THE TRUE COST OF PAIR PROGRAMMING: DEVELOPMENT OF A COMPREHENSIVE MODEL AND TEST

BY

©2011
Wenying Sun

Submitted to the graduate degree program in Business and the
Graduate Faculty of the University of Kansas
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Committee Members:

_____
Dr. George M. Marakas (chairperson)

_____
Dr. Andrew N. K. Chen

_____
Dr. James A. Heintz

_____
Dr. Gilbert Karuga

_____
Dr. Todd D. Little

Date Defended: _____

The Dissertation Committee for Wenying Sun
certifies that this is the approved version of the following dissertation:

THE TRUE COST OF PAIR PROGRAMMING: DEVELOPMENT OF A COMPREHENSIVE MODEL AND TEST

_____
Chairperson: Dr. George M. Marakas

Date Approved: _____

ABSTRACT

To advance the nomological net and theory, this dissertation proposed a comprehensive pair programming research model where the relationships among system complexity, programming methods, pair composition, effort, duration, defect rate, knowledge transfer, and various cost constructs were investigated. A multi-method, multi-study empirical approach was adopted. The survey method was employed for Study 1, and the bootstrap simulation method for Study 2. The reponses from 191 industry software developers and the simulation results suggest the previous conclusions regarding pair programming are limited in nature and the pair programming approach may not be as desirable in all situations as was previously assumed. The pair programming approach clearly adds value in situations where it is appropriate but certain conditions must be met for this goal to be achieved. Pair composition must be taken into account, and it is important to examine the interactions of multiple cost factors such as defect, effort, duration, and knowledge transfer and consider their combined effect on the ultimate goal of the project.

ACKNOWLEDGEMENT

First and foremost I would like to thank my advisor, George M. Marakas, who was abundantly helpful and offered invaluable assistance, support, and guidance throughout my study at KU. His positive attitude, passion for research, and willingness to share his knowledge made his seminars and our weekly dissertation meetings something to look forward to.

I am grateful to other members of my dissertation committee, Andrew N. K. Chen, Jim A. Heintz, Gilbert Karuga, and Todd D. Little. I appreciate their constructive feedback and commitment to see me through the long dissertation journey.

I would also like to express my gratitude to Jim A. Heintz, Director of Accounting and Information Systems. The warm gatherings at his home and his tireless support of the IS PhD program will always be remembered.

I am also indebted to Charly Edmonds and Deb Deering. Without their help in navigating processes, procedures, forms, and many other things, my study at KU would have been much less smooth.

Finally, I would like to thank my husband Baili and son Benjy for their loving support. Their understanding and endless love sustained me through the demanding process.

TABLE OF CONTENTS

1.INTRODUCTION

Software is used in virtually every part of human life. From embedded systems in our kitchen appliances to control systems for space exploration, from book keeping systems to monitoring systems for surgical operations, humans rely heavily on software for processing information and providing recommendations for actions and interactions.

Given the ubiquitous nature of software, it should not be surprising that the software industry has seen tremendous growth in the past decade and is expected to continue to grow in the future. According to reports by Data Monitor, a leading business information company specializing in industry analysis, the United States software market generated total revenues of $85.9 billion in 2007 and is forecast to have a value of $116.6 billion by 2012 (Datamonitor, 2008a); the global software market was valued at $251.5 billion in 2007 and is expected to reach $347.3 billion in 2012 (Datamonitor, 2008b). Industry estimates point to as many as 17 million professional software developers in the world including three million developers in the US (Woyke, 2008; Mahalo, 2009).

Despite its growth and rapid maturity, software development remains challenging. Yourdon (2004) classified many software development projects as "Death March" projects where participants helplessly watch the projects sink into the sea of failure. While there are conceivably an infinite number of issues that must be addressed across the wide spectrum of software development, arguably the two most important issues are the quality of the software and the cost of developing it.

The quality of software is of paramount importance. National Institute of Standards and Technology reports (Tassey, 2002) software errors cost the U.S. economy $59.5 billion annually. This represents over half of the 2007 software industry revenues generated by selling the products. Serious problems and significant losses may result when a system fails to deliver its

services as expected (Sommerville, 2007). For instance, the wrong dosage suggested by a medical expert system intended to be used in support of treatment could result in patient deaths (Kirby, 2007; Computing Cases, 2009); a defect in the computer controlling the space shuttle could cause catastrophic losses (Sommerville, 2009); a failure in the traffic control system could shut down traffic lights and cause chaotic traffic jams or potentially disasterous results (Geoghegan, 2005); a fault in the electronic switching systems of a telecommunications company could cause an outage, blocking millions of phone calls (Washington Technology, 1999); an error in calculating classroom exam results could misinform educators, students, and their parents, and a mistake in tax software could distort tax assessments and result in many taxpayers experiencing unnecessary audits and penalties (BBC News, 1998; 2009).

Regardless of the degree of criticality, software quality is essential. Even when liability is not a major concern, users are not likely to stay with software products full of inconvenience as a result of development deficiencies and errors. Companies producing poor quality software will likely not stay in the marketplace for too long.

The cost of developing software is high due to its increased size and complexity (Tassey, 2002). Brooks (1995) reported the IBM OS/360 project he managed required 5000 man-years in its design, construction, and documentation during a three-year period, and at peak time, had over 1000 people working on it. Microsoft NT operating systems was reported to have more than 35 million lines of code (Markoff, 1998). A commercial video game typically calls for over three million lines of code with budgets ranging from $5 million to $50 million (Hurt, 2009).

The cost of developing software is comprised of several elements: labor costs, training costs, and hardware and software costs, with labor as the largest cost component (Sommerville, 2007). According to the May 2010 labor statistics (Bureau of Labor Statistics, 2011), the

average hourly wage of a software developer in the US is $36.01, and the average annual wage is $74,900. Statistics based on 1,238 software projects from International Software Benchmarking Standards Group (ISBSG) report the average software effort among the projects is 8,414.80 man-hours and the maximum is 645,694 man-hours (Pendharkar and Rodger, 2009), which suggests the average direct labor cost for these projects is $303,000, and $23 million for the largest project.

Developing quality software within budget constraints is a universal challenge faced by all software projects. Quality comes at a cost. In programming, the bulk of cost is not simply labor associated with writing codes, but labor associated with writing *quality* codes. As Figure 1 illustrates, quality sits in the center of the triangle of money, scope, and time, and is affected by any changes made to any side of the triangle (Microsoft, 2009). When time and scope are fixed, money necessarily increases as quality goes up.

Figure 1.1 Quality and the Triangle (Microsoft, 2009)

Furthermore, Juran (1974) depicts a concave upward curve in Figure 1.2 as the relationship between cost and quality where he defines the "zone of improvement" and the "zone of perfectionism". The "zone of improvement" lies to the left of the optimum quality level, while the "zone of perfectionism" lies to the right. The graphs suggest quality in the improvement zone can be improved without incurring additional cost; as a matter of fact, it reduces cost per unit of product, but in the perfectionism zone, every little improvement in

quality comes with a high price. This can be metaphorically thought of in terms of the commonly used heuristic known as the "Pareto principle". The Pareto 80/20 principle states 80% of the effects come from 20% of the causes. When applied to quality and cost in software development, one can argue that generally 80 percent of the project quality is achieved by only 20 percent of the resources. This suggests, to achieve the remaining 20 percent of the total quality, up to 80 percent of the resources will be required.

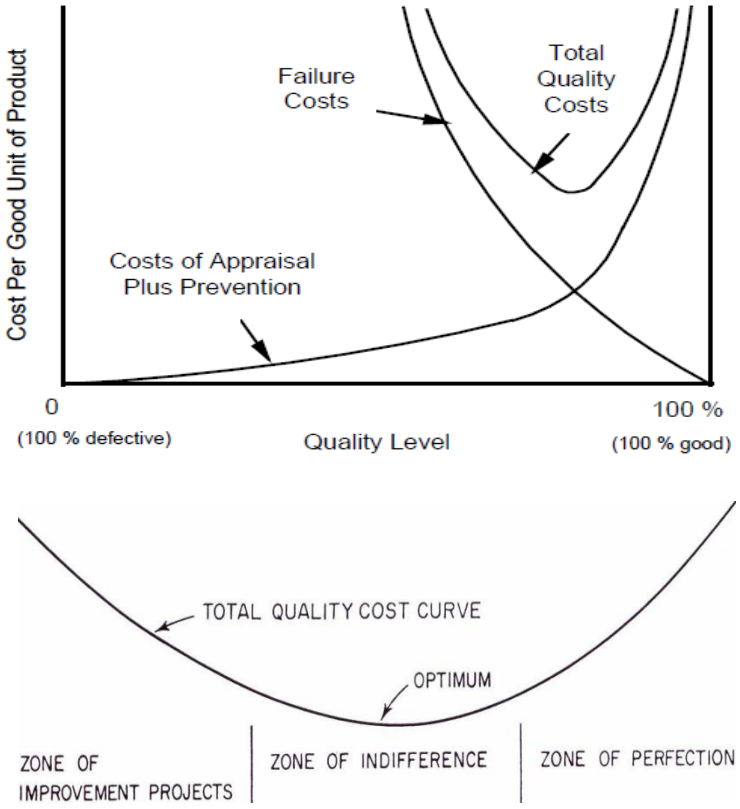


Figure 1.2 Quality and Cost (Juan, 1974)

Given the importance of this issue, numerous approaches have been developed to improve software quality while staying within cost constraints. Waterfall software development, rapid application development (RAD), object-oriented analysis and design (OOAD), and the

most recent agile software development (Pressman, 2005; Sommerville, 2007) all have the same goal: developing high quality software within budget. Although sharing the same goal, these approaches adopt different, sometimes even contradictory, techniques to achieve it (Dennis, Wixom, and Tegarden, 2005; Schach, 2006; Pfleeger and Atlee, 2008). For example, waterfall development endorses rigorous document "sign-off" processes thus minimizing changes as the project proceeds, while agile software development welcomes changing requirements even late in the project (Agile Manifesto, 2011). Waterfall does not deliver working software until the very end of the project while RAD develops some part of the system quickly and puts it into the hands of the users for suggestions to bring the system closer to what is needed. OOAD takes advantage of basic object-oriented characteristics such as classes, methods, inheritance, encapsulation, and polymorphism and focuses on reusability, which introduces concepts vastly different from the procedure-oriented analysis and design associated with Waterfall.

One of the most recent, and seemingly attractive, approaches is *pair programming*. Solo programming is the traditional programming method while pair programming has recently emerged as an attractive alternative to address the challenges solo programming faces such as quality and speed.

Pair programming is a programming method where two programmers work on the same programming task side by side in front of one computer (Beck, 2000; Williams, Kessler, Cunningham, and Jeffries, 2000; Arisholm, Gallis, Dybå, and Sjøberg, 2007). In pair programming, one programmer is the *driver*, and the other is the *navigator*. The driver sits in front of the computer screen, types the code, and pays close attention to the coding details. The navigator sits beside the driver, reviews the code, and takes the lead in developing alternative

strategies in the event of a problem. The programmers change roles periodically during the project to avoid role fatigue.

Pair programming is one of the twelve principles Extreme Programming (XP) follows (Sommerville, 2007). XP, the brainchild of Kent Beck and colleagues Ron Jeffries and Ward Cunningham (Beck, 2000), has emerged as the most successful and best-known of the agile software development method (Sommerville, 2007). Beck credits much of its success to the use of pair programming (Williams et al., 2000).

Despite the worldwide prevalence - used in 58.3% of the projects in India, 22.2% in Japan, 35.5% in US, and 27.2% in Europe and other countries (Cusumano, MacCormack, Kemerer, and Crandall, 2003) and the increased adoption – seven percent increase from 2007 to 2008 (VersionOne, 2008), our literature review suggests the adoption of pair programming seems to be based on limited and sometimes questionable studies reporting its effectiveness. The appeal of the pair programming method appears to come from the reported improvement in quality and shortened software development cycle. However, a close review of the literature reveals several problems with these conclusions.

First, with a few exceptions (Arisholm et al., 2007; Balijepally, Mahapatra, Nerur, and Price, 2009), the majority of the empirical studies were conducted using neither theory nor research framework. The comprehensive pair programming research framework suggested by Gallis, Arisholm, and Dybå (2003) and extended by Ally, Darroch, and Toleman (2005) has been largely ignored. Gallis et al. (2003) suggest individual factors and human interactions, as well as task characteristics are crucial elements in determining its effectiveness. Yet, few studies consider personal factors such as programmer expertise, prior pair programming experience, interpersonal factors such as pair composition, and external factors such as the complexity of the

programming task, as contributing forces and components in their research. Lack of theoretical guidance arguably puts the studies' conclusions in doubt.

Second, across the studies, there is no consensus on the measurement of software quality. For example, quality was measured by such varietal and opposing metrics as error rate (Jensen, 2003), readability and functionability (Nosek, 1998), correctness (Lui and Chan, 2003), percentage of test cases passed (Williams et al., 2000), and scores on classroom programming assignments (McDowell, Werner, Bullock, and Fernald, 2003; 2006). Consequently, findings from individual studies cannot be readily generalized beyond the measurement method the authors adopted.

Third, findings on how much more time is required by the pair programming method varied from a decrease in time (Lui and Chan, 2003), a 15 percent increase (Williams et al., 2000), a 41 percent increase (Nosek, 1998), to a 100 percent increase (Nawrocki and Wojciechowski, 2001), thus failing to provide a definitive answer to whether pair programming requires more programming hours than solo programming, and if so, by how much.

Finally, there has been very little empirical work, to date, on the issue of cost in the pair programming arena. In fact, only two major studies emerge from a comprehensive review of the literature. These two studies aimed to address the economics of pair programming but yielded different conclusions: in one study, pair programming was more cost effective than solo programming in all situations (Erdogmus and Williams, 2003), while in the other the economic benefit of pair programming depended on factors such as market pressure (Padberg and Müller, 2003). Moreover, both study results were hampered by the lack of reliable parameter values used in the simulation models. Erdogmus and Williams (2003) relied on data on programmer productivity, defect rate, and rework speed. Padberg and Müller (2003) depended on data about

pair speed advantage and pair defect advantage. However, in both cases, empirical evidence of the parameter values was very limited. As Erdogmus and Williams stated (p. 315-316) in the limitation section of their research, they derived the parameter data from different sources, but did not verify the data consistency among the sources. In addition, they did not have information on the software development methods of the companies involved in those statistics.

Given what we have learned from the present, albeit limited, body of research on pair programming, no empirical study has truly addressed all the issues associated with cost. From previous research, one cannot confidently answer the question: *is pair programming a cost effective method for software development?* While it might be true that pair programming can produce higher quality code with shortened development cycle, it may not come without a cost in other areas, such as increased programming hours. Since no study has addressed the entire complexity of cost as it relates to pair programming, an empirical study that thoroughly considers its many different contributing factors is warranted.

The purpose of this research is to propose and empirically validate a comprehensive theoretical model of pair programming as it relates to project cost. The outcome of the research aims to answer the following research questions:

1) Is pair programming a cost effective method compared to solo programming when overall project costs and quality are considered?

2) What are the conditions that determine the cost effectiveness, or lack thereof, of the pair programming method over solo programming?

Answers to these research questions will be derived via a multi-method empirical approach. The first method is a survey study with two primary objectives:

a). Gather information on practitioners' perceptions regarding the cost of pair programming and provide an initial validation of the research model;

b). Acquire data on project and developer characteristics to provide the necessary foundation for a simulation study.

The second method is a series of bootstrapping simulations. Using responses from the survey as input parameters, its objective is to determine in what situations pair programming is expected to be more cost effective than solo programming. We examine both the project and developer characteristics and vary parameter values to investigate the effects of the pair programming method on the overall cost of a project under a variety of conditions.

It is important to note that programming is only one phase of software development. We fully recognize activities both preceeding and after can have impacts on the total cost of the project. For the purpose of this research, however, we are interested in two different programming approaches and their associated effects on cost.

We believe this research has several theoretical and practical implications for researchers as well as practitioners. From a theoretical perspective, this research is expected to be one of the first few to examine several under-studied constructs and relationships: prior pair programming experience, knowledge transfer, the moderating effects of pair programming method on the relationship between system complexity and constructs such as effort, defect rate, and knowledge transfer, and all the cost-related constructs are new. We hope the mechanism we develop to acquire parameter values will provide a basis for future research in this area.

From the practical perspective, the study will likely provide foundational guidelines to the industry when one needs to decide which programming method to adopt in different project environments. Specifically, three situations and their associated project and developer

characteristics will be determined: situation 1 - pair programming is more cost effective than solo programming, situation 2 – the two methods are pretty much the same, and situation 3 – solo programming is more suitable than pair programming. Such findings are expected to provide suggestions to organizations in regards to which programming method to use given their unique project and developer characteristics.

The remainder of the research is organized into five chapters, a bibliography, and appendixes in the following manner. Chapter Two presents a review of the related literature on pair programming. Chapter Three presents the research model and hypotheses. Chapter Four delineates the research design and methodology of the study. An analysis of the data and a discussion of the findings are presented in Chapter Five. Chapter Six contains the identifiable limitations, summary, conclusions, and recommendations of the research. The research concludes with a bibliography and appendixes.

2. LITERATURE REVIEW

Practitioners and academia have both contributed to the body of pair programming knowledge and research. Despite occasional overlap, practitioners tend toward sharing their personal experiences and observations while academia typically contribute by collecting and analyzing data based on grounded theories and standard research methods.

It is important to note that the literature presented here represents a totality of both practitioner and academic work on pair programming. Suffice to say, the extant literature in pair programming is both immature and lacking strong empirical focus. Appended to the end of this chapter is Table 2.1 which contains a summary of the extant literature in pair programming. Despite the volume of the table, most of the studies are either practitioner-oriented anecdote (17 out of 85 = 20%) or published in conference proceedings (37 out of 85 = 43.5%). Among

refereed academic journal publications (29 out of 85 = 34%), nearly half of the studies focused on university computer science education rather than real world industrial practice. We can conclude from these descriptions that, in one sense, all studies become important when the extant literature is so young. In another sense, the academic and empirically focused portion of the literature is weak with respect to theory and foundation. It is our hope that this research effort will improve on this condition.

Practitioners have utilized several outlets to share their opinions and experiences. One major outlet is Internet websites such as *Agile Alliance, Agile Software Development, Agile Manifesto, Extreme Programming,* and *Wikipedia* on pair programming. The other outlet is practitioner's journals such as *Computer World, Information Week,* and *Dr. Dobb's*. In addition to the above, several popular news media have opinion pieces on pair programming. For example, a September 2009 New York Times article Olsen (2009), argued pair programming was the only way to work from the stand point of a programmer at a web development firm in Florida. A September 2009 Wall Street Journal article Price (2009), called pair programming a "trendy practice". The annual agile conference, which attracts over 1000 attendees, is another outlet for practitioners to share their industrial experience alongside academic colleagues.

The major outlets for academic publications are the Institute of Electrical and Electronics Engineers (IEEE), the Association for Computing Machinery (ACM), and premiere IS conference proceedings and journals. There has been substantial academic literature on pair programming research with most conducted by researchers trained in computer science and software engineering. The focus and the quality of the research varies greatly, ranging from opinion pieces based on observations, to studies that present descriptive statistics without conducting any significance test, to studies that rigorously adopt scientific research methods. We

include all the extant literature in this chapter for comprehensiveness, but we caution readers regarding the generalizability of the conclusions from some of the studies due to their lack of sound empiricism.

Among the studies we reviewed, two stand out as exemplars of important contributions: Gallis et al. (2003) and Dybå, Arisholm, Sjøberg, Hannay, and Shull (2007). These two studies present a comprehensive list of constructs that have been the focus of interest by most academic researchers in the pair programming arena.

Gallis et al. (2003) proposed a comprehensive research framework for pair programming. To address the problem of no available theoretical framework to support pair programming research, the authors developed such a framework based on existing studies and theories from group dynamics such as egoless programming (Weinberg, 1971), surgical team (Brooks, 1975), and dynamic duos (Constantine, 1995). Figure 2.1 illustrates this proposed research framework.

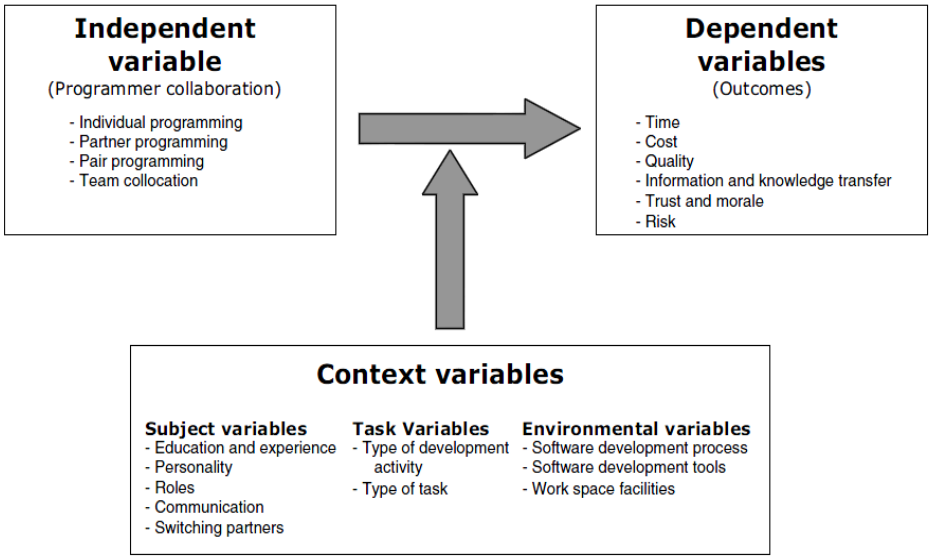


Figure 2.1 Research Framework for Pair Programming (Gallis et al., 2003)

This framework suggests that time, cost, quality, knowledge transfer, morale, and risk are

salient dependent variables with context variables such as subject, task, and environmental characteristics moderating the relationship between programmer collaboration methods and outcomes. This framework was later extended by adding organizational factors such as team building, pair management, human resource management, accountability, customer resistance, organizational culture, and collective code ownership to the moderating context variables (Ally, Darroch, and Toleman, 2005).

The second exemplar, Dybå et al. (2007), conducted a meta-analysis which represents the most current research aimed at synthesizing results from multiple empirical studies. The meta-analysis focused on the effect of programming method (solo vs. pair) on quality, duration, and effort. Figure 2.2 contains their findings.

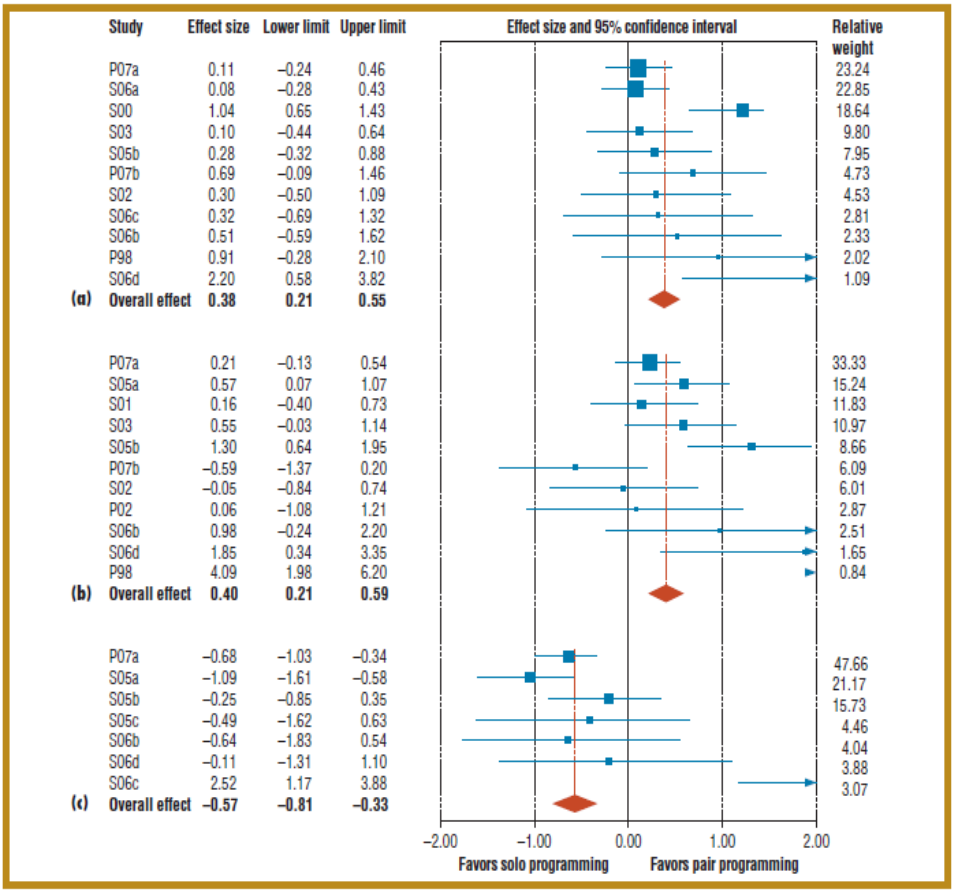


Figure 2.2 Findings from Meta-analysis (Dybå et al., 2007)

This meta-analysis, consisting of 15 studies, suggests pair programming leads to a medium-sized increase in quality (effect size = 0.38), a medium-sized overall reduction in duration (effect size = 0.40), and a medium-sized negative effect on effort (effect size = -0.57).[1]

In building upon the presentation of the two previous exemplars, the remainder of this literature review presents the findings from studies in pair programming using an organizing approach that separates the constructs identified in the two previous exemplars. We believe this approach presents the clearest understanding of the variables of interest in pair programming research in general, and this research effort in particular. Variables such as trust and risk were not included since few studies were identified to focus on those constructs. In addition, since the focus of this research effort is to compare pair programming to solo programming, literature that compares pair programming to other team development methods such as traditional team development, peer review, and inspection are also not presented or discussed.[2]

2.1. Quality

Numerous stories from the industry attest to the relationship between the use of pair programming and improved software quality. Tom Ayerst, an architectural consultant at a London-based investment bank, suggested that pair programmers made fewer coding mistakes and stupid choices because it was like having a pilot to focus on flying, while the navigator made strategic decisions about where to go next (Copeland, 2001a). At Royal & Sun Alliance Insurance Group, a $16 billion London-based insurer, pairing two developers on each assignment helped produce more stable code (Copeland, 2001a). Haungs (2001) noted the combined efforts

[1] For studies in software engineering, 0-0.37, 0.38-1.00, 1.01-3.40 represent small, medium, and large effects respectively (Kampenes, Dybå, Hannay, and Sjøberg, 2007).

[2] Interested readers are referred to Ciolkowski and Schlemmer (2002), Tomayko (2002), Heiberg, Puus, Salumaa, and Seeba (2003), Müller (2004; 2005), Chong (2005), Phongpaibul and Boehm (2006), and Xu and Rajlich (2006) for these areas of interest. Readers who are interested in research on distributed vs. co-located pair programming are referred to Baheti, Gehringer, and Stotts (2002), Prashant, Edward, and Stotts (2002), Natsu, Favela, Moran, Decouchant, and Martinez-Enriquez (2003), Canfora, Cimitile, Lucca, and Visaggio (2006), and Flor (2006).

in pair programming on the Chrysler 3 project produced a tool that was much better than the sum of its parts. At Iona Technologies and Wotif, the adoption of pair programming improved the quality of the final product (Poole and Huisman, 2001; Luck, 2004). The Quality Assurance teams at Symantec produced cleaner test classes and better coverage tests through the pair-programming process (Morales, 2002). Jensen, a consultant for the Software Technology Support Center, Hill Air Force Base, reported an error rate of 0.001 in a pair programming experience compared to the normal error rate experienced without using the pair approach. He further revealed integration of the first two components (approximately 10,000 source lines) was completed with only two coding errors and one design error; the third component was integrated with no errors; and the remaining three components had more errors, but the number of errors was significantly less than normal (Jensen, 2003). At Sabre Airline Solutions, actual coding was done in pairs by teams in open labs. Reports from Sabre suggest that the use of the pair approach cut defects dramatically: 100 defects for their Profit Manager project (500 KLOC),[3] zero defects for their Host Access project (15 KLOC), and four defects for their Peripheral Manager project (28 KLOC) (Anthes, 2004). At Intel pair-programmed components had the lowest defect density in the IXP2xx project and one of the paired teams achieved zero defect quality (Fitzgerald and Hartnett, 2005). Finally, Marchenko (2008a; 2008b; 2008c) stated his personal observations supported the notion of better quality when using the pair programming method.

The anecdotal claim about quality improvement through pair programming has been supported by academic-based empirical studies involving college students as well as practitioners. Nosek (1998) examined the effectiveness of pair programming with 15 full-time system programmers: five individuals and five pairs. The study results revealed pairs produced

[3] KLOC refers to thousand of lines of code.

more readable and functional solutions than solos. Lui and Chan (2003) used 15 full-time industrial programmers from different companies as subjects in their experiment, five pairs and five individuals, to solve two problems: a deduction problem and a procedural algorithm. Study results revealed that pairs outperformed individuals in terms of correctness. Nilsson (2003) conducted a pair programming survey that involved both students and practitioners and noted that most of the 67 survey respondents believed pair programming would reduce defect rate. Tessem (2003) conducted a field study with six students and researchers at the University of Bergen, Norway. The project lasted three weeks. All six programmers reported that they found pair programming led to higher quality. Canfora, Cimitile, Garcia, Piattini, and Visaggio (2007) conducted an experiment involving 18 developers in a software company in Spain. The authors reported that pairs consistently produced higher quality products than solos. Vanhanen and Lassenius (2007) surveyed 28 developers in a medium-sized Finnish software product company. Findings suggested a clearly positive effect for quality aspects such as understandability and maintainability of code, defect count, and customer satisfaction.

A series of experiments involving hundreds of college students in various programming classes were conducted by McDowell, Williams, and colleagues at the University of Utah, North Carolina State University, and the University of California Santa Cruz. Study results suggest, when compared to programs created by solos, those produced by paired students were significantly higher quality (Cockburn and Williams, 2001; McDowell, Werner, Bullock, and Fernald, 2002; McDowell, Hanks, and Werner, 2003; McDowell et al., 2003; 2006; Williams et al., 2003), passed more test cases (Williams et al., 2000), and had higher scores (Nagappan, Williams, Wiebe, Miller, Balik, Ferzli, Petlick, 2003; Nagappan, Williams, Ferzli, Wiebe, Yang, Miller, Balik, 2003). These same results were confirmed by several other student subject

experiments and surveys. Sanders (2001) conducted a student survey and reported based on 60 responses that the students believed pair programming would lead to higher quality of the program. Declue (2003) collected a survey in a CS2 course at Southwest Baptist University. Responses were highly skewed in favor of increased quality with pair programming. Mendes, Al-Fakhri, and Luxton_Reilly (2005) reported pairing improved the quality of assignments, examination scores, and percentage passing rate. Müller (2006) noticed the programs produced by solo students showed 14 percent more failures than the paired programs, and the failures of the solo programs were more severe than the failures of the paired programs. Wray (2010) applied theories such as expert programmer theory and change blindness to explain why pair programming was a superior approach over solo. He stated when two programmers are working together, one was more likely to ask a deep question that would prompt a novel inference from the stuck programmer. In addition, two people programming together wouldn't have the same prior categorization so one would spot some things faster and the other different things faster.

Despite the overwhelming support for the relationship between pair programming use and software quality, several studies yielded mixed results for the improved quality assertion.

Madeyski (2006) conducted an experiment on 188 students who developed a finance-accounting system using four different programming approaches: classic solo, test-driven development solo, classic pair, test-driven development pair. Quality, which was measured by relationships among the system's packages: coupling, stability, abstractness, and distance from main sequence, was not significantly affected by development methods. Hulkko and Abrahamsson (2005) revealed comment ratios were higher for pair programming than for solo, and pair programmed code was more readable, but there was no significant difference in defect density. Vanhanen and Lassenius (2005) found paired student programming teams wrote code

with fewer initial defects but delivered the final systems with more defects because they were less careful in system testing. Balijepally et al. (2009) concluded a pair was not necessarily better than a solo. They compared the performance of pairs with those of the best performers and the second best performers and found that pairs performed at the level above the second best performers but no better than the best performers.

Additionally, some studies found quality improvement was not consistent across tasks or projects. Gehringer (2003) required students to implement three projects simulating various aspects of a mciroarchitecture (cache, branch predictor, dynamic instruction scheduler) and revealed paired students obtained significantly higher grades on the first project but not in the subsequent projects. Hanks et al. (2004) asked students to work on three programming assignments – writing programs to play the card game blackjack, to implement a simple dice game, and to implement a text-based version of the Mine-sweeper game. The study found that paired students successfully implemented more features than solos in two of three assignments but no significant difference between the two groups on the second assignment. Vanhanen and Korpi (2007) reported a project carried out in a large telecommunications company in Finland where four developers were involved and pair programming was adopted from the beginning. All developers reported that pair programming lowered the number of defects, but in a team interview, the developers were somewhat uncertain because the navigators did not spot many defects during the programming.

Several studies noted quality improvement to be dependent on the complexity of the task. Al-Kilidar, Parkin, Aurum, and Jeffery (2005) indicated when requirements were simple, pairs produced a better quality product, but when requirements were complex, there was no significant difference in quality between pairs and solos. In direct contrast to Al-Kilidar, et al., (2005),

Arisholm et al. (2007) conducted a one day experiment using 295 professional java consultants with 99 individuals and 98 pairs. They reported that on more complex tasks, the pair programmers had a 48 percent increase in the proportion of correct solutions but for simpler systems there was no significant difference between pairs and solos.

In software development, quality is a complex concept with many different dimensions. Our review of the extant literature suggests little consensus on the measurement of software quality across studies. Quality was measured by defect rate (Vanhanen and Lassenius, 2005), readability and functionability (Nosek, 1998), correctness (Lui and Chan, 2003), percentage of test cases passed (Williams et al., 2000), and scores on classroom programming assignments (McDowell et al., 2003; McDowell et al., 2003; 2006). The lack of cumulative research on standard measures of quality makes it challenging to generalize conclusions across such studies and serves as a challenge for researchers focusing on this area.

2.2. Cost

As with quality in the previous section, studies are split in their conclusions regarding whether pair programming reduces the overall cost of a software development project compared to solo.

Some anecdotal stories suggest pair programming will reduce the overall cost of a project while others believe the benefits of pair programming do not justify the increased expense of the second programmer. Stephen Hutchinson, senior technical architect at Royal & Sun Alliance Insurance Group, claims pairing two developers on each assignment helped the company come in 15% lower than the projected budget (Copeland, April 2001). An application development manager at a major U.S. bank commented the cost issue was moot because through pair

programming there would be fewer defects and less time would be spent on bug fixing (Radding, 2002).

In contrast, Larry Zucker, executive director of application development at Dollar Rent-a-Car Systems in Tulsa, Oklahoma, said that while he appreciated the benefits of having two programmers on one task, the gains didn't justify the doubled expense. He also expressed the fear that the programming process could turn into a social event (Copeland, 2001a). This same view was echoed by the concern brought about several times by the programmers in Nilsson's survey (Nilsson, 2003): the benefits of pair programming did not cover the very expensive costs. Stephens and Rosenberg (2003) identified cost as the major issue facing a decision to employ pair programming (p. 150-151). Aiken's (2004) interview of three developers identified the same view: there are surely additional development costs, especially because productivity might suffer at first while people are adjusting. Luck (2004) reported a 15% extra cost from an industrial experience.

As with the practitioner community, conclusions drawn from the empirical studies are equally far from reaching consensus with regard to the relationship between pair programming and project cost. Müller (2006) found no difference in terms of development cost between a pair and a solo implementation if the cost for developing programs of a similar level of correctness was concerned, while Rostaher and Hericko (2002) revealed the average time spent to complete all three tasks by solo and pair programmers was very similar, which means pairs needed almost twice as much time and basically doubled the cost to complete the same amount of work compared to individuals.

Two major simulation studies attempted to address the economics of pair programming and yielded markedly different conclusions: in one study pair programming was more cost

effective than solo programming in all situations (Erdogmus and Williams, 2003), while in the other the realized economic benefit of pair programming depended on factors such as market pressure (Padberg and Müller, 2003).

Given that these two studies represent major efforts in addressing the cost benefits of pair programming vs. solo, we present a summary of the studies below. It should be noted that Müller and Padberg (2002) and Müller and Padberg (2003) appear to be earlier reports of studies similar to Padberg and Müller (2003). Since Padberg and Müller (2003) provided more comprehensive discussions of the study, only Padberg and Müller (2003) is presented here.

Erdogmus and Williams (2003) conducted a major research effort on the economics of pair programming. Three empirical parameters were crucial to their model: productivity (LOC/hour), defect rate (defects/LOC), and rework speed (defects fixed/hours). The abstract models for solo and pair used by the researchers is shown below:

where π is productivity, β is defect rate, and p is rework speed.

$$\text{Solo} = \{N=1, \pi=25.0, \beta=0.00585, p=0.0303\}$$

$$\text{Pair} = \{N=2, \pi=43.478, \beta=0.00351, p=0.0527\}$$

Based on these parameters, the authors compared solo and pair on three measures: efficiency, unit effort, and unit time and revealed that pair was better in all of the three metrics: nearly 100% improvement in efficiency, over 40% reduction in unit effort, and over 70% reduction in unit time.

The authors then considered two value realization models: single-point delivery (value realized at the end) and incremental delivery (value realized incrementally on a continuous basis). The comparison of solo and pair based on breakeven unit value ratio (solo breakeven unit value/pair breakeven unit value) suggested that pair was better in both situations.

Padberg and Müller (2003) constructed a mathematical model and applied the model in two different scenarios: conventional development and development using pair programming.

To realize the models, the authors adopted some values from previous studies. Productivity was defined as 250-550 LOC/month (Sommerville, 1996), pair speed advantage ranged from 1.3 to 1.8 (Nosek, 1998; Williams et al., 2000), defect density was assumed to be 0.03/LOC (Humphrey, 1995), pair defect advantage was set to 15% (Williams et al., 2000), and defect removal time was 5-20 hours/defect (Humphrey, 1989; 1995). In addition, several other values were assumed: discount rate was set to 25% to 100% per year, developer salary was set at $50,000 per year, leader salary at $60,000 per year, and work time was assumed to be 135 hours per month.

Based on the model analysis, several conclusions were drawn. First, the pair speed advantage, pair defect advantage, discount rate, and number of pairs each have a strong impact on the value of a pair programming project. Second, pair programming appears beneficial when the market pressure is really strong and programmers are much faster when working in pairs as compared to working alone. Third, if the workforce is limited, it will take a pair programming project a very strong market pressure, a large pair speed advantage, and a significant pair defect advantage to break even with the conventional project.

To summarize, these two economic models yielded different conclusions: Erdogmus and Williams (2003) suggested a positive economic picture for pair programming while Padberg and Müller (2003) concluded the benefit of pair programming depended on a variety of factors. Both models are severely restricted by one major limitation: the lack of reliable parameter values for the models. For example, Erdogmus and Williams (2003) heavily relied on productivity, defect rate, and rework speed, and Padberg and Müller (2003) depended on the data of pair speed

advantage and pair defect advantage. However, empirical evidence of these data items was very limited (Erdogmus and Williams, 2003; Padberg and Müller, 2003). The research study contained herein will attempt to mitigate this lack of reliable parameter values through the collection of data from a large sample of subjects experienced in pair and solo programming.

2.3. Effort

The majority of the extant literature focusing on effort in a pair environment concluded pair programming required more total development hours than solo programming. Nosek (1998) reported from an industrial experiment the average completion time for pairs was 40% more than solos. Williams et al. (2000) and Cockburn and Williams (2001) reported on average, pairs took 60% more programmer hours to complete the assignment, but after the adjustment period, this 60% decreased dramatically to a minimum of 15%. Rostaher and Hericko (2002) had 16 programmers, four solos and six pairs, to implement three tasks in a specified order. The experiment was limited to one day. The results suggested the average time spent to complete all three tasks by solo and pair programmers was very similar, which means pairs needed almost twice as much time to complete the same amount of work compared to individuals. This result was confirmed by student experiments conducted by Nawrocki and Wojciechowski (2001) and McDowell et al. (2003).

A few other studies shared similar findings. Arisholm et al. (2007) reported study results did not support the hypothesis that pair programming, in general, reduced the time required to solve the tasks correctly. Vanhanen and Lassenius (2007) surveyed 28 developers and found the development effort for individual features was higher for pair programming.

However, this finding was contradicted by several studies. Lui and Chan (2003) found even though pairs spent 20.9% more time, but in consideration of the same quality, pairs spent

4.2% less time than did solos on the same task. Canfora et al. (2007) reported despite cumulatively pairs using more effort, the finding was not consistent across the tasks. On the first round of tasks, the authors reported the average effort for pairs was half an hour less than that for solos (6.58 vs. 7.08 hours), but on the second round, they found the average for pairs was 2.7 hours more than that for solos (8.5 vs. 5.2 hours).

Once again, we find equivocality in the current empirical findings relating pair programming to effort.

