 141-183): Elsevier Science/JAI Press.

Domino, M. A. (2004). *Three Studies of Problem Solving in Collaborative Software Development.* Unpublished Ph.D. Dissertation, University of South Florida, Tampa, FL.

Douthitt, E. A., & Aiello, J. R. (2000). *The Impact of Computer Monitoring and Negative Affectivity on Task Performance and Satisfaction.* Paper presented at the Annual Meeting of the Academy of Management.

Dunegan, K. J., Uhl-Bien, M., & Duchon, D. (2002). LMX and Subordinate Performance: The Moderating Effects of Task Characteristics. *Journal of Business & Psychology, 17*(2), 275-285.

Durling, R., & Schick, C. (1976). Concept Attainment by Pairs and Individuals as a Function of Vocalization. *Journal of Educational Psychology, 68*, 83-91.

English, A. (2002). Extreme Programming: It's Worth a Look. *IT Pro, 4*, 48-50.

Eysenck, M. W., & Keane, M. T. (2000). *Cognitive Psychology: A Student's Handbook*. Philadelphia, PA: Psychology Press.

Faust, W. L. (1959). Group versus Individual Problem-Solving. *Journal of Abnormal & Social Psychology, 59,* 68-72.

Felsental, D. S. (1979). Group versus Individual Gambling Behavior: Reexamination and Limitation. *Behavioral Science, 24*, 334-345.

Fisher, C. D. (2003). Why do Lay People Believe that Satisfaction and Performance are Correlated? Possible Sources of a Commonsense Theory. *Journal of Organizational Behavior, 24*(6), 753.

Flor, N. V., & Hutchins, E., L. (1991). Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance. In Koenemann-Belleveau, J., Moher, T. G. & Robertson, S. P. (Eds.), *Proceedings of the fourth annual workshop on emprical studies of programmers* (pp. 36-63). Norwood, NJ: Ablex Publishing.

Forgas, J. P. (1981). Responsibility Attribution by Groups and Individuals: The Effects of the Interaction Episode. *European Journal of Social Psychology, 11,* 87-99.

Forsyth, D. R. (1999). *Group Dynamics* (Third ed.). Belmont, CA: Wadsworth Publishing.

Fox, D. J., & Lorge, I. (1962). The Relative Quality of Decisions Written by Individuals and by Groups as the Available Time for Problem Solving is Increased. *Journal of Social Psychology, 57*(1), 227-242.

Garibaldi, A. M. (1979). Affective Contributions of Cooperative and Group Goal Structures. *Journal of Educational Psychology, 71,* 788-794.

Geen, R. G. (1977). The Effects of Anticipation of Positive and Negative Outcomes on Audience Anxiety. *Journal of Consulting and Clinical Psychology, 45*(715-716).

Goldman, F. W., & Goldman, M. (1981). The Effects of Dyadic Group Experience in Subsequent Individual Performance. *Journal of Social Psychology, 115,* 83-88.

Goodman, P. S. (1986). Impact of Task and Technology on Group Performance. In Goodman, P. S. (Ed.), *Designing Effective Work Groups* (pp. 120-167). San Francisco, CA: Jossey-Bass.

Hackman, J. R. (1968). Effects of Task Characteristics on Group Products. *Journal of Experimental Social Psychology, 4*(2), 162-187.

Hackman, J. R., & Morris, C. G. (1976). Group Tasks, Group Interaction Processes and Group Performance Effectiveness: A Review and Proposed Integration. In Berkowitz, L. (Ed.), *Advances In Experimental Social Psychology* (Vol. 8, pp. 45-99). New York: Academic Press.

Hackman, J. R., Oldham, G., Janson, R., & Purdy, K. (1975). A New Strategy for Job Enrichment. *California Management Review, 17*(4), 57.

Hair, J. F., Anderson, R. E., Tatham, R. L., & Black, W. C. (1998). *Multivariate Data Analysis* (Fifth ed.). Upper Saddle River, NJ: Prentice-Hall.