2.4. Duration

When looking at overall project duration, most studies report pair programming served to shorten the project. Williams et al. (2000), Cockburn and Williams (2001), Williams and Kessler (2001), and Lui and Chan (2003) report findings from experiments suggesting pairs solved problems faster than solos. Rumpe and Schroer (2002) reported from a survey study that developers believed coding was completed much faster with pair programming. Vanhanen and Lassenius (2007) reported from another survey study that the opinions on the effect of pair programming on the probability of finishing a task on schedule were very positive, with the median being 6.0 using a 7 point Likert scale. Marchenko (2008c) stated his personal observations supported the claim of faster development when using the pair programming method.

In contrast, three studies suggest contradictory results. Rostaher and Hericko (2002) involved 16 developers who could choose six small tasks to form a simple insurance contract administration system. Each pair or individual had to implement as many of the tasks as possible in the exact order specified. Study results revealed that most of the developers finished four tasks and did not start the fifth one, suggesting pairs did not program faster than solos. Arisholm et al.

(2007) reported whether pair programming reduced duration depending on the task. The authors found that on more complex tasks the pair programmers had no significant differences in the time taken to solve the tasks correctly but for the simpler system, there was a 20 percent decrease in time taken. Dawande, Johar, Kumar, and Mookerjee (2008) developed analytical models to compare the performance of pair development, solo development, and mixed development under two separate objectives: effort minimization and time minimization. Their study results suggested solo programming was more appropriate for projects with a tight deadline than pair programming.

It appears from the literature that, while there is a level of equivocality with regard to the relationship between the use of pair programming and project duration, there is also material evidence to suggest the presence of an effect and the nature of that effect to be positive.

2.5. Knowledge Transfer and Learning

All industrial experience reports suggest that pair programming had a positive effect on both learning and knowledge transfer. Haungs (2001) noted pair programming on the Chrysler 3 project was very successful since it allowed him and his pair mate to synthesize unique individual expertise into an effective combination. At Iona Technologies, a great amount of knowledge transfer was witnessed through pair programming (Poole and Huisman, 2001). At Sabre Airline Solutions, the weaker people were paired with the stronger people and business knowledge and coding knowledge were transferred quickly (Anthes, 2004). At Wotif, the sharing of knowledge about the code has been perceived to be greatly improved through pair programming. (Luck, 2004). At Silver Platter Software, a startup company in California, a field experiment was conducted and pairing was found to be an effective means of knowledge transfer (Belshee, 2005). The authors stated when two people were paired, they shared knowledge. When

the pair split for a pair swap, the knowledge spread to all four participants. At Intel, two projects reported higher knowledge transfer through the pair programming method: in the IXP2xx project, developers believed they learned quite a lot from each other (Fitzgerald and Hartnett, 2005); in another project that developed firmware for processors, cross-training happened when pair programming was practiced even though it was not to the extent as the developers had hoped (Greene, 2004). Ambler (2007) reported pair programming allowed knowledge and skills to spread widely throughout the team and helped new employees learn the environment and build bonds with other team members quickly. Siobhan (2007) revealed pair programming offered learning opportunities for all team members and had a positive impact on team development due to effective communication through an open and transparent environment. At IBM, pair programming was a tool to mitigate the risk of relying on highly skilled individuals to produce results (Ambler, 2008). Marchenko (2008a) found pair programming was a very efficient tool for learning and competence transfer. Marchenko (2008c) stated, compared to pair programming, one major risk of solo programming was slow learning - especially for just graduated rookies.

These anecdotal findings were confirmed by some studies conducted by academics. Williams (1999) had 20 students in an experiment and most of the students (84%) stated they learned the materials faster and better when working with a partner. Müller and Tichy (2001) had 11 graduate students in an experiment and 43% of the participants stated that they learned something from pair programming even though this effect declined with the duration of the course. Sanders (2001) found the students believed pair programming led to higher knowledge transfer. Rumpe and Schroer (2002) stated there was immense knowledge transfer between the developers during pair programming. Janes et al. (2003) revealed pair programming was effective in sharing knowledge among 15 students who met occasionally in a three-month

summer internship. Tessem (2003) conducted a field study involving both students and practitioners and reported that all programmers found pair programming enhanced learning. Aiken (2004) found from his interviews of three industrial developers pair programming helped new people develop the knowledge of the system. VanDeGrift (2004) collected 293 responses from students and most students stated they learned about concepts covered in the course by working with a partner on the projects. Vanhanen and Lassenius (2005) conducted an experiment involving 20 college students in a programming class and found a better knowledge transfer among the pair programming teams than the solos. Vanhanen and Korpi (2007) reported from a case study that all four developers considered pair programming increased their knowledge of the system. Vanhanen and Lassenius (2007) surveyed 28 developers regarding the effects of pair programming and noted that the positive effect of pair programming was largest for learning. Dewande et al. (2008) concluded the pair programming approach was preferable for efficient knowledge sharing between developers.

There are a couple of exceptions, however. Cliburn (2003) found from a survey study that most of the students stated they learned more when they worked by themselves. In Hanks et al. (2004), the hypothesis of paired students showing a better understanding of the programming concepts were not supported.

Despite these two contrasting studies, it appears from practitioner and academic literature that pair programming is positively associated with the transfer of knowledge when compared to rival methods.

2.6. Confidence, Enjoyment, and Retention

Studies in this category were largely conducted by Williams, McDowell, and colleagues in programming classes at university settings. Virtually all of their findings suggest students

pairs were more confident in their solutions (Williams, 1999; William et al., 2000; Williams and Kessler, 2000; 2001; McDowell et al., 2003; 2006; Hanks, McDowell, Draper, and Krnjajic, 2004), enjoyed completing the assignments more (Williams, 1999; Williams et al., 2000; Williams and Kessler, 2000; 2001; Cockburn and Williams, 2001; Nagappan, Williams, Wiebe, Miller, Balik, Ferzli, Petlick, 2003; Nagappan, Williams, Ferzli, Wiebe, Yang, Miller, Balik, 2003; Williams, McDowell, Nagappan, Fernald, and Werner, 2003; McDowell et al., 2003; 2006; Hanks et al., 2004), and one year later were more likely to pursue computer science-related majors than students who programmed alone (McDowell et al., 2002; Nagappan, Williams, Wiebe, Miller, Balik, Ferzli, Petlick, 2003; Nagappan, Williams, Ferzli, Wiebe, Yang, Miller, Balik, 2003; Williams et al., 2003; McDowell et al., 2003; 2006).

Their results were supported by several other studies. Nosek (1998) reported paired developers expressed higher confidence about their work and enjoyment of the process. Poole and Huisman (2001) reported from an industry experience that developers enjoyed the pair programming process. Luck (2004) echoed these same findings from a different industrial experience. Succi, Pedrycz, Marchesi, and Williams (2002) reported from 108 survey responses collected from around the world that there was a very positive effect of pair programming on job satisfaction; Declue (2003) suggested pair programming increased confidence. Fitzgerald and Hartnett (2005) stated the paired developers in the project reported they had more fun, found the work more interesting and were more enthusiastic about their work. Müller & Tichy (2001), Sanders (2001), Cliburn (2003), Gehringer (2003), VanDeGrift (2004), and Mendes et al. (2005) all found the majority of the students enjoyed the pair programming experience.

As with much of the pair programming research, a few studies reported mixed results. Tessem (2003) stated although all programmers reported that pair programming was a positive

experience, it was contradicted by three of the six programmers who also used negative phrases such as “very exhausting”, “tiresome”. Vanhanen and Lassenius (2005) found half of the 12 student participants in the pair programming teams actually enjoyed solo programming more than pair programming. Vanhanen and Korpi (2007) revealed from a four-developer pair programming project that even though all of the four developers agreed pair programming promoted the formation of good team spirit in the beginning, only two developers liked pair programming more than solo programming and the other two found no difference. Finally, Balijepally et al. (2009) found programming pairs reported higher levels of satisfaction than those of the best and second-best performing members in nominal pairs. Regarding confidence, however, the confidence levels of pairs were no different from those of the best performing members in nominal pairs.

Such variables have clearly not been researched in either large samples or with consistent measurements. This remains an unanswered question with regard to pair programming.

2.7. Subject Variables

Several industrial comments and stories suggest it would take people with unique characteristics for successful pair programming. James Gosling, a former vice president and fellow at Sun Microsystems Inc., commented the company used some extreme programming (XP) techniques but passed on pair programming because he didn’t think people would do it. “[It gives] most of the people I know the creeps." (Copeland, 2001b). As an editor, in the Loyal Opposition section of IEEE Software, Glass (2001) disaggregated XP and examined its constituent elements in isolation. He listed pair programming as the number one Bad News. He stated he could not imagine holding ongoing conversations with a pair mate when he was operating in creative mode and did not believe that many programmers he knew would want to

operate that way. This view was echoed by Fitzgerald and Hartnett (2005) which noted the difficulty for one partner to reflect and concentrate with someone by his/her side. Taking this further, Greene (2004) reported that even though pair programming worked well, in general, at Intel, hard deadlines created problems for pair programming when one developer had more domain knowledge. He suggested at crunch time it was more expeditious for the expert to work on the implementation alone; pairing with another developer would reduce productivity.

Several studies identified pair composition as an important factor. Sanders (2001) found from a student survey that pair compatibility was considered a crucial attribute to pair programming success. In an opinion paper that provided guidelines on how to successfully implement pair programming in college programming courses, Bevan, Werner, and McDowell (2002) suggested it was important to pair students by skill level to avoid compatibility issues.

Even though studies agreed on the importance of pair composition, results differed in terms of how to achieve the best pair. Thomas, Ratcliffe, and Robertson (2003) paired student subjects based on their self perception of programming abilities. Study results revealed that students produced their best work when placed in pairs with students of similar self-confidence levels. In contrast, however, Jensen (2003) reported pairing programmers of the same experience and capability level was often counter-productive. The most troublesome pairs they dealt with during the experiment were two teams in which both members were near the same capability level. The worst-case team consisted of two *prima donna* programmers. They found teams functioned more smoothly, if one member was slightly more capable than the other. Katira, Williams, Wiebe, Miller, Balik, and Gehringer (2004) conducted experiments with 564 students. They examined compatibility among freshman, advanced undergraduate, and graduate students and found that the students' perception of their partner's skill level had a significant influence on

their compatibility. Bryant (2004) conducted a case study in a large internet banking company. Study results suggested novice pairs and more experienced pair programmers displayed different interaction patterns. For instance, novice pairs suggested and counter-suggested much more frequently. Also, where more experienced pair programmers worked together, there seemed to be a defined set of behaviors which remained common whoever was driving. By contrast, less experienced pair programmers seemed to behave quite differently from one another, even when filling the same role. The study revealed the effect of programmer experience on pair programming effectiveness. Cao and Xu (2005) conducted an experiment involving 23 college students. They compared activity patterns between different pair combinations and found differences in the activity patterns amongst different pair combinations. The High-High (ability) pairs enjoyed the pair programming process, passed most of the test cases, were confident on their programs, and believed there was higher knowledge transfer. Medium-Medium and High-Low pairs reported conflicting results on all aspects. Nedland (2005) suggested pairing with less motivated coworkers made it hard to maintain enthusiasm when every part of the practice was questioned and there was a general attitude of hostility. Lui and Chan (2006) noted novice-novice pairs against novice solos were much more productive in terms of completion time and software quality than expert-expert pairs against expert solos. In addition, a pair was much more productive when the pair was new to a programming problem and the problem was challenging. This suggests that pair programming may reduce costs when utilized in a more novice environment. Van Toll III, Lee, and Ahlswede (2007) confirmed Jensen's theory — pair programming worked best when the pairs were composed of slightly different skill levels. Programming with significantly less experience seemed to create problems: the programmer with more experience must be patient since he found himself constantly having to answer the same

questions over and over again. If there was a deadline set, he probably would have ended by doing the entire project himself. Dewande et al. (2008) reported the same finding: pair approach was better at leveraging expertise by pairing experts with less skilled partners.

A couple of studies noted the importance of pair rotation which suggests pair partners need to be changed on a regular basis. Tessem (2003) found from a field study that frequent partner changes were necessary to achieve optimal learning and also to increase the sense of collective ownership amongst the programmers. Srikanth et al. (2004) collected 287 survey responses and found the majority of students perceived pair rotation to be a desirable approach.

Several studies addressed other issues such as expertise, personal traits, and other developer characteristics. Dick and Zarnett (2002) opined that not all developers were suited for paired development. The team members should be selected with personality traits that were beneficial to paired programming, which could be determined through various interview techniques and the corresponding behavioral responses of the candidates. Nilsson (2003) found from a survey study that developers believed the "personal chemistry" was very important for pairs to work efficiently. Aiken (2004) interviewed three software engineers. The engineers did not think pair programming would work for everybody. Many software engineers were accustomed to working alone. Some people were uncomfortable to make a mistake in front of someone else. From a scheduling perspective, pairing up different capability levels slowed down the more productive leaders. However, the engineers believed pair programming would be a great way to get people up to speed on something, especially when somebody was new. Müller and Padberg (2004) had 38 student subjects in two controlled experiments and found a significant correlation between pair performance and how comfortably the developers feel with pair programming during the session (the "feelgood" factor). Langr (2005) listed several reasons

for resisting pair programming. He stated pair programming was not for everybody. Some people had fear of exposing personal weakness, while others might have personality issues such as introversion. Arisholm et al. (2007) reported from their one-day experiment that the moderating effect of system complexity depended on the programmer expertise: the observed benefits of pair programming in terms of correctness on the complex system applied mainly to juniors, whereas the reductions in duration to perform the tasks correctly on the simple systems applied mainly to intermediates and seniors. Chong and Hurlbutt (2007) conducted a four month ethnographic study of professional pair programmers from two software development teams. They found the programmers with more expertise dominated the interactions and had the final authority in decision making.

It can be seen from a review of the literature in this area that pair composition generates a wide variety of opinions and findings with no consensus forthcoming. This research effort intends to shed significant light in this area of investigation.

2.8. Task Variables

Several studies noted that pair programming was better suited for complex tasks. Nilsson (2003) reported from survey responses pair programming was ill suited for simple and routine tasks. Through experiments Lui and Chan (2003) found that pairs outperformed individuals in terms of effort and quality when there was a new and challenging problem. Fitzgerald and Hartnett (2005) found out pair programming helped solve difficult coding problems and was better suited for complex tasks. Hulkko and Abrahamsson (2005) revealed developers did not consider pair programming to be efficient for simple and routine-like tasks and many developers preferred to do simple tasks on their own. Arisholm et al. (2007) examined the moderating effect of task complexity in an one-day experiment and reported that on the more complex tasks, pair programmers had a 48 percent increase in the proportion of correct solutions but for simpler

systems there was no significant differences in the time taken to solve the tasks correctly. Vanhanen and Korpi (2007) discovered effort depended on the type of task. For complex tasks, the use of pair programming lowered the total effort, but the effort was considered higher for simple tasks than when using solo programming.

There are a couple of exceptions, however. While other studies suggested pair programming led to better quality when the task was complex, Al-Kilidar, Parkin, Aurum, and Jeffery (2005) found pairs produced better quality product when requirements were simple with no significant difference in quality between pairs and solos when requirements were complex. The authors attributed this result partly to the solos in the more complex module having developed and enhanced their skills through their earlier work experience. Vanhanen and Lassenius (2005) did not support the claim that pair programming was most useful for complex tasks. In their study, task complexity did not affect the effort differences between solo and pair programming.

Again we find equivocality and confound.

2.9. Environment and Organizational Variables

Several industrial reports and studies suggested the importance of work space facilities and management support for pair programming to work. Connextra Ltd., a London start-up and maker of Web browser software, reorganized its offices to accommodate XP, installing curved desks that let two developers sit side by side and share a computer (Copeland, 2001a) thus allowing for a pair programming environment. Jensen (2003) posited that coordination among the developers in a particular development setting he studied would have improved if the teams had been working in a common area. He used the term war room (or skunk works) to describe the ideal open environment, which would be a large area with worktables in the center and

cubicles around the outside. Software engineers in Aiken's interviews (2004) argued for getting a common work environment to remove distractions and to provide feelings of a team. Vanhanen, Lassenius, and Mäntylä (2007) presented experiences from a two-year study focused on the adoption of pair programming and noted that issues identified in the infrastructure of pair programming were solved through the adoption of the pair programming rooms.

The importance of management support was echoed by some studies. Jensen (2003) noted that managers must be supportive of the pair programming process. In a particular anecdote, a classic manager observed a programming pair working on a task over a period of time and suggested to the supervisor that one of the two programmers be laid off because only one was doing anything constructive (the driver always gets the credit). When the supervisor heard the suggestion, he replied that these programmers were the most productive people in the organization. The manager then asked that the programmers keep their office door closed so others would not reach the same wrong conclusion. Williams and Kessler (2003) noted it was crucial to overcome management resistance (chapter 4, pp.33-44): pair programmers create noise and resolving this problem may require a commitment from the organization in terms of a facility change or equipment purchase; pairing can also be difficult if cubicles or desks are arranged improperly. At Intel, managers tore down cubicle walls to lower the physical barriers to communication and introduced a new individual evaluation criterion: teamwork that reinforced the value of collaborative work (Greene, 2004).

2.10. Summary of Literature Review

To summarize this arguably small and disparate body of literature focusing on pair programming suggests both contributions to the body of knowledge and little consensus.

Possibly the greatest single accomplishment is the realization that pair programming has become one of the most studied principles in XP (Erickson, Lyytinen, and Siau 2005). A substantial number of studies have been conducted to study the variety of effects of pair programming in different contexts: industrial setting, classroom setting, comparing pair to solo, comparing pair to other group development methods, distributed pair programming, pair composition, and the economic aspects of pair programming. The results of these studies undoubtedly provide valuable information to the overall picture of the efficacy of pair programming but many disagreements and questions remain.

There are several caveats that must be noted when reviewing this body of literature. First, the majority of the studies were conducted using neither theory nor any recognizable or identified research framework. The one extant and comprehensive pair programming research framework suggested by Gallis et al. (2003) and extended by Ally et al. (2005) has been largely ignored. Few empirical studies have been conducted to test and refine the model. Actually, among the studies reviewed above, the only empirical studies that explicitly presented a research model are Arisholm et al. (2007) and Balijepally et al. (2009).

Second, most of the results were obtained from experimental studies in university settings. Only a few empirical studies involved industrial practitioners. And even with those studies, since professional programmers worked on short tasks, the complexity of real world software development was not reflected.

Third, most survey studies were conducted without academic rigor. None of the survey studies went through the standard cycle of reliability and validity checks, which brings the subsequent results further under question.

Finally, studies on the economic aspect of pair programming, upon which the study herein is focused, are limited. Only two major studies were identified. The two studies looked into different factors, used unsure parameters, and drew different conclusions. No other study, thus far, has attempted to reconcile the different conclusions from the two studies.

In conclusion, our literature review echoes the comments several authors made in their studies. Gallis et al. (2003) noted that results from existing empirical work contradicted each other due to the lack of a theoretical framework to support the pair programming research. Hulkko and Abrahamsson (2005) stated that the current body of knowledge in pair programming is scattered and unorganized. Parrish, Smith, Hale, and Hale (2004) suggested more empirical evidence from real industry projects is needed.

Table 2.1 is presented as a tabular summary of the extant literature on pair programming.

Table 2.1 Summary of Pair Programming Studies

LEGEND
Publication Outlet: **A**=Refereed Academic Journal **P**=Practitioner Journal or Practitioner-Oriented websites **B**=Book or Book Chapters **T**=Student Thesis **C**=Conference Proceeding
Research Method: **C**=Case Study **MA**=Meta Analysis **I**=Industrial Report **S**=Survey **E**=Experiment **O**=Opinion Paper **IV**=Interview **SM**=Simulation
Subject Type: **I**=Industrial Practitioner **S**=Student **M**=Mixed of Students and Practitioners
Statistical Analysis: **DC**=Descriptive Statistics Only **Y**=Performed Statistical Analysis **N**=No Statistical Analysis
Constructs of Interest: **Y**=Supported **N**=Not Supported **M**=Mixed Results

Study	Study Characteristics					Constructs of Interest											
	Publication Outlet	Research Method	Sample Size	Subject Type	Statistical Analysis	Better Quality	Higher Cost	Productivity	More Effort	Shorter Duration	Knowledge	Confidence	Enjoyment	Retention	Pair Characteristics	Task Characteristics	Environment Variables
Nosek (1998)	A	E	15	I	Y	Y			Y	Y		Y	Y				
Williams (1999)	C	S	20	S	D	Y		Y			Y	Y	Y				
Williams and Kessler (2000)	C																
Williams and Kessler (2001 Exp #1)	A																

Study	Study Characteristics					Constructs of Interest										
	Publication Outlet	Research Method	Sample Size	Subject Type	Statistical Analysis	Better Quality	Higher Cost	Productivity	More Effort	Shorter Duration	Knowledge	Confidence	Enjoyment	Retention	Pair Characteristics	Task Characteristics	Environment Variables
Williams et al. (2000), **Williams and Kessler (2001 Exp #2)** **Williams and Upchurch (2001)**	A A C	E	41	S	Y	Y			Y	Y		Y	Y				
Cockburn and Williams (2001)	B	O		M	D	Y	M		Y	Y	Y		Y				
Copeland (2001a)	P	I				Y	M										Y
Copeland (2001b)	P	I													Y		
Haungs (2001)	A	I				Y		Y			Y						
Müller & Tichy (2001)	C	S	11	S	N						Y		Y				
Nawrocki and Wojciechowski (2001)	A	E	21	S	D			N	Y								
Poole and Huisman (2001)	A	I	40-70	I	N	Y		Y			Y		Y				
Sanders (2001)	C	S	60	S	N	Y					Y				Y		
Bevan, Werner, and McDowell (2002)	C	O													Y		Y
Dick and Zarnett (2002)	C	O													Y		
McDowell et al. (2002)	C	E	600	S	Y	Y								Y			
Morales (2002)	P	I				Y											
Müller and Padberg (2002)	C	SM			Y	M	M		M						Y	Y	
Radding (2002)	P	I				Y	N										
Rostaher and Hericko (2002)	C	E	16	I	Y				Y	N							
Rumpe and Schroer (2002)	C	S	45	I	D					Y	Y						
Succi et al. (2002)	C	S	108	I	Y								Y				
Cliburn (2003)	A	S	17	S	N	Y					N		Y				
Declue (2003)	A	S	24	S	D	Y						Y					
Erdogmus and Williams (2003)	A	SM			Y		N										

Study	Study Characteristics					Constructs of Interest										
	Publication Outlet	Research Method	Sample Size	Subject Type	Statistical Analysis	Better Quality	Higher Cost	Productivity	More Effort	Shorter Duration	Knowledge	Confidence	Enjoyment	Retention	Pair Characteristics	Task Characteristics	Environment Variables
Gehringer (2003)	C	E	73/75/96/101	S	Y	M							Y				
Janes et al. (2003)	C	E	15	S	Y						Y						
Jensen (2003)	P	I				Y		Y							Y		Y
Lui and Chan (2003)	A	E	15	I	D	Y			M	Y						Y	
McDowell, Hanks, and Werner (2003)	A	E	288	S	Y	Y			Y								
McDowell, Werner, Bullock, and Fernald (2003, 2006)	C	E	554	S	Y	Y						Y	Y	Y			
Müller and Padberg (2003)	A	SM			Y		M								Y		Y
Nagappan, Williams, Wiebe, Miller, Balik, Ferzli, Petlick (2003)	A	E	412	S	Y	Y							Y	Y			
Nagappan, Williams, Ferzli, Wiebe, Yang, Miller, Balik (2003)	A	E	495	S	Y	Y							Y	Y			
Nilsson (2003)	T	S	67	M	D	Y	Y								Y		
Stephens and Rosenberg (2003)	B	O					Y								Y	Y	Y
Tessem (2003)	C	C	1	I	N	Y		M			Y		M		Y		
Thomas, Ratcliffe, and Robertson (2003)	C	E	64	S	Y	M							M		Y		
Williams and Kessler (2003)	B	O				Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
Williams, McDowell, Nagappan, Fernald, and Werner (2003)	C	E	1200	S	Y	Y							Y	Y			
Aiken (2004)	A	IV	3	I	N		Y			N	Y				Y		Y
Anthes (2004)	P	I				Y					Y						
Bryant (2004)	C	C	1	I	N										Y		
Greene (2004)	C	I						N			Y				Y		Y
Hanks et al. (2004)	A	E	150	S	Y	M					N	Y	Y				
Katira et al. (2004)	A	E	564	S	Y										Y		

Study	Study Characteristics					Constructs of Interest											
	Publication Outlet	Research Method	Sample Size	Subject Type	Statistical Analysis	Better Quality	Higher Cost	Productivity	More Effort	Shorter Duration	Knowledge	Confidence	Enjoyment	Retention	Pair Characteristics	Task Characteristics	Environment Variables
Luck (2004)	C	I				Y			Y		Y		Y				
Müller and Padberg (2004)	C	E	38	S	Y										Y		
Parrish et al. (2004)	A	E	48 Mod	I	Y			N									
Srikanth et al. (2004)	C	S	287	S	Y										Y		
VanDeGrift (2004)	A	S	293	S	D			Y			Y	Y	Y				
Williams, Shukla, and Anton (2004)	C	S	30	I	Y			Y									
Al-Kilidar et al. (2005)	C	E	150	S	Y	M										Y	
Belshee (2005)	C	E		I	N						Y						
Canfora et al. (2005)	C	E		S	Y				N								
Cao and Xu (2005)	C	E	23	S	N	M					M	M	M		Y		
Fitzgerald and Hartnett (2005)	A	C	1	I	N	Y				Y	Y		Y			Y	
Hulkko and Abrahamson (2005)	C	C	4	M	D	M		N								Y	
Langr (2005)	P	O													Y		
Mendes et al. (2005)	C	E	300	S	Y	Y							Y				
Nedland (2005)	P	I						Y							Y		Y
Preston (2005)	A	O									Y						
Vanhanen and Lassenius (2005)	C	E	20	S	Y	N		N			Y		M			N	
Lui and Chan (2006)	A	E	40	S	D			M							Y	Y	
Madeyski (2006)	C	E	188	S	Y	N											
Müller (2006)	A	E	18	S	Y	Y	M										
Ambler (2007)	P	I									Y						
Arisholm et al. (2007)	A	E	298	I	Y	M			M	M					Y		Y
Canfora et al. (2007)	A	E	18	I	Y	Y			M								
Chong and Hurlbutt (2007)	C	C	2	I	N										Y		
Dybå et al. (2007)	A	MA	15	M	Y	Y			Y	Y							
Siobhan (2007)	P	O									Y						
Van Toll III, Lee, and Ahlswede (2007)	C	E	NR	S	N										Y		
Vanhanen and Korpi (2007)	C	C	4	I	Y	M			M		Y		M			Y	

Study	Study Characteristics					Constructs of Interest										
	Publication Outlet	Research Method	Sample Size	Subject Type	Statistical Analysis	Better Quality	Higher Cost	Productivity	More Effort	Shorter Duration	Knowledge	Confidence	Enjoyment	Retention	Pair Characteristics	Task Characteristics	Environment Variables
Vanhanen and Lassenius (2007)	C	S	28	I	Y	Y			Y		Y		Y				
Vanhanen, Lassenius, and Mäntylä (2007)	C	C	1	I	Y												Y
Ambler (2008)	P	I									Y						
Dawande, Johar, Kumar, and Mookerjee (2008)	A	SM									Y				Y	Y	
Marchenko (2008a)	P	O				Y					Y						
Marchenko (2008b)	P	O				Y					Y						
Marchenko (2008c)	P	O				Y				Y							
Balijepally et al. (2009)	A	E	120	S	Y	M						M	Y				
Wray (2010)	A	O				Y					Y						

3. RESEARCH MODEL AND HYPOTHESES

When considering a synthesis of the reviewed literature, findings from pair programming research suggest the research model shown in Figure 3.1. In this model, the programming method solo vs. pair is the independent variable while the project characteristics such as complexity and the developer characteristics such as expertise moderate the relationships between programming method and the dependent variables. All previous pair programming studies that presented an explicit research model (Gallis et al., 2003; Arisholm et al., 2007; Balijepally et al., 2009) followed this approach, even though none of the studies empirically tested the knowledge transfer construct.

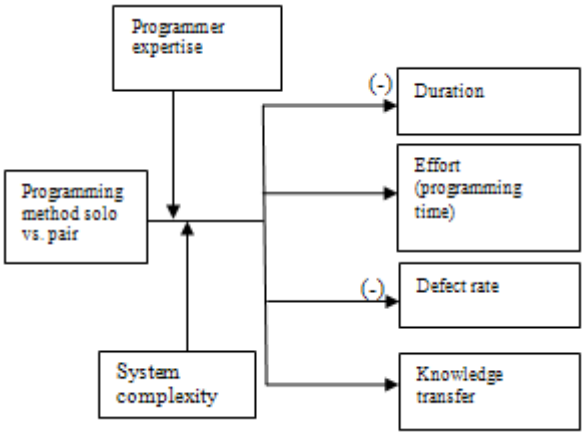


Figure 3.1 Research Model based on Previous Pair Programming Studies

Building upon the known relationships shown in Figure 3.1, we extend these findings in an effort to advance the nomological net and propose the research model for this study in Figure 3.2. As shown in the figure, we argue the nature of the project (system complexity) has a direct impact on the project outcome, and programming method can be used to mitigate the strength of the relationship between the project and dependent variables such as effort, defect rate, and knowledge transfer. Therefore, we propose system complexity as the independent variable and programming method as a moderator.

We also argue pair composition affects the effectiveness of pair programming method. While this argument is in line with suggestions from previous pair programming studies, in our model, pair composition is presented as a moderator to the effect of programming method. Additionally, as discussed later in this chapter, we believe our operationalization of the salient constructs to be more comprehensive than previously accomplished.

We further argue project duration can be derived from effort and, therefore, does not need to be viewed as a stand-alone dependent variable. Another extension in this model, as we are interested in the cost of the project as it relates to programming method, is the inclusion of

several cost-related constructs. Note that the constructs within the dashed box are derived from logic rather than effect. As such, they require no specific hypothesis testing. They are included to demonstrate how cost is considered in this research. These variables are more important in Study 2 than Study 1 (as illustrated in chapter 4).

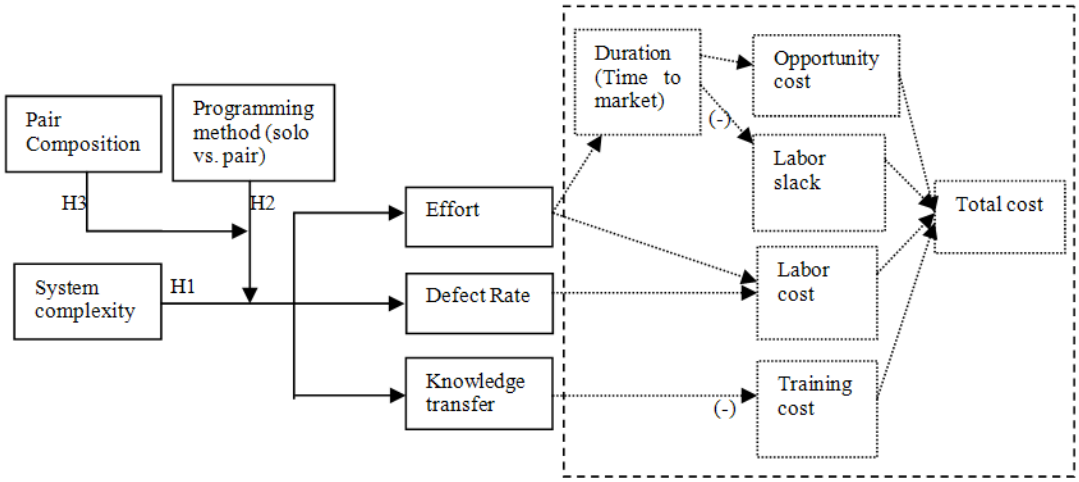


Figure 3.2 Research Model for this Study

3.1. System Complexity, Effort, Defect Rate, and Knowledge Transfer

Some software projects are simple in nature while others can be highly complex. Writing a program to display the message "Hello World!" is a simple task that requires a few lines of code and can be accomplished in minutes. Developing a web-based business application for retail sales represents a medium-level complexity requiring thousands of lines of code and project durations measured in months. Constructing an operating system such as Windows 7 is an extremely complex task as it constitutes millions of lines of code and demands years of hard work.

According to March and Simon (1958), complexity can be generally characterized by unknown or uncertain alternatives, unknown means-ends connections, and existence of a number of subtasks that may not be easily factored into independent parts. As system complexity

increases, the number of alternatives, the number of unknown connections, and the number of interrelated tasks all increase. In order to identify the best or optimal solution among the many possibilities, developers have to weigh different alternatives, clarify various connections, and sort out the variety of tasks involved. This process becomes more time consuming and error prone as complexity increases.

However, despite the challenges of system complexity, a positive note is that a complex system requires more diverse knowledge in order to identify a good solution. Compared to a simple system where a solution is obvious and does not warrant much discussion, a complex system forces developers to share, process, and synthesize a variety of information, therefore increasing knowledge transfer among the team members.

In keeping with the above, we hypothesize:

H1a. *Regardless of programming approach, as system complexity increases, the programming effort increases.*

H1b. *Regardless of programming approach, as system complexity increases, defect rate increases.*

H1c. *Regardless of programming approach, as system complexity increases, there is higher knowledge transfer rate amongst the project team members.*

In this study, system complexity is operationalized as the number of modules in a given project, and then classified into low, medium, and high. A module is a self-contained program that carries out a clearly defined task. As the number of modules grows, more communications amongst the modules are required, thus increasing system complexity. This argument is in line with what is suggested by several studies in project complexity (Banker, Davis, and Slaughter, 1998; Kemerer, 1995; Espinosa, Slaughter, Kraut, and Herbsleb, 2007).

Previous pair programming studies have adopted a variety of measures for complexity: application control style-delegated vs. centralized (Arisholm et al., 2007), deduction vs. procedural problem (Lui and Chan, 2003), modification of five methods in two classes vs. seven methods in five classes (Balijepally et al., 2009). Although these measures were reasonable for their respective experimental studies, they could not be readily generalized to other studies: control style is only applicable for projects that adopt different control styles, whether developers solve a deduction or procedural problem depends on the project domain, and the concepts of methods and classes pertain only to object-oriented development. In this study, we adopt a measure that can be generalized across the projects.

Effort is defined as the total number of hours spent by a developer or developers on the programming aspect of a project. In solo programming, for a particular programming task, the effort is the number of hours one developer spends on the task. In pair programming, since two developers work on the same task at the same time, the effort is two times the number of hours one developer spends on the task.

This definition of effort is widely used for software project effort measurement (Pressman, 2005; Sommerville, 2007), and was adopted by many previous pair programming studies (Nosek, 1998; Williams et al., 2000; Nawrocki and Wojciechowski, 2001; Rostaher and Hericko, 2002; McDowell et al., 2003; Dybå et al., 2007).

A defect is a quality problem in the source code that is found after the software has been released to the end-users, and defect rate is the number of defects per thousand lines of codes (defects/KLOC). This definition represents a commonly used software engineering metric developed to measure the correctness aspect of software quality (Pressman, 2005; Sommerville 2007). It has been adopted by the industry and several empirical studies on software

development (Abdel-Hamid, Sengupta, and Swett, 1999; Ji, Mookerjee, and Sethi, 2005; Kandt, 2009).

The previous pair programming studies reported results on defects, error rate, and defect density without providing explicit definitions of the constructs. In this study, we intend to enhance the nomological net of pair programming research by adopting a commonly agreed-upon metric for software defects.

Knowledge transfer is the communication of knowledge from a source so that it is learned and applied by a recipient (Ko, Kirsch, and King, 2005). In this research effort, three aspects of knowledge transfer are measured: knowledge transfer with regard to programming syntax and logic, understanding of the program itself and its relationship to the overall system, and the approaches to solve problems in general.

None of the previous pair programming studies, to the best of our knowledge, developed explicit measures for knowledge transfer. In this study, we develop measures for knowledge transfer through tailoring concepts introduced in information systems.

3.2. Programming Method as a Moderator

Pair programming studies generally found pair programming incurred more programming effort than solo. Nosek (1997) reported 41% more effort compared to solo. Williams et al. (2000) and Cockburn and Williams (2001) noted 15% more effort compared to solo. Nawrocki and Wojciechowski (2001), Rostaher and Hericko (2002), and McDowell et al. (2003) found 100% more effort compared to solo. The meta analysis study conducted by Dybå et al. in 2007 reported a medium-sized negative effect due to pair programming (effect-size=-0.57).

We argue the magnitude of effort increase by pair programming is associated with system complexity. With a low complexity system, a solution is obvious regardless of whether one

programmer or two programmers work on the task, therefore pairing up two programmers on such a task is likely to waste one of the programmers' time, thus doubling the programming effort. As complexity increases, however, the amount and diversity of information needed to solve the problem increases — possibly exponentially. In pair programming, two programmers can share the information processing load and therefore have the potential to reach a workable solution more efficiently. With an extremely complex system, it is possible the information is so overwhelming that it takes a solo programmer a long time to figure out the solution. Conversely, in a pair programming environment, the programmers share the information processing load and have the chance to create synergy thus solving the problem more quickly. Following this logic, it is feasible to assume the possibility of pair programming resulting in a decrease in programming effort when compared to solo.

The above argument is in line with findings from Vanhanen and Korpi (2007) which suggest pair programming reduced the total effort for complex tasks but increased effort for simple tasks.

Thus, we hypothesize:

H2a. *When compared to solo programming, as system complexity increases, pair programming will moderate the effect of system complexity on programming effort thus reducing effort.*

In a low complexity system, defect rate is generally low regardless of the programming method adopted. Assuming pair programming has the potential to reduce the defect rate when compared to solo programming, in low complexity systems there isn't much room to demonstrate improvement. For example, any programmer with a bit of training can write the "Hello World" program with zero defects, so pairing two programmers to work on such a program does not help with defect rate since the best they can do is to produce a program with

zero defects. Though admittedly contrived and simplistic when compared to actual programming efforts, this example nonetheless illustrates the small effect pair programming can have to reduce defect rate when the baseline is already very low.

However, in the case of a complex system where a good solution hides among many possible alternatives, pair programming enables the programmers to bring a variety of skills to the table, to more thoroughly weigh different alternatives, and to have a better chance to identify the best solution. This argument is in line with the theories of distributed cognition (Flor and Hutchins, 1991; Williams and Kessler, 2001; Williams and Upchurch, 2001) and co-discovery (Papert, 1980; Lim, Ward, and Benbasat, 1997). The theories state by engaging in deeper level thinking and searching through larger spaces of alternatives, working as a pair helps improve one's mental model and reduces the chances of selecting a bad plan. It is also in line with findings from group/team research in organizational behavior which suggests complex tasks benefit more from discussions amongst the group members on alternative solutions (Robbins, 2000). This argument is further supported by several pair programming studies (Lui and Chan, 2003; Nilsson, 2003; Fitzgerald and Hartnett, 2005; Hulkko and Abrahamsson, 2005).

We hypothesize:

H2b. *When compared to solo programming, as system complexity increases, pair programming will reduce the effect of complexity on defect rate thus reducing the overall defect rate.*

To our knowledge, no empirical studies in pair programming have examined the relationship between project complexity and knowledge transfer. We argue a complex task requires more diverse knowledge and heavier load of information processing in order to identify a good solution. Compared to a simple task, where a solution is obvious and does not warrant

much discussion, a complex task will force developers to share, process, and synthesize a variety of information, therefore transferring more knowledge amongst themselves. This argument is in line with the cognitive load theory in a group setting which states the cognitive load can be shared amongst group members enabling them to deal with more complex problems than individuals (Kirschner, Paas, and Kirschner, 2008). Therefore, we hypothesize:

> **H2c**. *When compared to solo programming, as system complexity increases, pair programming will increase the effect of system complexity on knowledge transfer thus enhancing knowledge transfer.*

3.3. Pair Composition

Studies generally suggested pair composition as an important factor affecting the overall success of a pair programming effort (Sanders, 2001; Bevan et al., 2002; Gallis et al., 2003; Arisholm et al., 2007; Dybå et al., 2007). In this study, we consider programmer expertise and prior pair programming experience as two salient characteristics of pair composition. No prior research, to our knowledge, has empirically tested the effect of prior pair programming experience on pair effectiveness.

Several studies report the effectiveness of pair programming depends on the programmer expertise. Nosek (1998) found programmers with more years of experience performed better than programmers with fewer years of experience in a pair environment. Jensen (2003) and Van Toll III et al. (2007) suggested pair programming worked best when the pairs were of a slightly different skill level. Lui and Chan (2006) concluded novice-novice pairs against novice solos were more productive than expert-expert pairs against expert solos. Chong and Hurlbutt (2007) noted pairing a less knowledgeable programmer with a more knowledgeable programmer was effective when the less knowledgeable one was new to both the team and the code base. Finally,

Arisholm et al. (2007) tested programmer expertise as a moderator in their study and found the effect of pair programming on duration and defects dependent upon expertise.

We argue the effectiveness of pair programming varies when pairs of different compositions - junior-junior, senior-senior, and junior-senior are compared to junior/senior solos. As such, we hypothesize:

> **H3a***: Programmer expertise moderates the effectiveness of the pair programming method.*

In this study, we adopt the terms of *junior* and *senior* to create a binary representation for levels of programmer expertise in a project. A senior programmer is defined as one who has at least six years of experience within the project domain. A junior programmer is defined as one who has less than two years of experience within the project domain. The area occupied by these two experience anchors can be considered to be a mid-level programmer. For this study, we are focusing on the extremes of the continuum and, as such, a mid-level programmer will not be specifically considered.

Among the pair programming studies that reported on the effect of programmer expertise, only a few reported explicit measures for this construct. Arisholm et al. (2007) measured programmer expertise by two indicators: programmer skill category as reported by the project managers and skill levels based on the results of a pretest programming task. Balijepally et al. (2009) treated programming ability as a control variable and determined it by computing the subject's weighted average GPA in information systems courses taken at the university. In our study, since we adopt different research methods than have been previously employed (described in detail in chapter 4), it is not possible to adopt either of these measures. However, our method of using the number of years of experience to differentiate expertise is in keeping with common

practice in information systems research in general and in line with the recommendations made by studies such as Harel and McLean (1985) in software development literature.

Regarding prior pair programming experience, despite no study empirically testing this construct, two studies suggest its importance on programmer interaction patterns and project outcomes. Bryant (2004) found there were different interaction frequencies and interaction types between novice and expert pair programmers: expert pair programmers averaged 27% fewer interactions per hour and novice pair programmers suggested and counter-suggested more frequently (82.5 times per session) than experts (42.5 times per session).

The concept of pair jelling was discussed in Williams et al. (2000). The authors suggested that programmers went through an initial adjustment period in the transition from solitary to collaborative programming. This adjustment period varied from hours to days. The authors reported that in their student experiments, after the adjustment in the first assignment, the paired students performed much better in the subsequent tasks. For example, on average, pairs took 60% more programmer hours than their solo counterparts to complete the assignment, but after the adjustment period, this 60% decreased to 15%.

We argue prior pair programming experience allows the programmers to get adjusted to each other and become productive quickly. As such, we hypothesize:

> **H3b***: Prior pair programming experience moderates the effectiveness of the pair programming method.*

Prior pair programming experience is measured by whether a programmer has programmed in pairs before. Since none of the prior studies provided an explicit definition on this construct, one of the contributions of this study is to define this construct explicitly.

3.7. Duration

Numerous anecdotes and several empirical studies support the notion that pair programming allows the faster delivery of a project when compared to solo. Lui and Chan (2003) suggested pairs were faster than solos. Williams et al. (2000) and Williams and Kessler (2001) both found that pairs produced programs with shorter cycle time. Rumpe and Schroer (2002) and Vanhanen and Lassenius (2007) reported from survey studies that developers believed coding was completed much faster with pair programming. Dybå et al. (2007) noted a medium-sized overall reduction of the project duration (effect-size = 0.40) in their meta-analysis.

Project duration is defined as time elapsed from the start to the delivery of a project. While previous studies usually develop hypotheses to test the effects of programming method on duration, we argue duration, *ceteris paribus*, can be derived from a measure of effort. For instance, if the measured efforts in solo and pair programming are 140 and 210 hours respectively, then assuming a seven hours workday, solo programming will finish the project in 20 days (140/7) while pair will finish it in 15 days (210/2/7). While we freely acknowledge there are other aspects to a project besides the level of programming effort, we believe the choice of programming method will not dramatically alter those aspects of the project and, therefore, we believe there is no need of additional data collection for explicit hypothesis testing.

3.8. Cost

Study results are mixed regarding whether pair programming increases overall project cost. Several studies suggest pair programming increases cost (Cockburn and Williams, 2001; Rostaher and Hericko, 2002; Nilsson, 2003: Aiken, 2004; Luck, 2004) while others argue pair programming to be a cost saving mechanism (Copeland, 2001a; Radding, 2002; Erdogmus and Williams, 2003). A few studies found the project cost depends upon other factors such as market

pressure and the nature of the task (Müller and Padberg, 2002; Müller and Padberg, 2003; Padberg and Müller, 2003).

In this study, we follow the cost definition provided by Sommerville (2007). According to Sommerville, project cost is primarily the costs of the labors involved. These direct labor costs are the costs of paying software developers for the hours spent on a project.

Therefore, for each task, the direct labor cost is the number of hours developers spend on the task times their corresponding pay rate. For the overall project, the direct labor cost is the sum of the costs for all tasks involved. Since the direct labor cost is a function of hours and pay rate, assuming constant pay rates, the more hours developers spend on the project, the higher the direct labor cost will be.

One direct labor cost is the labor spent on programming and defect fixes. As developers spend more time on programming, the labor cost increases. As defects are uncovered, developers will incur rework time to fix the problems. Therefore, labor cost increases as developers spend more programming hours on the project, and labor cost decreases as there are fewer defects in the system.

The other category of cost is training cost. Trainings are mechanisms to educate the developers so they have the skills and information to successfully complete the project. Trainings are essential when developers do not have the technical skills to implement the project, do not have a good understanding of the system, and/ or do not have good problem solving skills. We do not expect knowledge transfer between the pairs to replace all trainings. However, we argue that knowledge transfer between the pairs helps developers acquire a better understanding of the modules and the overall system, improve technical and general problem solving skills, and, thus, reduces training cost.

Opportunity cost is defined as costs associated with opportunities an organization loses for undertaking a given project. An example of opportunity cost is the profit the organization could have made by working on a different project. With a short-term project, organizations can jump into other business opportunities rather quickly, thus minimizing opportunity cost. When a project goes on forever, an organization's resources are tied up in the project, and therefore cannot be made available to undertake other projects. As a result, opportunity cost increases.

Labor slack occurs when a project is finished ahead of schedule. When an organization has labor slack, more work, presumably, can be brought to the work force, so more projects can be started and finished. Alternatively, if labor slack is consistent, head count can be reduced thus creating a cost savings to the organization. This argument suggests labor slack contributes positively to the project cost by representing a deduction from total project cost as a result of shortened duration.

Other costs include costs of hardware, software and computer networks needed to support the project, costs of office space, utilities, central facilities, and employee benefits like pensions and health insurance (Sommerville, 2007). These overhead costs tend to be fixed. In this study, we consider the total cost of a software development project as a function of labor cost, training cost, opportunity cost, and labor slack. The total cost changes as any of the four costs changes. Because these costs can be either calculated or derived, we offer no specific hypotheses with regard to them.

4. METHODOLOGY

To facilitate the testing of our hypotheses and draw sufficient conclusions to answer the research questions at hand, we adopted a multi-method, multi-study empirical approach. The survey method was employed for Study 1 (S1) and the bootstrap simulation method for Study 2

(S2). For S1, we developed and administered a survey instrument to a targeted subject population of industrial software developers. This survey instrument had two objectives: (1) to gather data on practitioner perceptions with regard to the cost and effectiveness of the pair programming when compared to the more traditional solo approach and to provide sufficient data for testing our hypotheses and provide an initial validation of the research model (2) to acquire parameter data on a variety of project and developer characteristics to provide the necessary foundation for a simulation study.

Study 2 is a series of bootstrapping simulations. Using responses from the survey as input parameters, this study had two objectives: (1) to provide additional validation of the research hypotheses presented in the research model and (2) to determine in what situation pair programming is more cost effective than solo programming.

4.1. Survey

4.1.1. Survey Instrument

Because the level of complexity and focus contained herein had not been conducted in prior pair programming studies, our instrument for S1 required both generation and validation of the majority of survey items. With the exception of the knowledge transfer related questions adapted from Ko, Kirsch, and King (2005), the remaining questions were generated by the researchers. Table 4.1 provides a side-by-side match between the survey items intended to collect data for hypothesis testing and their associated hypothesis. Additionally, the complete survey instrument is attached in Appendix A.

Table 4.1 Instrument Items and Associated Hypothesis

H	Q#	Instrument Items
H1a	**Q1a**	Estimate effort in programming hours assuming a • Low, Medium, and High-complexity project.

H1b	**Q1b**	Estimate defect rate (# of defects / 1000 LOC) assuming a • Low, Medium, and High-complexity project.
H1c	**Q1c**	As system complexity increases • more communication will occur regarding the understanding of a module and how this module integrates with other modules • it improves a programmer's knowledge of the programming language in use. • it improves a programmer's general problem solving skills
H2a	**Q2a**	Estimate the percentage difference in effort between solo programming and pair programming, assuming a • Low, Medium, and High-complexity project.
H2b	**Q2b**	Estimate the percentage difference in defect rate between solo programming and pair programming assuming • Low, Medium, and High-complexity project.
H2c	**Q2c**	Estimate the percentage difference in knowledge transfer between solo programming and pair programming assuming a • Low, Medium, and High-complexity project.
Note: The following questions are repeated for low, medium, and high complexity projects.		
H3a	**Q3a_1**	Estimate the percentage difference in effort between a JUNIOR SOLO programmer and a pair if a pair is comprised of: • Two SENIOR programmers • A JUNIOR programmer and a SENIOR programmer • Two JUNIOR programmers
	Q3a_2	Estimate the percentage difference in effort between a SENIOR SOLO programmer and a pair if a pair is comprised of: • Two SENIOR programmers • A JUNIOR programmer and a SENIOR programmer • Two JUNIOR programmers
	Q3a_3	Estimate the percentage difference in defect rate between a JUNIOR SOLO programmer and a pair if a pair is comprised of: • Two SENIOR programmers • A JUNIOR programmer and a SENIOR programmer • Two JUNIOR programmers
	Q3a_4	Estimate the percentage difference in defect rate between a SENIOR SOLO programmer and a pair if a pair is comprised of: • Two SENIOR programmers • A JUNIOR programmer and a SENIOR programmer • Two JUNIOR programmers.
	Q3a_5	Estimate the percentage difference in knowledge transfer between a SOLO programmer and a pair if a pair is comprised of • Two SENIOR programmers • A JUNIOR programmer and a SENIOR programmer • Two JUNIOR programmers.
H3b	**Q3b_1**	Estimate the percentage difference in effort between solo programming and pair programming, assuming a pair is comprised of two • BOTH have prior pair programming experience • ONE of whom has prior pair programming experience • NEITHER of whom has prior pair programming experience
	Q3b_2	Estimate the percentage difference in defect rate between solo programming and pair programming, assuming a pair is comprised of two • BOTH have prior pair programming experience • ONE of whom has prior pair programming experience • NEITHER of whom has prior pair programming experience

	Q3b_3	Estimate the percentage difference in knowledge transfer between solo programming and pair programming, assuming a pair is comprised of two • BOTH programmers have prior pair programming experience. • ONE of the two programmers has prior pair programming experience. • NEITHER of the two programmers has prior pair programming experience.

4.1.2. Construct Validity and Reliability

To ensure the content validity of the survey instrument, an extensive review of prior literature was conducted and, whenever possible, pre-existing questionnaire items were considered and incorporated. Due to lack of literature in pair programming surveys, the adoption of existing questionnaires was found to be rather limited. However, commonly accepted measures by the computer science, software engineering, and information systems were available for review and appropriate items were adapted for use herein. Furthermore, following initial creation of the instrument, an informed pilot involving appropriate faculty and PhD students was conducted.

Most of the constructs in the instrument are best represented as single-item constructs, therefore, the traditional construct validity technique was not applicable. Knowledge transfer is a reflective construct with three measurement items. Confirmatory factor analysis utilizing varimax rotation (Johnson and Wichern 1992; Banker, Davis, and Slaughter 1998) was employed to verify the validity of this construct. The factor loadings from the three measurement items were 0.747, 0.868, and 0.862 respectively, which provides satisfactory evidence of convergent validity and internal consistency. Since knowledge transfer is the only reflective construct in this study, discriminant validity is not applicable. The Cronbach Alpha for the knowledge transfer construct is 0.764 which exceeded the 0.70 criterion (Nunnally 1978), suggesting the construct is reliable.

To improve survey reliability, only complete surveys were used for the data analysis. During data collection, the authors received numerous comments from respondents stating reasons why they didn't complete the survey. Not having sufficient knowledge of the measures was the primary reason. Therefore, it is reasonable to believe respondents who didn't know the measures or didn't feel comfortable reporting the measures abandoned the survey and, therefore, were not appropriate respondents. The respondent profile presented in Chapter 5 suggests the responses came from well-educated and highly experienced practitioners (91.4 percent have college or graduate degrees, 77.8 percent have over 10 years of industrial experience). Hence, it is fair to assume what was reported in the survey reflected the projects that went on in the industry.

4.2 Bootstrap Simulation

4.2.1 Bootstrap

The bootstrap, introduced by Efron (1979), is a statistical technique which allows a description of the variability of a statistic based on a unique finite sample. The bootstrap estimates standard errors by resampling with replacement of the original finite sample. The samples obtained are "pseudo sample" or "bootstrap sample" which are used to estimate the statistics of interest. The bootstrap samples are expected to behave similarly to the underlying distribution of the data and serve to control and check the stability of the results. Bootstrap is recommended in situations where the distribution of a statistic of interest is unknown, or when the sample size is insufficient for statistical inference (Adèr et al. 2008). It is a suitable technique for this research because the distributions of many of the parameters we gather from the survey and need to use for analysis (e.g. project hours, project defects, team size) are

unknown. For a more detailed description of the bootstrap technique see Efron (1979, 1982), Freedman and Peters (1984), and Efron and Tibshirani (1993), among others.

The number of bootstrap samples recommended in the literature has increased as available computing power has increased. To reduce the effects of random sampling errors from the bootstrap procedure, we have chosen to do 5,000 bootstrap samples. The original data set contained 191 records. We took 5,000 bootstrap samples of size 191 each with replacement from the original dataset and computed the means on pertinent data variables for each of these 5,000 bootstrap samples. The bootstrap estimates then became a sample of size 5,000 from which further analyses were conducted. The program that extracted the bootstrap samples and calculated the mean estimates is attached in Appendix B.

4.2.2. Cost Calculation

Since one primary goal is to investigate the cost constructs, columns were added to the original survey data to capture the cost information before the bootstrap process started. As specified in the research model in chapter 3, the cost constructs are *labor cost*, *training cost*, *opportunity cost*, *labor slack,* and *total cost*. Labor cost has two components: one is effort cost that resulted from initial system development and the other is defect cost which is cost associated with fixing defects. Since opportunity cost and labor slack are direct derivations of the project duration, we include duration in our calculation as well.

As stated in chapter 3, the measures of the cost constructs were calculated from their precedents. Therefore, one primary step in file preparation is to follow the formulas to calculate the costs. Table 4.2 below describes the formulas used for the cost calculation. Hours, defects, defect fixing speed, team size are supplied by the survey respondents. Hourly pay rates are from

the Bureau of Labor Statistics (2011). Team size was rounded to an even number so both solo and pair programming methods have the same number of developers on the team.

Table 4.2 Formulas for Cost Calculation

Construct	Formula
Effort cost	Hours spent * hourly pay rate
Defect cost	(Defects * defect fixing speed) * hourly pay rate
Duration	Total hours / (team size * working hours per day)
Opportunity cost/ Labor slack cost	(Difference between actual duration and ideal duration) * Team size * Pay rate. The ideal duration is the average duration calculated through 5000 bootstraps on the duration data.
Training cost	Original training cost – savings on training cost due to knowledge transfer.
Total cost	Effort cost + Defect cost + Training cost + Opportunity cost - Labor slack cost

A random generating function was used to return a random value given a range as we calculate the cost on several variables: hourly pay rates, survey choices, and knowledge transfer coefficients. For example, the hourly pay rate for a senior software developer was randomly assigned by the following function *(rand (lowerLimit, upperLimit))*. The lowerLimit and upperLimit are the range of hourly pay rates for a senior software developer obtained from Labor Statistics. Following the same logic, the questions that ask the percentage difference between programming methods are randomly assigned values between 1 and 20 given 1 as the survey choice, 21 and 40 given 2 as the choice, etc. The Rand() function returns a random value from the specified range. The random function is well suited since it introduces randomness while keeping the value within the range of interest.

The program that performs all the file preparation steps is attached in Appendix C.

5. RESULTS

This chapter presents the sample characteristics, results from the hypotheses testing, and results on the cost constructs.

5.1 Survey Sample

The sample for the survey consisted of software practitioners who have professional programming experience in the industry. Over 2,500 email addresses were collected through conference attendance, association memberships, paper authorships, and referrals. Each participant was contacted by the authors through a personalized email requesting participation in the survey. One reminder, 10 days after the initial message, was sent to the non-respondents. Data collection lasted for approximately one month. 191 surveys were returned complete, yielding a net response rate of approximately seven percent.

To the best of our knowledge, this is the first industry-wide survey focused on pair programming. Due to the lack of baseline data, it is hard to judge whether the response rate was high, low, or comparable. The fact that past literature only published a handful of survey studies which either used students as respondents or collected responses from specific organizations seems to imply that conducting an industry wide survey on this topic is a challenging task. We, therefore, consider the seven percent response rate reasonable.

We tested for the possibility of response bias by comparing the responses from the first 20 percent of the responses received to those from the last 20 percent received. Statistical tests revealed that all relevant measurement items shared the same results ($p < 0.05$). We thus conclude there is little evidence of a difference in responses between early and late responders. This suggests that nonresponse bias is unlikely to be an issue in this study (Chatterjee, Grewal, and Sambamurthy 2002).

The profile of the respondents and their associated organizations is shown in Table 5.0. The profile indicates the respondents worked in various industries with projects of different sizes. The profile also suggests the responses came from well-educated and experienced

practitioners. 91.5 percent of the respondents have baccalaureate or advanced degrees. 75.9 percent have over 10 years of working experience in the information technology field. 128 respondents have pair programming experience, and 63 respondents indicated no pair programming experience. Only 10.0% of the responses were female. This is most likely a reflection of the gender imbalance in the software development industry.

Table 5.0 Respondents Profile

	N	%		N	%
Pair programming experience			**Experience in IS/IT field**		
Never	63	32.9%	< 2 years	4	2.1%
<1 year	27	14.1%	2-5 years	8	4.1%
1-2 years	34	17.8%	6-10 years	34	17.8%
3-4 years	28	14.6%	11-20 years	88	46.1%
5-6 years	22	11.5%	20+ years	57	29.8%
7-8 years	6	3.1%			
9-10 years	4	2.1%			
>10 years	7	3.6%			
Current Position			**Highest education**		
Developer	22	11.5%	High school or equivalent	1	0.5%
Project manager	41	21.4%	Some college	15	7.8%
Team Leader	19	9.9%	College degree	74	38.7%
VP/Director of Development	25	13.0%	Graduate degree	101	52.8%
Development Manager	18	9.4%			
Consultant/Trainer	19	9.7%			
Architect	13	6.8%	**Location**		
QA/Tester	16	8.3%	North America	160	83.7%
Product Manager	3	1.6%	Europe	13	6.8%
IT Staff	4	2.0%	Asia	15	7.8%
CIO/CTO/CEO/President	1	0.5%	Australia/NZ	3	1.6%
Other	9	4.7%			
Age			**Gender**		
<30	19	9.9%	Male	172	90.0%
30-39	70	36.6%	Female	19	10.0%
40-49	64	33.5%			
50-59	30	15.7%			
60-69	7	3.6%			
>69	1	0.5%			
Industry			**Total Software Organization**		
e-Commerce	13	6.8%	(# of employees in software		
Financial	19	10.0%	development and delivery)		
Government	13	6.8%	<5	13	6.8%
IT Consulting	33	17.2%	5-20	34	17.8%
Manufacturing	8	4.1%	21-50	56	29.3%
Retail	2	1.0%	51-100	26	13.6%
Technology	69	36.1%	101-250	22	11.5%
Other	34	17.8%	>250	40	20.9%
Average Project Budget					
<$10K	9	4.7%			
$10K-$49K	22	11.5%			
$50K-99K	51	26.7%			
100K-499K	59	30.8%			
500K-999K	22	11.5%			
1,000K-$2,000K	18	9.4%			
>2,000K	10	5.2%			

5.2. Hypothesis Testing

Hypothesis testing was conducted on both the original survey data and the simulation data. In this section, we present the hypotheses, results, and a brief explanation of the results.

H1a. Regardless of programming approach, as system complexity increases, the programming effort increases.
H1b. Regardless of programming approach, as system complexity increases, defect rate increases.
H1c. Regardless of programming approach, as system complexity increases, there is higher knowledge transfer rate amongst the project team members.

To test for support of H1a and H1b based on the original survey responses, non-parametric related sampled tests were applied. This approach is the most appropriate for several reasons. First, respondents have experience on projects of different sizes in different organizations. As they estimate the number of modules and the defect rate for projects at three levels of complexity (low, medium, high), there is inevitably a large dispersion of entries where distribution cannot be assumed and outliers are common. Rank-based nonparametric tests do not make distribution assumptions and are not affected by outliers (Hollander and Wolfe 1999). Second, the survey was designed such that the same subject is asked to provide responses for low, medium and high complexity projects, thus, the data points are not independent of each other and related sample tests are suitable.

To test for support of H1a and H1b based on the simulation data, multivariate tests on repeated measures were employed. Data examination on the simulation data suggests through the bootstrapping process data on effort and defect approximate normal (skewness -0.12 to 0.92, Kurtosis -0.03 to 1.11), therefore, there is no need to conduct non-parametric testing.

To test for support of H1c, a one sample t-test was adopted on both the survey and simulation data. T-test is appropriate in this case because we intend to test the difference between the sample mean and the population mean. All the measurement items, as well as the

grand mean which is calculated from the three measurement items, meet the assumptions for t-test: independence and normal distribution (Warner 2007).

Table 5.1 is a summary of the results for H1a, H1b, and H1c. Each variable's basic statistics are reported as well. For variables where no distribution can be assumed, their mean ranks are reported. The three items to measure knowledge transfer approximate normal distributions, therefore, their means and standard deviations are reported.

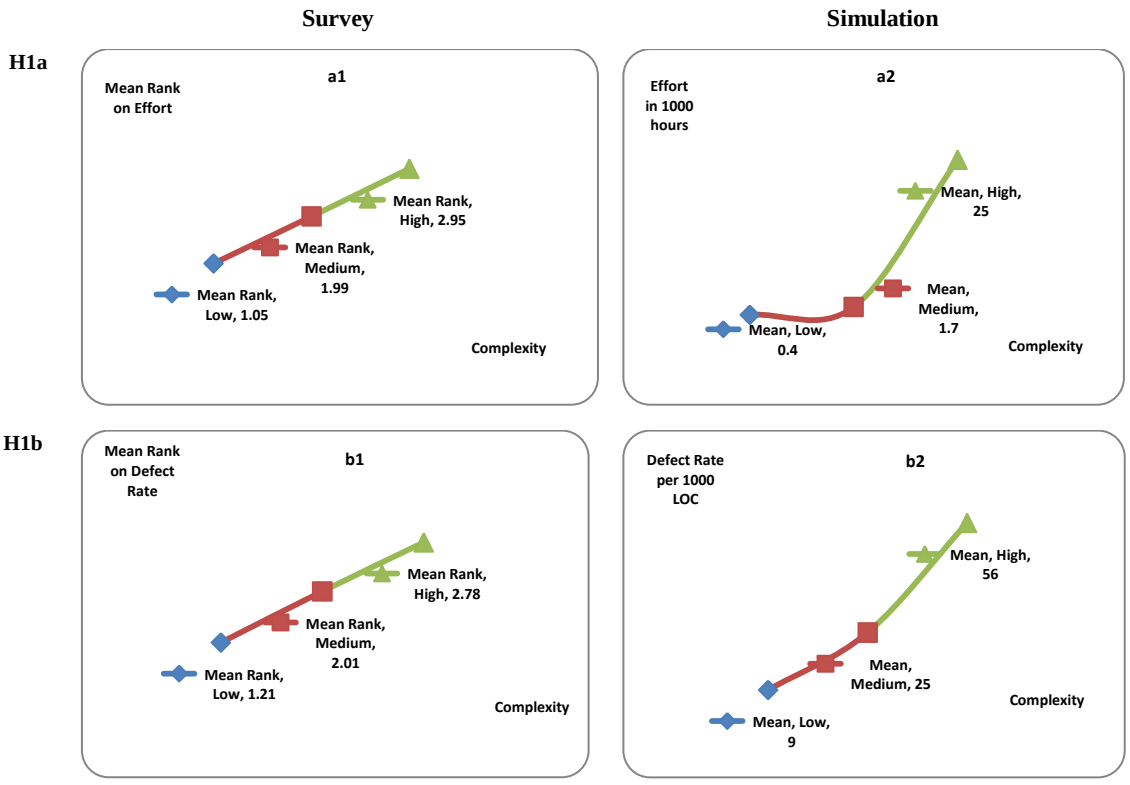
Table 5.1 Results for H1a, H1b, and H1c

	From Survey (N=191)					From Simulation (N=5000)				Overall
H	Item	Mean Rank	SD	p-value =<	H Support	Mean	SD	P-value =<	H Support	H Support
H1a	Q1a Effort in		NA	0.001	Yes			0.001	Yes	Yes
	LCP	1.05				459	70			
	MCP	1.99				1699	196			
	HCP	2.95				25011	6303			
H1b	Q1b Defect rate in		NA	0.001	Yes			0.001	Yes	Yes
	LCP	1.21				9	1.38			
	MCP	2.01				25	4.07			
	HCP	2.78				55	7.63			
H1c	Q1c	Mean		0.001	Yes			0.001	Yes	Yes
	Module understanding	5.24	1.73			5.25	0.12			
	Programming language	4.46	1.61			4.43	0.11			
	Problem solving	4.51	1.59			4.51	0.11			
	Grand mean	4.74	1.35			4.74	0.10			
LCP=Low complexity project MCP=Medium complexity project HCP=High complexity project										

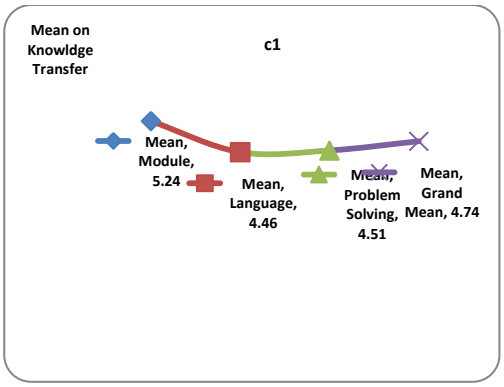
These results are further illustrated in Figure 5.1. Figure 5.1-a1 shows the mean ranks of effort at three system complexity levels: low, medium, and high. The upward trend suggests effort increases as system complexity increases. Wilcoxon Signed Ranks Tests on related samples suggest the ranks between different complexity levels (medium vs. low, high vs. medium) are statistically significant (p-value < 0.001), thus, supporting H1a. Figure 5.1-b1 depicts the mean ranks of defect rate at three complexity levels and suggests as system complexity increases defect rate increases. Wilcoxon Signed Ranks Tests on related samples

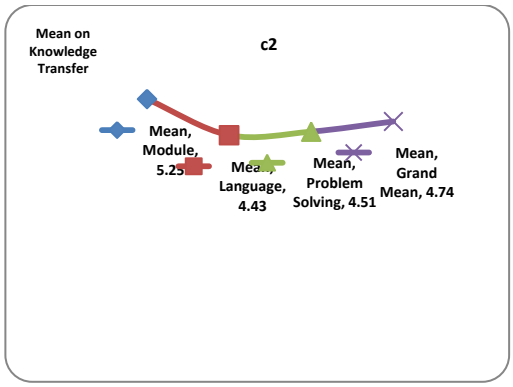
suggest the differences in ranks between complexity levels (medium vs. low, high vs. medium) are statistically significant (p-value < 0.001), supporting H1b. Figure 5.1-c1 plots the mean from each of the three measurement items for knowledge transfer as well as the grand mean from the three items. The sample mean was compared to the population mean (4-Neutral). Results indicate each of the sample means is significantly different from the population mean (p-value < 0.001), suggesting the respondents generally agree that as the system complexity increases, there is a higher knowledge transfer. Therefore, H1c is supported. As demonstrated in Figure 5.1-a2, 5.1-b2, and 5.1-c2 which present the means on effort, defect rate, and knowledge transfer, and statistics in Table 5.1 (p-value < 0.001), results based on the simulation data suggest the same conclusions.

Figure 5.1 Results for H1a, H1b, and H1c



H1c





H2a. When compared to solo programming, as system complexity increases, pair programming will moderate the effect of system complexity on programming effort thus reducing effort.
H2b. When compared to solo programming, as system complexity increases, pair programming will reduce the effect of complexity on defect rate thus reducing the overall defect rate.
H2c. When compared to solo programming, as system complexity increases, pair programming will increase the effect of system complexity on knowledge transfer thus enhancing knowledge transfer.

To test for support of H2a, H2b, and H2c, multivariate tests on repeated measures were employed. These tests are appropriate for several reasons. First, each survey respondent was asked to provide answers at multiple specified conditions. Second, data screening suggests the variables meet the assumption of multivariate repeated measures: the variables are quantitative and approximately normally distributed, scores on the repeated measure variables have a multivariate normal distribution, and relationships among repeated measures are linear (Warner 2007).

As discussed in Chapter 4, a significant concern was that perceptions on the effectiveness of pair programming could differ between practitioners who have and do not have pair programming experience. Thus, each respondent was coded based on whether he/she had pair programming experience and the coded variable was used as the between-subject factor in the multivariate test to examine any possible differences between the two subject groups. Table 5.2 is a summary of the hypothesis testing results along with each variable's basic statistics.

Table 5.2 Results for H2a, H2b, and H2c

Survey									Simulation							Overall
H	**Item**	**With PPE (N=128)**		**Without PPE (N=63)**		**WSE p-value =<**	**H Supp**	**BSE p - value =<**	**With PPE (N=5000)**		**Without PPE (N=5000)**		**WSE**	**H Supp**	**BSE p-value =<**	
	% diff. b/t SP and PP in … *	**μ**	**SE**	**μ**	**SE**				**μ**	**SE**	**μ**	**SE**				
H2a	Q2a Effort					0.001	Yes	0.012					0.001	Yes	0.001	Yes
	LCP	0.85	0.22	1.4	0.32				0.92	0.01	1.39	0.006				
	MCP	-0.28	0.18	0.59	0.25				-0.13	0.01	0.62	0.006				
	HCP	-1.09	0.2	-0.19	0.28				-1.01	0.004	-0.14	0.004				
H2b	Q2b Defect rate					0.001	Yes	0.001					0.001	Yes	0.001	Yes
	LCP	-1.61	0.14	-0.66	0.2				-1.72	0.01	-0.86	0.006				
	MCP	-2.22	0.13	-1.19	0.19				-2	0	-1.04	0.002				
	HCP	-2.61	0.16	-1.77	0.23				-2.62	0.01	-1.86	0.006				
H2c	Q2 Knowledge transfer					0.001	Yes	0.088					0.001	Yes	0.001	Yes
	LCP	1.35	0.2	0.5	0.29				1.13	0.01	0.48	0.006				
	MCP	1.81	0.21	1.21	0.3				1.79	0.01	1.1	0.005				
	HCP	2.23	0.25	1.77	0.35				2.04	0.01	1.84	0.005				

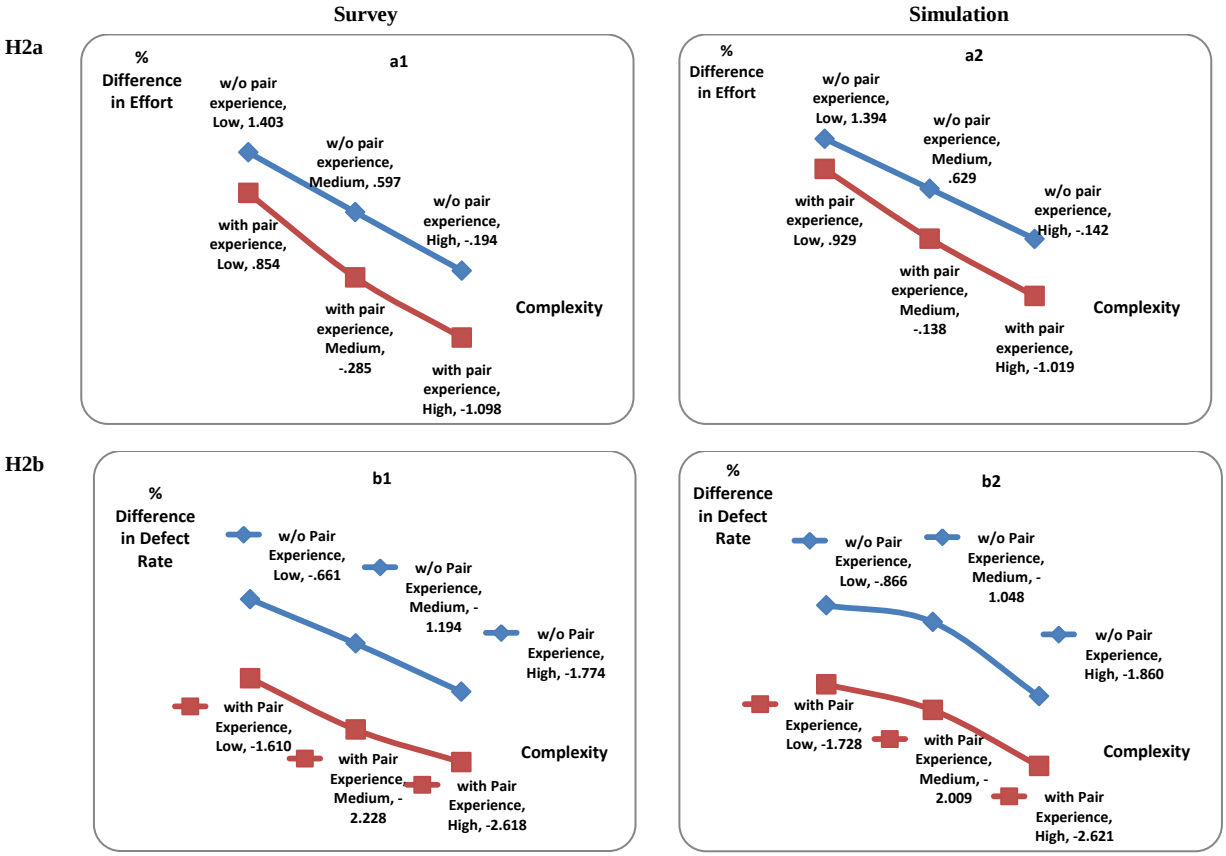
WSE=within subject effect BSE=between subject effect
PPE=pair programming experience
SP=solo programming PP=pair programming
LCP=low complexity project MCP=medium complexity project HCP=high complexity project

* Measurement scales:
0=No difference
Pair increases: 1=1-20% 2-21-40% 3+41-60% 4=61-80% 5=81-100% 6=>100%
Pair decreases: -1=1-20% -2=21-40% -3=41-60% -4=61-80% -5=81-99%

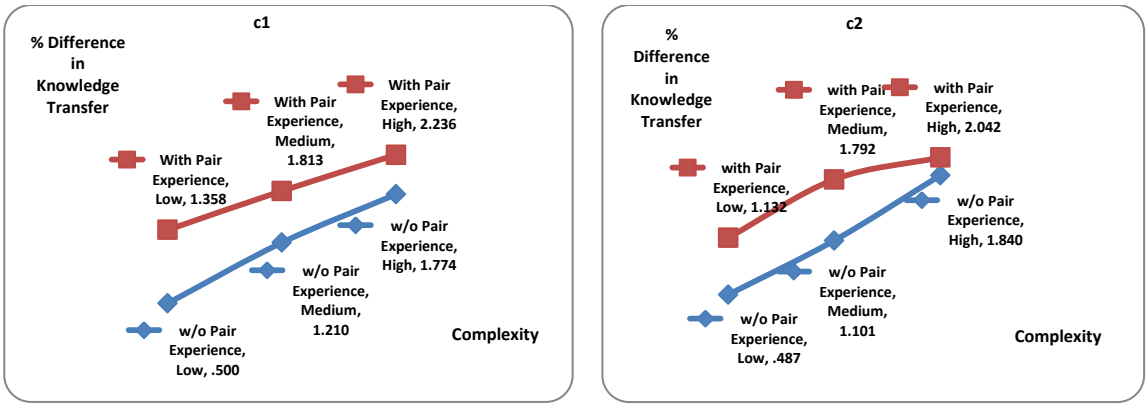
The results are further illustrated in Figure 5.2. Figure 5.2-a1 and a2 show a downward trend, which suggests both respondent groups agree as system complexity increases, compared to the solo programming method, pair programming will result in a decrease in programming effort (p-value < 0.001), supporting H2a. However, despite sharing the same trend, the two groups disagree significantly on the magnitude of change on effort at all complexity levels (p-value = 0.012 from the original responses, p-value < 0.001 from simulation). Specifically, in all three complexity levels, practitioners with pair programming experience viewed pair programming more positively than practitioners without pair programming experience. Figure 5.2-b1 and b2 demonstrate a downward trend regarding defect rate, suggesting as system complexity increases,

pair programming method will decrease the defect rate (p-value < 0.001), supporting H2b. Once again the two respondent groups disagree on the magnitude of decrease (p-value < 0.001). Figure 5.2-c1 and c2 show an upward trend, implying as the system complexity increases, compared to solo programming, pair programming method will result in higher knowledge transfer (p-value < 0.001), supporting H2c. The two respondent groups differ on the magnitude of increase on knowledge transfer. While this difference based on simulation is statistically significant (p-value < 0.001), the difference based on the original survey responses is not (p-value = 0.088).

Figure 5.2 Results for H2a, H2b, and H2c



H2c



Mean responses from practitioners without pair programming experience
Mean responses from practitioners with pair programming experience

Y-Axis: 0=No difference
Pair increases: 1=1-20% 2=21-40% 3=41-60% 4=61-80% 5=81-100% >100%
Pair decreases: -1=1-20% -2=21-40% -3=41-60% -4=61-80% -5=81-99%

H3a: Programmer expertise moderates the effectiveness of the pair programming method.
H3b: Prior pair programming experience moderates the effectiveness of the pair programming method.

To test for support of H3a and H3b, multivariate repeated measures were employed for the same reasons as stated previously. Table 5.3 provides a summary of the hypothesis testing results and each variable's basic statistics.

Table 5.3 Results for H3a and H3b

		Survey							Simulation							Overall
H	Item	With PPE (N=128)		Without PPE (N=63)		WSE p-value =<	H Supp	BSE p-value =<	With PPE (N=5000)		Without PPE (N=5000)		WSE p-value =<	H Supp	BSE p-value =<	
	% diff. b/t SP and PP in…	**μ**	**SE**	**μ**	**SE**				**μ**	**SE**	**μ**	**SE**				
H3a	Q3a Effort															
	Jr-Jr	1.02	0.22	1.8	0.3	0.001	Yes	0.006	1.00	0.004	1.86	0.00	0.001	Yes	0.001	Yes
	Jr-Sr	-0.43	0.21	0.64	0.3				-0.43	0.007	0.70	0.01				
	Sr-Sr	-0.92	0.21	-0.06	0.3				-0.98	0.003	-0.06	0.00				
	Defect rate															
	Jr-Jr	-1.1	0.14	-0.38	0.2	0.001	Yes	0.001	-1.00	0.004	-0.19	0.004	0.001	Yes	0.001	Yes
	Jr-Sr	-2.19	0.13	-1.41	0.19				-2.01	0.005	-1.26	0.005				
	Sr-Sr	-2.79	0.15	-1.83	0.21				-2.96	0.003	-1.94	0.003				
	Knowledge transfer															
	Jr-Jr	1.32	0.19	0.83	0.27	0.001	Yes	0.28	1.08	0.004	0.94	0.004	0.001	Yes	0.001	Yes
	Jr-Sr	2.24	0.24	1.85	0.34				2.06	0.004	1.9	0.004				
	Sr-Sr	1.89	0.22	1.61	0.31				1.87	0.006	1.65	0.006				

	Q3b Effort															
	Prior PP0	1.34	0.21	2.03	0.3				1.25	0.006	2.03	0.006				
	Prior PP1	0.016	0.19	0.82	0.27	0.001	Yes	0.006	-0.01	0.004	0.89	0.004	0.001	Yes	0.001	Yes
	Prior PP2	-1.24	0.21	-0.8	0.29				-1.17	0.005	-0.06	0.005				
	Defect rate															
	Prior PP0	-0.92	0.15	-0.4	0.21				-0.99	0.004	-0.24	0.004				
H3b	Prior PP1	-1.9	0.14	-1.12	0.2	0.001	Yes	0.003	-1.99	0.002	-1.01	0.002	0.001	Yes	0.001	Yes
	Prior PP2	-2.65	0.15	-1.82	0.22				-2.82	0.005	-1.86	0.005				
	Knowledge transfer															
	Prior PP0	1.24	0.25	0.79	0.25				1.05	0.003	0.93	0.003				
	Prior PP1	1.98	0.21	1.37	0.29	0.001	Yes	0.162	1.93	0.005	1.28	0.005	0.001	Yes	0.001	Yes
	Prior PP2	2.08	0.24	1.69	0.33				1.95	0.005	1.79	0.005				

Jr-Jr=junior-junior pair Jr-Sr=junior-senior pair Sr-Sr=senior-senior pair
Prior PP0 =pair neither has prior pair programming experience
Prior PP1 = pair one has prior pair programming experience
Prior PP2 = pair both have prior pair programming experience

Figure 5.3 depicts the impact of one aspect of pair composition - the expertise of the developers, on the effectiveness of pair programming method. Figure 5.3-a1 and a2 show the junior-junior composition will result in an increase in effort when applying the pair programming method compared to the solo programming method, the junior-senior composition will result in an increase in effort according to practitioners who do not have pair programming experience, but will result in a decrease in effort according to practitioners who have pair programming experience, and the senior-senior composition will lead to a decrease in effort from the perspectives of both respondent groups. Figure 5.3-b1 and b2 suggest a junior-junior composition will result in the least decrease in defect rate while a senior-senior composition will result in the largest decrease. Figure 5.3-c1 and c2 demonstrates the junior-senior composition leads to the highest amount of knowledge transfer, while the junior-junior composition generates the least amount of knowledge transfer.