Hare, A. P. (1995). Individual Versus Group. In Hare, A. P., Blumberg, H. H., Davies, M. F. & Kent, M. V. (Eds.), *Small Group Research: A Handbook* (pp. 261-270). Norwood, NJ: Ablex Publishing Corp.

Harkins, S. G. (1987). Social Loafing and Social Facilitation. *Journal of Experimental Social Psychology, 23,* 1-18.

Harkins, S. G., & Jackson, J. M. (1985). The Role of Evaluation in Eliminating Social Loafing. *Personality and Social Psychology, 11*, 575-584.

Harkins, S. G., & Petty, R., E. (1982). Effects of Task Difficulty and Task Uniqueness on Social Loafing. *Journal of Personality and Social Psychology, 43*(6), 1241-1229.

Harkins, S. G., & Szymanski, K. (1989). Social Loafing and Group Evauation. *Journal of Personality and Social Psychology, 56*, 934-941.

Hart, J. W., Bridgett, K. J., & Karau, S. J. (2001). Coworker Abiity and Effort as Determinants of Individual Effort on a Collective Task. *Group Dynamics, 5*(3), 181-190.

Hashiguchi, K. (1974). The Number of Decision Makers and the Level of Risk Taking Within a Group. *Japanese Journal of Experimental Social Psychology, 14*(2), 121-131.

Hastie, R. (1986). Review Essay: Experimental Evidence on Group Accuracy. In Owen, G. & Grofman, B. (Eds.), *Information Pooling and Group Decision-Making* (pp. 129-157). Westport, CT: JAI Press.

Heath, C., & Gonzalez, R. (1995). Interaction with Others Increases Decision Confidence but Not Decision Quality: Evidence against Information Collection Views of Interactive Decision Making. *Organizational Behavior & Human Decision Processes, 61*(3), 305-326.

Highsmith, J. (2002a). *Agile Software Development Ecosystems*. Boston: Addison-Wesley.

Highsmith, J. (2002b). *Extreme Programming* (White paper): Agile Project Management Advisory Service, Cutter Consortium.

Highsmith, J. (2003). Agile Project Management: Principles and Tools. *Agile Project Management Advisory Service, 4*(2), 37.

Highsmith, J., & Cockburn, A. (2001a). Agile Software Development 1: The Business of Innovation. *IEEE Computer, 34*(9), 120-122.

Highsmith, J., & Cockburn, A. (2001b). Agile Software Development 1: The Business of Innovation. *IEEE Computer, 34*(9), 120-127.

Hill, G. W. (1982). Group versus Individual Performance: Are N + 1 Heads Better than One? *Psychological Bulletin, 91*, 517-539.

Hinsz, V. B. (1995). Goal Setting by Groups Performing an Additive Task: A Comparison with Individual Goal Setting. *Journal of Applied Social Psychology, 25*(11), 965-990.

Hinsz, V. B., & Nickell, G. S. (2004). Positive Reactions to Working in Groups in a Study of Group and Individual Goal Decision Making. *Group Dynamics, 8*(4), 253-264.

Hinsz, V. B., Tindale, R. S., & Vollrath, D. A. (1997). The Emerging Conceptualization of Groups as Information Processors. *Psychological Bulletin, 121*(1), 43-64.

Hoegl, M., Parboteeah, K. P., & Gemuenden, H. G. (2003). When Teamwork Quality Matters: Task Innovativeness as a Moderator of the Teamwork-Performance Relationship in Software Development Projects. *Journal of Engineering Technology Management, 20*, 281-302.

Hollan, J., Hutchins, E., L., & Kirsh, D. (2000). Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research. *ACM Transactions on Human-Computer Interaction, 7*(2), 174-196.

Hom, H. L., & Berger, M. (1994). The Effects of Cooperative and Individualistic Reward on Intrinsic Motivation. *Journal of Genetic Psychology, 155*(1), 87.

Howell, W. C., Gettys, C. F., Martin, D. W., Nawrocki, L. H., & Johnston, W. A. (1970). Evaluation of Diagnostic Tests by Individuals and Small Groups. *Organizational Behavior & Human Performance, 5*(3), 211.