The results further indicate the effectiveness of pair programming varies when pairs of different compositions - junior-junior, senior-senior, and junior-senior are compared to solos (p-value < 0.001). Therefore, H3a is supported. Again, even though the respondent groups agree

on the general trend (p-value < 0.001), they disagree on the magnitude of the changes. Results from simulation suggest all disagreements are statistically significant (p-value < 0.001). Results based on the original survey data indicate the disagreements on effort and defect rate are statistically significant (p-value = 0.006 for effort, < 0.001 for defect rate) but the disagreement on knowledge transfer is not (p-value =0.28).

Figure 5.3 Results for H3a

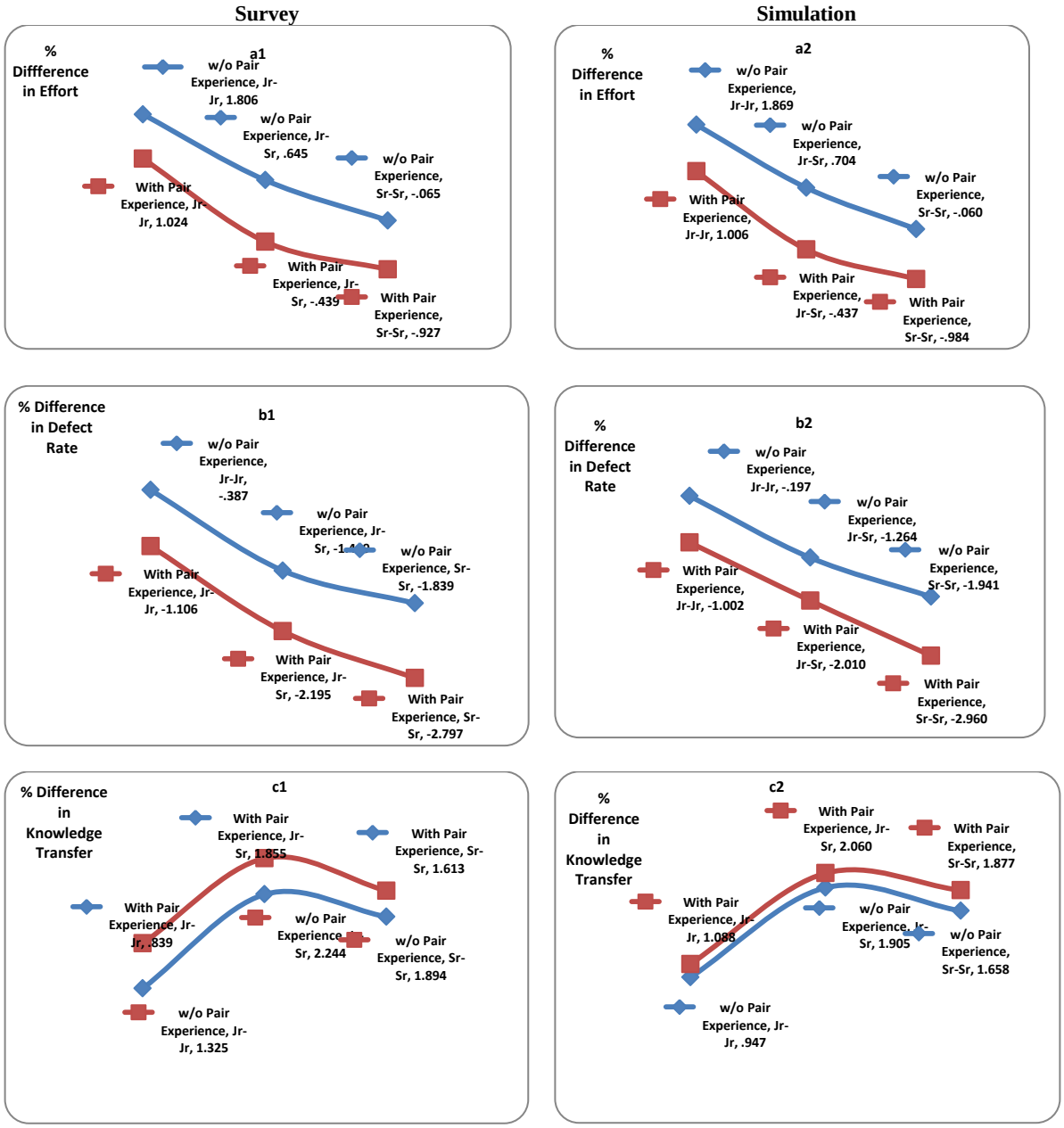


Figure 5.4 depicts the impact of another aspect of pair composition - prior pair programming experience, on the effectiveness of pair programming. Figure 5.4-a1 and a2 suggest a pair without prior pair programming experience will lead to the highest amount of effort increase, a pair with one having prior pair programming experience will lead to a slight effort increase, while a pair with prior pair programming experience will result in a decrease in effort.

Figure 5.4-b1 and b2 reveal a pair with prior pair programming experience will lead to the highest decrease in defect rate, a pair with one having prior pair programming experience will result in the second highest decrease, and a pair without prior pair programming experience will have the least decrease in defect rate.

Figure 5.4-c1 and c2 suggests a pair with prior pair programming experience will have the highest amount of knowledge transfer, and a pair without pair programming experience will generate the least amount of knowledge transfer. Results indicate the effectiveness of pair programming varies when pairs of different compositions based on whether they have prior pair programming experience – neither has, one has, and both have are compared to solos (p-value < 0.001). Therefore, H3b is supported. Results based on the original survey data suggest the difference between the two respondents groups is significant in effort (p-value = 0.006) and defect rate (p-value=0.003), but the group difference is not significant on knowledge transfer (p-value=0.162). Results from simulation indicate all differences are significant (p-value < 0.001).

Figure 5.4 Results for H3b

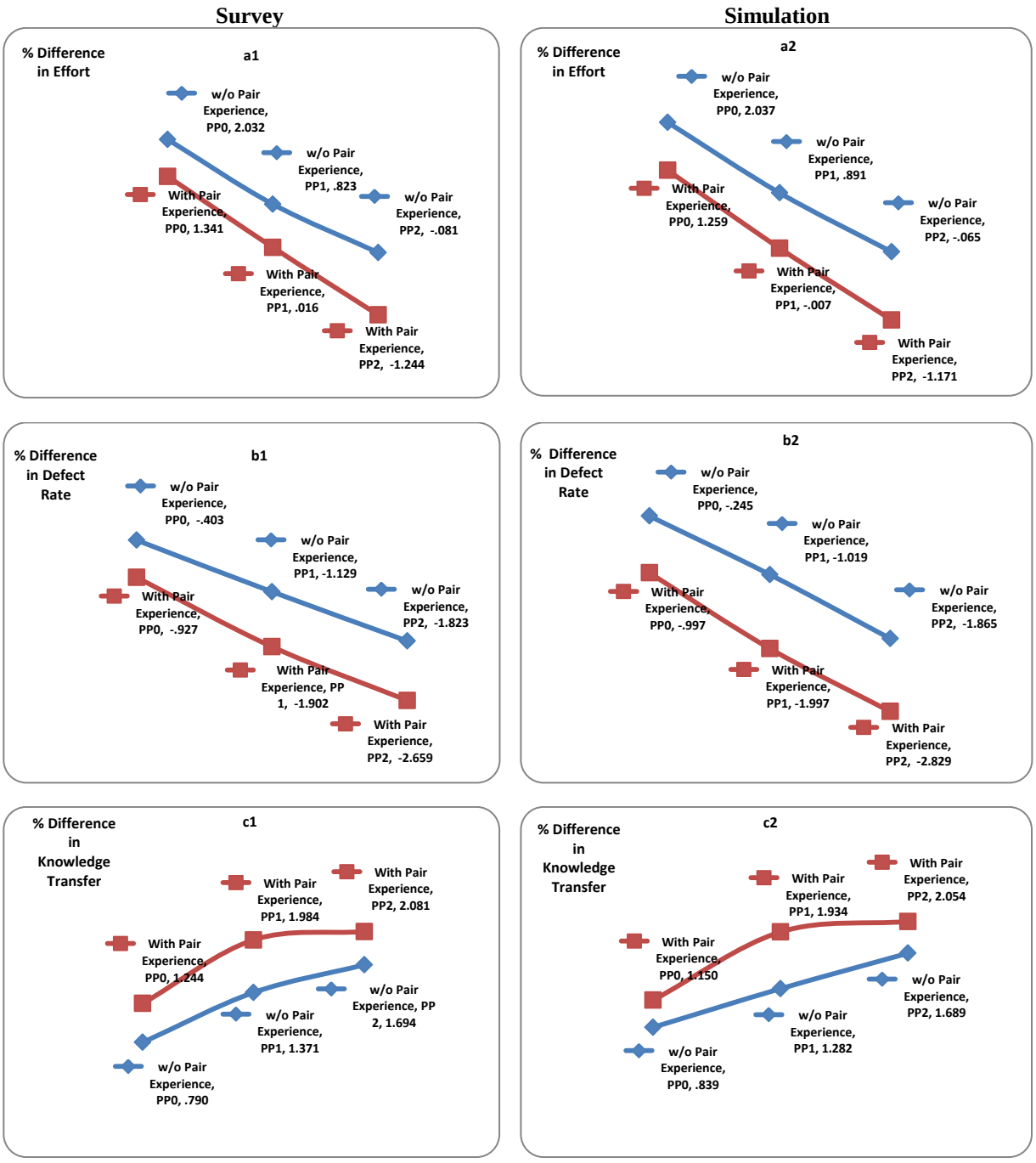


Table 5.4 is a summary of all the statistical tests and their outcomes. As one can see from the results, all hypotheses stated in the Research Model and Hypotheses section are supported by both the original survey responses and simulation data.

Table 5.4 Summary of the Hypotheses Testing Results

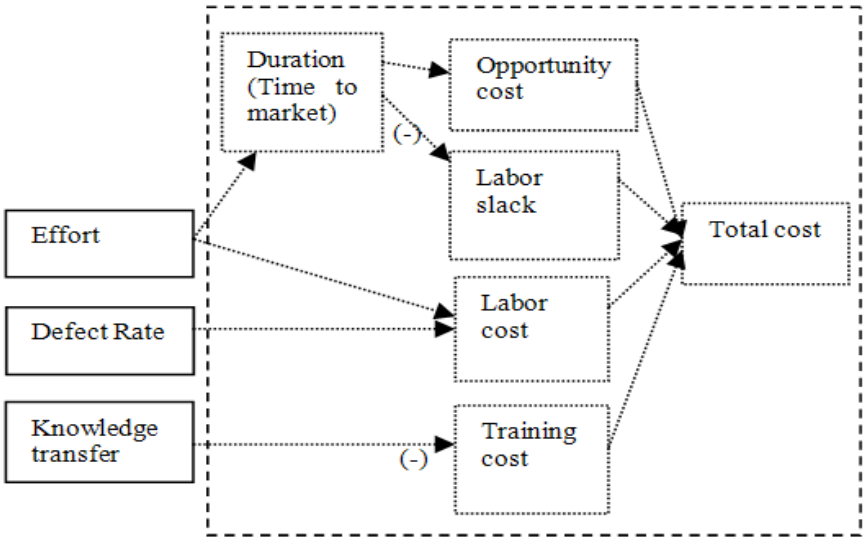
		From Survey				From Simulation				Overall
H	Path	Test	Within Subject p-value =<	Between Subject Effect p-value =<	H Supp	Test	Within Subject p-value =<	Between Subject Effect p-value =<	H Supp	
H1a	Complexity → Effort	NP	0.001	NA	Yes	GLM-RM	0.001	NA	Yes	Yes
H1b	Complexity → Defect Rate	NP	0.001	NA	Yes	GLM-RM	0.001	NA	Yes	Yes
H1c	Complexity → Knowledge Transfer	t-test	0.001	NA	Yes	t-test	0.001	NA	Yes	Yes
H2a	Pair Programming moderates Complexity and Effort	GLM-RM	0.001	0.012	Yes	GLM-RM	0.001	0.001	Yes	Yes
H2b	Pair Programming moderates Complexity and Defect Rate	GLM-RM	0.001	0.001	Yes	GLM-RM	0.001	0.001	Yes	Yes
H2c	Pair Programming moderates Complexity and Knowledge Transfer	GLM-RM	0.001	0.088	Yes	GLM-RM	0.001	0.001	Yes	Yes
	Programmer Expertise affects the effectiveness of pair programming in									
H3a1	Effort	GLM-RM	0.001	0.006	Yes	GLM-RM	0.001	0.001	Yes	Yes
H3a2	Defect Rate	GLM-RM	0.001	0.001	Yes	GLM-RM	0.001	0.001	Yes	Yes
H3a3	Knowledge Transfer	GLM-RM	0.001	0.28	Yes	GLM-RM	0.001	0.001	Yes	Yes
	Prior Pair Programming Experience affects the effectiveness of pair programming in									
H3b1	Effort	GLM-RM	0.001	0.006	Yes	GLM-RM	0.001	0.001	Yes	Yes
H3b2	Defect Rate	GLM-RM	0.001	0.003	Yes	GLM-RM	0.001	0.001	Yes	Yes
H3b3	Knowledge Transfer	GLM-RM	0.001	0.162	Yes	GLM-RM	0.001	0.001	Yes	Yes

NP = nonparametric related sample tests – Wilcoxon Signed Ranks Test
GLM_RM=general linear model on repeated measures

5.3 Cost

Figure 5.5 is the cost aspect of the research model presented in Chapter 3. As illustrated in the figure, in this project, the overall project cost consists of *opportunity cost, labor slack, labor cost,* and *training cost*. Opportunity cost and labor slacks are derived from duration, labor cost consists of effort cost and defect cost, and training cost is calculated as a result of knowledge transfer. In this section, we discuss results on these cost constructs from the bootstrap simulation.

Figure 5.5 Research Model – Cost Constructs



As described in Chapter 4, in the survey we asked responses regarding eight groups: *generic solo, generic pair, junior-junior pair, junior-senior pair, senior-senior pair, pair without prior programming experience, pair with one having prior pair programming experience,* and *pair with prior pair programming*. Generic solo refers to solos with no predefined skill levels. Generic pair refers to pairs without consideration of their compositions. As such, our analysis and discussions in this study are limited to these eight groups. We recognize the importance of other plausible groups, e.g. junior solo, senior solo, and other variations of pair compositions.

However, they are beyond the scope of this research and are left for future research as discussed in the Future Research section in the next chapter.

To enable group comparisons, z-scores were computed ($(x - \mu) / \sigma$). It is important to note, despite dollar amounts presented along with the z-scores, the emphasis of the study is not on specific dollar amounts, but rather on the relative contributions to cost associated with the different programming methods represented by the eight groups described above. Bonferroni pair-wise comparisons were conducted on all groups at each project complexity level on all constructs to ensure against alpha build-up. Results suggest most of the group differences are statistically significant (p-value < 0.05). The ones that are not statistically significant are marked with an ampersand (&), and the ones that are not significantly different from the means are marked with the pound sign (#).

5.3.1 Labor Cost

Labor cost has two components: *effort cost* which is the labor cost incurred during initial development, and *defect cost* which are labor costs associated with fixing defects. We first present results on effort cost, then defect cost, and finally overall labor cost.

Effort Cost

Table 5.5 presents the standardized scores on effort cost as well as the dollar amounts from the means. Dollar amounts below the mean (meaning the cost is lower than an average project) are in parentheses. Means and standard deviations for low, medium, and high complexity projects are presented at the end of the table. Those represent the μ and σ used in the z-score calculation.

Table 5.5 Effort Cost

	Z-Score			$ Amount from the Mean		
	Low	**Medium**	**High**	**Low**	**Medium**	**High**
Solo	-.208	-.005#	-.098&	(2,809)	(385)	(111,237)
Pair	.032&	-.125	-.102&	437	(8,864)	(116,158)
Jr-Jr	-.154	-.073	-.404	(2,083)	(5,191)	(458,503)
Jr-Sr	.029&#	-.177	-.194	396	(12,627)	(219,519)
Sr-Sr	.091	.116	.097	1,230	8,252	109,656
Prior PP0	.214	.430	.653	2,887	30,624	740,291
Prior PP1	.126	.169	.202	1,699	12,053	228,676
Prior PP2	-.130	-.335	-.153	(1,756)	(23,862)	(173,205)
Mean	30,742	183,340	2,688,511			
SD	13,330	69,702	1,138,701			

Jr-Jr=junior-junior pair Jr-Sr=junior-senior pair Sr-Sr=senior-senior pair
Prior PP0=pair neither having prior pair programming experience
Prior PP1=pair one having prior pair programming experience
Prior PP2=pair both having prior pair programming experience

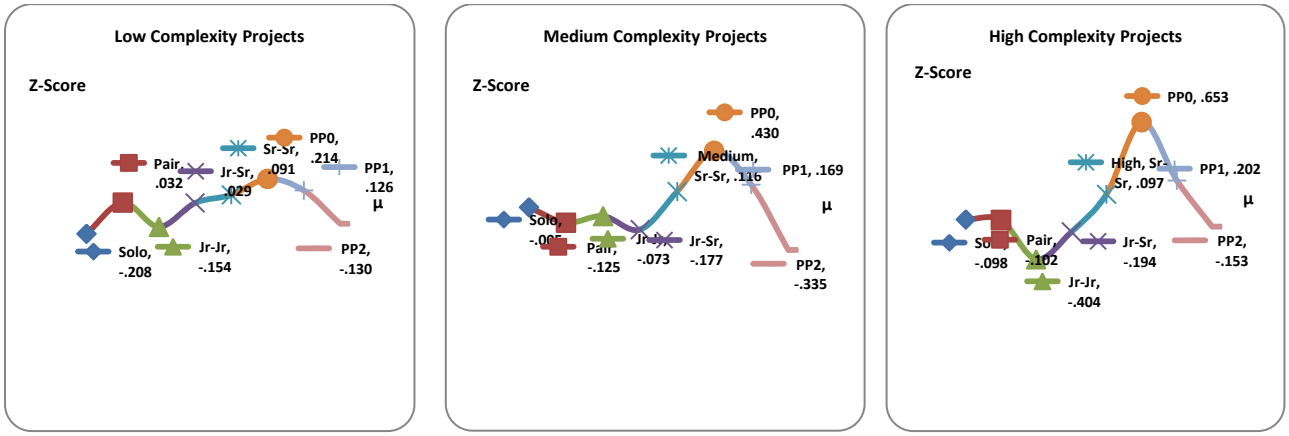
Results suggest project complexity matters. As complexity increases, effort cost increases (p-value < 0001). In low complexity projects, solo is the most cost effective. However, in medium and high complexity projects, junior-junior, junior-senior, and pair both with prior pair programming experience are more cost effective than solo.

Results further suggest pair composition matters. For example, considering expertise in pair composition, junior-senior is the most cost effective in medium complexity projects, junior-junior is the most cost effective in low and high complexity projects, and senior-senior is the most expensive in all cases. Considering prior pair programming experience in pair composition, pairs with prior pair programming experience are consistently the most cost effective while pairs without prior pair programming experience are consistently the most expensive across all project complexity levels.

This finding is further illustrated in Figure 5.5. Figure 5.5 indicates, considering all the groups, solo incurs the least effort cost in low complexity projects, pairs with prior pair programming experience are the least expensive in medium complexity projects, and the junior-

junior pair is the most cost effective in high complexity projects. In addition, the generic pair appears to behave just like a generic pair. Its performance presents neither the best pair nor the worst pair. It is between the best and the worst. For example, generic pair does not perform as well as pairs with prior pair programming experience, but it consistently outperforms pairs without prior pair programming experience. The same trend is identified as the generic pair is compared to pairs comprised of different levels of expertise.

Figure 5.5 Effort Cost



Defect Cost

Table 5.6 presents the standardized scores on defect cost as well as the dollar amounts from the means.

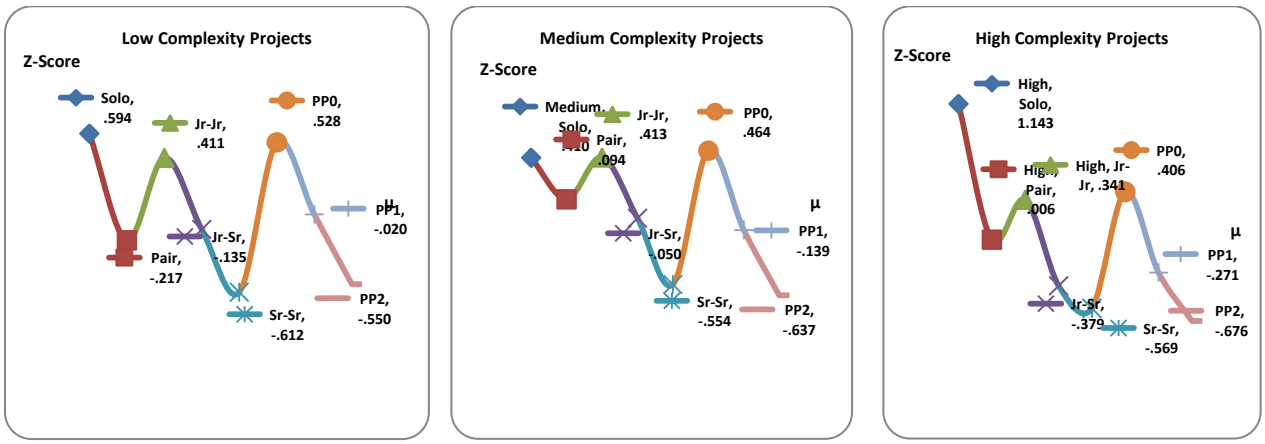
Table 5.6 Defect Cost

	Z-Score			$ Amount from the Mean		
	Low	Medium	High	Low	Medium	High
Solo	.594	.410&	1.143	15,924	153,391	2,448,516
Pair	-.217	.094	.006#	(5,809)	35,251	12,284
Jr-Jr	.411	.413&	.341	11,007	154,406	730,037
Jr-Sr	-.135	-.050	-.379	(3,607)	(18,693)	(812,628)
Sr-Sr	-.612	-.554	-.569	(16,389)	(207,170)	(1,219,902)
Prior PP0	.528	.464	.406	14,139	173,512	870,407
Prior PP1	-.020#	-.139	-.271	(524)	(52,170)	(580,955)
Prior PP2	-.550	-.637	-.676	(14,740)	(238,526)	(1,447,756)
Mean	56,302	628,166	4,905,760			
SD	26,791	374,236	2,142,952			

Results suggest in low complexity projects, solo, junior-junior pairs, and pairs without prior pair programming experience incur a relatively high defect cost. In medium complexity projects, junior-junior pairs and pairs without prior pair programming experience incur higher defect costs than solo but all the other pairs incur less defect costs than solo. In high complexity projects, all pairs incur less defect costs than solo. Considering expertise in pair composition, senior-senior is the most cost effective and junior-junior is the most expensive. Considering prior pair programming experience in pair composition, pairs with prior pair programming experience are the most cost effective while pairs without prior pair programming experience are the most expensive.

This finding is further illustrated in Figure 5.6. Figure 5.6 demonstrates considering all the groups, senior-senior incurs the least defect cost in low complexity projects, and pairs with prior pair programming experience incur the least cost in medium and high complexity projects. The defect cost is high for solo, junior-junior pair, and pair without prior pair programming experience in all situations. The generic pair incurs less defect cost than solo in all circumstances.

Figure 5.6 Defect Cost



Labor Cost

Table 5.7 presents the standardized scores on overall labor cost as well as the dollar amounts from the means.

Table 5.7 Labor Cost

	Z-Score			$ Amount from the Mean		
	Low	**Medium**	**High**	**Low**	**Medium**	**High**
Solo	.406	.389&	.878	10,870	145,727	1,882,040
Pair	-.166	.068	-.039	(4,446)	25,306	(84,053)
Jr-Jr	.276	.380&	.101	7,398	142,213	217,130
Jr-Sr	-.099	-.079	-.388	(2,659)	(29,578)	(832,030)
Sr-Sr	-.468	-.507	-.417	(12,546)	(189,609)	(893,804)
Prior PP0	.524	.518	.607	14,051	193,806	1,299,759
Prior PP1	.036	-.103	-.132	956	(38,446)	(282,943)
Prior PP2	-.509	-.666	-.609	(13,624)	(249,418)	(1,306,094)
Mean	87,045	811,507	7,594,271			
SD	26,791	374,236	2,142,952			

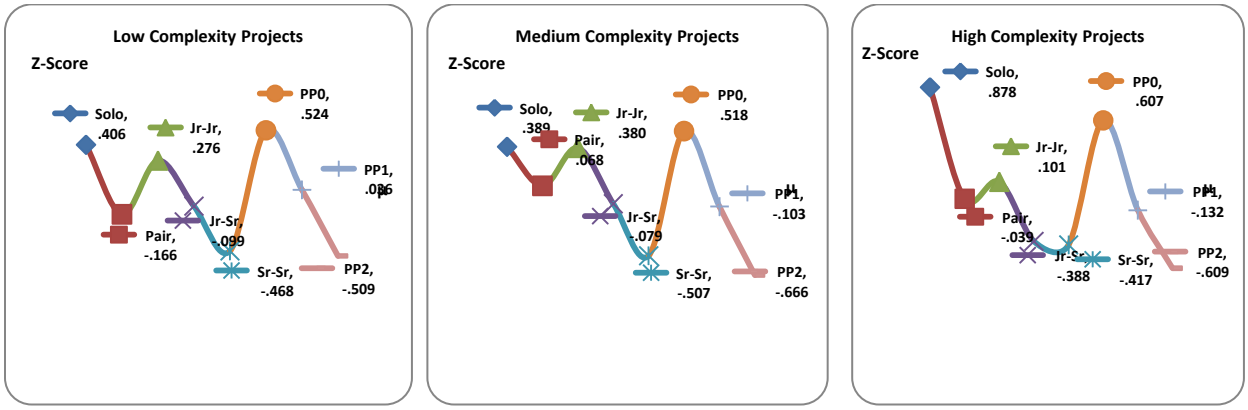
In low and medium complexity projects, solo does not incur the highest labor cost. In low complexity projects, pairs without prior pair programming experience are more expensive than solo. In medium complexity projects, besides pair without prior pair programming experience, junior-junior pairs also incur higher labor costs than solo. In high complexity projects, however, all pairs regardless of their pair composition incur less labor cost than solo.

Considering expertise in pair composition, senior-senior pairs are the best in low and medium complexity projects. In high complexity projects, junior-senior pairs are the most cost effective. Considering prior pair programming experience in pair composition, pairs with prior pair programming experience appear to be the most cost effective.

This finding is further illustrated in Figure 5.7. Figure 5.7 indicates considering all groups, pairs with prior pair programming experience are the most cost effective. In low and medium complexity projects, pairs without prior pair programming experience are the most

expensive. In high complexity projects, solo is the most expensive. The generic pair is more cost effective than solo in all situations.

Figure 5.7 Labor Cost



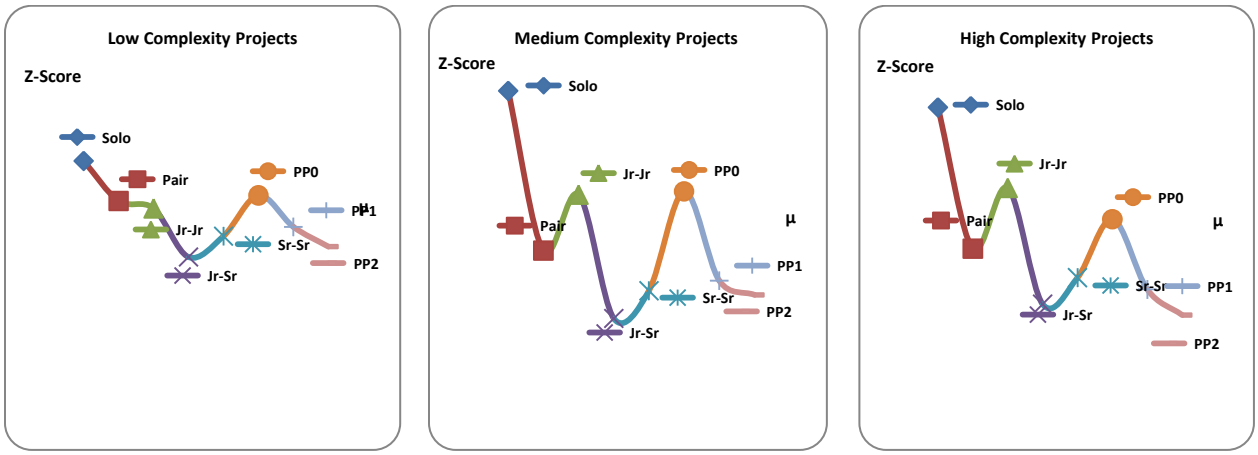
5.3.2. Training Cost

No data is available to suggest how much impact knowledge transfer through different programming methods has on overall training cost. Therefore, to provide the widest possible insight into this relationship, a series of analyses were conducted assuming knowledge transfer ranges of 10% to 100% impact on training cost. The detailed z-scores and their associated means and standard deviations are included in Appendix D.

The z-score data from appendix D suggests regardless of percent of impact, the relative contribution of programming methods to training cost as a result of knowledge transfer demonstrates the same trend. This is illustrated in Figure 5.8. To be specific, in low complexity projects, junior-senior pairs result in the highest savings in training cost, followed by pairs with prior pair programming experience, pairs one with prior pair programming experience, and senior-senior pairs. In medium complexity projects, the same trend continues except the generic pair has training costs lower than the mean. In high complexity projects, pairs with prior pair

programming experience results in the lowest training cost. Solo incurs the highest training cost in all situations.

Figure 5.8 Training Cost – General Trend

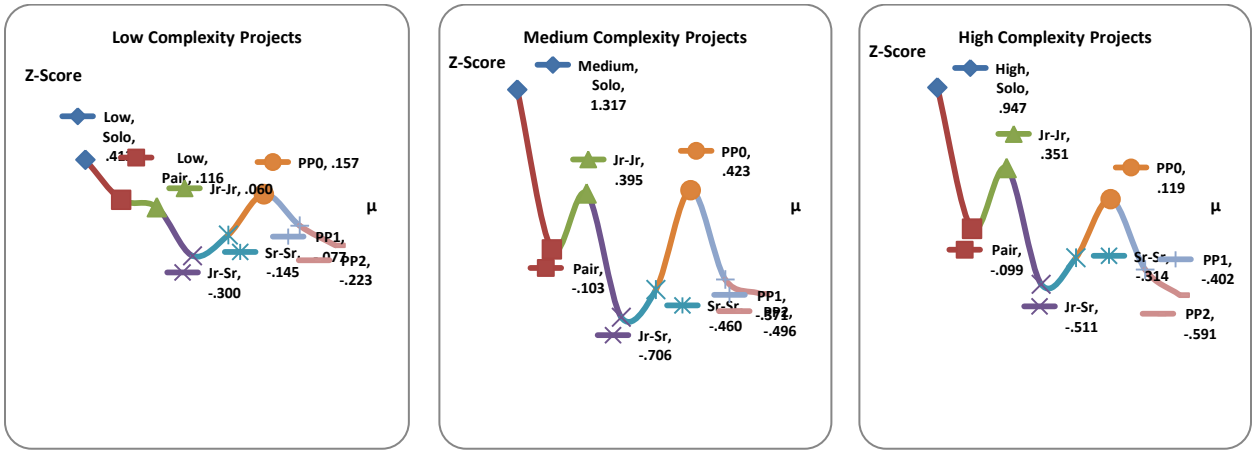


To illustrate the findings further, a single point in our testing range 40% impact on training cost - is chosen to present the detailed findings. Table 5.8 diplays the standardized scores on training cost for the 40% impact range as well as the dollar amounts from the means. The results are demonstrated in Figure 5.9 as well.

Table 5.8 Training Cost – 40% Impact

	Z-Score - 40% Impact			$ Amount from the Mean		
	Low	**Medium**	**High**	**Low**	**Medium**	**High**
Solo	.412	1.317	.947	53	911	983
Pair	.116	-.103	-.099	15	(71)	(103)
JJ	.060	.395	.351	8	273	364
JS	-.300	-.706	-.511	(39)	(488)	(530)
SS	-.145	-.460	-.314	(19)	(318)	(326)
None	.157	.423	.119	20	293	124
One	-.077	-.371	-.402	(10)	(257)	(417)
Both	-.223	-.496	-.591	(29)	(343)	(613)
Mean	3,384	16,962	45,978			
SD	129	692	1,037			

Figure 5.9 Training Cost – 40% Impact



Even though Bonferroni pair-wise comparisons on the z-scores suggest all pair differences are statistically significant (p-value < 0.05), amounts from the means indicate there may not be any practical significance resulting from knowledge transfer. In the 40% impact scenario, the largest amount below the average is $613, and the largest amount above the average is $983. Examinations on the dollar amounts from the means on all percentage impact scenarios reveal that the largest deduction is $1,125, and highest increase is $2,518. Whether twenty-five hundred dollars carries any practical significance to the entire project cost will be highly dependent on the budget situation in the organization. Nonetheless, the potential impact in any given situation could easily be material and, as such, we must consider it in this research.

5.3.3. Duration, Opportunity Cost, and Labor Slack

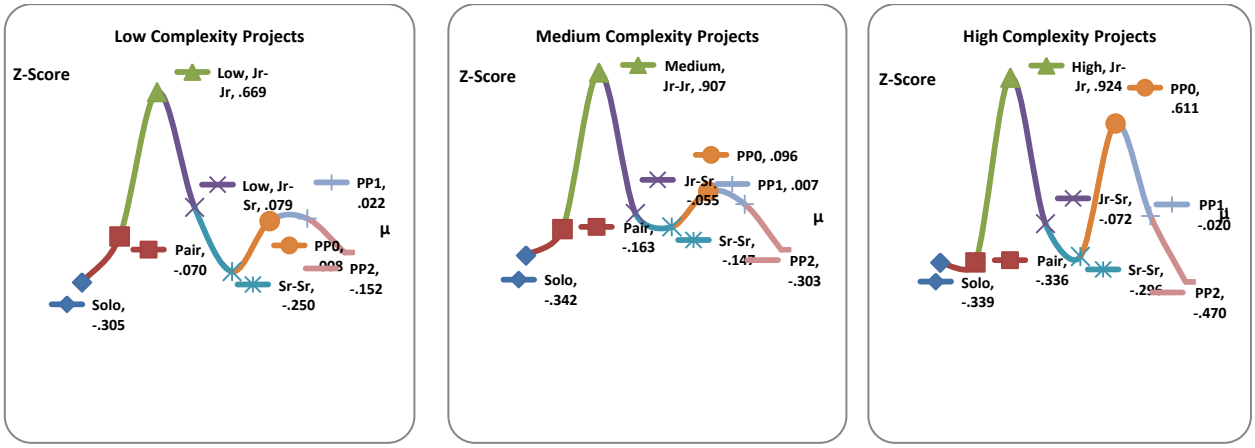
Table 5.9 presents project duration in days by programming methods at three different project complexity levels. Based on results of the simulation data, the mean duration for a low complexity project is 20 days, the mean duration for a medium complexity project is 93 days, and the mean duration for a high complexity projects is 1,010 days. The differences are statistically significant (p-value < 0.001).

The data suggests in both low and medium complexity projects, solo is the fastest programming method. In high complexity projects, however, pairs with prior pair programming experience are the fastest, followed by solo and generic pairs (the difference between solo and generic pairs is not statistically significant, p-value = 1.0). Among all programming methods and pair compositions, the junior-junior pairs are the slowest, followed by pairs without prior pair programming experience. These finding are further illustrated in Figure 5.10.

Table 5.9 Duration

	Z-Score			Days from the Mean		
	Low	**Medium**	**High**	**Low**	**Medium**	**High**
Solo	-.305	-.342	-.339&	(3)	(15)	(129)
Pair	-.070	-.163	-.336&	(1)	(7)	(128)
Jr-Jr	.669	.907	.924	7	40	350
Jr-Sr	.079	-.055	-.072	1	(2)	(27)
Sr-Sr	-.250	-.147	-.296	(3)	(7)	(112)
Prior PP0	.008#	.096	.611	0	4	232
Prior PP1	.022#	.007#	-.020#	0	0	(8)
Prior PP2	-.152	-.303	-.470	(2)	(13)	(178)
Mean	20	93	1,010			
SD	10	44	379			

Figure 5.10 Duration

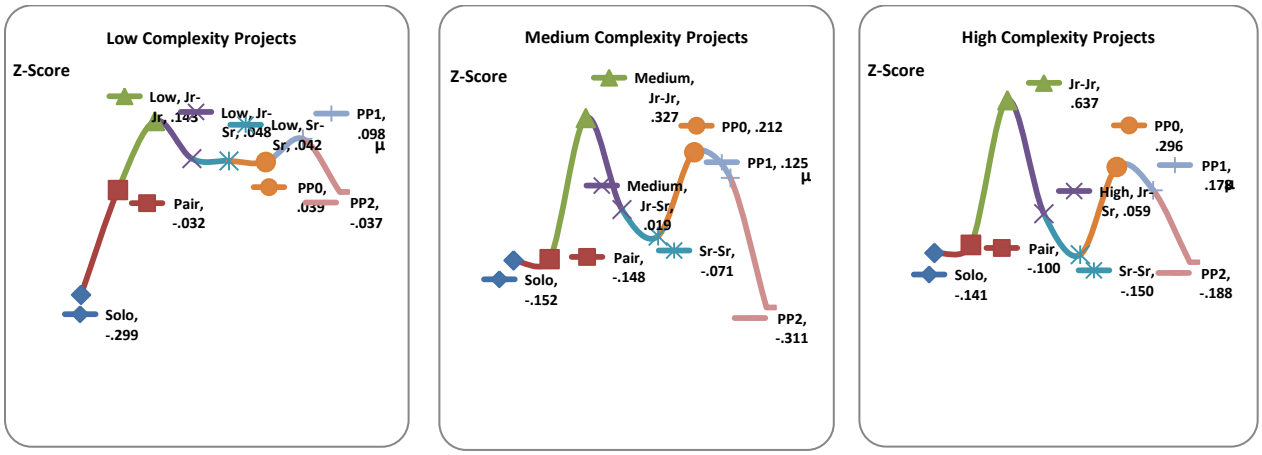


Opportunity cost is cost associated with those opportunities an organization potentially loses for committing to, and therefore undertaking a given project. When a project falls behind schedule (in this case, the duration mean), an organization's resources are tied up in the project, and therefore cannot be made available to undertake other projects. As a result, opportunity costs are incurred. Results suggest in low complexity projects, solo incurs the lowest opportunity cost, however in medium and high complexity projects, pairs with prior pair programming experience incur the least opportunity cost. Furthermore, junior-junior pairs incur the highest opportunity cost in all situations, followed by solo. Table 5.10 presents the z-scores on opportunity cost as well as the dollar amounts from the means. The finding is further illustrated in Figure 5.11.