Hunt, P. J., & Hillery, J. M. (1973). Social Facilitation in a Coaction Setting: An Examination of the Effects over Learning Trials. *Journal of Experimental Social Psychology, 9*, 563-571.

Iaffaldano, M. T., & Muchinsky, P. M. (1985). Job Satisfaction and Job Performance: A Meta-Analysis. *Psychological Bulletin, 97*(2), 251-273.

Ito, J. K., & Peterson, R. B. (1986). Effects of Task Difficulty and Interunit Interdependence on Information Processing Systems. *Academy of Management Journal, 29*(1), 139.

Jackson, J. M., & Harkins, S. G. (1985a). Equity in Effort: An Explanation of the Social Loafing Effect. *Journal of Personality and Social Psychology, 49*, 1199-1206.

Jackson, J. M., & Harkins, S. G. (1985b). Equity in Effort: An Explanation of the Social Loafing Effect. *Journal of Personality and Social Psychology, 49*(5), 1199-1206.

Jackson, J. M., & Williams, K. D. (1985). Social Loafing on Difficult Tasks: Working Collectively can Improve Performance. *Journal of Personality and Social Psychology, 49*(4), 937-942.

Jain, B. A., & Solomon, J. S. (2000). The Effect of Task Complexity and Conflict Handling Styles on Computer-Supported Negotiations. *Information & Management, 37*, 161-168.

Jehn, K. A., Northcraft, G. B., & Neale, M. A. (1999). Why Differences Make a Difference: A Field Study of Diversity, Conflict, and Performance in Workgroups. *Administrative Science Quarterly, 44*(4), 741-763.

Johnson, D. W., Johnson, R. T., & Scott, L. (1978). The Effects of Cooperative and Individualized Instruction on Student Attitudes and Achievement. *Journal of Social Psychology, 104*(2), 207.

Johnson-Laird, P. N. (1981). Mental Models in Cognitive Science. In Norman, D. A. (Ed.), *Perspectives on Cognitive Science* (pp. 147-191). Norwood, NJ: Ablex.

Johnson-Laird, P. N. (1983). *Mental Models*. Cambridge, England: Cambridge University Press.

Johnson-Laird, P. N. (2001). Mental Models and Human Reasoning. In Dupoux, E. (Ed.), *Language, Brain and Cognitve Development: Essays in Honor of Jacques Mehler* (pp. 85-102). Cambridge, MA: MIT Press.

Judge, T. A., Thoresen, C. J., Bono, J. E., & Patton, G. K. (2001). The Job Satisfaction-Job Performance Relationship: A Qualitative and Quantitative Review. *Psychological Bulletin, 127*(3), 376-407.

Kanekar, S. (1982). Individual and Group Performance on an Anagrams Task. *Australian Journal of Psychology, 34*, 337-344.

Kanekar, S., Libby, C., Engels, J., & Jahn, G. (1978). Group Performance as a Function of Group Type, Task Condition and Scholastic Level. *European Journal of Social Psychology, 8*, 439-451.

Karau, S. J., & Williams, K. D. (1993a). Social Loafing: A Meta-Analytic Review and Theoretical Integration. *Journal of Personality and Social Psychology, 65*(4), 681-706.

Karau, S. J., & Williams, K. D. (1993b). Social Loafing: A Meta-Analytic Review and Theoretical Integration. *Journal of Personality and Social Psychology, 65*(4), 681-706.

Kerlinger, F. N. (1986). *Foundations of Behavioral Research*. Fort Worth, TX: Holt, Rinehart & Winston, Inc.

Kernan, M. C., Bruning, N. S., & Miller-Guhde, L. (1994). Individual and Group Performance: Effects of Task Complexity and Information. *Human Performance, 7*(4), 273-289.

Kerr, N. L. (1983). Motivation Losses in Groups: A Social Dilemma Analysis. *Journal of Personality and Social Psychology, 45*, 819-828.