Table 5.10 Opportunity Cost

	Z-Score			Opportunity Cost		
	Low	**Medium**	**High**	**Low**	**Medium**	**High**
Solo	-.299	-.152&	-.141	(4,462)	(10,615)	(163,955)
Pair	-.032&	-.148&	-.100	(479)	(10,332)	(116,082)
Jr-Jr	.143	.327	.637	2,141	22,790	740,834
Jr-Sr	.048	.019#	.059	710	1,325	68,467
Sr-Sr	.042&&	-.071	-.150	628	(4,934)	(174,496)
Prior PP0	.039&&	.212	.296	583	14,734	344,411
Prior PP1	.098	.125	.178	1,464	8,725	207,324
Prior PP2	-.037&	-.311	-.188	(556)	(21,624)	(218,500)
Mean	22,279	127,663	2,384,615			
SD	14,928	69,636	1,163,733			

Figure 5.11 Opportunity Cost



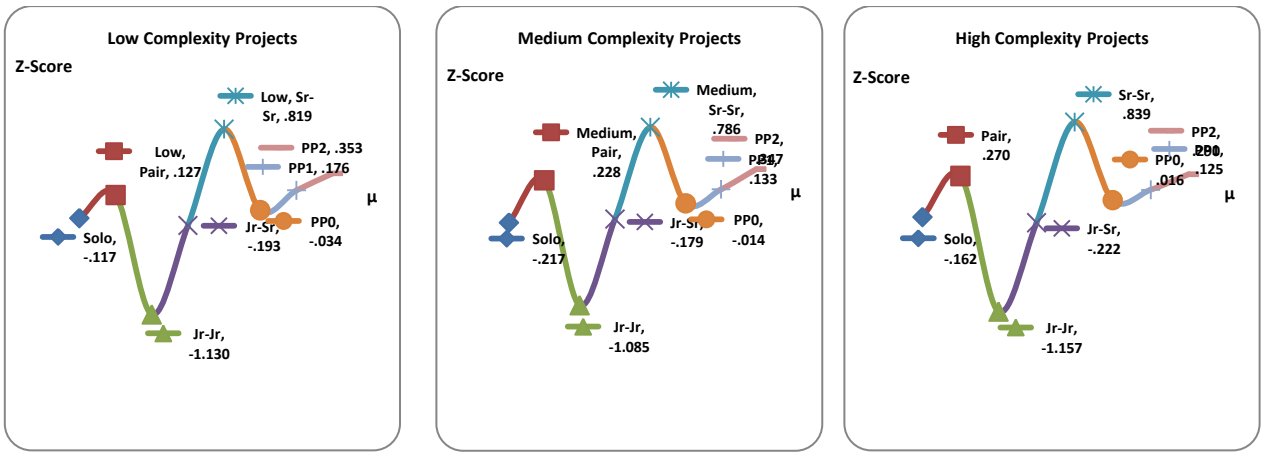
In contrast to opportunity costs, labor slack occurs when a project is finished ahead of schedule. When an organization has labor slack, more work, presumably, can be brought to the work force, so more projects can be started and finished. Alternatively, if labor slack is consistent, head count can be reduced thus creating a cost savings to the organization. This argument suggests labor slack contributes positively to the project cost by representing a deduction from total project cost as a result of shortened duration.

Table 5.11 presents the z-scores on labor slack as well as the dollar amounts from the means. Results suggest senior-senior pairs incur the highest labor slack at all three project complexity levels, followed by pairs with prior pair programming experience, pairs with one having prior pair programming experience, and generic pairs. Despite the z-scores demonstrating the same trend across three project complexity levels, the magnitude of increase in dollar amount is significant as the project complexity increases. The finding is illustrated in Figure 5.12.

Table 5.11 Labor Slack

	Z-Score			Labor Slack		
	Low	**Medium**	**High**	**Low**	**Medium**	**High**
Solo	-.117	-.217	-.162	(1,244)	(9,977)	(80,684)
Pair	.127	.228	.270	1,347	10,479	134,669
Jr-Jr	-1.130	-1.085	-1.157	(11,980)	(49,903)	(576,340)
Jr-Sr	-.193	-.179	-.222	(2,047)	(8,216)	(110,431)
Sr-Sr	.819	.786	.839	8,683	36,166	417,981
Prior PP0	-.034	-.014#	.016#	(363)	(638)	8,047
Prior PP1	.176	.133	.125	1,861	6,127	62,415
Prior PP2	.353	.347	.290	3,743	15,962	144,343
Mean	41,404	173,148	2,063,020			
SD	10,602	46,005	498,345			

Figure 5.12 Labor Slack



5.3.4. Total Cost

Table 5.12 presents the standardized scores on total project cost as well as the dollar amounts from the average.

Table 5.12 Total Project Cost

	Z-Score			$ Amount from the Mean		
	Low	**Medium**	**High**	**Low**	**Medium**	**High**
Solo	.220	.355	.622	9,982	153,288	2,207,970
Pair	-.158	.013#	-.100	(7,188)	5,689	(355,221)
Jr-Jr	.476	.496	.142	21,584	214,213	503,463
Jr-Sr	.020&	-.065	-.309	898	(28,263)	(1,097,673)
Sr-Sr	-.511	-.542	-.411	(23,180)	(234,249)	(1,459,621)

Prior PP0	.396	.526	.661	17,960	227,170	2,345,984
Prior PP1	.017&#	-.088	-.058	749	(38,027)	(206,822)
Prior PP2	-.459	-.694	-.546	(20,805)	(299,821)	(1,938,083)
Mean	71,304	782,984	7,961,844			
SD	45,361	432,003	3,550,279			

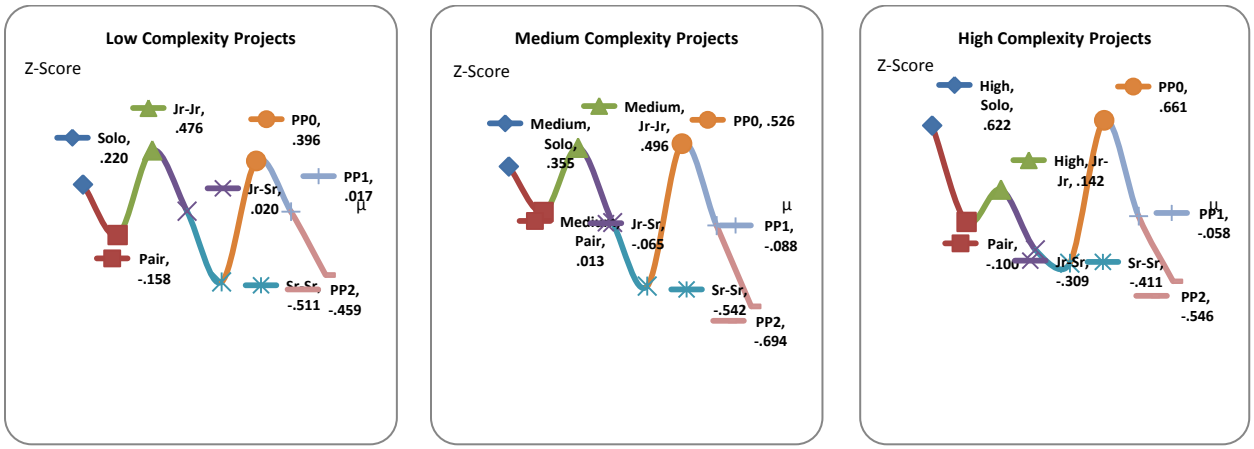
Results suggest project complexity clearly matters. As project complexity increases, the project cost goes up. The cost differences at three complexity levels are statistically significant (p-value < 0.001). Results also suggest the savings associated with pair programming become substantially significant as project complexity increases. The savings range from thousands of dollars in low complexity projects to over one million dollars in high complexity projects.

Results indicate, despite the pair programming method's effectiveness in mitigating the increased project costs, as complexity increases, pair composition matters. Considering expertise in pair composition, even though junior-senior and senior-senior pairs consistently perform better than solos, the junior-junior pairs do not perform as well unless it is a high complexity project. In addition, projects done by junior-senior pairs cost approximately the same as an average project in the low complexity situation, but incur measurable savings in both medium and high complexity projects.

Considering prior pair programming experience in pair composition, projects finished by pairs without pair programming experience are more costly than those finished by solos. Also, pairs with prior pair programming experience are consistently the most cost effective group across the three project complexity levels, while pairs with one having prior pair programming experience only result in savings in medium and high complexity projects. The generic pair is more cost effective than solo and its performance lies between the best and the worst pair combinations.

The findings are illustrated in Figure 5.11. Considering all programming methods and pair compositions, the method that results in highest project cost is, surprisingly enough not solo, but rather the junior-junior pairs for low complexity projects, and pairs without prior pair programming experience for medium and high complexity projects. It seems apparent that the junior-junior pair does not seem to be a good cost saving mechanism unless the situation is a high complexity project. Senior-senior pairs result in the highest savings in low complexity projects while pairs with prior pair programming experience result in highest savings in medium and high complexity projects. Nonetheless, it seems reasonable to conclude that pair programming represents a cost saving mechanism but system complexity and pair composition must be considered if such cost savings are to be realized.

Figure 5.11 Total Project Cost



6. DISCUSSIONS, LIMITATIONS, AND FUTURE RESEARCH

6.1.Discussion

One principal objective of this study was to provide a validation of the research model proposed in the Research Model and Hypotheses section. The results obtained have provided the

desired insights. The following discussion is organized around hypotheses and the implications of the results.

H1 argued as system complexity increases, effort increases, defect rate increases, knowledge transfer increases. While the arguments appeared intuitive, there have been very few empirical studies providing support of these arguments. The studies we identified to be somewhat close to our approach did not consider knowledge transfer, and instead of system complexity, primarily focused on program size and its relationship between program effort and defect (e.g. Kemerer 1987; Banker and Kermerer 1989; Mukhopadhyay et al. 1992; Krishnan et al. 2000; Matson et al. 2002). Furthermore, in pair programming research, the handful of studies that considered system complexity in their research models (Arisholm et al. 2007; Balijepally et al. 2009) treated system complexity as a moderator instead of an independent variable as in this research. Therefore, our study is one of the few that provides strong empirical evidence to support the claim that system complexity *is* the salient driver with regard to the outcome of a software development project.

H2a proposed when compared to solo programming, as system complexity increases, pair programming will moderate the effect of system complexity on programming effort, thus reducing effort. Results suggest pair programming will incur less programming effort than solo programming in high complexity projects but more effort in low complexity projects; in terms of medium complexity projects, the perceptions were split: practitioners who had pair programming experience believed pair programming would reduce effort while practitioners who did not have pair programming experience thought pair programming would increase effort. Despite the different views in medium complexity projects, it is clear that practitioners agree pair programming can be an effort-reduction mechanism in high complexity projects. The

implication of this result is the pair programming method is best suited for a high complexity project and ill suited for a low complexity project given the goal to reduce overall programming effort. This also serves to explain the equivocality in findings reported by prior pair programming studies: some noted effort increase (Nosek 1998; Williams et al. 2000; Rostaher and Hericko 2002), others revealed effort decrease (Lui and Chan 2003; Canfora et al. 2005), and one presented inconsistent findings across the tasks (Canfora et al. 2007). Our study suggests pair programming studies must consider the complexity of the programming tasks in order to draw a sensible conclusion.

H2b states when compared to solo programming, as system complexity increases, pair programming will reduce the effect of complexity on defect rate, thus reducing the overall defect rate. Results suggest pair programming helps reduce defect rate in projects at all complexity levels, but as complexity increases, there is a higher reduction in defect rate. The implication of this result is pair programming is an effective mechanism to reduce defects, which is particularly true in high complexity projects. This result supports the finding shared by the majority of the previous pair programming research, although, in prior studies, the effect of complexity was largely ignored.

H2c argued when compared to solo programming, as system complexity increases, pair programming will increase the effect of system complexity on knowledge transfer thus enhancing knowledge transfer. Results suggest pair programming increases knowledge transfer in projects at all complexity levels and the effect is higher as complexity increases. The implication of this result is pair programming is an effective mechanism to transfer knowledge among the team members, and this is particularly true in high complexity projects. This study represents the first pair programming study that operationalized knowledge transfer, validated its

validity, and provided strong empirical evidence on the effect of pair programming on knowledge transfer.

H3a proposed programmer expertise would affect the effectiveness of the pair programming method. The study examined three pair expertise compositions: junior-junior, junior-senior, and senior-senior. Results suggest senior-senior pairs will incur the least amount of programming effort and produce programs with the lowest defect rate, junior-senior pairs fall second, and junior-junior pairs will achieve the worst results. In terms of knowledge transfer, junior-senior pairs work the best, senior-senior pairs the second, and junior-junior pairs the last. The implication of these results is to assist in determining what optimal pair composition to adopt depending on the goals/constraints of the project: if the goal is to complete the project with the least effort or the least defect, then senior-senior pairs should be the choice; if the goal is to share knowledge, then junior-senior pairs will be the best. Our results suggest junior-junior pairs do not seem to be a good option in any of the three circumstances. Although several studies examined pair compatibility, the majority of them did not operationalize pair composition, and none examined its effect on knowledge transfer. Therefore, this study is the first that took a comprehensive and systematic approach to the study of pair composition in relation to the outcome of the project.

H3b stated prior pair programming experience would affect the effectiveness of the pair programming method. Results indicate pairs with prior pair programming experience achieved the best results in all three aspects (least effort, lowest defect rate, and highest knowledge transfer) while pairs without prior pair programming experience work the worst (highest effort, highest defect rate, and lowest knowledge transfer). While the results appeared to be intuitive, this study is the first one to provide empirical evidence to support the claim prior pair

programming experience is an important factor: it directly contributes to how effectively a pair performs.

Furthermore, based on whether they have or do not have pair programming experience, perceptions are different. In general, practitioners who had pair programming experience tended to favor the pair programming method over the solo. For instance, in the case of medium complexity projects, practitioners with pair programming experience thought pair programming would reduce effort while practitioners without pair programming experience believed pair programming would increase effort. The same was true when considering junior-senior composition: practitioners with pair programming experience thought junior-senior pairs would reduce effort while practitioners without pair programming experience believed junior-senior pairs would increase effort.

Another principal objective of the study is to investigate the cost constructs to determine in what situations pair programming is more cost effective than solo programming. Simulation results suggests the selection of the most cost effective approach depends on the complexity of the project and pair composition. In particular, several findings are suggested. First, our results make clear that *project complexity is the driving force of project cost*. As project complexity increases, all costs go up: labor cost, training cost, opportunity cost, labor slack, and overall project cost. In addition, even though there are cost variations among the programming methods, no method is likely to reduce the cost of a medium or high complexity project to the cost range of a low complexity project, as suggested by the dollar ranges of projects at different complexity levels: a low complexity project ranges from $48,000 to $92,000, a medium complexity project runs from $400,000 to a million, and a high complexity project is from five million to ten million. While this finding is not surprising, this study is, as above, the first in pair programming

research to treat complexity as the independent variable and derive dollar ranges on projects at different complexity levels.

Second, results suggest pairs with prior pair programming experience incur the lowest labor cost in all situations while pairs without prior pair programming experience incur the highest labor cost in low and medium complexity projects and solos incur the highest labor cost in high complexity projects. This finding illustrates the importance of prior pair programming experience in pair composition and suggests pairs with prior pair programming experience can achieve the best results yet pairs without prior pair programming experience will achieve the worst outcome. Also, the impact of hourly pay rate and defect cost are important factors to consider: for example, junior-junior pairs have the lowest effort cost because of the low hourly pay rate, but senior-senior pairs have the lowest overall labor cost despite their high hourly pay rate due to savings in defect cost, and solos are the least cost effective in overall labor cost in high complexity projects due to high defect cost despite its low effort cost. Previous studies usually investigated effort and defect separately and the typical conclusion was pairs incurred more effort but lowered defects. Our study suggests *it is important to examine the combined effect of effort cost and defect cost on the overall labor cost*.

Third, most previous studies concluded, when compared to solo programming, pair programming serves to shorten the project duration (e.g. Williams et al. 2000; Cockburn and Williams 2001; Rumpe and Schroer 2002; Williams and Kessler 2003). Our study yielded contradictory results. Our findings suggest solos are faster than pairs except in one situation: pairs with prior pair programming experience are found to be faster than solos in high complexity projects. A closer examination of the previous studies revealed that previous studies tended to compare pair (two developers) to solo (one developer) and their conclusions,

intuitively enough, suggested two developers worked faster than one developer. However, comparing two developers to one developer is not a fair comparison. A fair comparison should be a pair (two developers) to two solos (two developers), which is the approach we followed in this study. In pair programming, each pair works together on one particular task, while in solo programming, the same task is divided into two, and each solo works on a different subtask concurrently. Consider the following: a given project is divided into 12 programming tasks with a team size of four (two pairs vs. four solo developers). In pair programming, since there are two pairs, two tasks are being worked on simultaneously whereas in solo programming, since there are four developers, four tasks are being worked on simultaneously. Using the same team size for both pair and solo programming, it is not surprising that solos will usually finish the project sooner than pairs given the fact that in solo programming more tasks are being worked on simultaneously. Our finding concurs with the conclusion drawn by Parrish et al. (2004). Their study showed, as developers divided the tasks into subtasks and worked on subtasks independently, they were several times faster than developers working together on the same tasks. Our study did, however, discover one circumstance where pairs are faster than solos. Results suggest project complexity and pair composition are influential factors: *when the complexity of the project is high and pairs are comprised of developers with prior pair programming experience, pairs are faster than solos*. Besides considering duration, we also investigated the impact of shortened or prolonged duration on the project cost: opportunity cost and labor slack. As far as we know, this is, again, the first study to include opportunity cost and labor slack in the total cost calculation.

Fourth, this study is the first to quantify knowledge transfer and to relate knowledge transfer to training cost. Since no data is available to suggest how knowledge transfer influences

training cost, we conducted a series of sensitivity analyses assuming knowledge transfer ranges from 10% to 100% impact on training cost. Results suggest solos incur the highest training cost in all situations, junior-senior pairs result in highest savings in training cost in low and medium complexity projects and pairs with prior pair programming experience result in highest savings in training cost in high complexity projects. A close examination of all of the impact scenarios suggests the dollar savings range from $500 to $2500. Therefore, based on our calculation, the impact of knowledge transfer on training cost may not carry practical significance. Nonetheless, it is important to consider as a means to reduce overall training costs in a particular project. As we learn more about this effect, we may find it to be a more significant variable in the equation.

Finally, regarding the overall project cost, two prior studies research the total project cost and drew different conclusions: one concluded pairs are more cost effective than solos in all situations (Erdogmus and Williams 2003), and the other suggested pairs are more cost effective when market pressure is high (Padberg and Müller 2003). *Our results indicate pair programming is not more cost effective than solo programming in all situations. Whether pairs are more cost effective than solos depends on pair composition and project complexity*. Considering expertise in pair composition, junior-junior pairs are not cost effective compared to solos unless the project is a high complexity project; considering prior pair programming experience in pair composition, pairs without prior pair programming experience are consistently more expensive than solos. In low complexity projects, junior-senior pairs, senior-senior pairs, pairs with one having prior pair programming experience, pairs with prior pair programming experience are more cost effective than solos while junior-junior pairs and pairs without prior pair programming experience are more costly than solos; in medium complexity projects, the same trend continues; in high complexity projects, however, all pairs except pairs without prior

programming experience are more cost effective than solos. Furthermore, as discussed along with duration earlier, our study suggests, with the same team size, when market pressure is high, the best choice is solo programming over pair programming since in solo programming more tasks can be worked on simultaneously by different developers. Our finding concludes more dimensions need to be added to the cost consideration: the complexity of the project and pair composition. Our findings also illustrate the importance of prior pair programming experience which is a construct not previously studied empirically.

6.2.Limitations

While the results appear encouraging with regard to furthering our understanding of the role of pair programming in software development, as with any empirical endeavor, several limitations must be taken into consideration when interpreting them.

The first area of limitation lies with the exploratory nature of the questionnaire. In an ideal situation, one would adopt questionnaires that have been proven to reach the desired levels of validity and reliability. Unfortunately, this was not possible in this case because no such instrument existed. A potential threat to validity and reliability of the newly developed measures in this study is inaccurate reporting from the respondents. However, we believe this threat is minimized by the following mechanisms. First, most of the measures are widely adopted industrial standards software developers are familiar with. In addition, in the survey tool, the definition of each measure was explicitly presented to the respondents before they answered associated questions. Given the above reasons, it is unlikely that the respondents would misunderstand the questions and unintentionally report wrong information.

Second, only complete surveys were used for analysis. During data collection, the authors received numerous comments from respondents stating reasons why they didn't

complete the survey. Not knowing the answers to the questions was the primary reason. Therefore, it is reasonable to believe respondents who didn't know or didn't feel comfortable reporting the answers abandoned the survey and, therefore, were not suitable or appropriate respondents. Despite all the preventive effort, it is still important to bear in mind that future studies are needed to perfect the instrument and confirm its validity and reliability.

As is true with any other survey study, another limitation regarding this research effort is the possibility of sampling bias. Since there is no centralized file that would allow us to draw a random sample of software developers in the industry, we had to solicit participants based on the resources available. Furthermore, there is always the possibility of self-selection bias. However, the respondents' profile presented in the results section suggests the respondents were quite diverse in terms of age, experience, and industry. In addition, we checked non-response bias by comparing the responses from the first 20 percent of the responses received to those from the last 20 percent received and results indicated non-response bias didn't seem to be a problem. Finally, assuming effect size 0.35, alpha 0.05, and levels of 3, the power table for one-way repeated measures suggests a power value of 0.99 with sample size of 21 (Jaccard and Becker 2002 pp.588; Warner 2008 pp.917). Thus, we consider 191 to be a sample size more than large enough to address some of the potential biases.

The other limitation lies with the nature of survey data. The participants were asked to provide responses based on experience and/or perception. Nevertheless, the source of the analysis results is perception data. Since there is always a possibility that perception may not align with reality seamlessly, cautions need to be given as we interpret the data.

One area to be clarified is the survey items that were single-item constructs. In general, this approach is not deemed to be appropriate and multiple item constructs are preferred. In this

case, however, many of the constructs used were, in fact, best described using a single-item approach. For example, while it is easy to conceive a variety of methods to describe the construct *effort*, the common industry-accepted approach is to describe the construct in terms of programmer hours – a single-item approach. In addition, it must be noted that the survey instrument combined two different data collection efforts: 1) to collect data related to the constructs in the research model and 2) to collect parameter data for use in the development of the simulation conducted in Study 2. Given this, we believe the single-item constructs we employed were appropriate in their use and do not detract from either the validity or the results or their associated conclusions.

A final area warranting discussion is we used linear functions (e.g. labor cost = effort cost + defect cost; effort cost = hours * hourly pay rate) in all of the cost calculations. It is possible that some of the functions are more sophisticated and may be curvilinear or in some other higher order forms. A series of future studies should be conducted and focus on the relationships of the various cost constructs in the software development cost model. For that purpose industrial data from actual software development projects through computer logs and other automatic data capturing methods is desirable.

6.3 Conclusions

As noted above, this study makes several contributions. First, it documents the most comprehensive literature review on pair programming contained in the extant literature, which lays the foundation for future research in the field. Second, it developed a survey instrument and conducted one of the first industrial-wide surveys on pair programming. Third, it proposed and empirically tested a unique and comprehensive pair programming research model during which several under-studied constructs and relationships (e.g. system complexity, pair composition,

knowledge transfer) were thoroughly examined. Finally, the bootstrap simulation yielded several findings regarding when and where pairs are more cost effective than solos, which, hopefully answers some of the fundamental questions a business organization will raise when considering the adoption of the pair programming method.

As the ultimate goal of all academic research is to move from description to prescription, we believe our findings allow for such an event. Several guidelines, albeit preliminary in nature, can be offered to the industry based on the results of this study.

First, *pair programming is a good choice when system complexity is high*. In high complexity projects, pair programming reduces programming effort, improves quality, and reduces overall labor cost compared to solo.

Second, *pair composition must be taken into account*. In most situations, whether pairs are more effective than solos depends on how the pairs are comprised.

Third, *prior pair programming experience is an important factor to consider*. Pairs with prior pair programming experience perform well and pairs without prior pair programming experience perform poorly, which suggests, to achieve optimal outcome, organizations should invest money and time for developers to gain pair programming experience on small tasks before assigning them to projects.

Fourth, *solo is a good choice when the market pressure is high*. Solos are faster than all pairs except in one situation: pairs with prior pair programming experience in a high complexity project.

Finally, while every project has its primary goal, *it is important to examine the interactions of multiple cost factors such as defect, effort, duration, and knowledge transfer and consider their combined effect on the ultimate goal of the project*. For example, junior-junior

pairs result in the lowest effort cost, junior-senior pairs have the highest knowledge transfer, while senior-senior pairs produce the highest quality project hence have the lowest defect cost. As we combine all the factors and consider the overall project cost, the senior-senior pair was found to be the most cost effective among the three.

6.4 Future Research

Through the results obtained from the two studies contained herein, several areas of future research are identified. To begin, since the results of this study are based on perceptual data, it is desirable to acquire data from specific software projects through log files and other automatic data capturing mechanisms to further validate the research model. Next, following the findings in the group interaction literature, other pair compositions variables, such as personality, gender, and age should be examined. Considerations should also be given to solos at different experience levels when comparing pairs to solos. Furthermore, mixed programming methods in one team – a mix of solos and pairs, as well as mixed compositions in the pairs in one team should be investigated.

Additionally, a focused investigation of the relationships of cost constructs regarding the software development cost model is desirable. In our study, the relationships were assumed to be linear. However, there might exist more sophisticated higher order relationships. Continuing with this approach, other variables, e.g. environmental factors, could shed additional light on the relative effectiveness of the two programming methods. Finally, it is desirable to consider programming as part of the overall software development and examine the relationship between the choice of programming method and changes in other project phases such as planning, analysis, design, and maintenance.

In closing, our study makes clear that the previous conclusions regarding pair programming are limited in nature and the development approach may not be as clearly desirable in all situations as was previously assumed. The pair programming approach clearly adds value in situations where it is appropriate but certain conditions must be met for this goal to be achieved. It is our hope that this research will serve to inform for research and practice with regard to the pair programming approach and will, additionally, serve to generate future research efforts based upon this work that serve to further our understanding of the various salient constructs.

7. APPENDIXES

A. Survey Instrument

Dear respondents:

Thank you for your participation in this academic research project. **If you have pair programming experience, please answer the survey questions based on your EXPERIENCE. Otherwise, please answer the questions based on your PERCEPTION of pair programming.**

This project seeks to better understand the benefits and drawbacks of pair programming in relation to the more conventional solo programming approach commonly found in industrial settings.

Pair programming is defined as two (2) programmers working side by side on the same programming project using a single workstation. One programmer is referred to as the "Driver" and the other is referred to as the "Navigator". The Driver types the code and pays close attention to the coding details. The navigator reviews the code and is responsible for developing and suggesting alternative strategies. The two programmers change roles periodically as the need for various skill sets and expertise changes.

Solo programming is defined as one (1) programmer working alone and performing all programming tasks.

We believe the results of this research effort will serve to contribute to both the academic and the applied communities. If you are interested, we will be happy to share aggregated information once the research is completed.

The questionnaire is expected to take approximately 10-15 minutes to complete. Your name and affiliated organization will not be associated in any way with the research findings. The survey responses are stored on secured severs and only aggregated results will be reported. If you have any questions about this survey or our research, please direct them to nansun@ku.edu.

Completion of the survey indicates your willingness to participate in this project and that you are at least age eighteen. If you have any questions about your rights as a research participant, you may call (785) 864-7429 or write to the Human Subjects Committee Lawrence Campus (HSCL), University of Kansas, 2385 Irving Hill Road, Lawrence, Kansas 66045-7563, or email dhann@ku.edu.

Sincerely,

George Marakas, Professor of Information Systems
Nan Sun, PhD Candidate in Information Systems

Solo Programming | Pair Programming

How long have you personally used the pair programming method?

Never | < 1 year | 1-2 years | 3-4 years | 5-6 years | 7-8 years | 9-10 years | > 10 years

Your current position in your company:

Developer | Project Manager | Team Lead | VP/Director of Development | Development Manager | Consultant/Trainer | Architect | QA

Other (please specify)

Your experience in IS/IT field:

None | < 2 years | 2-5 years | 6-10 years | 11-20 years | 20+ years

Where are you located?

North America | South/Central America | Europe | Asia | Australia/NZ | Africa

Your highest level of education:

High School or Equivalent | Some College | College Degree | Graduate Degree

Your gender:

Male | Female

Your age:

<30 | 30-39 | 40-49 | 50-59 | 60-69 | >69

What is the size of a typical software development project in your organization?

Please enter # of lines of code in 1000 (KLOC)

A module is a program that is designed to accomplish a well-defined task.

Estimate the number of modules in a typical ...

Low-complexity project

Medium-complexity project

High-complexity project

Q1a. Estimate the number of programming hours, assuming a typical ...

Low-complexity project

Medium-complexity project

High-complexity project

Q1b. Estimate the number of defects (per 1000 lines of code), assuming a typical...

Low-complexity project

Medium-complexity project

High-complexity project

Q1c. As system complexity increases, ...

	1 Strongly Disagree	2	3	4 Neutral	5	6	7 Strongly Agree
More communication will occur regarding the understanding of a module and how this module integrates with other modules.	○	○	○	○	○	○	○
It tends to improve a programmer's knowledge of the programming language in use.	○	○	○	○	○	○	○
It tends to improve a programmer's general problem solving skills.	○	○	○	○	○	○	○

How many software development projects ...

Does your organization usually complete in one year?

Does your organization usually undertake concurrently?

Did your organization finish in the past three years?

Out of the total number of projects your organization finishes each year, what percentage of the projects will you classify as ... (Please enter numbers between 0 and 100 and make sure they add up to 100.)

Low-complexity project?

Medium-complexity project?

High-complexity project?

Please be noted of the following definitions:

Programming effort - The total number of hours developers spend on the programming aspect of a project.
Defect rate - The number of defects per 1000 lines of code.
Knowledge transfer - The communication of knowledge from a source so that it is learned and applied by a recipient.

Estimate the number of years of experience within the project domain ...

A senior programmer has at least.

A junior programmer has less than.

Q2a/Q3. Please estimate the percentage difference in programming EFFORT between solo programming and pair programming, assuming a ...

	81-99%	61-80%	41-60%	21-40%	1-20% Pair decrease effort	<- 0 ->	1-20% Pair increase effort	21-40%	41-60%	61-80%	81-100%	> 100%
Low-complexity project.												
Medium-complexity project.												
High-complexity project.												
Pair comprising of two juniors.												
Pair comprising of a junior and a senior.												
Pair comprising of two seniors.												
Pair none of whom have prior pair programming experience.												
Pair one of whom have prior pair programming experience.												
Pair both of whom have prior pair programming experience.												

Q2b/Q3. Please estimate the percentage difference in DEFECT RATE between solo programming and pair programming, assuming a ...

	81-99%	61-80%	41-60%	21-40%	1-20% Pair decrease defect	<- 0 ->	1-20% Pair increase defect	21-40%	41-60%	61-80%	81-100%	> 100%
Low-complexity project.												
Medium-complexity project.												
High-complexity project.												
Pair comprising of two juniors.												
Pair comprising of a junior and a senior.												
Pair comprising of two seniors.												
Pair none of whom have prior pair programming experience.												
Pair one of whom have prior pair programming experience.												
Pair both of whom have prior pair programming experience.												

Q2c/Q3c. Please estimate the percentage difference in KNOWLEDGE TRANSFER between solo programming and pair programming, assuming a ...

	81-99%	61-80%	41-60%	21-40%	1-20% Pair decrease KT	<- 0 ->	1-20% Pair increase KT	21-40%	41-60%	61-80%	81-100%	> 100%
Low-complexity project.	○	○	○	○	○	○	○	○	○	○	○	○
Medium-complexity project.	○	○	○	○	○	○	○	○	○	○	○	○
High-complexity project.	○	○	○	○	○	○	○	○	○	○	○	○
Pair comprising of two juniors.	○	○	○	○	○	○	○	○	○	○	○	○
Pair comprising of a junior and a senior.	○	○	○	○	○	○	○	○	○	○	○	○
Pair comprising of two seniors.	○	○	○	○	○	○	○	○	○	○	○	○
Pair none of whom have prior pair programming experience.	○	○	○	○	○	○	○	○	○	○	○	○
Pair one of whom have prior pair programming experience.	○	○	○	○	○	○	○	○	○	○	○	○
Pair both of whom have prior pair programming experience.	○	○	○	○	○	○	○	○	○	○	○	○

Please estimate ...

	81-99%	61-80%	41-60%	21-40%	1-20% Pair decrease total project cost	<- 0 ->	1-20% Pair increase total project cost	21-40%	41-60%	61-80%	81-100%	> 100%
The percentage difference in TOTAL PROJECT COST between solo programming and pair programming assuming all else equal.	○	○	○	○	○	○	○	○	○	○	○	○

Which of the following is the most commonly used programming language in your organization?

○ Java ○ C++ ○ .net

○ Other (please specify)

Assuming using the most common programming language in your organization, estimate how many lines of code, on average, ...

A SOLO programmer produces in an hour. _____

Two programmers working as a PAIR produce in an hour. _____

Defects are mistakes left unnoticed and uncorrected in the final program.

Assuming an 8-hour working day, please estimate the number of defects, on average, ...

A SOLO programmer can identify and fix per day. _____

Two programmers working as a PAIR can identify and fix per day. _____

Please indicate the percentage of development cost in each of the following areas for a typical software development project in your organization. (Please enter numbers between 0 and 100 and make sure they add up to 100.)

Programming is _ % of overall project cost.

Rework (due to defects) is _% of overall project cost.

Training is _% of overall project cost.

Other activities are _% of overall project cost.

How long (in months) does a typical software development project last?

What is the typical development team size (# of developers) for the projects in your organization?

What is the average project budget (in $1,000)?

< $10K | $10K-$49K | $50K-$99K | $100K-$499K | $500K-$999K | $1,000K-$2,000K | > $2,000K

Please answer the following: (Please enter numbers between 0 and 100)

What percent of all software projects in your organization have been developed by using the pair programming method?

What percent of total programming time is pair programming employed in a typical project?

Your organization intends to use pair programming for future projects.

Strongly Disagree | Moderately Disagree | Disagree | Neutral | Agree | Moderately Agree | Strongly Agree

Your organization intends to use pair programming for future projects.

Strongly Disagree | Moderately Disagree | Disagree | Neutral | Agree | Moderately Agree | Strongly Agree

How large is your total software organization? (including employees related to all aspects of software development and delivery)

<5 | 5-20 | 21-50 | 51-100 | 101-250 | >250

What industry is your organization primarily in?

Comments:

B. Bootstrap Sampling Program

```
* Purpose - This program bootstraps (sample with replacements) the survey data along with calculated cost columns
5,000 times.  From each bootstrap, it calculates the average information from the data variables and saves the
information in the cost file.  Three files are used: 1) costdata.dbf contains the raw data with calculated cost columns
and has 191 records:  2) cost1.dbf holds data from each bootstrap. It has 5,000 records from each bootstrap and its
content is emptied and the file is ready to hold data from next  bootstrap once the calculation is done  and the results
are saved;  3) cost.dbf contains the calculated average information from each bootstrap and has 5,000 records at the
end of the processing.  It takes about 25 minutes to run this program.
* Software the program is written in: Microsoft Visual FoxPro 9.0
* Author - Sun
* Date - 3/28/2011

* Prepare the files
SET DEFAULT TO f:\dbf
CLOSE ALL
USE cost EXCLUSIVE
ZAP
USE cost1EXCLUSIVE
ZAP

* Set the seed for the random number generator and specify the range the random number needs to fall
* between.
?RAND(20110101)
Upper = 191
Lower = 1

* Bootstrap, calculate and save the average information.  This process is repeated 5000 times.
pickNo = 1
USE costdata
FOR loopNo = 1 TO 5000

        * Bootstrap on costdata.dbf and save the sampling records in cost1.dbf
        FOR Count = 1 TO 191
                pickNo = INT((Upper - Lower + 1) * RAND() + Lower)
                INSERT INTO cost1 SELECT * FROM costdata WHERE RECNO()  = pickNo
        ENDFOR

        * Calculate the average information from cost1.dbf and save the information in cost.dbf
        INSERT INTO cost SELECT AVG(team), AVG(fixingS),  AVG(lcx_hrs), AVG(mcx_hrs),
        AVG(hcx_hrs), AVG(lcx_defect), AVG(mcx_defect), AVG(hcx_defect), AVG(kt1), AVG(kt2),
        AVG(kt3), AVG(sl_cost), AVG(sm_cost), AVG(sh_cost), AVG(pl_cost), AVG(pm_cost), AVG(ph_cost),
        AVG(pl_jjcost), AVG(pm_jjcost), AVG(ph_jjcost), AVG(pl_jscost), AVG(pm_jscost), AVG(ph_jscost),
        AVG(pl_sscost), AVG(pm_sscost), AVG(ph_sscost), AVG(pl_p0cost), AVG(pm_p0cost),
        AVG(ph_p0cost), AVG(pl_p1cost), AVG(pm_p1cost), AVG(ph_p1cost), AVG(pl_p2cost),
        AVG(pm_p2cost), AVG(ph_p2cost), AVG(sl_cost2), AVG(sm_cost2), AVG(sh_cost2), AVG(pl_cost2),
        AVG(pm_cost2), AVG(ph_cost2), AVG(pl_jjcost2), AVG(pm_jjcost2), AVG(ph_jjcost2),
        AVG(pl_jscost2), AVG(pm_jscost2), AVG(ph_jscost2), AVG(pl_sscost2), AVG(pm_sscost2),
        AVG(ph_sscost2), AVG(pl_p0cost2), AVG(pm_p0cost2), AVG(ph_p0cost2), AVG(pl_p1cost2),
        AVG(pm_p1cost2), AVG(ph_p1cost2), AVG(pl_p2cost2), AVG(pm_p2cost2), AVG(ph_p2cost2),
        AVG(sl_edcost), AVG(sm_edcost), AVG(sh_edcost), AVG(pl_edcost), AVG(pm_edcost),
        AVG(ph_edcost), AVG(pl_jjedco), AVG(pm_jjedco), AVG(ph_jjedco), AVG(pl_jsedco),
        AVG(pm_jsedco), AVG(ph_jsedco), AVG(pl_ssedco), AVG(pm_ssedco), AVG(ph_ssedco),
        AVG(pl_p0edco), AVG(pm_p0edco), AVG(ph_p0edco), AVG(pl_p1edco), AVG(pm_p1edco),
        AVG(ph_p1edco), AVG(pl_p2edco), AVG(pm_p2edco), AVG(ph_p2edco), AVG(sl_ecost),
```

```
        AVG(sl_dcost), AVG(sl_dr), AVG(sl_ocost), AVG(sl_scost), AVG(sl_TC),  AVG(sm_ecost),
        AVG(sm_dcost), AVG(sm_dr), AVG(sm_ocost), AVG(sm_scost), AVG(sm_TC),  AVG(sh_ecost),
        AVG(sh_dcost), AVG(sh_dr), AVG(sh_ocost), AVG(sh_scost), AVG(sh_TC),  AVG(pl_ecost),
        AVG(pl_dcost), AVG(pl_dr), AVG(pl_ocost), AVG(pl_scost), AVG(pl_TC),  AVG(pm_ecost),
        AVG(pm_dcost), AVG(pm_dr), AVG(pm_ocost), AVG(pm_scost), AVG(pm_TC),  AVG(ph_ecost),
        AVG(ph_dcost), AVG(ph_dr), AVG(ph_ocost), AVG(ph_scost), AVG(ph_TC),  AVG(pl_jjeco),
        AVG(pl_jjdco), AVG(pl_jjdr), AVG(pl_jjoco), AVG(pl_jjsco), AVG(pl_jjTC),  AVG(pm_jjeco),
        AVG(pm_jjdco), AVG(pm_jjdr), AVG(pm_jjoco), AVG(pm_jjsco),  AVG(pm_jjTC),  AVG(ph_jjeco),
        AVG(ph_jjdco), AVG(ph_jjdr), AVG(ph_jjoco), AVG(ph_jjsco), AVG(ph_jjTC), AVG(pl_jseco),
        AVG(pl_jsdco), AVG(pl_jsdr), AVG(pl_jsoco), AVG(pl_jssco), AVG(pl_jsTC), AVG(pm_jseco),
        AVG(pm_jsdco), AVG(pm_jsdr), AVG(pm_jsoco), AVG(pm_jssco), AVG(pm_jsTC), AVG(ph_jseco),
        AVG(ph_jsdco), AVG(ph_jsdr), AVG(ph_jsoco), AVG(ph_jssco), AVG(ph_jsTC), AVG(pl_sseco),
        AVG(pl_ssdco), AVG(pl_ssdr), AVG(pl_ssoco), AVG(pl_sssco),  AVG(pl_ssTC), AVG(pm_sseco),
        AVG(pm_ssdco), AVG(pm_ssdr), AVG(pm_ssoco), AVG(pm_sssco), AVG(pm_ssTC), AVG(ph_sseco),
        AVG(ph_ssdco), AVG(ph_ssdr), AVG(ph_ssoco), AVG(ph_sssco), AVG(ph_ssTC), AVG(pl_p0eco),
        AVG(pl_p0dco), AVG(pl_p0dr), AVG(pl_p0oco), AVG(pl_p0sco), AVG(pl_p0TC), AVG(pm_p0eco),
        AVG(pm_p0dco), AVG(pm_p0dr), AVG(pm_p0oco), AVG(pm_p0sco), AVG(pm_p0TC),
        AVG(ph_p0eco), AVG(ph_p0dco), AVG(ph_p0dr), AVG(ph_p0oco), AVG(ph_p0sco), AVG(ph_p0TC),
        AVG(pl_p1eco), AVG(pl_p1dco), AVG(pl_p1dr), AVG(pl_p1oco), AVG(pl_p1sco), AVG(pl_p1TC),
        AVG(pm_p1eco), AVG(pm_p1dco), AVG(pm_p1dr), AVG(pm_p1oco), AVG(pm_p1sco),
        AVG(pm_p1TC),  AVG(ph_p1eco), AVG(ph_p1dco), AVG(ph_p1dr), AVG(ph_p1oco), AVG(ph_p1sco),
        AVG(ph_p1TC),  AVG(pl_p2eco), AVG(pl_p2dco), AVG(pl_p2dr), AVG(pl_p2oco), AVG(pl_p2sco),
        AVG(pl_p2TC),  AVG(pm_p2eco), AVG(pm_p2dco), AVG(pm_p2dr), AVG(pm_p2oco),
        AVG(pm_p2sco), AVG(pm_p2TC), AVG(ph_p2eco), AVG(ph_p2dco), AVG(ph_p2dr), AVG(ph_p2oco),
        AVG(ph_p2sco), AVG(ph_p2TC)
        FROM cost1

        * Empty the records in cost1.dbf so cost1.dbf is ready for the next bootstrap
        SELECT cost1
        ZAP
ENDFOR
```