Kerr, N. L., & Brunn, S. (1983). The Dispensability of Member Effort and Group Motivation Loses: Free Rider Effects. *Journal of Personality and Social Psychology, 44*, 78-94.

Kerr, N. L., & Tindale, R. S. (2004). Group Performance and Decision Making. *Annual Review of Psychology, 55*(1), 623-655.

Kim, J., & Lerch, F. J. (1997). Why is Programming (Sometimes) so Difficult? Programming as Scientific Discovery in Multiple Problem Spaces. *Information Systems Research, 8*(1), 25.

King, A. (1989). Verbal Interaction and Problem-Solving Within Computer-Assisted Coperative Learning Groups. *Journal of Educational Computing Research, 5*(1), 1-15.

Koehler, D. J. (1991). Explanation, Imagination, and Confidence in Judgment. *Psychological Bulletin, 110*(3), 499-519.

Kolb, K. J., & Aiello, J. R. (1996). The Effects of Electronic Performance Monitoring on Stress: Locus of control as a Moderator Variable. *Computers in Human Behavior, 12*(3), 407-423.

Langan-Fox, J., Anglim, J., & Wilson, J. R. (2004). Mental Models, Team Mental Models, and Performance: Process, Development, and Future Directions. *Human Factors and Ergonomics in Manufacturing, 14*(4), 331-352.

Langan-Fox, J., Code, S., & Langfield-Smith, K. (2000). Team mental models: Techniques, methods, and analytic approaches. *Human Factors, 42*(2), 242-271.

Larey, T. S., & Paulus, P. B. (1995). Social Comparison and Goal Setting in Brainstorming Groups. *Journal of Applied Social Psychology, 25*(18), 1579-1596.

Latane, B. (1981). The Psychology of Social Impact. *American Psychologist, 36*, 343-356.

Laughlin, P. R., & Adamopoulos, J. (1980). Social Combination Processes and Individual Learning for Six-Person Cooperative Groups on an Intellective Task. *Journal of Personality and Social Psychology, 38*, 941-947.

Laughlin, P. R., & Barth, J. M. (1981). Group-to-Individual and Individual-to-Group Problem-Solving Transfer. *Journal of Personality and Social Psychology, 41*, 1087-1093.

Laughlin, P. R., Bonner, B. L., & Miner, A. G. (2002). Groups Perform Better than the Best Individuals on Letter-to-Numbers Problems. *Organizational Behavior and Human Decision Processes, 88*, 605-620.

Laughlin, P. R., & Ellis, A. L. (1986). Demonstrability and Social Combination Processes on Mathematical Intellective Tasks. *Journal of Experimental Social Psychology, 22*, 177-189.

Laughlin, P. R., VanderStoep, S. W., & Hollingshead, A. B. (1991). Collective Versus Individual Induction: Recognition of Truth, Rejection of Error, and Collective Information Processing. *Journal of Personality and Social Psychology, 61*(1), 50-67.

Laughlin, P. R., Zander, M. L., Knievel, E. M., & Tan, T. K. (2003). Groups Perform Better Than the Best Individuals on Letters-to-Numbers Problems: Informative Equations and Effective Strategies. *Journal of Personality and Social Psychology, 85*(4), 684-694.

Letovsky, S. (1986). Cognitive Processes in Program Comprehension. In Soloway, E. & Iyengar, S. (Eds.), *Empirical Studies of Programmers* (pp. 58-79). Norwood, NJ: Ablex Publishing.

Levi, D. (2001). *Group Dynamics for Teams*. Thousand Oaks, CA: Sage Publications.

Levine, J. M., & Moreland, R. L. (1998). Small Groups. In Gilbert, D. T. & Fiske, S. T. (Eds.), *Handbook of Social Psychology* (4 ed., Vol. 2, pp. 415-469). New York, NY: McGraw-Hill.

Lim, K. H., Ward, L. M., & 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.

Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Tesorier, R., Williams, L., & Zelkowitz, M. (2002). *Empirical Findings in Agile Methods.* Paper presented at the Second XP Universe and First Agile Universe Conference, Chicago, IL.

Locke, E. A. (1968). Toward a Theory of Task Motivation and Incentives. *Organizational Behavior and Human Performance, 3,* 157-189.

Locke, E. A. (1970). Job Satisfaction and Job Performance: A Theoretical Analysis. *Organizational Behavior & Human Performance, 5*(5), 484-500.

Locke, E. A. (1976). The Nature and Causes of Job Satisfaction. In Dunnette, M. D. (Ed.), *Handbook of Industrial and Organizational Psychology* (pp. 1297-1349). Chicago: Rand McNally.

Locke, E. A., & Latham, G. P. (1990). Work Motivation and Satisfaction: Light at the End of the Tunnel. *Psychological Science, 1*(4), 240-246.

Locke, E. A., & Latham, G. P. (2004). What Should we Do About Motivation Theory? Six Recommendations for the Twenty-First Century. *Academy of Management Review, 29*(3), 388-403.

MacCormack. (2001). How Internet Companies Build Software. *MIT Sloan Management Review*, 75-84.

March, J., & Simon, H. (1958). *Organizations*. New York: Wiley.

Markus, H. (1978). Mere Presence and Social Facilitation. *Journal of Experimental Social Psychology, 14*, 389-397.

McGrath, J. E. (1984). *Groups: Interaction and Performance*. Englewood Cliffs, N. J.: Prentice-Hall, Inc.

Miner, F. C. (1984). Group versus Individual Decision Making: An Investigation of Performance Measures, Decision Strategies, and Process Losses/Gains. *Organizational Behavior and Human Performance, 33*, 112-124.

Mohammed, S., Klimoski, R. J., & Rentsch, J. R. (2000). The Measurement of Team Mental Models: We Have no Shared Schema. *Organizational Research Methods, 3*(2), 123-165.

Mullen, B. (1983). Operationalizing the Effect of the Group on the Individual: A Self-Attention Perspective. *Journal of Experimental Social Psychology, 19*, 295-322.

Mullen, B., & Salas, J. C. (1991). Productivity Loss in Brainstorming Groups: A Meta-Analytic Integration. *Basic Applied Social Psychology, 12*(1), 3-23.

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. (2003). *Improving the CS1 Experience with Pair Programming.* Paper presented at the 34th SIGCSE Technical Symposium on Computer Science Education, Reno, Navada.

Nawrocki, J., & Wojciechowski, A. (2001). *Experimental Evaluation of Pair Programming.* Paper presented at the 12th European Software Control and Metrics Conference, ESCOM, London, UK.

Neter, J., Kutner, M. H., Nachtsheim, C. J., & Wasserman, W. (1996). *Applied Linear Models* (Fourth ed.). Chicago: Irvin.

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

Nosek, J. T. (1998). The Case for Collaborative Programming. *Communications of the ACM, 41*(3), 105-108.

Nunnally, J. C. (1978). *Psychometric Theory*. New York, NY: McGraw-Hill.

O'Leary-Kelly, A. M., Martocchio, J. J., & Frink, D. D. (1994). A Review of the Influence of Group Goals on Group Performance. *Academy of Management Journal, 37*(5), 1285.

Orr, K. (2002). CMM Versus Agile Development: Religious Wars and Software Development. *Agile Project Management Advisory Service, 3*(7), 29.

Panina, D., & Aiello, J. R. (2005 (In press)). Acceptance of Electronic Monitoring and its Consequences in Different Cultural Contexts: A Conceptual Model. *Journal of International Business*, 1-39.

Paulus, P. B. (1983). Group Influence on Individual Task Performance. In Paulus, P. B. (Ed.), *Basic Group Processes* (pp. 97-120). New York: Springer-Verlag.

Paulus, P. B., Dzindolet, M. T., Poletes, G., & Camacho, L. M. (1993). Perception of Performance in Group Brainstorming: The Illusion of Group Productivity. *Personality & Social Psychology Bulletin, 19*(1), 78-89.