C. Cost Calculation Program

```
* Purpose - This program creates the columns needed to store the cost-related information and calculates the cost
information.  The calculations are done for each of the following programming methods: solo, pair, junior-junior
pair, junior-senior pair, senior-senior pair, pair with prior pair programming experience, pair one having prior pair
programming experience, and pair without prior pair programming experience.
* Software the program is written in: Microsoft Visual FoxPro 9.0
* Author - Sun
* Date - 3/3/2011

* Clear the environment and set the seed for the random number generator.
CLEAR ALL
CLOSE ALL
SET DEFAULT TO f:\dbf
?RAND(20110101)

*********************************************************************************************
Variables needed later.  They specify the ranges the randomomly generated number needs to fall between.
********************************************************************************************* Pay
rate range for a typical programmer, a junior programmer, and a senior programmer
PayRateL = 19.54
PayRateH = 54.51
PayRateJL= 19.54
```

PayRateJH = 29.93
PayRateSL = 29.94
PayRateSH = 54.51

* Range of percentages assigned to the budget choices
Budget1L = 1000
Budget1H = 9000
Budget2L = 10000
Budget2H = 49000
Budget3L = 50000
Budget3H = 99000
Budget4L = 100000
Budget4H = 499000
Budget5L = 500000
Budget5H = 999000
Budget6L = 1000000
Budget6H = 2000000
Budget7L = 2100000
Budget7H = 4000000

* Range of percentages assigned to the survey question choices
DIF1L = 0.01
DIF1H = 0.20
DIF2L = 0.21
DIF2H = 0.40
DIF3L = 0.41
DIF3H = 0.60
DIF4L = 0.61
DIF4H = 0.80
DIF5L = 0.81
DIF5H = 1.00
DIF6L = 1.01
DIF6H = 2.00
DIF5L2 = 0.81
DIF5H2 = 0.99

* KT coefficients based on complexity
L_KTL = 0.01
L_KTH = 0.33
M_KTL = 0.34
M_KTH = 0.66
H_KTL = 0.67
H_KTH = 1.00

* Assume KT impacts 10 - 100% of the training cost
Pct10 = 0.10
Pct20 = 0.20
Pct30 = 0.30
Pct40 = 0.40
Pct50 = 0.50
Pct60 = 0.60
Pct70 = 0.70
Pct80 = 0.80
Pct90 = 0.90
Pct100 = 1.00

```
*********************************************************************************
* Retrieve data from the survey file and prepare the columns
*********************************************************************************
* Retrieve complete records from the original survey data
SELECT Respondent, Pair_Exp, Lcx_Hrs, Mcx_Hrs, Hcx_Hrs, Lcx_Defect, Mcx_Defect, Hcx_Defect, kt1, kt2,
kt3, Lcx_ef, Mcx_ef, Hcx_ef, Jr2_ef, Jrsr_ef, Sr2_ef, Prior0_ef, Prior1_ef, Prior2_ef,
Lcx_df, Mcx_df, Hcx_df, Jr2_df, Jrsr_df, Sr2_df, Prior0_df, Prior1_df, Prior2_df,
Lcx_kt, Mcx_kt, Hcx_kt, Jr2_kt, Jrsr_kt, Sr2_kt, Prior0_kt, Prior1_kt, Prior2_kt,
Teamsize, Solo_fixSp, Pair_fixSp, solo_prod, Budget, Training, prjcomp1yr, prjconcurr, lcx_prj, mcx_prj, hcx_prj
FROM data
WHERE Budget <> 0
INTO table simuC

* Create C1.dbf.  C1 has the columns needed for cost calculation for solo and pair
SELECT *, 0 as PairYes, 00.00 as payRate, 00.00 as payRate2, 00.00 as payRateJ, 00.00 as payRateJ2, 00.00 as
payRateS, 00.00 as payRateS2, 00.000 as FixingS, 0000000000.00 as BudgetC,
000000.00 as TrainingC, 000 as team, 00000000.000 as l_size, 0000000.000 as m_size,
0000000.000 as h_size, 000.00 as l_Ideal, 000.00 as m_Ideal, 000.00 as h_Ideal,
000000.00 as SL_Effort, 000000.00 as SL_ECost, 000000.00 as SL_Defect, 0000000000.00 as SL_DCost,
000000.00 as SL_DR, 000000.00 as SL_OCost, 000000.00 as SL_Slack, 000000.00 as SL_SCost,
00.00 as SL_KT, 000000.00 as SL_TC10, 000000.00 as SL_TC20, 000000.00 as SL_TC30, 000000.00 as
SL_TC40, 000000.00 as SL_TC50, 000000.00 as SL_TC60, 000000.00 as SL_TC70, 000000.00 as SL_TC80,
000000.00 as SL_TC90, 000000.00 as SL_TC100,  000000.00 as SM_Effort, 000000.00 as SM_ECost, 000000.00
as SM_Defect, 0000000000.00 as SM_DCost, 000000.00 as SM_DR, 000000.00 as SM_OCost, 000000.00 as
SM_Slack, 000000.00 as SM_SCost, 00.00 as SM_KT, 000000.00 as SM_TC10, 000000.00 as SM_TC20,
000000.00 as SM_TC30, 000000.00 as SM_TC40, 000000.00 as SM_TC50, 000000.00 as SM_TC60, 000000.00 as
SM_TC70, 000000.00 as SM_TC80, 000000.00 as SM_TC90, 000000.00 as SM_TC100,  000000.00 as SH_Effort,
000000.00 as SH_ECost, 000000.00 as SH_Defect, 0000000000.00 as SH_DCost, 000000.00 as SH_DR, 000000.00
as SH_OCost, 000000.00 as SH_Slack, 000000.00 as SH_SCost, 00.00 as SH_KT, 000000.00 as SH_TC10,
000000.00 as SH_TC20, 000000.00 as SH_TC30, 000000.00 as SH_TC40, 000000.00 as SH_TC50, 000000.00 as
SH_TC60, 000000.00 as SH_TC70, 000000.00 as SH_TC80, 000000.00 as SH_TC90, 000000.00 as SH_TC100,
000000.00 as PL_Effort, 000000.00 as PL_ECost, 000000.00 as PL_Defect, 0000000000.00 as PL_DCost,
000000.00 as PL_DR, 000000.00 as PL_OCost, 000000.00 as PL_Slack, 000000.00 as PL_SCost, 00.00 as PL_KT,
000000.00 as PL_TC10, 000000.00 as PL_TC20, 000000.00 as PL_TC30, 000000.00 as PL_TC40, 000000.00 as
PL_TC50, 000000.00 as PL_TC60, 000000.00 as PL_TC70, 000000.00 as PL_TC80, 000000.00 as PL_TC90,
000000.00 as PL_TC100,  000000.00 as PM_Effort, 000000.00 as PM_ECost, 000000.00 as PM_Defect,
0000000000.00 as PM_DCost, 000000.00 as PM_DR, 000000.00 as PM_OCost, 000000.00 as PM_Slack,
000000.00 as PM_SCost, 00.00 as PM_KT, 000000.00 as PM_TC10, 000000.00 as PM_TC20, 000000.00 as
PM_TC30, 000000.00 as PM_TC40, 000000.00 as PM_TC50, 000000.00 as PM_TC60, 000000.00 as PM_TC70,
000000.00 as PM_TC80, 000000.00 as PM_TC90, 000000.00 as PM_TC100,  000000.00 as PH_Effort, 000000.00
as PH_ECost, 000000.00 as PH_Defect, 0000000000.00 as PH_DCost, 000000.00 as PH_DR, 000000.00 as
PH_OCost, 000000.00 as PH_Slack, 000000.00 as PH_SCost, 00.00 as PH_KT, 000000.00 as PH_TC10, 000000.00
as PH_TC20, 000000.00 as PH_TC30, 000000.00 as PH_TC40, 000000.00 as PH_TC50, 000000.00 as PH_TC60,
000000.00 as PH_TC70, 000000.00 as PH_TC80, 000000.00 as PH_TC90, 000000.00 as PH_TC100
FROM simuC;
into TABLE c1

SELECT c1

* Code soloPair with 0=No pair experience, 1=Has pair experience based on survey responses
Replace ALL PairYes WITH 1 FOR pair_exp <> 1

* Calculate the size of the projects in KLOC based on survey responses
Replace ALL l_size WITH ROUND((lcx_hrs * solo_prod) / 1000, 3)
Replace ALL m_size WITH ROUND((mcx_hrs * solo_prod) / 1000, 3)
Replace ALL h_size WITH ROUND((hcx_hrs * solo_prod) / 1000, 3)
```

```
* Randomly return a number from the range of pay rates
Replace ALL payRate WITH ROUND((PayRateH - PayRateL + 0.01) * RAND() + PayRateL,2)
Replace ALL payRate2 WITH ROUND((PayRateH - PayRateL + 0.01) * RAND() + PayRateL,2)
Replace ALL payRateJ WITH ROUND((PayRateJH - PayRateJL + 0.01) * RAND() + PayRateJL,2)
Replace ALL payRateJ2 WITH ROUND((PayRateJH - PayRateJL + 0.01) * RAND() + PayRateJL,2)
Replace ALL payRateS WITH ROUND((PayRateSH - PayRateSL + 0.01) * RAND() + PayRateSL,2)
Replace ALL payRateS2 WITH ROUND((PayRateSH - PayRateSL + 0.01) * RAND() + PayRateSL,2)

* Calculate the training cost for the project based on survey responses
Replace ALL BudgetC WITH INT((Budget1H - Budget1L + 1) * RAND() + Budget1L) FOR budget = 1
Replace ALL BudgetC WITH INT((Budget2H - Budget2L + 1) * RAND() + Budget2L)  FOR budget = 2
Replace ALL BudgetC WITH INT((Budget3H - Budget3L + 1) * RAND() + Budget3L)  FOR budget = 3
Replace ALL BudgetC WITH INT((Budget4H - Budget4L + 1) * RAND() + Budget4L)  FOR budget = 4
Replace ALL BudgetC WITH INT((Budget5H - Budget5L + 1) * RAND() + Budget5L)  FOR budget = 5
Replace ALL BudgetC WITH INT((Budget6H - Budget6L + 1) * RAND() + Budget6L)  FOR budget = 6
Replace ALL BudgetC WITH INT((Budget7H - Budget7L + 1) * RAND() + Budget7L)  FOR budget = 7
Replace ALL TrainingC WITH (Training/100) * BudgetC

* Ensure team size is an even number
Replace ALL Team WITH teamsize FOR MOD(teamsize, 2) = 0
Replace ALL Team WITH teamsize + 1 FOR MOD(teamsize, 2) <> 0

* Calculate KT coefficients based on complexity
Replace ALL SL_KT WITH ROUND((L_KTH - L_KTL + 0.01) * RAND() + L_KTL,2)
Replace ALL SM_KT WITH ROUND((M_KTH - M_KTL + 0.01) * RAND() + M_KTL,2)
Replace ALL SH_KT WITH ROUND((H_KTH - H_KTL + 0.01) * RAND() + H_KTL,2)

* Average duration for low, medium, and high complexity projects calculated from 5000 bootstraps on the duration
columns.
Replace ALL L_Ideal WITH 20
Replace ALL M_Ideal WITH 93
Replace ALL H_Ideal WITH 1010

*********************************************************************************
*   Generic solo and generic pair in low, medium, and high complexity projects
*********************************************************************************

*********************************************************************************
** SOLO LOW complexity projects
*********************************************************************************
* Effort based on survey responses
Replace ALL SL_Effort WITH lcx_hrs
Replace ALL SL_ECost WITH SL_Effort * payRate

* Defect based on survey responses
Replace ALL SL_Defect WITH lcx_Defect
Replace ALL SL_DCost WITH SL_Defect * l_size * FixingS * payRate
Replace ALL lcx_defect WITH sl_defect

* Duration
Replace ALL SL_DR WITH SL_Effort/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL SL_Slack WITH SL_DR - L_Ideal
Replace ALL SL_SCost WITH -1 * SL_Slack * HrsPerDay8 * payRate * Team FOR SL_Slack <=0
```

```
Replace ALL SL_OCost WITH SL_Slack * HrsPerDay8 * payRate * Team FOR SL_Slack > 0

* Knowledge Transfer
* Assume knowledge transfer impacts different percentage of training cost: 10%, 20%, -, 100%
Replace ALL SL_TC10 WITH (TrainingCL - Pct10 * TrainingCL * SL_KT)
Replace ALL SL_TC20 WITH (TrainingCL - Pct20 * TrainingCL * SL_KT)
Replace ALL SL_TC30 WITH (TrainingCL - Pct30 * TrainingCL * SL_KT)
Replace ALL SL_TC40 WITH (TrainingCL - Pct40 * TrainingCL * SL_KT)
Replace ALL SL_TC50 WITH (TrainingCL - Pct50 * TrainingCL * SL_KT)
Replace ALL SL_TC60 WITH (TrainingCL - Pct60 * TrainingCL * SL_KT)
Replace ALL SL_TC70 WITH (TrainingCL - Pct70 * TrainingCL * SL_KT)
Replace ALL SL_TC80 WITH (TrainingCL - Pct80 * TrainingCL * SL_KT)
Replace ALL SL_TC90 WITH (TrainingCL - Pct90 * TrainingCL * SL_KT)
Replace ALL SL_TC100 WITH (TrainingCL - Pct100 * TrainingCL * SL_KT)


*******************************************************************************
** SOLO MEDIUM complexity projects
*******************************************************************************
* Effort based on survey responses
Replace ALL SM_Effort WITH mcx_hrs
Replace ALL SM_ECost WITH SM_Effort * payRate

* Defect based on survey responses
Replace ALL SM_Defect WITH mcx_Defect
Replace ALL SM_DCost WITH SM_Defect * m_size  * FixingS * payRate
Replace ALL mcx_defect WITH sm_defect

* Duration
Replace ALL SM_DR WITH SM_Effort/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL SM_Slack WITH SM_DR - M_Ideal
Replace ALL SM_SCost WITH -1 * SM_Slack * HrsPerDay8 * payRate * Team FOR SM_Slack <=0
Replace ALL SM_OCost WITH SM_Slack * HrsPerDay8 * payRate * Team FOR SM_Slack > 0

* Knowledge Transfer
Replace ALL SM_TC10 WITH (TrainingCM - Pct10 * TrainingCM * SM_KT)
Replace ALL SM_TC20 WITH (TrainingCM - Pct20 * TrainingCM * SM_KT)
Replace ALL SM_TC30 WITH (TrainingCM - Pct30 * TrainingCM * SM_KT)
Replace ALL SM_TC40 WITH (TrainingCM - Pct40 * TrainingCM * SM_KT)
Replace ALL SM_TC50 WITH (TrainingCM - Pct50 * TrainingCM * SM_KT)
Replace ALL SM_TC60 WITH (TrainingCM - Pct60 * TrainingCM * SM_KT)
Replace ALL SM_TC70 WITH (TrainingCM - Pct70 * TrainingCM * SM_KT)
Replace ALL SM_TC80 WITH (TrainingCM - Pct80 * TrainingCM * SM_KT)
Replace ALL SM_TC90 WITH (TrainingCM - Pct90 * TrainingCM * SM_KT)
Replace ALL SM_TC100 WITH (TrainingCM - Pct100 * TrainingCM * SM_KT)


*******************************************************************************
** SOLO HIGH complexity projects
*******************************************************************************
* Effort based on survey responses
Replace ALL SH_Effort WITH hcx_hrs
Replace ALL SH_ECost WITH SH_Effort * payRate
```

```
* Defect based on survey responses
Replace ALL SH_Defect WITH hcx_Defect
Replace ALL SH_DCost WITH SH_Defect * h_size  * FixingS * payRate
Replace ALL hcx_defect WITH sh_defect

* Duration
Replace ALL SH_DR WITH SH_Effort/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL SH_Slack WITH SH_DR - H_Ideal
Replace ALL SH_SCost WITH -1 * SH_Slack * HrsPerDay8 * payRate * Team FOR SH_Slack <=0
Replace ALL SH_OCost WITH SH_Slack * HrsPerDay8 * payRate * Team FOR SH_Slack > 0

* Knowledge Transfer
Replace ALL SH_TC10 WITH (TrainingCH - Pct10 * TrainingCH * SH_KT)
Replace ALL SH_TC20 WITH (TrainingCH - Pct20 * TrainingCH * SH_KT)
Replace ALL SH_TC30 WITH (TrainingCH - Pct30 * TrainingCH * SH_KT)
Replace ALL SH_TC40 WITH (TrainingCH - Pct40 * TrainingCH * SH_KT)
Replace ALL SH_TC50 WITH (TrainingCH - Pct50 * TrainingCH * SH_KT)
Replace ALL SH_TC60 WITH (TrainingCH - Pct60 * TrainingCH * SH_KT)
Replace ALL SH_TC70 WITH (TrainingCH - Pct70 * TrainingCH * SH_KT)
Replace ALL SH_TC80 WITH (TrainingCH - Pct80 * TrainingCH * SH_KT)
Replace ALL SH_TC90 WITH (TrainingCH - Pct90 * TrainingCH * SH_KT)
Replace ALL SH_TC100 WITH (TrainingCH - Pct100 * TrainingCH * SH_KT)

*****************************************************************************
** PAIR LOW complexity projects
*****************************************************************************
* Effort based on survey responses
Replace all PL_Effort WITH lcx_hrs for lcx_ef = 0
Replace ALL PL_Effort WITH lcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for lcx_ef
= 1
Replace ALL PL_Effort WITH lcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))   for
lcx_ef = 2
Replace ALL PL_Effort WITH lcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))   for
lcx_ef = 3
Replace ALL PL_Effort WITH lcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))  for lcx_ef
= 4
Replace ALL PL_Effort WITH lcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2))   for
lcx_ef = 5
Replace ALL PL_Effort WITH lcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2))   for
lcx_ef = 6
Replace ALL PL_Effort WITH lcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
lcx_ef = -1
Replace ALL PL_Effort WITH lcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for lcx_ef
= -2
Replace ALL PL_Effort WITH lcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))  for
lcx_ef = -3
Replace ALL PL_Effort WITH lcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))  for
lcx_ef = -4
Replace ALL PL_Effort WITH lcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  for
lcx_ef = -5

Replace ALL PL_ECost WITH PL_Effort * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
```

```
Replace ALL PL_Defect WITH lcx_Defect FOR lcx_df = 0
Replace ALL PL_Defect WITH lcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
lcx_df = 1
Replace ALL PL_Defect WITH lcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))  FOR
lcx_df = 2
Replace ALL PL_Defect WITH lcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))  FOR
lcx_df = 3
Replace ALL PL_Defect WITH lcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))  FOR
lcx_df = 4
Replace ALL PL_Defect WITH lcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2))  FOR
lcx_df = 5
Replace ALL PL_Defect WITH lcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2))  FOR
lcx_df = 6
Replace ALL PL_Defect WITH lcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
lcx_df = -1
Replace ALL PL_Defect WITH lcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
lcx_df = -2
Replace ALL PL_Defect WITH lcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
lcx_df = -3
Replace ALL PL_Defect WITH lcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
lcx_df = -4
Replace ALL PL_Defect WITH lcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR lcx_df = -5

* Defect cost
Replace ALL PL_DCost WITH PL_Defect *  l_size  * FixingS * payRate

* Duration
Replace ALL PL_DR WITH PL_Effort/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PL_Slack WITH PL_DR - L_Ideal
Replace ALL PL_SCost WITH -1 * PL_Slack *  HrsPerDay8 * Team * ROUND((payRate + payRate2)/2, 2) FOR
PL_Slack <=0
Replace ALL PL_OCost WITH PL_Slack * HrsPerDay8 * Team * ROUND((payRate + payRate2)/2, 2) FOR
PL_Slack > 0

* Knowledge Transfer based on survey responses
Replace ALL PL_KT WITH SL_KT  FOR lcx_kt = 0
Replace ALL PL_KT WITH SL_KT * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR lcx_kt
= 1
Replace ALL PL_KT WITH SL_KT * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))  FOR lcx_kt
= 2
Replace ALL PL_KT WITH SL_KT * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR lcx_kt
= 3
Replace ALL PL_KT WITH SL_KT * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR lcx_kt
= 4
Replace ALL PL_KT WITH SL_KT * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR lcx_kt
= 5
Replace ALL PL_KT WITH SL_KT * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR lcx_kt
= 6
Replace ALL PL_KT WITH  SL_KT * (1 - ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR lcx_kt
= -1
Replace ALL PL_KT WITH SL_KT * (1 - ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR lcx_kt
= -2
```

```
Replace ALL PL_KT WITH SL_KT * (1 - ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR lcx_kt
= -3
Replace ALL PL_KT WITH SL_KT * (1 - ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR lcx_kt
= -4
Replace ALL PL_KT WITH SL_KT * (1 - ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  FOR
lcx_kt = -5

* Convert KT into training cost
Replace ALL PL_TC10 WITH (TrainingCL - Pct10 * TrainingCL * PL_KT)
Replace ALL PL_TC20 WITH (TrainingCL - Pct20 * TrainingCL * PL_KT)
Replace ALL PL_TC30 WITH (TrainingCL - Pct30 * TrainingCL * PL_KT)
Replace ALL PL_TC40 WITH (TrainingCL - Pct40 * TrainingCL * PL_KT)
Replace ALL PL_TC50 WITH (TrainingCL - Pct50 * TrainingCL * PL_KT)
Replace ALL PL_TC60 WITH (TrainingCL - Pct60 * TrainingCL * PL_KT)
Replace ALL PL_TC70 WITH (TrainingCL - Pct70 * TrainingCL * PL_KT)
Replace ALL PL_TC80 WITH (TrainingCL - Pct80 * TrainingCL * PL_KT)
Replace ALL PL_TC90 WITH (TrainingCL - Pct90 * TrainingCL * PL_KT)
Replace ALL PL_TC100 WITH (TrainingCL - Pct100 * TrainingCL * PL_KT)

*************************************************************************************
** PAIR MEDIUM complexity projects
*************************************************************************************
* Effort based on survey responses
Replace ALL PM_Effort WITH mcx_hrs for mcx_ef = 0
Replace ALL PM_Effort WITH mcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
mcx_ef = 1
Replace ALL PM_Effort WITH mcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))  for
mcx_ef = 2
Replace ALL PM_Effort WITH mcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
mcx_ef = 3
Replace ALL PM_Effort WITH mcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
mcx_ef = 4
Replace ALL PM_Effort WITH mcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2))  for
mcx_ef = 5
Replace ALL PM_Effort WITH mcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
mcx_ef = 6
Replace ALL PM_Effort WITH mcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
mcx_ef = -1
Replace ALL PM_Effort WITH mcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
mcx_ef = -2
Replace ALL PM_Effort WITH mcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))  for
mcx_ef = -3
Replace ALL PM_Effort WITH mcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))  for
mcx_ef = -4
Replace ALL PM_Effort WITH mcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for
mcx_ef = -5

Replace ALL PM_ECost WITH PM_Effort * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
Replace ALL PM_Defect WITH mcx_Defect FOR mcx_df = 0
Replace ALL PM_Defect WITH mcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR mcx_df = 1
Replace ALL PM_Defect WITH mcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR mcx_df = 2
```

```
Replace ALL PM_Defect WITH mcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR mcx_df = 3
Replace ALL PM_Defect WITH mcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR mcx_df = 4
Replace ALL PM_Defect WITH mcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2))
FOR mcx_df = 5
Replace ALL PM_Defect WITH mcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2))
FOR mcx_df = 6
Replace ALL PM_Defect WITH mcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR mcx_df = -1
Replace ALL PM_Defect WITH mcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR mcx_df = -2
Replace ALL PM_Defect WITH mcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR mcx_df = -3
Replace ALL PM_Defect WITH mcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR mcx_df = -4
Replace ALL PM_Defect WITH mcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR mcx_df = -5

* Defect cost
Replace ALL PM_DCost WITH PM_Defect * m_size * FixingS* payRate

* Duration
Replace ALL PM_DR WITH PM_Effort/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PM_Slack WITH PM_DR - M_Ideal
Replace ALL PM_SCost WITH -1 * PM_Slack * HrsPerDay8 *Team * ROUND((payRate + payRate2)/2, 2) FOR
PM_Slack <=0
Replace ALL PM_OCost WITH PM_Slack * HrsPerDay8 * Team * ROUND((payRate + payRate2)/2, 2) FOR
PM_Slack > 0

* Knowledge Transfer based on survey responses
Replace ALL PM_KT WITH SM_KT  FOR mcx_kt = 0
Replace ALL PM_KT WITH SM_KT * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
mcx_kt = 1
Replace ALL PM_KT WITH SM_KT * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))  FOR
mcx_kt = 2
Replace ALL PM_KT WITH SM_KT * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
mcx_kt = 3
Replace ALL PM_KT WITH SM_KT * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
mcx_kt = 4
Replace ALL PM_KT WITH SM_KT * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
mcx_kt = 5
Replace ALL PM_KT WITH SM_KT * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
mcx_kt = 6
Replace ALL PM_KT WITH SM_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
mcx_kt = -1
Replace ALL PM_KT WITH SM_KT * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
mcx_kt = -2
Replace ALL PM_KT WITH SM_KT * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
mcx_kt = -3
Replace ALL PM_KT WITH SM_KT * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
mcx_kt = -4
Replace ALL PM_KT WITH SM_KT * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) FOR
mcx_kt = -5
```

```
* Convert KT into training cost
Replace ALL PM_TC10 WITH (TrainingCM - Pct10 * TrainingCM * PM_KT)
Replace ALL PM_TC20 WITH (TrainingCM - Pct20 * TrainingCM * PM_KT)
Replace ALL PM_TC30 WITH (TrainingCM - Pct30 * TrainingCM * PM_KT)
Replace ALL PM_TC40 WITH (TrainingCM - Pct40 * TrainingCM * PM_KT)
Replace ALL PM_TC50 WITH (TrainingCM - Pct50 * TrainingCM * PM_KT)
Replace ALL PM_TC60 WITH (TrainingCM - Pct60 * TrainingCM * PM_KT)
Replace ALL PM_TC70 WITH (TrainingCM - Pct70 * TrainingCM * PM_KT)
Replace ALL PM_TC80 WITH (TrainingCM - Pct80 * TrainingCM * PM_KT)
Replace ALL PM_TC90 WITH (TrainingCM - Pct90 * TrainingCM * PM_KT)
Replace ALL PM_TC100 WITH (TrainingCM - Pct100 * TrainingCM * PM_KT)

*********************************************************************************
** PAIR HIGH complexity projects
*********************************************************************************
* Effort based on survey responses
Replace ALL PH_Effort WITH hcx_hrs for hcx_ef = 0
Replace ALL PH_Effort WITH hcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
hcx_ef = 1
Replace ALL PH_Effort WITH hcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
hcx_ef = 2
Replace ALL PH_Effort WITH hcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
hcx_ef = 3
Replace ALL PH_Effort WITH hcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))  for
hcx_ef = 4
Replace ALL PH_Effort WITH hcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
hcx_ef = 5
Replace ALL PH_Effort WITH hcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2))  for
hcx_ef = 6
Replace ALL PH_Effort WITH hcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
hcx_ef = -1
Replace ALL PH_Effort WITH hcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
hcx_ef = -2
Replace ALL PH_Effort WITH hcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))  for
hcx_ef = -3
Replace ALL PH_Effort WITH hcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
hcx_ef = -4
Replace ALL PH_Effort WITH hcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for
hcx_ef = -5

Replace ALL PH_ECost WITH PH_Effort * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
Replace ALL PH_Defect WITH hcx_Defect FOR hcx_df = 0
Replace ALL PH_Defect WITH hcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
hcx_df = 1
Replace ALL PH_Defect WITH hcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
hcx_df = 2
Replace ALL PH_Defect WITH hcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
hcx_df = 3
Replace ALL PH_Defect WITH hcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
hcx_df = 4
Replace ALL PH_Defect WITH hcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
hcx_df = 5
```

```
Replace ALL PH_Defect WITH hcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
hcx_df = 6
Replace ALL PH_Defect WITH hcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
hcx_df = -1
Replace ALL PH_Defect WITH hcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
hcx_df = -2
Replace ALL PH_Defect WITH hcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
hcx_df = -3
Replace ALL PH_Defect WITH hcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
hcx_df = -4
Replace ALL PH_Defect WITH hcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR hcx_df = -5
* Defect cost
Replace ALL PH_DCost WITH PH_Defect * h_size * FixingS * payRate

* Duration
Replace ALL PH_DR WITH PH_Effort/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PH_Slack WITH PH_DR - H_Ideal
Replace ALL PH_SCost WITH -1 * PH_Slack *  HrsPerDay8 * Team * ROUND((payRate + payRate2)/2, 2) FOR
PH_Slack <=0
Replace ALL PH_OCost WITH PH_Slack * HrsPerDay8 * Team * ROUND((payRate + payRate2)/2, 2) FOR
PH_Slack > 0

* Knowledge Transfer based on survey responses
Replace ALL PH_KT WITH SH_KT  FOR hcx_kt = 0
Replace ALL PH_KT WITH SH_KT * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR hcx_kt
= 1
Replace ALL PH_KT WITH SH_KT * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR hcx_kt
= 2
Replace ALL PH_KT WITH SH_KT * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR hcx_kt
= 3
Replace ALL PH_KT WITH SH_KT * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR hcx_kt
= 4
Replace ALL PH_KT WITH SH_KT * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR hcx_kt
= 5
Replace ALL PH_KT WITH SH_KT * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR hcx_kt
= 6
Replace ALL PH_KT WITH SH_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR hcx_kt
= -1
Replace ALL PH_KT WITH SH_KT * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR hcx_kt
= -2
Replace ALL PH_KT WITH SH_KT * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR hcx_kt
= -3
Replace ALL PH_KT WITH SH_KT * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR hcx_kt
= -4
Replace ALL PH_KT WITH SH_KT * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) FOR
hcx_kt = -5

* Convert KT into training cost
Replace ALL PH_TC10 WITH (TrainingCH - Pct10 * TrainingCH * PH_KT)
Replace ALL PH_TC20 WITH (TrainingCH - Pct20 * TrainingCH * PH_KT)
Replace ALL PH_TC30 WITH (TrainingCH - Pct30 * TrainingCH * PH_KT)
Replace ALL PH_TC40 WITH (TrainingCH - Pct40 * TrainingCH * PH_KT)
Replace ALL PH_TC50 WITH (TrainingCH - Pct50 * TrainingCH * PH_KT)
```

```
Replace ALL PH_TC60 WITH (TrainingCH - Pct60 * TrainingCH * PH_KT)
Replace ALL PH_TC70 WITH (TrainingCH - Pct70 * TrainingCH * PH_KT)
Replace ALL PH_TC80 WITH (TrainingCH - Pct80 * TrainingCH * PH_KT)
Replace ALL PH_TC90 WITH (TrainingCH - Pct90 * TrainingCH * PH_KT)
Replace ALL PH_TC100 WITH (TrainingCH - Pct100 * TrainingCH * PH_KT)

********************************************************************************************** PAIR
COMPOSITION - JUNIOR-JUNIOR, JUNIOR-SENIOR, SENIOR-SENIOR
**********************************************************************************************
* Create C2 since the maximum number of columns a table can have is 256.  C2 has the columns needed for cost
calculation for junior-junior, junior-senior, and senior-senior

SELECT respondent, trainingC, trainingCL, trainingCM, trainingCH, team, l_team, m_team, h_team, comp1yr,
lcx_hrs, mcx_hrs, hcx_hrs, jr2_ef, jr2_df, jr2_kt, jrsr_ef, jrsr_df, jrsr_kt, sr2_ef, sr2_df, sr2_kt, lcx_defect,
mcx_defect, hcx_defect, payRate, payRate2, payRateJ, payRateJ2, payRateS, payRateS2, FixingS, l_size, m_size,
h_size, L_Ideal, M_Ideal, H_Ideal, sl_kt, sm_kt, sh_kt, 000000.00 as PL_JJEf, 000000.00 as PL_JJECo, 000000.00
as PL_JJDef, 0000000000.00 as PL_JJDCo, 000000.00 as PL_JJDR, 000000.00 as PL_JJOCo, 000000.00 as
PL_JJSlack, 000000.00 as PL_JJSCo, 00.00 as PL_JJKT, 000000.00 as PL_JJTC10, 000000.00 as PL_JJTC20,
000000.00 as PL_JJTC30, 000000.00 as PL_JJTC40, 000000.00 as PL_JJTC50, 000000.00 as PL_JJTC60,
000000.00 as PL_JJTC70, 000000.00 as PL_JJTC80, 000000.00 as PL_JJTC90, 000000.00 as PL_JJTC100,
000000.00 as PM_JJEf, 000000.00 as PM_JJECo, ;
000000.00 as PM_JJDef, 0000000000.00 as PM_JJDCo, 000000.00 as PM_JJDR, 000000.00 as PM_JJOCo,
000000.00 as PM_JJSlack, 000000.00 as PM_JJSCo, 00.00 as PM_JJKT, 000000.00 as PM_JJTC10, 000000.00 as
PM_JJTC20, 000000.00 as PM_JJTC30, 000000.00 as PM_JJTC40, 000000.00 as PM_JJTC50, 000000.00 as
PM_JJTC60, 000000.00 as PM_JJTC70, 000000.00 as PM_JJTC80, 000000.00 as PM_JJTC90, 000000.00 as
PM_JJTC100, 000000.00 as PH_JJEf, 000000.00 as PH_JJECo, ;
000000.00 as PH_JJDef, 0000000000.00 as PH_JJDCo, 000000.00 as PH_JJDR, 000000.00 as PH_JJOCo,
000000.00 as PH_JJSlack, 000000.00 as PH_JJSCo, 00.00 as PH_JJKT, 000000.00 as PH_JJTC10, 000000.00 as
PH_JJTC20, 000000.00 as PH_JJTC30, 000000.00 as PH_JJTC40, 000000.00 as PH_JJTC50, 000000.00 as
PH_JJTC60, 000000.00 as PH_JJTC70, 000000.00 as PH_JJTC80, 000000.00 as PH_JJTC90, 000000.00 as
PH_JJTC100,  000000.00 as PL_JSEf, 000000.00 as PL_JSECo, 000000.00 as PL_JSDef, 0000000000.00 as
PL_JSDCo, 000000.00 as PL_JSDR, 000000.00 as PL_JSOCo, 000000.00 as PL_JSSlack, 000000.00 as PL_JSSCo,
00.00 as PL_JSKT,  000000.00 as PL_JSTC10, 000000.00 as PL_JSTC20, 000000.00 as PL_JSTC30, 000000.00 as
PL_JSTC40, 000000.00 as PL_JSTC50, 000000.00 as PL_JSTC60, 000000.00 as PL_JSTC70, 000000.00 as
PL_JSTC80, 000000.00 as PL_JSTC90, 000000.00 as PL_JSTC100, 000000.00 as PM_JSEf, 000000.00 as
PM_JSECo, 000000.00 as PM_JSDef, 0000000000.00 as PM_JSDCo, 000000.00 as PM_JSDR, 000000.00 as
PM_JSOCo, 000000.00 as PM_JSSlack, 000000.00 as PM_JSSCo, 00.00 as PM_JSKT, 000000.00 as PM_JSTC10,
000000.00 as PM_JSTC20, 000000.00 as PM_JSTC30, 000000.00 as PM_JSTC40, 000000.00 as PM_JSTC50,
000000.00 as PM_JSTC60, 000000.00 as PM_JSTC70, 000000.00 as PM_JSTC80, 000000.00 as PM_JSTC90,
000000.00 as PM_JSTC100,  000000.00 as PH_JSEf, 000000.00 as PH_JSECo,
000000.00 as PH_JSDef, 0000000000.00 as PH_JSDCo, 000000.00 as PH_JSDR, 000000.00 as PH_JSOCo,
000000.00 as PH_JSSlack, 000000.00 as PH_JSSCo, 00.00 as PH_JSKT, 000000.00 as PH_JSTC10, 000000.00 as
PH_JSTC20, 000000.00 as PH_JSTC30, 000000.00 as PH_JSTC40, 000000.00 as PH_JSTC50, 000000.00 as
PH_JSTC60, 000000.00 as PH_JSTC70, 000000.00 as PH_JSTC80, 000000.00 as PH_JSTC90, 000000.00 as
PH_JSTC100,  000000.00 as PL_SSEf, 000000.00 as PL_SSECo, 000000.00 as PL_SSDef, 0000000000.00 as
PL_SSDCo, 000000.00 as PL_SSDR, 000000.00 as PL_SSOCo, 000000.00 as PL_SSSlack, 000000.00 as
PL_SSSCo, 00.00 as PL_SSKT, 000000.00 as PL_SSTC10, 000000.00 as PL_SSTC20, 000000.00 as PL_SSTC30,
000000.00 as PL_SSTC40, 000000.00 as PL_SSTC50, 000000.00 as PL_SSTC60, 000000.00 as PL_SSTC70,
000000.00 as PL_SSTC80, 000000.00 as PL_SSTC90, 000000.00 as PL_SSTC100,  000000.00 as PM_SSEf,
000000.00 as PM_SSECo, 000000.00 as PM_SSDef, 0000000000.00 as PM_SSDCo, 000000.00 as PM_SSDR,
000000.00 as PM_SSOCo, 000000.00 as PM_SSSlack,000000.00 as PM_SSSCo, 00.00 as PM_SSKT, 000000.00
as PM_SSTC10, 000000.00 as PM_SSTC20, 000000.00 as PM_SSTC30, 000000.00 as PM_SSTC40, 000000.00 as
PM_SSTC50, 000000.00 as PM_SSTC60, 000000.00 as PM_SSTC70, 000000.00 as PM_SSTC80, 000000.00 as
PM_SSTC90, 000000.00 as PM_SSTC100,  000000.00 as PH_SSEf, 000000.00 as PH_SSECo, 000000.00 as
PH_SSDef, 0000000000.00 as PH_SSDCo, ;
000000.00 as PH_SSDR, 000000.00 as PH_SSOCo, 000000.00 as PH_SSSlack,000000.00 as PH_SSSCo, ;
```

```
00.00 as PH_SSKT, 000000.00 as PH_SSTC10, 000000.00 as PH_SSTC20, 000000.00 as PH_SSTC30, 000000.00
as PH_SSTC40, 000000.00 as PH_SSTC50, 000000.00 as PH_SSTC60, 000000.00 as PH_SSTC70, 000000.00 as
PH_SSTC80, 000000.00 as PH_SSTC90, 000000.00 as PH_SSTC100
FROM c1
into TABLE c2

SELECT c2

*********************************************************************************************
** PAIR LOW complexity projects and EXPERTISE COMPOSITION (JUNIOR-JUNIOR)
*********************************************************************************************
* Effort based on survey responses
Replace ALL PL_JJEf WITH lcx_hrs for Jr2_ef = 0
Replace ALL PL_JJEf WITH lcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Jr2_ef =
1
Replace ALL PL_JJEf WITH lcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Jr2_ef =
2
Replace ALL PL_JJEf WITH lcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Jr2_ef =
3
Replace ALL PL_JJEf WITH lcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Jr2_ef =
4
Replace ALL PL_JJEf WITH lcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for Jr2_ef =
5
Replace ALL PL_JJEf WITH lcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for Jr2_ef =
6
Replace ALL PL_JJEf WITH lcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Jr2_ef =
-1
Replace ALL PL_JJEf WITH lcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Jr2_ef =
-2
Replace ALL PL_JJEf WITH lcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Jr2_ef =
-3
Replace ALL PL_JJEf WITH lcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Jr2_ef =
-4
Replace ALL PL_JJEf WITH lcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for
Jr2_ef = -5

Replace ALL PL_JJECo WITH PL_JJEf * ROUND((payRateJ + payRateJ2)/2, 2)

* Defect based on survey responses
Replace ALL PL_JJDef WITH lcx_Defect FOR Jr2_df = 0
Replace ALL PL_JJDef WITH lcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jr2_df = 1
Replace ALL PL_JJDef WITH lcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jr2_df = 2
Replace ALL PL_JJDef WITH lcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jr2_df = 3
Replace ALL PL_JJDef WITH lcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jr2_df = 4
Replace ALL PL_JJDef WITH lcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Jr2_df = 5
Replace ALL PL_JJDef WITH lcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Jr2_df = 6
Replace ALL PL_JJDef WITH lcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jr2_df = -1
Replace ALL PL_JJDef WITH lcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jr2_df = -2
```

```
Replace ALL PL_JJDef WITH lcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jr2_df = -3
Replace ALL PL_JJDef WITH lcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jr2_df = -4
Replace ALL PL_JJDef WITH lcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Jr2_df = -5

Replace ALL PL_JJDCo WITH (PL_JJDef * l_size * FixingS) * payRate

* Duration
Replace ALL PL_JJDR WITH PL_JJEf/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PL_JJSlack WITH PL_JJDR - L_Ideal
Replace ALL PL_JJSCo WITH -1 * PL_JJSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateJ2)/2, 2)
FOR PL_JJSlack <=0
Replace ALL PL_JJOCo WITH PL_JJSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateJ2)/2, 2) FOR
PL_JJSlack > 0

* Knowledge Transfer based on survey responses
Replace ALL PL_JJKT WITH SL_KT FOR Jr2_kt = 0
Replace ALL PL_JJKT WITH SL_KT * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jr2_kt = 1
Replace ALL PL_JJKT WITH SL_KT * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jr2_kt = 2
Replace ALL PL_JJKT WITH SL_KT * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jr2_kt = 3
Replace ALL PL_JJKT WITH SL_KT * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jr2_kt = 4
Replace ALL PL_JJKT WITH SL_KT * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Jr2_kt = 5
Replace ALL PL_JJKT WITH  SL_KT * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Jr2_kt = 6
Replace ALL PL_JJKT WITH SL_KT * (1  - ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jr2_kt = -1
Replace ALL PL_JJKT WITH SL_KT * (1  - ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jr2_kt = -2
Replace ALL PL_JJKT WITH SL_KT * (1  - ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jr2_kt = -3
Replace ALL PL_JJKT WITH SL_KT * (1  - ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jr2_kt = -4
Replace ALL PL_JJKT WITH SL_KT * (1  - ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) FOR
Jr2_kt = -5

* Convert KT into training cost
Replace ALL PL_JJTC10 WITH (TrainingCL - Pct10 * TrainingCL * PL_JJKT)
Replace ALL PL_JJTC20 WITH (TrainingCL - Pct20 * TrainingCL * PL_JJKT)
Replace ALL PL_JJTC30 WITH (TrainingCL - Pct30 * TrainingCL * PL_JJKT)
Replace ALL PL_JJTC40 WITH (TrainingCL - Pct40 * TrainingCL * PL_JJKT)
Replace ALL PL_JJTC50 WITH (TrainingCL - Pct50 * TrainingCL * PL_JJKT)
Replace ALL PL_JJTC60 WITH (TrainingCL - Pct60 * TrainingCL * PL_JJKT)
Replace ALL PL_JJTC70 WITH (TrainingCL - Pct70 * TrainingCL * PL_JJKT)
Replace ALL PL_JJTC80 WITH (TrainingCL - Pct80 * TrainingCL * PL_JJKT)
Replace ALL PL_JJTC90 WITH (TrainingCL - Pct90 * TrainingCL * PL_JJKT)
Replace ALL PL_JJTC100 WITH (TrainingCL - Pct100 * TrainingCL * PL_JJKT)
```

```
********************************************************************************************** PAIR MEDIUM complexity projects and EXPERTISE COMPOSITION (JUNIOR-JUNIOR)
********************************************************************************************** Effort based on survey responses
Replace ALL PM_JJEf WITH mcx_hrs for Jr2_ef = 0
Replace ALL PM_JJEf WITH mcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Jr2_ef = 1
Replace ALL PM_JJEf WITH mcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Jr2_ef = 2
Replace ALL PM_JJEf WITH mcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Jr2_ef = 3
Replace ALL PM_JJEf WITH mcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Jr2_ef = 4
Replace ALL PM_JJEf WITH mcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for Jr2_ef = 5
Replace ALL PM_JJEf WITH mcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for Jr2_ef = 6
Replace ALL PM_JJEf WITH mcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Jr2_ef = -1
Replace ALL PM_JJEf WITH mcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Jr2_ef = -2
Replace ALL PM_JJEf WITH mcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Jr2_ef = -3
Replace ALL PM_JJEf WITH mcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Jr2_ef = -4
Replace ALL PM_JJEf WITH mcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for Jr2_ef = -5

Replace ALL PM_JJECo WITH PM_JJEf * ROUND((payRateJ + payRateJ2)/2, 2)

* Defect based on survey responses
Replace ALL PM_JJDef WITH mcx_Defect FOR Jr2_df = 0
Replace ALL PM_JJDef WITH mcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR Jr2_df = 1
Replace ALL PM_JJDef WITH mcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR Jr2_df = 2
Replace ALL PM_JJDef WITH mcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR Jr2_df = 3
Replace ALL PM_JJDef WITH mcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR Jr2_df = 4
Replace ALL PM_JJDef WITH mcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR Jr2_df = 5
Replace ALL PM_JJDef WITH mcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR Jr2_df = 6
Replace ALL PM_JJDef WITH mcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR Jr2_df = -1
Replace ALL PM_JJDef WITH mcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR Jr2_df = -2
Replace ALL PM_JJDef WITH mcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR Jr2_df = -3
Replace ALL PM_JJDef WITH mcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR Jr2_df = -4
Replace ALL PM_JJDef WITH mcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) FOR Jr2_df = -5

Replace ALL PM_JJDCo WITH (PM_JJDef * m_size * FixingS) * payRate
```

```
* Duration
Replace ALL PM_JJDR WITH PM_JJEf / (Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PM_JJSlack WITH PM_JJDR - M_Ideal
Replace ALL PM_JJSCo WITH -1 * PM_JJSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateJ2)/2, 2)
FOR PM_JJSlack <=0
Replace ALL PM_JJOCo WITH PM_JJSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateJ2)/2, 2) FOR
PM_JJSlack > 0

* Knowledge Transfer based on survey responses
Replace ALL PM_JJKT WITH SM_KT FOR Jr2_kt = 0
Replace ALL PM_JJKT WITH SM_KT * (1  + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jr2_kt = 1
Replace ALL PM_JJKT WITH SM_KT * (1  + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jr2_kt = 2
Replace ALL PM_JJKT WITH SM_KT * (1  + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jr2_kt = 3
Replace ALL PM_JJKT WITH SM_KT * (1  + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jr2_kt = 4
Replace ALL PM_JJKT WITH SM_KT * (1  + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Jr2_kt = 5
Replace ALL PM_JJKT WITH SM_KT * (1  + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Jr2_kt = 6
Replace ALL PM_JJKT WITH SM_KT * (1  - ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jr2_kt = -1
Replace ALL PM_JJKT WITH SM_KT * (1  - ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jr2_kt = -2
Replace ALL PM_JJKT WITH SM_KT * (1  - ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jr2_kt = -3
Replace ALL PM_JJKT WITH SM_KT * (1  - ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jr2_kt = -4
Replace ALL PM_JJKT WITH SM_KT * (1  - ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) FOR
Jr2_kt = -5

* Convert KT into training cost
Replace ALL PM_JJTC10 WITH (TrainingCM - Pct10 * TrainingCM * PM_JJKT)
Replace ALL PM_JJTC20 WITH (TrainingCM - Pct20 * TrainingCM * PM_JJKT)
Replace ALL PM_JJTC30 WITH (TrainingCM - Pct30 * TrainingCM * PM_JJKT)
Replace ALL PM_JJTC40 WITH (TrainingCM - Pct40 * TrainingCM * PM_JJKT)
Replace ALL PM_JJTC50 WITH (TrainingCM - Pct50 * TrainingCM * PM_JJKT)
Replace ALL PM_JJTC60 WITH (TrainingCM - Pct60 * TrainingCM * PM_JJKT)
Replace ALL PM_JJTC70 WITH (TrainingCM - Pct70 * TrainingCM * PM_JJKT)
Replace ALL PM_JJTC80 WITH (TrainingCM - Pct80 * TrainingCM * PM_JJKT)
Replace ALL PM_JJTC90 WITH (TrainingCM - Pct90 * TrainingCM * PM_JJKT)
Replace ALL PM_JJTC100 WITH (TrainingCM - Pct100 * TrainingCM * PM_JJKT)

*********************************************************************************
** PAIR HIGH complexity projects and EXPERTISE COMPOSITION (JUNIOR-JUNIOR)
*********************************************************************************
* Effort based on survey responses
Replace ALL PH_JJEf WITH hcx_hrs for Jr2_ef = 0
Replace ALL PH_JJEf WITH hcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Jr2_ef
= 1
```

```
Replace ALL PH_JJEf WITH hcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Jr2_ef
= 2
Replace ALL PH_JJEf WITH hcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Jr2_ef
= 3
Replace ALL PH_JJEf WITH hcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Jr2_ef
= 4
Replace ALL PH_JJEf WITH hcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for Jr2_ef
= 5
Replace ALL PH_JJEf WITH hcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for Jr2_ef
= 6
Replace ALL PH_JJEf WITH hcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Jr2_ef
= -1
Replace ALL PH_JJEf WITH hcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Jr2_ef
= -2
Replace ALL PH_JJEf WITH hcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Jr2_ef
= -3
Replace ALL PH_JJEf WITH hcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Jr2_ef
= -4
Replace ALL PH_JJEf WITH hcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for
Jr2_ef = -5

Replace ALL PH_JJECo WITH PH_JJEf * ROUND((payRateJ + payRateJ2)/2, 2)

* Defect based on survey responses
Replace ALL PH_JJDef WITH hcx_Defect FOR Jr2_df = 0
Replace ALL PH_JJDef WITH hcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jr2_df = 1
Replace ALL PH_JJDef WITH hcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jr2_df = 2
Replace ALL PH_JJDef WITH hcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jr2_df = 3
Replace ALL PH_JJDef WITH hcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jr2_df = 4
Replace ALL PH_JJDef WITH hcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Jr2_df = 5
Replace ALL PH_JJDef WITH hcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Jr2_df = 6
Replace ALL PH_JJDef WITH hcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jr2_df = -1
Replace ALL PH_JJDef WITH hcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jr2_df = -2
Replace ALL PH_JJDef WITH hcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jr2_df = -3
Replace ALL PH_JJDef WITH hcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jr2_df = -4
Replace ALL PH_JJDef WITH hcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Jr2_df = -5

Replace ALL PH_JJDCo WITH (PH_JJDef * h_size * FixingS) * payRate

* Duration
Replace ALL PH_JJDR WITH PH_JJEf/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PH_JJSlack WITH PH_JJDR - H_Ideal
```

```
Replace ALL PH_JJSCo WITH -1 * PH_JJSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateJ2)/2, 2)
FOR PH_JJSlack <=0
Replace ALL PH_JJOCo WITH PH_JJSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateJ2)/2, 2) FOR
PH_JJSlack > 0

* Knowledge Transfer based on survey responses
Replace ALL PH_JJKT WITH SH_KT FOR Jr2_kt = 0
Replace ALL PH_JJKT WITH SH_KT * (1  + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jr2_kt = 1
Replace ALL PH_JJKT WITH SH_KT * (1  + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jr2_kt = 2
Replace ALL PH_JJKT WITH SH_KT * (1  + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jr2_kt = 3
Replace ALL PH_JJKT WITH SH_KT * (1  + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jr2_kt = 4
Replace ALL PH_JJKT WITH SH_KT * (1  + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Jr2_kt = 5
Replace ALL PH_JJKT WITH SH_KT * (1  + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Jr2_kt = 6
Replace ALL PH_JJKT WITH SH_KT * (1  -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jr2_kt = -1
Replace ALL PH_JJKT WITH SH_KT * (1  -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jr2_kt = -2
Replace ALL PH_JJKT WITH SH_KT * (1  -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jr2_kt = -3
Replace ALL PH_JJKT WITH SH_KT * (1  -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jr2_kt = -4
Replace ALL PH_JJKT WITH SH_KT * (1  -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) FOR
Jr2_kt = -5

* Convert KT into training cost
Replace ALL PH_JJTC10 WITH (TrainingCH - Pct10 * TrainingCH * PH_JJKT)
Replace ALL PH_JJTC20 WITH (TrainingCH - Pct20 * TrainingCH * PH_JJKT)
Replace ALL PH_JJTC30 WITH (TrainingCH - Pct30 * TrainingCH * PH_JJKT)
Replace ALL PH_JJTC40 WITH (TrainingCH - Pct40 * TrainingCH * PH_JJKT)
Replace ALL PH_JJTC50 WITH (TrainingCH - Pct50 * TrainingCH * PH_JJKT)
Replace ALL PH_JJTC60 WITH (TrainingCH - Pct60 * TrainingCH * PH_JJKT)
Replace ALL PH_JJTC70 WITH (TrainingCH - Pct70 * TrainingCH * PH_JJKT)
Replace ALL PH_JJTC80 WITH (TrainingCH - Pct80 * TrainingCH * PH_JJKT)
Replace ALL PH_JJTC90 WITH (TrainingCH - Pct90 * TrainingCH * PH_JJKT)
Replace ALL PH_JJTC100 WITH (TrainingCH - Pct100 * TrainingCH * PH_JJKT)


********************************************************************************************** PAIR
LOW complexity projects and EXPERTISE COMPOSITION (JUNIOR-SENIOR)
**********************************************************************************************
* Effort based on survey responses
Replace ALL PL_JSEf WITH lcx_hrs for Jrsr_ef = 0
Replace ALL PL_JSEf WITH lcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Jrsr_ef
= 1
Replace ALL PL_JSEf WITH lcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Jrsr_ef
= 2
Replace ALL PL_JSEf WITH lcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Jrsr_ef
= 3
Replace ALL PL_JSEf WITH lcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Jrsr_ef
= 4
```

```
Replace ALL PL_JSEf WITH lcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for Jrsr_ef
= 5
Replace ALL PL_JSEf WITH lcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for Jrsr_ef
= 6
Replace ALL PL_JSEf WITH lcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Jrsr_ef
= -1
Replace ALL PL_JSEf WITH lcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Jrsr_ef
= -2
Replace ALL PL_JSEf WITH lcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Jrsr_ef
= -3
Replace ALL PL_JSEf WITH lcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Jrsr_ef
= -4
Replace ALL PL_JSEf WITH lcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for
Jrsr_ef = -5

Replace ALL PL_JSECo WITH PL_JSEf * ROUND((payRateJ + payRateS)/2, 2)

* Defect based on survey responses
Replace ALL PL_JSDef WITH lcx_Defect FOR Jrsr_df = 0
Replace ALL PL_JSDef WITH lcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jrsr_df = 1
Replace ALL PL_JSDef WITH lcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jrsr_df = 2
Replace ALL PL_JSDef WITH lcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jrsr_df = 3
Replace ALL PL_JSDef WITH lcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jrsr_df = 4
Replace ALL PL_JSDef WITH lcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Jrsr_df = 5
Replace ALL PL_JSDef WITH lcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Jrsr_df = 6
Replace ALL PL_JSDef WITH lcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jrsr_df = -1
Replace ALL PL_JSDef WITH lcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jrsr_df = -2
Replace ALL PL_JSDef WITH lcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jrsr_df = -3
Replace ALL PL_JSDef WITH lcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jrsr_df = -4
Replace ALL PL_JSDef WITH lcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Jrsr_df = -5

Replace ALL PL_JSDCo WITH (PL_JSDef * l_Size * FixingS) * payRate

* Duration
Replace ALL PL_JSDR WITH PL_JSEf/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PL_JSSlack WITH PL_JSDR - L_Ideal
Replace ALL PL_JSSCo WITH -1 * PL_JSSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateS)/2, 2)
FOR PL_JSSlack <=0
Replace ALL PL_JSOCo WITH PL_JSSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateS)/2, 2) FOR
PL_JSSlack > 0

* Knowledge Transfer based on survey responses
Replace ALL PL_JSKT WITH SL_KT FOR Jrsr_kt = 0
```

```
Replace ALL PL_JSKT WITH SL_KT * (1  +  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jrsr_kt = 1
Replace ALL PL_JSKT WITH SL_KT * (1  +  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jrsr_kt = 2
Replace ALL PL_JSKT WITH SL_KT * (1  +  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jrsr_kt = 3
Replace ALL PL_JSKT WITH SL_KT * (1  +  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jrsr_kt = 4
Replace ALL PL_JSKT WITH SL_KT * (1  +  ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Jrsr_kt = 5
Replace ALL PL_JSKT WITH SL_KT * (1  +  ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Jrsr_kt = 6
Replace ALL PL_JSKT WITH SL_KT * (1  -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jrsr_kt = -1
Replace ALL PL_JSKT WITH SL_KT * (1  -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jrsr_kt = -2
Replace ALL PL_JSKT WITH SL_KT * (1  -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jrsr_kt = -3
Replace ALL PL_JSKT WITH SL_KT * (1  -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jrsr_kt = -4
Replace ALL PL_JSKT WITH SL_KT * (1  -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) FOR
Jrsr_kt = -5

* Convert KT into training cost
Replace ALL PL_JSTC10 WITH (TrainingCL - Pct10 * TrainingCL * PL_JSKT)
Replace ALL PL_JSTC20 WITH (TrainingCL - Pct20 * TrainingCL * PL_JSKT)
Replace ALL PL_JSTC30 WITH (TrainingCL - Pct30 * TrainingCL * PL_JSKT)
Replace ALL PL_JSTC40 WITH (TrainingCL - Pct40 * TrainingCL * PL_JSKT)
Replace ALL PL_JSTC50 WITH (TrainingCL - Pct50 * TrainingCL * PL_JSKT)
Replace ALL PL_JSTC60 WITH (TrainingCL - Pct60 * TrainingCL * PL_JSKT)
Replace ALL PL_JSTC70 WITH (TrainingCL - Pct70 * TrainingCL * PL_JSKT)
Replace ALL PL_JSTC80 WITH (TrainingCL - Pct80 * TrainingCL * PL_JSKT)
Replace ALL PL_JSTC90 WITH (TrainingCL - Pct90 * TrainingCL * PL_JSKT)
Replace ALL PL_JSTC100 WITH (TrainingCL - Pct100 * TrainingCL * PL_JSKT)

*********************************************************************************
** PAIR MEDIUM complexity projects and EXPERTISE COMPOSITION (JUNIOR-SENIOR)
********************************************************************************* Effort
based on survey responses
Replace ALL PM_JSEf WITH mcx_hrs for Jrsr_ef = 0
Replace ALL PM_JSEf WITH mcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Jrsr_ef = 1
Replace ALL PM_JSEf WITH mcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Jrsr_ef = 2
Replace ALL PM_JSEf WITH mcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Jrsr_ef = 3
Replace ALL PM_JSEf WITH mcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Jrsr_ef = 4
Replace ALL PM_JSEf WITH mcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
Jrsr_ef = 5
Replace ALL PM_JSEf WITH mcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
Jrsr_ef = 6
Replace ALL PM_JSEf WITH mcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Jrsr_ef = -1
Replace ALL PM_JSEf WITH mcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Jrsr_ef = -2
```

```
Replace ALL PM_JSEf WITH mcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Jrsr_ef = -3
Replace ALL PM_JSEf WITH mcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Jrsr_ef= -4
Replace ALL PM_JSEf WITH mcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for
Jrsr_ef = -5

Replace ALL PM_JSECo WITH PM_JSEf * ROUND((payRateJ + payRateS)/2, 2)

* Defect based on survey responses
Replace ALL PM_JSDef WITH mcx_Defect FOR Jrsr_df = 0
Replace ALL PM_JSDef WITH mcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR Jrsr_df = 1
Replace ALL PM_JSDef WITH mcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR Jrsr_df = 2
Replace ALL PM_JSDef WITH mcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR Jrsr_df = 3
Replace ALL PM_JSDef WITH mcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR Jrsr_df = 4
Replace ALL PM_JSDef WITH mcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2))
FOR Jrsr_df = 5
Replace ALL PM_JSDef WITH mcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2))
FOR Jrsr_df = 6
Replace ALL PM_JSDef WITH mcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR Jrsr_df = -1
Replace ALL PM_JSDef WITH mcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR Jrsr_df = -2
Replace ALL PM_JSDef WITH mcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR Jrsr_df = -3
Replace ALL PM_JSDef WITH mcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR Jrsr_df = -4
Replace ALL PM_JSDef WITH mcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Jrsr_df = -5

Replace ALL PM_JSDCo WITH (PM_JSDef * m_Size * FixingS) * payRate

* Duration
Replace ALL PM_JSDR WITH PM_JSEf/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PM_JSSlack WITH PM_JSDR - M_Ideal
Replace ALL PM_JSSCo WITH -1 * PM_JSSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateS)/2, 2)
FOR PM_JSSlack <=0
Replace ALL PM_JSOCo WITH PM_JSSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateS)/2, 2) FOR
PM_JSSlack > 0

* Knowledge Transfer based on survey responses
Replace ALL PM_JSKT WITH SM_KT FOR Jrsr_kt = 0
Replace ALL PM_JSKT WITH SM_KT * (1  +  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jrsr_kt = 1
Replace ALL PM_JSKT WITH SM_KT * (1  +  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jrsr_kt = 2
Replace ALL PM_JSKT WITH SM_KT * (1  +  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jrsr_kt = 3
Replace ALL PM_JSKT WITH SM_KT * (1  +  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jrsr_kt = 4
```

```
Replace ALL PM_JSKT WITH SM_KT * (1  +  ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Jrsr_kt = 5
Replace ALL PM_JSKT WITH SM_KT * (1  +  ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Jrsr_kt = 6
Replace ALL PM_JSKT WITH SM_KT * (1  -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jrsr_kt = -1
Replace ALL PM_JSKT WITH SM_KT * (1  -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jrsr_kt = -2
Replace ALL PM_JSKT WITH SM_KT * (1  -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jrsr_kt = -3
Replace ALL PM_JSKT WITH SM_KT * (1  -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jrsr_kt = -4
Replace ALL PM_JSKT WITH SM_KT * (1  -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Jrsr_kt = -5

* Convert KT into training cost
Replace ALL PM_JSTC10 WITH (TrainingCM - Pct10 * TrainingCM * PM_JSKT)
Replace ALL PM_JSTC20 WITH (TrainingCM - Pct20 * TrainingCM * PM_JSKT)
Replace ALL PM_JSTC30 WITH (TrainingCM - Pct30 * TrainingCM * PM_JSKT)
Replace ALL PM_JSTC40 WITH (TrainingCM - Pct40 * TrainingCM * PM_JSKT)
Replace ALL PM_JSTC50 WITH (TrainingCM - Pct50 * TrainingCM * PM_JSKT)
Replace ALL PM_JSTC60 WITH (TrainingCM - Pct60 * TrainingCM * PM_JSKT)
Replace ALL PM_JSTC70 WITH (TrainingCM - Pct70 * TrainingCM * PM_JSKT)
Replace ALL PM_JSTC80 WITH (TrainingCM - Pct80 * TrainingCM * PM_JSKT)
Replace ALL PM_JSTC90 WITH (TrainingCM - Pct90 * TrainingCM * PM_JSKT)
Replace ALL PM_JSTC100 WITH (TrainingCM - Pct100 * TrainingCM * PM_JSKT)


*********************************************************************************
** PAIR HIGH complexity projects and EXPERTISE COMPOSITION (JUNIOR-SENIOR)
*********************************************************************************
* Effort based on survey responses
Replace ALL PH_JSEf WITH hcx_hrs for Jrsr_ef = 0
Replace ALL PH_JSEf WITH hcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Jrsr_ef
= 1
Replace ALL PH_JSEf WITH hcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Jrsr_ef
= 2
Replace ALL PH_JSEf WITH hcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Jrsr_ef
= 3
Replace ALL PH_JSEf WITH hcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Jrsr_ef
= 4
Replace ALL PH_JSEf WITH hcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for Jrsr_ef
= 5
Replace ALL PH_JSEf WITH hcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for Jrsr_ef
= 6
Replace ALL PH_JSEf WITH hcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Jrsr_ef
= -1
Replace ALL PH_JSEf WITH hcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Jrsr_ef
= -2
Replace ALL PH_JSEf WITH hcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Jrsr_ef
= -3
Replace ALL PH_JSEf WITH hcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Jrsr_ef
= -4
Replace ALL PH_JSEf WITH hcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for
Jrsr_ef= -5
```

```
Replace ALL PH_JSECo WITH PH_JSEf * ROUND((payRateJ + payRateS)/2, 2)

* Defect based on survey responses
Replace ALL PH_JSDef WITH hcx_Defect FOR Jrsr_df = 0
Replace ALL PH_JSDef WITH hcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jrsr_df = 1