Pedhazur, E. J., & Pedhazur-Schmelkin, L. (1991). *Measurement, Design, and Analysis: An Integrated Approach*. Hillsdale, NJ: Lawrence Erlbaum.

Peterson, D. K., & Pitz, G. F. (1988). Confidence, Uncertainty, and the Use of Information. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 14*(1), 85-92.

Petty, M. M., McGee, G. W., & Cavender, J. W. (1984). A Meta-Analysis of the Relationships Between Individual Job Satisfaction and Individual Performance., *Academy of Management Review* (Vol. 9, pp. 712): Academy of Management.

Petty, R., E., Harkins, S. G., Williams, K. D., & Latane, B. (1977). The Effects of Group Size on Cognitive Effort and Evaluation. *Personality and Social Psychology Bulletin, 3*, 575-578.

Reusser, K. (1990). From Text to Situation to Equation: Cognitive Simulation of Understanding and Solving Mathematical Word Problems. In Mandl, H., DeCorte, E., Bennett, N. & Friedrich, H. F. (Eds.), *Learning and Instruction: Analysis of Complex Skills and Complex Knowledge Domains* (Vol. 2.2, pp. 477-498). Elmsford, NY: Pergamon Press.

Riley, M. S., & Greeno, J. G. (1988). Developmental Analysis of Understanding Language About Quantities and of Solving Problems. *Cognition & Instruction, 5*(1), 49.

Rist, R. (1989). Schema Creation in Programming. *Cognitive Science, 13*, 389-414.

Robbins, S. P. (1998). *Organizational Behavior: Concepts, Controversies, Applications* (Fourth ed.). Upper Sadler River, NJ: Simon and Schuster.

Rosencrance, L. (2004, Aug 2). Computer Glitch Hits Two Airlines. *Computer World*.

Rubin, H., Johnson, M., & Iventosch, S. (2002). The US Software Industry. *IEEE Software, 19*(1), 95-97.

Rumpe, B., & Schroder, A. (2002). *Quantitative Survey on Extreme Programming Projects.* Paper presented at the Third International Conference on Extreme Programming and Flexible Processes in Software Development (XP2002), Alghero, Italy.

Sanders, G. S., & Baron, R. S. (1975). The Motivating Effects of Distraction on Task Performance. *Journal of Personality and Social Psychology, 32*, 956-963.

Sanna, L. J. (1992). Self-Efficacy Theory: Implications for Social Facilitation and Social Loafing. *Journal of Personality and Social Psychology, 62*(5), 774-786.

Santhanam, R., & Sein, M. K. (1994). Improving End-user Proficiency: Effects of Conceptual Training and Nature of Interaction. *Information Systems Research, 5*(4), 378-399.

Schoner, B., Rose, G. L., & Hoyt, G. C. (1974). Quality of Decisions: Individuals versus Real and Synthetic Groups. *Journal of Applied Psychology, 59*(4), 424-432.

Schraw, G., & Nietfeld, J. (2003). Mental Models. In Guthrie, J. W. (Ed.), *Encyclopedia of Education* (pp. 1600-1602). New York: Thomson.

Schroder, H., Driver, M., & Streufert, S. (1967). *Human Information Processing*. New York: Holt, Rinehart and Winston.

Schvaneveldt, R. W. (1990). *Pathfinder Associative Networks: Studies in Knowledge Organization*. Norwood, NJ: Ablex Publishing Corp.

Scott, W. E. (1966). Activation Theory and Task Design. *Organizational Behavior & Human Performance, 1*(1), 3-30.

Shaw, J. D., Duffy, M. K., & Stark, E. M. (2000). Interdependence and Preference for Group Work: Main and Congruence Effects on the Satisfaction and Performance of Group Members. *Journal of Management, 26*(2), 259-279.

Shaw, M. E. (1954). Some Effects of Problem Complexity upon Problem Solution Efficiency in Different Communication Nets. *Journal of Experimental Psychology, 48*, 211-217.

Shaw, M. E. (1981). *Group Dynamics: The Psychology of Small Group Behavior* (Third ed.). New York: McGraw-Hill Publishing Company.

Shaw, M. E., & Ashton, N. (1976). Do Assembly Bonus Effects Occur on Disjunctive Tasks? A Test of Steiner's Theory. *Bulletin of the Psychonomic Society, 8*(6), 469-471.

Shepperd, J. A., & Taylor, K. M. (1999). Social Loafing and Expectancy-Value Theory. *Personality and Social Psychology Bulletin, 23*, 1147-1158.

Shih, Y.-F., & Alessi, S. M. (1993). Mental Models and Transfer of Learning in Computer Programming. *Journal of Research on Computing in Education, 26*(2), 154-175.

Smith, B. N., Kerr, N. A., Markus, M. J., & Stasson, M. F. (2001). Individual Differences in Social Loafing: Need for Cognition as a Motivator in Collective Performance. *Group Dynamics, 5*(2), 150-158.

Sniezek, J. A. (1992). Groups under Uncertainty: An Examination of Confidence in Group Decision Making. *Organizational Behavior & Human Decision Processes, 52*(1), 124.

StandishGroup. (2001). *Extreme Chaos*: The Standish Group International Inc.

Stanton, J. M., & Barnes-Farrell, J. L. (1996). Effects of Electronic Performance Monitoring on Personal Control, Task Satisfaction, and Task Performance. *Journal of Applied Psychology, 81*(6), 738-745.

Steele-Johnson, D., Beauregard, R. S., Hoover, P. B., & Schmidt, A. M. (2000). Goal Orientation and Task Demand Effects on Motivation, Affect, and Performance. *Journal of Applied Psychology, 85*(5), 724-738.

Steiner, I. D. (1972). *Group Process and Productivity*. New York: Academic Press.

Stephenson, G. M., & Wagner, W. (1989). Origins of the Misplaced Confidence Effect in Collaborative Recall. *Applied Cognitive Psychology, 3*(3), 227-236.

Stern, E. (1993). What Makes Certain Arithmetic Word Problems Involving the Comparison of Sets so Difficult for Children?. *Journal of Educational Psychology, 85*(1), 7-23.

Stroebe, W., Diehl, M., & Abakoumkin, G. (1996). Social Compensation and the Kohler Effect: Toward a Theoretical Explanation of Motivation Gains in Group Productivity. In Witte, E. H. & Davis, J. H. (Eds.), *Understanding Group*

Behavior: Consensual Action by Small Groups (Vol. 2, pp. 37-65). Mahwah, NJ: Erlbaum.

Stubbart, C. I. (1989). Managerial Cognition: A Missing Link in Strategic Management Research. *Journal of Management Studies, 26*(4), 325-347.

Taylor, M. S. (1981). The Motivational Effects of Task Challenge: A Laboratory Investigation. *Organizational Behavior and Human Decision Processes, 27*, 255-278.

Tindale, R. S., & Larson, J. R. (1992). Assembly Bonus Effect or Typical Group Performance?: A Comment on Michaelsen, Watson, and Black (1989). *Journal of Applied Psychology, 77*(1), 102-105.

Triplett, N. (1898). The Dynamogenic Factors in Pacemaking and Competition. *American Journal of Psychology, 9*, 507-533.

Van der Veer, G. C., & Melguizo, M. d. C. P. (2003). Mental Models. In Jacko, J. A. & Sears, A. (Eds.), *The Human-Computer Interaction Handbook* (pp. 52-80). Mahwah, NJ: Lawrence Erlbaum.

Vandenbosch, B., & Higgins, C. (1996). Information Acquisition and Mental Models: An Investigation into the Relationship Between Behaviour and Learning. *Information Systems Research, 7*(2), 198-214.

Vygotsky, L. S. (1978). Internalization of Higher Cognitive Functions. In Cole, M., John-Steiner, V., Scribner, S. & Souberman, E. (Eds.), *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, MA: Harvard University Press.

Wageman, R. (1995). Interdependence and Group Effectiveness. *Administrative Science Quarterly, 40*(1), 145-180.

Wagner, J. A., & Moch, M. K. (1986). Individualism-Collectivism: Concept and Measure. *Group and Organization Studies, 11*, 280-304.

Weinberg, G. M. (1971). *The Psychology of Computer Programming*. New York, NY: Van Nostrand Reinhold.

Weingart, L. R. (1992). Impact of Group Goals, Task Component Complexity, Effort, and Planning on Group Performance. *Journal of Applied Psychology, 5*, 682-693.

Williams, K. D., Harkins, S. G., & Karau, S. J. (2003). Social Performance. In Hogg, M. A. & Cooper, J. (Eds.), *The SAGE Handbook of Social Psychology* (pp. 494-511). Thousand Oaks, CA: Sage Publications.

Williams, K. D., Harkins, S. G., & Latane, B. (1981). Identifiability as a Deterrent to Social Loafing: Two Cheering Experiments. *Journal of Personality and Social Psychology, 40*, 303-311.

Williams, K. D., & Karau, S. J. (1991). Social Loafing and Social Compensation: The Effects of Expectations of Co-Worker Performance. *Journal of Personality and Social Psychology, 61*(4), 570-581.

Williams, L. (2000). *The Collaborative Software Process.* Unpublished Ph.D. Dissertation, University of Utah.

Williams, L., A., & Kessler, R. R. (2000). All I Really Need to Know About Pair Programming I Learned in Kindergarten. *Communications of the ACM, 43*(5), 108-114.

Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the Case for Pair Programming. *IEEE Software*, *17*(4), 19-25.

Wilson, J. R. (2000). Mental Models. In Karwowski, W. (Ed.), *International Encyclopedia of Ergonomics and Human Factors* (pp. 493-496). London: Taylor and Francis.

Wilson, J. R., & Rutherford, A. (1989). Mental Models: Theory and Application in Human Factors. *Human Factors, 31*(6), 617-634.

Witte, E. H. (1989). Kohler Rediscovered: the Anti-Ringelmann Effect. *European Journal of Social Psychology*, *19*(2), 147-154.

Wood, R. E. (1986). Task Complexity - Definition of the Construct. *Organizational Behavior and Human Decision Processes, 37*(1), 60-82.

Wood, R. E., & Mento, A. J. (1987). Task Complexity as Moderator of Goal Effects: A Meta-Analysis. *Journal of Applied Psychology*, *72*(3), 416-425.

Yinon, Y., Jaffe, Y., & Feshbach, S. (1975). Risky Aggression in Individuals and Groups. *Journal of Personality and Social Psychology*, *31*, 808-815.

Yourdon, E. (1997). *Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects*. New Jersey: Prentice Hall Computer Books.

Zajonc, R. B. (1965). Social Facilitation. *Science*, *149*, 269-274.

Zajonc, R. B. (1980). Copresence. In Paulus, P. B. (Ed.), *Psychology of Group Behavior* (pp. 35-60). Hillsdale, NJ: Erlbaum.

Zalesny, M. D., & Ford, J. K. (1990). Extending the Social Information Processing Perspective: New Links to Attitudes, Behaviors, and Perceptions. *Organizational Behavior and Human Decision Processes, 47*, 205-246.

Zander, A. F. (1974). Productivity and Group Success: Team Spirit vs. the Individual Achiever. *Psychology Today*, *8*(6), 64-68.

Zarnoth, P., & Sniezek, J. A. (1997). The Social Influence of Confidence in Group Decision Making. *Journal of Experimental Social Psychology, 33*(4), 345-366.

BIOGRAPHICAL INFORMATION

Venugopal Balijepally received his doctorate in Business Administration with a major in Information Systems from the University of Texas at Arlington. He holds Postgraduate Diploma in Management (equivalent to MBA) from Management Development Institute, India and Master of Technology in civil engineering from Indian Institute of Technololgy, India. He received his Bachelor of Engineering in civil engineering from Osmania University, India.

He has over ten years of industry experience in management positions. His current research interests include software development, social networks, knowledge management, IS management, and research methodologies.