Replace ALL PH_JSDef WITH hcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jrsr_df = 2
Replace ALL PH_JSDef WITH hcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jrsr_df = 3
Replace ALL PH_JSDef WITH hcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jrsr_df = 4
Replace ALL PH_JSDef WITH hcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Jrsr_df = 5
Replace ALL PH_JSDef WITH hcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Jrsr_df = 6
Replace ALL PH_JSDef WITH hcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jrsr_df = -1
Replace ALL PH_JSDef WITH hcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jrsr_df = -2
Replace ALL PH_JSDef WITH hcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jrsr_df = -3
Replace ALL PH_JSDef WITH hcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jrsr_df = -4
Replace ALL PH_JSDef WITH hcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Jrsr_df= -5

Replace ALL PH_JSDCo WITH (PH_JSDef * h_Size * FixingS) * payRate

* Duration
Replace ALL PH_JSDR WITH PH_JSEf/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PH_JSSlack WITH PH_JSDR - H_Ideal
Replace ALL PH_JSSCo WITH -1 * PH_JSSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateS)/2, 2)
FOR PH_JSSlack <=0
Replace ALL PH_JSOCo WITH PH_JSSlack * HrsPerDay8 * Team * ROUND((payRateJ + payRateS)/2, 2) FOR
PH_JSSlack > 0

* Knowledge Transfer based on survey responses
Replace ALL PH_JSKT WITH SH_KT FOR Jrsr_kt = 0
Replace ALL PH_JSKT WITH SH_KT * (1  +  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jrsr_kt = 1
Replace ALL PH_JSKT WITH SH_KT * (1  +  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jrsr_kt  = 2
Replace ALL PH_JSKT WITH SH_KT * (1  +  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jrsr_kt  = 3
Replace ALL PH_JSKT WITH SH_KT * (1  +  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jrsr_kt = 4
Replace ALL PH_JSKT WITH SH_KT * (1  +  ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Jrsr_kt  = 5
Replace ALL PH_JSKT WITH SH_KT * (1  +  ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Jrsr_kt  = 6
Replace ALL PH_JSKT WITH SH_KT * (1  -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Jrsr_kt  = -1
```

```
Replace ALL PH_JSKT WITH SH_KT * (1  -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Jrsr_kt  = -2
Replace ALL PH_JSKT WITH SH_KT * (1  -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Jrsr_kt  = -3
Replace ALL PH_JSKT WITH SH_KT * (1  -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Jrsr_kt  = -4
Replace ALL PH_JSKT WITH SH_KT * (1  -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) FOR
Jrsr_kt = -5

* Convert KT into training cost
Replace ALL PH_JSTC10 WITH (TrainingCH - Pct10 * TrainingCH * PH_JSKT)
Replace ALL PH_JSTC20 WITH (TrainingCH - Pct20 * TrainingCH * PH_JSKT)
Replace ALL PH_JSTC30 WITH (TrainingCH - Pct30 * TrainingCH * PH_JSKT)
Replace ALL PH_JSTC40 WITH (TrainingCH - Pct40 * TrainingCH * PH_JSKT)
Replace ALL PH_JSTC50 WITH (TrainingCH - Pct50 * TrainingCH * PH_JSKT)
Replace ALL PH_JSTC60 WITH (TrainingCH - Pct60 * TrainingCH * PH_JSKT)
Replace ALL PH_JSTC70 WITH (TrainingCH - Pct70 * TrainingCH * PH_JSKT)
Replace ALL PH_JSTC80 WITH (TrainingCH - Pct80 * TrainingCH * PH_JSKT)
Replace ALL PH_JSTC90 WITH (TrainingCH - Pct90 * TrainingCH * PH_JSKT)
Replace ALL PH_JSTC100 WITH (TrainingCH - Pct100 * TrainingCH * PH_JSKT)


*********************************************************************************
** PAIR LOW complexity projects and EXPERTISE COMPOSITION (SENIOR-SENIOR)
*********************************************************************************
* Effort based on survey responses
Replace ALL PL_SSEf WITH lcx_hrs for Sr2_ef = 0
Replace ALL PL_SSEf WITH lcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Sr2_ef
= 1
Replace ALL PL_SSEf WITH lcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Sr2_ef
= 2
Replace ALL PL_SSEf WITH lcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Sr2_ef
= 3
Replace ALL PL_SSEf WITH lcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Sr2_ef
= 4
Replace ALL PL_SSEf WITH lcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for Sr2_ef
= 5
Replace ALL PL_SSEf WITH lcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for Sr2_ef
= 6
Replace ALL PL_SSEf WITH lcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Sr2_ef
= -1
Replace ALL PL_SSEf WITH lcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Sr2_ef
= -2
Replace ALL PL_SSEf WITH lcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Sr2_ef
= -3
Replace ALL PL_SSEf WITH lcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Sr2_ef
= -4
Replace ALL PL_SSEf WITH lcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for
Sr2_ef = -5

Replace ALL PL_SSECo WITH PL_SSEf * ROUND((payRateS + payRateS2)/2, 2)

* Defect based on survey responses
Replace ALL PL_SSDef WITH lcx_Defect FOR Sr2_df = 0
Replace ALL PL_SSDef WITH lcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Sr2_df = 1
```

```
Replace ALL PL_SSDef WITH lcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Sr2_df = 2
Replace ALL PL_SSDef WITH lcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Sr2_df = 3
Replace ALL PL_SSDef WITH lcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Sr2_df = 4
Replace ALL PL_SSDef WITH lcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Sr2_df = 5
Replace ALL PL_SSDef WITH lcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Sr2_df = 6
Replace ALL PL_SSDef WITH lcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Sr2_df = -1
Replace ALL PL_SSDef WITH lcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Sr2_df = -2
Replace ALL PL_SSDef WITH lcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Sr2_df = -3
Replace ALL PL_SSDef WITH lcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Sr2_df = -4
Replace ALL PL_SSDef WITH lcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Sr2_df = -5

Replace ALL PL_SSDCo WITH (PL_SSDef * l_Size * FixingS) * payRate

* Duration
Replace ALL PL_SSDR WITH PL_SSEf/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PL_SSSlack WITH PL_SSDR - L_Ideal
Replace ALL PL_SSSCo WITH -1 * PL_SSSlack * HrsPerDay8 * Team * ROUND((payRateS + payRateS2)/2, 2)
FOR PL_SSSlack <=0
Replace ALL PL_SSOCo WITH PL_SSSlack * HrsPerDay8 * Team * ROUND((payRateS + payRateS2)/2, 2) FOR
PL_SSSlack > 0

* Knowledge Transfer based on survey responses
Replace ALL PL_SSKT WITH SL_KT FOR Sr2_kt = 0
Replace ALL PL_SSKT WITH SL_KT * (1  +  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Sr2_kt = 1
Replace ALL PL_SSKT WITH SL_KT * (1  +  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Sr2_kt = 2
Replace ALL PL_SSKT WITH SL_KT * (1  +  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Sr2_kt = 3
Replace ALL PL_SSKT WITH SL_KT * (1  +  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Sr2_kt = 4
Replace ALL PL_SSKT WITH SL_KT * (1  +  ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Sr2_kt = 5
Replace ALL PL_SSKT WITH SL_KT * (1  +  ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Sr2_kt = 6
Replace ALL PL_SSKT WITH SL_KT * (1  -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Sr2_kt = -1
Replace ALL PL_SSKT WITH SL_KT * (1  -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Sr2_kt = -2
Replace ALL PL_SSKT WITH SL_KT * (1  -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Sr2_kt = -3
Replace ALL PL_SSKT WITH SL_KT * (1  -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Sr2_kt = -4
```

```
Replace ALL PL_SSKT WITH SL_KT * (1  -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) FOR
Sr2_kt = -5

* Convert KT into training cost
Replace ALL PL_SSTC10 WITH (TrainingCL - Pct10 * TrainingCL * PL_SSKT)
Replace ALL PL_SSTC20 WITH (TrainingCL - Pct20 * TrainingCL * PL_SSKT)
Replace ALL PL_SSTC30 WITH (TrainingCL - Pct30 * TrainingCL * PL_SSKT)
Replace ALL PL_SSTC40 WITH (TrainingCL - Pct40 * TrainingCL * PL_SSKT)
Replace ALL PL_SSTC50 WITH (TrainingCL - Pct50 * TrainingCL * PL_SSKT)
Replace ALL PL_SSTC60 WITH (TrainingCL - Pct60 * TrainingCL * PL_SSKT)
Replace ALL PL_SSTC70 WITH (TrainingCL - Pct70 * TrainingCL * PL_SSKT)
Replace ALL PL_SSTC80 WITH (TrainingCL - Pct80 * TrainingCL * PL_SSKT)
Replace ALL PL_SSTC90 WITH (TrainingCL - Pct90 * TrainingCL * PL_SSKT)
Replace ALL PL_SSTC100 WITH (TrainingCL - Pct100 * TrainingCL * PL_SSKT)

********************************************************************************************** PAIR
MEDIUM complexity projects and EXPERTISE COMPOSITION (SENIOR-SENIOR)
**********************************************************************************
* Effort based on survey responses
Replace ALL PM_SSEf WITH mcx_hrs for Sr2_ef = 0
Replace ALL PM_SSEf WITH mcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Sr2_ef = 1
Replace ALL PM_SSEf WITH mcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Sr2_ef = 2
Replace ALL PM_SSEf WITH mcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Sr2_ef = 3
Replace ALL PM_SSEf WITH mcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Sr2_ef = 4
Replace ALL PM_SSEf WITH mcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
Sr2_ef = 5
Replace ALL PM_SSEf WITH mcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
Sr2_ef = 6
Replace ALL PM_SSEf WITH mcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Sr2_ef = -1
Replace ALL PM_SSEf WITH mcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Sr2_ef = -2
Replace ALL PM_SSEf WITH mcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Sr2_ef = -3
Replace ALL PM_SSEf WITH mcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Sr2_ef= -4
Replace ALL PM_SSEf WITH mcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for
Sr2_ef = -5

Replace ALL PM_SSECo WITH PM_SSEf * ROUND((payRateS + payRateS2)/2, 2)

* Defect based on survey responses
Replace ALL PM_SSDef WITH mcx_Defect FOR Sr2_df = 0
Replace ALL PM_SSDef WITH mcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR Sr2_df = 1
Replace ALL PM_SSDef WITH mcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR Sr2_df = 2
Replace ALL PM_SSDef WITH mcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR Sr2_df = 3
Replace ALL PM_SSDef WITH mcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR Sr2_df = 4
```

```
Replace ALL PM_SSDef WITH mcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2))
FOR Sr2_df = 5
Replace ALL PM_SSDef WITH mcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2))
FOR Sr2_df = 6
Replace ALL PM_SSDef WITH mcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR Sr2_df = -1
Replace ALL PM_SSDef WITH mcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR Sr2_df = -2
Replace ALL PM_SSDef WITH mcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR Sr2_df = -3
Replace ALL PM_SSDef WITH mcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR Sr2_df = -4
Replace ALL PM_SSDef WITH mcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Sr2_df = -5

Replace ALL PM_SSDCo WITH (PM_SSDef * m_Size * FixingS) * payRate

* Duration
Replace ALL PM_SSDR WITH PM_SSEf/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PM_SSSlack WITH PM_SSDR - M_Ideal
Replace ALL PM_SSSCo WITH -1 * PM_SSSlack * HrsPerDay8 * Team * ROUND((payRateS + payRateS2)/2, 2)
FOR PM_SSSlack <=0
Replace ALL PM_SSOCo WITH PM_SSSlack * HrsPerDay8 * Team * ROUND((payRateS + payRateS2)/2, 2)
FOR PM_SSSlack > 0

* Knowledge Transfer based on survey responses
Replace ALL PM_SSKT WITH SM_KT FOR Sr2_kt = 0
Replace ALL PM_SSKT WITH SM_KT * (1 +  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Sr2_kt = 1
Replace ALL PM_SSKT WITH SM_KT * (1 +  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Sr2_kt = 2
Replace ALL PM_SSKT WITH SM_KT * (1 +  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Sr2_kt = 3
Replace ALL PM_SSKT WITH SM_KT * (1 +  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Sr2_kt = 4
Replace ALL PM_SSKT WITH SM_KT * (1 +  ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Sr2_kt = 5
Replace ALL PM_SSKT WITH SM_KT * (1 +  ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Sr2_kt = 6
Replace ALL PM_SSKT WITH SM_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Sr2_kt = -1
Replace ALL PM_SSKT WITH SM_KT * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Sr2_kt = -2
Replace ALL PM_SSKT WITH SM_KT * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Sr2_kt = -3
Replace ALL PM_SSKT WITH SM_KT * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Sr2_kt = -4
Replace ALL PM_SSKT WITH SM_KT * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) FOR
Sr2_kt = -5

* Convert KT into training cost
Replace ALL PM_SSTC10 WITH (TrainingCM - Pct10 * TrainingCM * PM_SSKT)
Replace ALL PM_SSTC20 WITH (TrainingCM - Pct20 * TrainingCM * PM_SSKT)
Replace ALL PM_SSTC30 WITH (TrainingCM - Pct30 * TrainingCM * PM_SSKT)
```

```
Replace ALL PM_SSTC40 WITH (TrainingCM - Pct40 * TrainingCM * PM_SSKT)
Replace ALL PM_SSTC50 WITH (TrainingCM - Pct50 * TrainingCM * PM_SSKT)
Replace ALL PM_SSTC60 WITH (TrainingCM - Pct60 * TrainingCM * PM_SSKT)
Replace ALL PM_SSTC70 WITH (TrainingCM - Pct70 * TrainingCM * PM_SSKT)
Replace ALL PM_SSTC80 WITH (TrainingCM - Pct80 * TrainingCM * PM_SSKT)
Replace ALL PM_SSTC90 WITH (TrainingCM - Pct90 * TrainingCM * PM_SSKT)
Replace ALL PM_SSTC100 WITH (TrainingCM - Pct100 * TrainingCM * PM_SSKT)

********************************************************************************************** PAIR
HIGH complexity projects and EXPERTISE COMPOSITION (SENIOR-SENIOR)
**********************************************************************************************
* Effort based on survey responses
Replace ALL PH_SSEf WITH hcx_hrs for Sr2_ef = 0
Replace ALL PH_SSEf WITH hcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Sr2_ef
= 1
Replace ALL PH_SSEf WITH hcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Sr2_ef
= 2
Replace ALL PH_SSEf WITH hcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Sr2_ef
= 3
Replace ALL PH_SSEf WITH hcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Sr2_ef
= 4
Replace ALL PH_SSEf WITH hcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for Sr2_ef
= 5
Replace ALL PH_SSEf WITH hcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for Sr2_ef
= 6
Replace ALL PH_SSEf WITH hcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for Sr2_ef
= -1
Replace ALL PH_SSEf WITH hcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for Sr2_ef
= -2
Replace ALL PH_SSEf WITH hcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for Sr2_ef
= -3
Replace ALL PH_SSEf WITH hcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for Sr2_ef
= -4
Replace ALL PH_SSEf WITH hcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for
Sr2_ef= -5

Replace ALL PH_SSECo WITH PH_SSEf * ROUND((payRateS + payRateS2)/2, 2)

* Defect based on survey responses
Replace ALL PH_SSDef WITH hcx_Defect FOR Sr2_df = 0
Replace ALL PH_SSDef WITH hcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Sr2_df = 1
Replace ALL PH_SSDef WITH hcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Sr2_df = 2
Replace ALL PH_SSDef WITH hcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Sr2_df = 3
Replace ALL PH_SSDef WITH hcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Sr2_df = 4
Replace ALL PH_SSDef WITH hcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Sr2_df = 5
Replace ALL PH_SSDef WITH hcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Sr2_df= 6
Replace ALL PH_SSDef WITH hcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Sr2_df = -1
Replace ALL PH_SSDef WITH hcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Sr2_df = -2
```

```
Replace ALL PH_SSDef WITH hcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Sr2_df = -3
Replace ALL PH_SSDef WITH hcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Sr2_df = -4
Replace ALL PH_SSDef WITH hcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Sr2_df= -5

Replace ALL PH_SSDCo WITH (PH_SSDef * h_Size * FixingS) * payRate

* Duration
Replace ALL PH_SSDR WITH PH_SSEf/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PH_SSSlack WITH PH_SSDR - H_Ideal
Replace ALL PH_SSSCo WITH -1 * PH_SSSlack * HrsPerDay8 * Team * ROUND((payRateS + payRateS2)/2, 2)
FOR PH_SSSlack <=0
Replace ALL PH_SSOCo WITH PH_SSSlack * HrsPerDay8 * Team * ROUND((payRateS + payRateS2)/2, 2)
FOR PH_SSSlack > 0

* Knowledge Transfer based on survey responses
Replace ALL PH_SSKT WITH SH_KT FOR Sr2_kt = 0
Replace ALL PH_SSKT WITH SH_KT * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Sr2_kt = 1
Replace ALL PH_SSKT WITH SH_KT * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Sr2_kt  = 2
Replace ALL PH_SSKT WITH SH_KT * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Sr2_kt  = 3
Replace ALL PH_SSKT WITH SH_KT * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Sr2_kt = 4
Replace ALL PH_SSKT WITH SH_KT * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Sr2_kt  = 5
Replace ALL PH_SSKT WITH SH_KT * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Sr2_kt  = 6
Replace ALL PH_SSKT WITH SH_KT * (1 - ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Sr2_kt  = -1
Replace ALL PH_SSKT WITH SH_KT * (1 - ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Sr2_kt  = -2
Replace ALL PH_SSKT WITH SH_KT * (1 - ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Sr2_kt  = -3
Replace ALL PH_SSKT WITH SH_KT * (1 - ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Sr2_kt  = -4
Replace ALL PH_SSKT WITH SH_KT * (1 - ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) FOR
Sr2_kt = -5

* Convert KT into training cost
Replace ALL PH_SSTC10 WITH (TrainingCH - Pct10 * TrainingCH * PH_SSKT)
Replace ALL PH_SSTC20 WITH (TrainingCH - Pct20 * TrainingCH * PH_SSKT)
Replace ALL PH_SSTC30 WITH (TrainingCH - Pct30 * TrainingCH * PH_SSKT)
Replace ALL PH_SSTC40 WITH (TrainingCH - Pct40 * TrainingCH * PH_SSKT)
Replace ALL PH_SSTC50 WITH (TrainingCH - Pct50 * TrainingCH * PH_SSKT)
Replace ALL PH_SSTC60 WITH (TrainingCH - Pct60 * TrainingCH * PH_SSKT)
Replace ALL PH_SSTC70 WITH (TrainingCH - Pct70 * TrainingCH * PH_SSKT)
Replace ALL PH_SSTC80 WITH (TrainingCH - Pct80 * TrainingCH * PH_SSKT)
Replace ALL PH_SSTC90 WITH (TrainingCH - Pct90 * TrainingCH * PH_SSKT)
Replace ALL PH_SSTC100 WITH (TrainingCH - Pct100 * TrainingCH * PH_SSKT)
```

```
*********************************************************************************************
* PAIR COMPOSTION - NEITHER in the pair has prior pair programming experience (PRIOR0), ONE has
(PRIOR1), BOTH have (PRIOR2)
*********************************************************************************************
* Create C3 since the maximum number of columns a table can have is 256. C3 has the cost columns needed for
prior0, prior1, and prior2

SELECT respondent, trainingC, trainingCL, trainingCM, trainingCH, team, l_team, m_team, h_team, comp1yr,
lcx_hrs, mcx_hrs, hcx_hrs, prior0_ef, prior1_ef, prior2_ef,  lcx_defect, mcx_defect, hcx_defect, prior0_df,
prior1_df, prior2_df, sl_kt, sm_kt, sh_kt, prior0_kt, prior1_kt, prior2_kt,
payRate, payRate2, payRateJ, payRateJ2, payRateS, payRateS2, L_Size, M_Size, H_Size, FixingS, L_Ideal,
M_Ideal, H_Ideal, 000000.00 as PL_P0Ef, 000000.00 as PL_P0ECo, 000000.00 as PL_P0Def, 0000000000.00 as
PL_P0DCo, 000000.00 as PL_P0DR, 000000.00 as PL_P0OCo, 000000.00 as PL_P0Slack, 000000.00 as
PL_P0SCo,  00.00 as PL_P0KT,  000000.00 as PL_P0TC10, 000000.00 as PL_P0TC20, 000000.00 as PL_P0TC30,
000000.00 as PL_P0TC40, 000000.00 as PL_P0TC50, 000000.00 as PL_P0TC60, 000000.00 as PL_P0TC70,
000000.00 as PL_P0TC80, 000000.00 as PL_P0TC90, 000000.00 as PL_P0TC100,  000000.00 as PM_P0Ef,
000000.00 as PM_P0ECo, 000000.00 as PM_P0Def, 0000000000.00 as PM_P0DCo, 000000.00 as PM_P0DR,
000000.00 as PM_P0OCo, 000000.00 as PM_P0Slack, 000000.00 as PM_P0SCo, 00.00 as PM_P0KT, 000000.00
as PM_P0TC10, 000000.00 as PM_P0TC20, 000000.00 as PM_P0TC30, 000000.00 as PM_P0TC40, 000000.00 as
PM_P0TC50, 000000.00 as PM_P0TC60, 000000.00 as PM_P0TC70, 000000.00 as PM_P0TC80, 000000.00 as
PM_P0TC90, 000000.00 as PM_P0TC100,  000000.00 as PH_P0Ef, 000000.00 as PH_P0ECo, ;
000000.00 as PH_P0Def, 0000000000.00 as PH_P0DCo, 000000.00 as PH_P0DR, 000000.00 as PH_P0OCo,
000000.00 as PH_P0Slack, 000000.00 as PH_P0SCo,  00.00 as PH_P0KT, 000000.00 as PH_P0TC10, 000000.00
as PH_P0TC20, 000000.00 as PH_P0TC30, 000000.00 as PH_P0TC40, 000000.00 as PH_P0TC50, 000000.00 as
PH_P0TC60, 000000.00 as PH_P0TC70, 000000.00 as PH_P0TC80, 000000.00 as PH_P0TC90, 000000.00 as
PH_P0TC100,  000000.00 as PL_P1Ef, 000000.00 as PL_P1ECo, 000000.00 as PL_P1Def, 0000000000.00 as
PL_P1DCo, 000000.00 as PL_P1DR, 000000.00 as PL_P1OCo, 000000.00 as PL_P1Slack, 000000.00 as
PL_P1SCo, 00.00 as PL_P1KT, 000000.00 as PL_P1TC10, 000000.00 as PL_P1TC20, 000000.00 as PL_P1TC30,
000000.00 as PL_P1TC40, 000000.00 as PL_P1TC50, 000000.00 as PL_P1TC60, 000000.00 as PL_P1TC70,
000000.00 as PL_P1TC80, 000000.00 as PL_P1TC90, 000000.00 as PL_P1TC100,  000000.00 as PM_P1Ef,
000000.00 as PM_P1ECo, 000000.00 as PM_P1Def, 0000000000.00 as PM_P1DCo, 000000.00 as PM_P1DR,
000000.00 as PM_P1OCo, 000000.00 as PM_P1Slack, 000000.00 as PM_P1SCo,
00.00 as PM_P1KT, 000000.00 as PM_P1TC10, 000000.00 as PM_P1TC20, 000000.00 as PM_P1TC30, 000000.00
as PM_P1TC40, 000000.00 as PM_P1TC50, 000000.00 as PM_P1TC60, 000000.00 as PM_P1TC70, 000000.00 as
PM_P1TC80, 000000.00 as PM_P1TC90, 000000.00 as PM_P1TC100,
000000.00 as PH_P1Ef, 000000.00 as PH_P1ECo, 000000.00 as PH_P1Def, 0000000000.00 as PH_P1DCo,
000000.00 as PH_P1DR, 000000.00 as PH_P1OCo, 000000.00 as PH_P1Slack, 000000.00 as PH_P1SCo, 00.00 as
PH_P1KT, 000000.00 as PH_P1TC10, 000000.00 as PH_P1TC20, 000000.00 as PH_P1TC30, 000000.00 as
PH_P1TC40, 000000.00 as PH_P1TC50, 000000.00 as PH_P1TC60,
000000.00 as PH_P1TC70, 000000.00 as PH_P1TC80, 000000.00 as PH_P1TC90, 000000.00 as PH_P1TC100,
000000.00 as PL_P2Ef, 000000.00 as PL_P2ECo, 000000.00 as PL_P2Def, 0000000000.00 as PL_P2DCo,
000000.00 as PL_P2DR, 000000.00 as PL_P2OCo, 000000.00 as PL_P2Slack, 000000.00 as PL_P2SCo, 00.00 as
PL_P2KT, 000000.00 as PL_P2TC10, 000000.00 as PL_P2TC20, 000000.00 as PL_P2TC30, 000000.00 as
PL_P2TC40, 000000.00 as PL_P2TC50, 000000.00 as PL_P2TC60, 000000.00 as PL_P2TC70, 000000.00 as
PL_P2TC80, 000000.00 as PL_P2TC90, 000000.00 as PL_P2TC100,  000000.00 as PM_P2Ef, 000000.00 as
PM_P2ECo, 000000.00 as PM_P2Def, 0000000000.00 as PM_P2DCo, 000000.00 as PM_P2DR, 000000.00 as
PM_P2OCo, 000000.00 as PM_P2Slack, 000000.00 as PM_P2SCo, 00.00 as PM_P2KT, 000000.00 as
PM_P2TC10, 000000.00 as PM_P2TC20, 000000.00 as PM_P2TC30, 000000.00 as PM_P2TC40, 000000.00 as
PM_P2TC50, 000000.00 as PM_P2TC60, 000000.00 as PM_P2TC70, 000000.00 as PM_P2TC80, 000000.00 as
PM_P2TC90, 000000.00 as PM_P2TC100,  000000.00 as PH_P2Ef, 000000.00 as PH_P2ECo, ;
000000.00 as PH_P2Def, 0000000000.00 as PH_P2DCo, 000000.00 as PH_P2DR, 000000.00 as PH_P2OCo,
000000.00 as PH_P2Slack, 000000.00 as PH_P2SCo, 00.00 as PH_P2KT, 000000.00 as PH_P2TC10, 000000.00 as
PH_P2TC20, 000000.00 as PH_P2TC30, 000000.00 as PH_P2TC40, 000000.00 as PH_P2TC50, 000000.00 as
PH_P2TC60, 000000.00 as PH_P2TC70, 000000.00 as PH_P2TC80, 000000.00 as PH_P2TC90, 000000.00 as
PH_P2TC100
```

```
FROM c1
into TABLE c3

SELECT c3

*****************************************************************************************
** PAIR LOW complexity projects and PRIOR PAIR EXPERIENCE (NEITHER HAS EXPERIENCE)
*****************************************************************************************
* based on survey responses
Replace ALL PL_P0Ef WITH lcx_hrs for Prior0_ef = 0
Replace ALL PL_P0Ef WITH lcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Prior0_ef = 1
Replace ALL PL_P0Ef WITH lcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior0_ef = 2
Replace ALL PL_P0Ef WITH lcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior0_ef = 3
Replace ALL PL_P0Ef WITH lcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior0_ef = 4
Replace ALL PL_P0Ef WITH lcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
Prior0_ef = 5
Replace ALL PL_P0Ef WITH lcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
Prior0_ef = 6
Replace ALL PL_P0Ef WITH lcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Prior0_ef = -1
Replace ALL PL_P0Ef WITH lcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior0_ef = -2
Replace ALL PL_P0Ef WITH lcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior0_ef = -3
Replace ALL PL_P0Ef WITH lcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior0_ef = -4
Replace ALL PL_P0Ef WITH lcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  for
Prior0_ef = -5

Replace ALL PL_P0ECo WITH PL_P0Ef * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
Replace ALL PL_P0Def WITH lcx_Defect FOR Prior0_df = 0
Replace ALL PL_P0Def WITH lcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior0_df = 1
Replace ALL PL_P0Def WITH lcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior0_df = 2
Replace ALL PL_P0Def WITH lcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior0_df = 3
Replace ALL PL_P0Def WITH lcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior0_df = 4
Replace ALL PL_P0Def WITH lcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior0_df = 5
Replace ALL PL_P0Def WITH lcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior0_df = 6
Replace ALL PL_P0Def WITH lcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior0_df = -1
Replace ALL PL_P0Def WITH lcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior0_df = -2
Replace ALL PL_P0Def WITH lcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior0_df = -3
```

```
Replace ALL PL_P0Def WITH lcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior0_df = -4
Replace ALL PL_P0Def WITH lcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior0_df = -5

Replace ALL PL_P0DCo WITH (PL_P0Def * l_Size * FixingS) * payRate

* Duration
Replace ALL PL_P0DR WITH PL_P0Ef/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PL_P0Slack WITH PL_P0DR - L_Ideal
Replace ALL pl_p0Sco WITH  -1 * pl_p0slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR pl_p0Slack
<= 0
Replace ALL pl_p0Oco WITH  pl_p0slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR pl_p0Slack > 0

* Knowledge Transfer based on survey responses
Replace ALL PL_P0KT WITH SL_KT FOR Prior0_kt = 0
Replace ALL PL_P0KT WITH SL_KT * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior0_kt = 1
Replace ALL PL_P0KT WITH SL_KT * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior0_kt = 2
Replace ALL PL_P0KT WITH SL_KT * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior0_kt = 3
Replace ALL PL_P0KT WITH SL_KT * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior0_kt = 4
Replace ALL PL_P0KT WITH SL_KT * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior0_kt = 5
Replace ALL PL_P0KT WITH SL_KT * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior0_kt = 6
Replace ALL PL_P0KT WITH SL_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior0_kt = -1
Replace ALL PL_P0KT WITH SL_KT * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior0_kt = -2
Replace ALL PL_P0KT WITH SL_KT * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior0_kt = -3
Replace ALL PL_P0KT WITH SL_KT * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior0_kt = -4
Replace ALL PL_P0KT WITH SL_KT * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  FOR
Prior0_kt = -5
Replace ALL PL_P0KT WITH 0 FOR PL_P0KT < 0
Replace ALL PL_P0KT WITH 1 FOR PL_P0KT > 1

* Convert KT into training cost
Replace ALL PL_P0TC10 WITH (TrainingCL - Pct10 * TrainingCL * PL_P0KT)
Replace ALL PL_P0TC20 WITH (TrainingCL - Pct20 * TrainingCL * PL_P0KT)
Replace ALL PL_P0TC30 WITH (TrainingCL - Pct30 * TrainingCL * PL_P0KT)
Replace ALL PL_P0TC40 WITH (TrainingCL - Pct40 * TrainingCL * PL_P0KT)
Replace ALL PL_P0TC50 WITH (TrainingCL - Pct50 * TrainingCL * PL_P0KT)
Replace ALL PL_P0TC60 WITH (TrainingCL - Pct60 * TrainingCL * PL_P0KT)
Replace ALL PL_P0TC70 WITH (TrainingCL - Pct70 * TrainingCL * PL_P0KT)
Replace ALL PL_P0TC80 WITH (TrainingCL - Pct80 * TrainingCL * PL_P0KT)
Replace ALL PL_P0TC90 WITH (TrainingCL - Pct90 * TrainingCL * PL_P0KT)
Replace ALL PL_P0TC100 WITH (TrainingCL - Pct100 * TrainingCL * PL_P0KT)

*********************************************************************************
```

```
** PAIR MEDIUM complexity projects and PRIOR PAIR EXPERIENCE (NEITHER HAS EXPERIENCE)
*********************************************************************************************
* Effort based on survey responses
Replace ALL PM_P0Ef WITH mcx_hrs for Prior0_ef = 0
Replace ALL PM_P0Ef WITH mcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
Prior0_ef = 1
Replace ALL PM_P0Ef WITH mcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior0_ef = 2
Replace ALL PM_P0Ef WITH mcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior0_ef = 3
Replace ALL PM_P0Ef WITH mcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior0_ef = 4
Replace ALL PM_P0Ef WITH mcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
Prior0_ef = 5
Replace ALL PM_P0Ef WITH mcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
Prior0_ef = 6
Replace ALL PM_P0Ef WITH mcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Prior0_ef = -1
Replace ALL PM_P0Ef WITH mcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior0_ef = -2
Replace ALL PM_P0Ef WITH mcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior0_ef = -3
Replace ALL PM_P0Ef WITH mcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior0_ef= -4
Replace ALL PM_P0Ef WITH mcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2)) for
Prior0_ef = -5

Replace ALL PM_P0ECo WITH PM_P0Ef * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
Replace ALL PM_P0Def WITH mcx_Defect FOR Prior0_df = 0
Replace ALL PM_P0Def WITH mcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR Prior0_df = 1
Replace ALL PM_P0Def WITH mcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR Prior0_df = 2
Replace ALL PM_P0Def WITH mcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR Prior0_df = 3
Replace ALL PM_P0Def WITH mcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR Prior0_df = 4
Replace ALL PM_P0Def WITH mcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2))
FOR Prior0_df = 5
Replace ALL PM_P0Def WITH mcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2))
FOR Prior0_df = 6
Replace ALL PM_P0Def WITH mcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR Prior0_df = -1
Replace ALL PM_P0Def WITH mcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR Prior0_df = -2
Replace ALL PM_P0Def WITH mcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR Prior0_df = -3
Replace ALL PM_P0Def WITH mcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR Prior0_df = -4
Replace ALL PM_P0Def WITH mcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior0_df = -5

Replace ALL PM_P0DCo WITH (PM_P0Def * m_Size * FixingS) * payRate
```

```
* Duration
Replace ALL PM_P0DR WITH PM_P0Ef/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PM_P0Slack WITH PM_P0DR - M_Ideal
Replace ALL pm_p0Sco WITH  -1 * pm_p0slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR
pm_p0Slack <= 0
Replace ALL pm_p0Oco WITH  pm_p0slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR pm_p0Slack >
0

* Knowledge Transfer based on survey responses
Replace ALL PM_P0KT WITH SM_KT FOR Prior0_kt = 0
Replace ALL PM_P0KT WITH SM_KT * (1 +  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior0_kt = 1
Replace ALL PM_P0KT WITH SM_KT * (1 +  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior0_kt = 2
Replace ALL PM_P0KT WITH SM_KT * (1 +  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior0_kt = 3
Replace ALL PM_P0KT WITH SM_KT * (1 +  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior0_kt = 4
Replace ALL PM_P0KT WITH SM_KT * (1 +  ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior0_kt = 5
Replace ALL PM_P0KT WITH SM_KT * (1 +  ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior0_kt = 6
Replace ALL PM_P0KT WITH SM_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior0_kt = -1
Replace ALL PM_P0KT WITH SM_KT * (1  -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior0_kt = -2
Replace ALL PM_P0KT WITH SM_KT * (1  -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior0_kt = -3
Replace ALL PM_P0KT WITH SM_KT * (1  -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior0_kt = -4
Replace ALL PM_P0KT WITH SM_KT * (1  -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior0_kt = -5

* Convert KT into training cost
Replace ALL PM_P0TC10 WITH (TrainingCM - Pct10 * TrainingCM * PM_P0KT)
Replace ALL PM_P0TC20 WITH (TrainingCM - Pct20 * TrainingCM * PM_P0KT)
Replace ALL PM_P0TC30 WITH (TrainingCM - Pct30 * TrainingCM * PM_P0KT)
Replace ALL PM_P0TC40 WITH (TrainingCM - Pct40 * TrainingCM * PM_P0KT)
Replace ALL PM_P0TC50 WITH (TrainingCM - Pct50 * TrainingCM * PM_P0KT)
Replace ALL PM_P0TC60 WITH (TrainingCM - Pct60 * TrainingCM * PM_P0KT)
Replace ALL PM_P0TC70 WITH (TrainingCM - Pct70 * TrainingCM * PM_P0KT)
Replace ALL PM_P0TC80 WITH (TrainingCM - Pct80 * TrainingCM * PM_P0KT)
Replace ALL PM_P0TC90 WITH (TrainingCM - Pct90 * TrainingCM * PM_P0KT)
Replace ALL PM_P0TC100 WITH (TrainingCM - Pct100 * TrainingCM * PM_P0KT)


**********************************************************************************
** PAIR HIGH complexity projects and PRIOR PAIR EXPERIENCE (NEITHER HAS EXPERIENCE)
********************************************************************************** Effort
based on survey responses
Replace ALL PH_P0Ef WITH hcx_hrs for Prior0_ef = 0
Replace ALL PH_P0Ef WITH hcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
Prior0_ef = 1
```

```
Replace ALL PH_P0Ef WITH hcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior0_ef = 2
Replace ALL PH_P0Ef WITH hcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior0_ef = 3
Replace ALL PH_P0Ef WITH hcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior0_ef = 4
Replace ALL PH_P0Ef WITH hcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
Prior0_ef = 5
Replace ALL PH_P0Ef WITH hcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
Prior0_ef = 6
Replace ALL PH_P0Ef WITH hcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Prior0_ef = -1
Replace ALL PH_P0Ef WITH hcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior0_ef = -2
Replace ALL PH_P0Ef WITH hcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior0_ef = -3
Replace ALL PH_P0Ef WITH hcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior0_ef = -4
Replace ALL PH_P0Ef WITH hcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  for
Prior0_ef= -5

Replace ALL PH_P0ECo WITH PH_P0Ef * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
Replace ALL PH_P0Def WITH hcx_Defect FOR Prior0_df = 0
Replace ALL PH_P0Def WITH hcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior0_df = 1
Replace ALL PH_P0Def WITH hcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior0_df = 2
Replace ALL PH_P0Def WITH hcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior0_df = 3
Replace ALL PH_P0Def WITH hcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior0_df = 4
Replace ALL PH_P0Def WITH hcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior0_df = 5
Replace ALL PH_P0Def WITH hcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior0_df= 6
Replace ALL PH_P0Def WITH hcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior0_df = -1
Replace ALL PH_P0Def WITH hcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior0_df = -2
Replace ALL PH_P0Def WITH hcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior0_df = -3
Replace ALL PH_P0Def WITH hcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior0_df = -4
Replace ALL PH_P0Def WITH hcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior0_df= -5

Replace ALL PH_P0DCo WITH (PH_P0Def * h_Size * FixingS) * payRate

* Duration
Replace ALL PH_P0DR WITH PH_P0Ef/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PH_P0Slack WITH PH_P0DR - H_Ideal
```

```
Replace ALL ph_p0Sco WITH  -1 * ph_p0slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR ph_p0Slack
<= 0
Replace ALL ph_p0Oco WITH  ph_p0slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR ph_p0Slack > 0

* Knowledge Transfer based on survey responses
Replace ALL PH_P0KT WITH SH_KT FOR Prior0_kt = 0
Replace ALL PH_P0KT WITH SH_KT * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior0_kt = 1
Replace ALL PH_P0KT WITH SH_KT * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior0_kt  = 2
Replace ALL PH_P0KT WITH SH_KT * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior0_kt  = 3
Replace ALL PH_P0KT WITH SH_KT * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior0_kt = 4
Replace ALL PH_P0KT WITH SH_KT * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior0_kt  = 5
Replace ALL PH_P0KT WITH SH_KT * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior0_kt  = 6
Replace ALL PH_P0KT WITH SH_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior0_kt  = -1
Replace ALL PH_P0KT WITH SH_KT * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior0_kt  = -2
Replace ALL PH_P0KT WITH SH_KT * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior0_kt  = -3
Replace ALL PH_P0KT WITH SH_KT * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior0_kt  = -4
Replace ALL PH_P0KT WITH SH_KT * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  FOR
Prior0_kt = -5

* Convert KT into training cost
Replace ALL PH_P0TC10 WITH (TrainingCH - Pct10 * TrainingCH * PH_P0KT)
Replace ALL PH_P0TC20 WITH (TrainingCH - Pct20 * TrainingCH * PH_P0KT)
Replace ALL PH_P0TC30 WITH (TrainingCH - Pct30 * TrainingCH * PH_P0KT)
Replace ALL PH_P0TC40 WITH (TrainingCH - Pct40 * TrainingCH * PH_P0KT)
Replace ALL PH_P0TC50 WITH (TrainingCH - Pct50 * TrainingCH * PH_P0KT)
Replace ALL PH_P0TC60 WITH (TrainingCH - Pct60 * TrainingCH * PH_P0KT)
Replace ALL PH_P0TC70 WITH (TrainingCH - Pct70 * TrainingCH * PH_P0KT)
Replace ALL PH_P0TC80 WITH (TrainingCH - Pct80 * TrainingCH * PH_P0KT)
Replace ALL PH_P0TC90 WITH (TrainingCH - Pct90 * TrainingCH * PH_P0KT)
Replace ALL PH_P0TC100 WITH (TrainingCH - Pct100 * TrainingCH * PH_P0KT)


********************************************************************************************** PAIR
LOW complexity projects and PRIOR PAIR EXPERIENCE (ONE HAS EXPERIENCE)
**********************************************************************************
* Effort based on survey responses
Replace ALL PL_P1Ef WITH lcx_hrs for Prior1_ef = 0
Replace ALL PL_P1Ef WITH lcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
Prior1_ef = 1
Replace ALL PL_P1Ef WITH lcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior1_ef = 2
Replace ALL PL_P1Ef WITH lcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior1_ef = 3
Replace ALL PL_P1Ef WITH lcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior1_ef = 4
```

```
Replace ALL PL_P1Ef WITH lcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
Prior1_ef = 5
Replace ALL PL_P1Ef WITH lcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
Prior1_ef = 6
Replace ALL PL_P1Ef WITH lcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Prior1_ef = -1
Replace ALL PL_P1Ef WITH lcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior1_ef = -2
Replace ALL PL_P1Ef WITH lcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior1_ef = -3
Replace ALL PL_P1Ef WITH lcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior1_ef = -4
Replace ALL PL_P1Ef WITH lcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  for
Prior1_ef = -5

Replace ALL PL_P1ECo WITH PL_P1Ef * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
Replace ALL PL_P1Def WITH lcx_Defect FOR Prior1_df = 0
Replace ALL PL_P1Def WITH lcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior1_df = 1
Replace ALL PL_P1Def WITH lcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior1_df = 2
Replace ALL PL_P1Def WITH lcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior1_df = 3
Replace ALL PL_P1Def WITH lcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior1_df = 4
Replace ALL PL_P1Def WITH lcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior1_df = 5
Replace ALL PL_P1Def WITH lcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior1_df = 6
Replace ALL PL_P1Def WITH lcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior1_df = -1
Replace ALL PL_P1Def WITH lcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior1_df = -2
Replace ALL PL_P1Def WITH lcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior1_df = -3
Replace ALL PL_P1Def WITH lcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior1_df = -4
Replace ALL PL_P1Def WITH lcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior1_df = -5

Replace ALL PL_P1DCo WITH (PL_P1Def * l_Size * FixingS) * payRate

* Duration
Replace ALL PL_P1DR WITH PL_P1Ef/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PL_P1Slack WITH PL_P1DR - L_Ideal
Replace ALL pl_p1Sco WITH  -1 * pl_p1slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR pl_p1Slack
<= 0
Replace ALL pl_p1Oco WITH  pl_p1slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR pl_p1Slack > 0

* Knowledge Transfer based on survey responses
Replace ALL PL_P1KT WITH SL_KT FOR Prior1_kt = 0
```

```
Replace ALL PL_P1KT WITH SL_KT * (1 +  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior1_kt = 1
Replace ALL PL_P1KT WITH SL_KT * (1 +  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior1_kt = 2
Replace ALL PL_P1KT WITH SL_KT * (1 +  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior1_kt = 3
Replace ALL PL_P1KT WITH SL_KT * (1 +  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior1_kt = 4
Replace ALL PL_P1KT WITH SL_KT * (1 +  ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior1_kt = 5
Replace ALL PL_P1KT WITH SL_KT * (1 +  ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior1_kt = 6
Replace ALL PL_P1KT WITH SL_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior1_kt = -1
Replace ALL PL_P1KT WITH SL_KT * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior1_kt = -2
Replace ALL PL_P1KT WITH SL_KT * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior1_kt = -3
Replace ALL PL_P1KT WITH SL_KT * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior1_kt = -4
Replace ALL PL_P1KT WITH SL_KT * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  FOR
Prior1_kt = -5

* Convert KT into training cost
Replace ALL PL_P1TC10 WITH (TrainingCL - Pct10 * TrainingCL * PL_P1KT)
Replace ALL PL_P1TC20 WITH (TrainingCL - Pct20 * TrainingCL * PL_P1KT)
Replace ALL PL_P1TC30 WITH (TrainingCL - Pct30 * TrainingCL * PL_P1KT)
Replace ALL PL_P1TC40 WITH (TrainingCL - Pct40 * TrainingCL * PL_P1KT)
Replace ALL PL_P1TC50 WITH (TrainingCL - Pct50 * TrainingCL * PL_P1KT)
Replace ALL PL_P1TC60 WITH (TrainingCL - Pct60 * TrainingCL * PL_P1KT)
Replace ALL PL_P1TC70 WITH (TrainingCL - Pct70 * TrainingCL * PL_P1KT)
Replace ALL PL_P1TC80 WITH (TrainingCL - Pct80 * TrainingCL * PL_P1KT)
Replace ALL PL_P1TC90 WITH (TrainingCL - Pct90 * TrainingCL * PL_P1KT)
Replace ALL PL_P1TC100 WITH (TrainingCL - Pct100 * TrainingCL * PL_P1KT)

********************************************************************************************** PAIR
MEDIUM complexity projects and PRIOR PAIR EXPERIENCE (ONE HAS EXPERIENCE)
**********************************************************************************
* Effort based on survey responses
Replace ALL PM_P1Ef WITH mcx_hrs for Prior1_ef = 0
Replace ALL PM_P1Ef WITH mcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
Prior1_ef = 1
Replace ALL PM_P1Ef WITH mcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior1_ef = 2
Replace ALL PM_P1Ef WITH mcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior1_ef = 3
Replace ALL PM_P1Ef WITH mcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior1_ef = 4
Replace ALL PM_P1Ef WITH mcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
Prior1_ef = 5
Replace ALL PM_P1Ef WITH mcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
Prior1_ef = 6
Replace ALL PM_P1Ef WITH mcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Prior1_ef = -1
Replace ALL PM_P1Ef WITH mcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior1_ef = -2
```

```
Replace ALL PM_P1Ef WITH mcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior1_ef = -3
Replace ALL PM_P1Ef WITH mcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior1_ef= -4
Replace ALL PM_P1Ef WITH mcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  for
Prior1_ef = -5

Replace ALL PM_P1ECo WITH PM_P1Ef * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
Replace ALL PM_P1Def WITH mcx_Defect FOR Prior1_df = 0
Replace ALL PM_P1Def WITH mcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR Prior1_df = 1
Replace ALL PM_P1Def WITH mcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR Prior1_df = 2
Replace ALL PM_P1Def WITH mcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR Prior1_df = 3
Replace ALL PM_P1Def WITH mcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR Prior1_df = 4
Replace ALL PM_P1Def WITH mcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2))
FOR Prior1_df = 5
Replace ALL PM_P1Def WITH mcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2))
FOR Prior1_df = 6
Replace ALL PM_P1Def WITH mcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR Prior1_df = -1
Replace ALL PM_P1Def WITH mcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR Prior1_df = -2
Replace ALL PM_P1Def WITH mcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR Prior1_df = -3
Replace ALL PM_P1Def WITH mcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR Prior1_df = -4
Replace ALL PM_P1Def WITH mcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior1_df = -5

Replace ALL PM_P1DCo WITH (PM_P1Def * m_Size * FixingS) * payRate

* Duration
Replace ALL PM_P1DR WITH PM_P1Ef/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PM_P1Slack WITH PM_P1DR - M_Ideal
Replace ALL pm_p1Sco WITH  -1 * pm_p1slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR
pm_p1Slack <= 0
Replace ALL pm_p1Oco WITH  pm_p1slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR pm_p1Slack >
0

* Knowledge Transfer based on survey responses
Replace ALL PM_P1KT WITH SM_KT FOR Prior1_kt = 0
Replace ALL PM_P1KT WITH SM_KT * (1 +  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior1_kt = 1
Replace ALL PM_P1KT WITH SM_KT * (1 +  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior1_kt = 2
Replace ALL PM_P1KT WITH SM_KT * (1 +  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior1_kt = 3
Replace ALL PM_P1KT WITH SM_KT * (1 +  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior1_kt = 4
```

```
Replace ALL PM_P1KT WITH SM_KT * (1 +  ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior1_kt = 5
Replace ALL PM_P1KT WITH SM_KT * (1 +  ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior1_kt = 6
Replace ALL PM_P1KT WITH SM_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior1_kt = -1
Replace ALL PM_P1KT WITH SM_KT * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior1_kt = -2
Replace ALL PM_P1KT WITH SM_KT * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior1_kt = -3
Replace ALL PM_P1KT WITH SM_KT * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior1_kt = -4
Replace ALL PM_P1KT WITH SM_KT * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior1_kt = -5

* Convert KT into training cost
Replace ALL PM_P1TC10 WITH (TrainingCM - Pct10 * TrainingCM * PM_P1KT)
Replace ALL PM_P1TC20 WITH (TrainingCM - Pct20 * TrainingCM * PM_P1KT)
Replace ALL PM_P1TC30 WITH (TrainingCM - Pct30 * TrainingCM * PM_P1KT)
Replace ALL PM_P1TC40 WITH (TrainingCM - Pct40 * TrainingCM * PM_P1KT)
Replace ALL PM_P1TC50 WITH (TrainingCM - Pct50 * TrainingCM * PM_P1KT)
Replace ALL PM_P1TC60 WITH (TrainingCM - Pct60 * TrainingCM * PM_P1KT)
Replace ALL PM_P1TC70 WITH (TrainingCM - Pct70 * TrainingCM * PM_P1KT)
Replace ALL PM_P1TC80 WITH (TrainingCM - Pct80 * TrainingCM * PM_P1KT)
Replace ALL PM_P1TC90 WITH (TrainingCM - Pct90 * TrainingCM * PM_P1KT)
Replace ALL PM_P1TC100 WITH (TrainingCM - Pct100 * TrainingCM * PM_P1KT)


********************************************************************************************** PAIR
HIGH complexity projects and PRIOR PAIR EXPERIENCE (ONE HAS EXPERIENCE)
**********************************************************************************
* Effort based on survey responses
Replace ALL PH_P1Ef WITH hcx_hrs for Prior1_ef = 0
Replace ALL PH_P1Ef WITH hcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
Prior1_ef = 1
Replace ALL PH_P1Ef WITH hcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior1_ef = 2
Replace ALL PH_P1Ef WITH hcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior1_ef = 3
Replace ALL PH_P1Ef WITH hcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior1_ef = 4
Replace ALL PH_P1Ef WITH hcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
Prior1_ef = 5
Replace ALL PH_P1Ef WITH hcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
Prior1_ef = 6
Replace ALL PH_P1Ef WITH hcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Prior1_ef = -1
Replace ALL PH_P1Ef WITH hcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior1_ef = -2
Replace ALL PH_P1Ef WITH hcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior1_ef = -3
Replace ALL PH_P1Ef WITH hcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior1_ef = -4
Replace ALL PH_P1Ef WITH hcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  for
Prior1_ef= -5
```

```
Replace ALL PH_P1ECo WITH PH_P1Ef * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
Replace ALL PH_P1Def WITH hcx_Defect FOR Prior1_df = 0
Replace ALL PH_P1Def WITH hcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior1_df = 1
Replace ALL PH_P1Def WITH hcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior1_df = 2
Replace ALL PH_P1Def WITH hcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior1_df = 3
Replace ALL PH_P1Def WITH hcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior1_df = 4
Replace ALL PH_P1Def WITH hcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior1_df = 5
Replace ALL PH_P1Def WITH hcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior1_df= 6
Replace ALL PH_P1Def WITH hcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior1_df = -1
Replace ALL PH_P1Def WITH hcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior1_df = -2
Replace ALL PH_P1Def WITH hcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior1_df = -3
Replace ALL PH_P1Def WITH hcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior1_df = -4
Replace ALL PH_P1Def WITH hcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior1_df= -5

Replace ALL PH_P1DCo WITH (PH_P1Def * h_Size * FixingS) * payRate

* Duration
Replace ALL PH_P1DR WITH PH_P1Ef/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PH_P1Slack WITH PH_P1DR - H_Ideal
Replace ALL ph_p1Sco WITH  -1 * ph_p1slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR ph_p1Slack
<= 0
Replace ALL ph_p1Oco WITH  ph_p1slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR ph_p1Slack > 0

* Knowledge Transfer based on survey responses
Replace ALL PH_P1KT WITH SH_KT FOR Prior1_kt = 0
Replace ALL PH_P1KT WITH SH_KT * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior1_kt = 1
Replace ALL PH_P1KT WITH SH_KT * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior1_kt  = 2
Replace ALL PH_P1KT WITH SH_KT * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior1_kt  = 3
Replace ALL PH_P1KT WITH SH_KT * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior1_kt = 4
Replace ALL PH_P1KT WITH SH_KT * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior1_kt  = 5
Replace ALL PH_P1KT WITH SH_KT * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior1_kt  = 6
Replace ALL PH_P1KT WITH SH_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior1_kt  = -1
Replace ALL PH_P1KT WITH SH_KT * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior1_kt  = -2
```

```
Replace ALL PH_P1KT WITH SH_KT * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior1_kt  = -3
Replace ALL PH_P1KT WITH SH_KT * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior1_kt  = -4
Replace ALL PH_P1KT WITH SH_KT * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  FOR
Prior1_kt = -5

* Convert KT into training cost
Replace ALL PH_P1TC10 WITH (TrainingCH - Pct10 * TrainingCH * PH_P1KT)
Replace ALL PH_P1TC20 WITH (TrainingCH - Pct20 * TrainingCH * PH_P1KT)
Replace ALL PH_P1TC30 WITH (TrainingCH - Pct30 * TrainingCH * PH_P1KT)
Replace ALL PH_P1TC40 WITH (TrainingCH - Pct40 * TrainingCH * PH_P1KT)
Replace ALL PH_P1TC50 WITH (TrainingCH - Pct50 * TrainingCH * PH_P1KT)
Replace ALL PH_P1TC60 WITH (TrainingCH - Pct60 * TrainingCH * PH_P1KT)
Replace ALL PH_P1TC70 WITH (TrainingCH - Pct70 * TrainingCH * PH_P1KT)
Replace ALL PH_P1TC80 WITH (TrainingCH - Pct80 * TrainingCH * PH_P1KT)
Replace ALL PH_P1TC90 WITH (TrainingCH - Pct90 * TrainingCH * PH_P1KT)
Replace ALL PH_P1TC100 WITH (TrainingCH - Pct100 * TrainingCH * PH_P1KT)


********************************************************************************** PAIR
LOW complexity projects and PRIOR PAIR EXPERIENCE (BOTH HAVE EXPERIENCE)
**********************************************************************************
* Effort based on survey responses
Replace ALL PL_P2Ef WITH lcx_hrs for Prior2_ef = 0
Replace ALL PL_P2Ef WITH lcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
Prior2_ef = 1
Replace ALL PL_P2Ef WITH lcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior2_ef = 2
Replace ALL PL_P2Ef WITH lcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior2_ef = 3
Replace ALL PL_P2Ef WITH lcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior2_ef = 4
Replace ALL PL_P2Ef WITH lcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
Prior2_ef = 5
Replace ALL PL_P2Ef WITH lcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
Prior2_ef = 6
Replace ALL PL_P2Ef WITH lcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Prior2_ef = -1
Replace ALL PL_P2Ef WITH lcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior2_ef = -2
Replace ALL PL_P2Ef WITH lcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior2_ef = -3
Replace ALL PL_P2Ef WITH lcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior2_ef = -4
Replace ALL PL_P2Ef WITH lcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  for
Prior2_ef = -5

Replace ALL PL_P2ECo WITH PL_P2Ef * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
Replace ALL PL_P2Def WITH lcx_Defect FOR Prior2_df = 0
Replace ALL PL_P2Def WITH lcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior2_df = 1
Replace ALL PL_P2Def WITH lcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior2_df = 2
```

```
Replace ALL PL_P2Def WITH lcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior2_df = 3
Replace ALL PL_P2Def WITH lcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior2_df = 4
Replace ALL PL_P2Def WITH lcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior2_df = 5
Replace ALL PL_P2Def WITH lcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior2_df = 6
Replace ALL PL_P2Def WITH lcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior2_df = -1
Replace ALL PL_P2Def WITH lcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior2_df = -2
Replace ALL PL_P2Def WITH lcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior2_df = -3
Replace ALL PL_P2Def WITH lcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior2_df = -4
Replace ALL PL_P2Def WITH lcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior2_df = -5

Replace ALL PL_P2DCo WITH (PL_P2Def * l_Size * FixingS) * payRate

* Duration
Replace ALL PL_P2DR WITH PL_P2Ef/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PL_P2Slack WITH PL_P2DR - L_Ideal
Replace ALL pl_p2Sco WITH  -1 * pl_p2slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR pl_p2Slack
<= 0
Replace ALL pl_p2Oco WITH  pl_p2slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR pl_p2Slack > 0

* Knowledge Transfer based on survey responses
Replace ALL PL_P2KT WITH SL_KT FOR Prior2_kt = 0
Replace ALL PL_P2KT WITH SL_KT * (1 +  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior2_kt = 1
Replace ALL PL_P2KT WITH SL_KT * (1 +  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior2_kt = 2
Replace ALL PL_P2KT WITH SL_KT * (1 +  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior2_kt = 3
Replace ALL PL_P2KT WITH SL_KT * (1 +  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior2_kt = 4
Replace ALL PL_P2KT WITH SL_KT * (1 +  ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior2_kt = 5
Replace ALL PL_P2KT WITH SL_KT * (1 +  ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior2_kt = 6
Replace ALL PL_P2KT WITH SL_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior2_kt = -1
Replace ALL PL_P2KT WITH SL_KT * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior2_kt = -2
Replace ALL PL_P2KT WITH SL_KT * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior2_kt = -3
Replace ALL PL_P2KT WITH SL_KT * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior2_kt = -4
Replace ALL PL_P2KT WITH SL_KT * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  FOR
Prior2_kt = -5

* Convert KT into training cost
```

```
Replace ALL PL_P2TC10 WITH (TrainingCL - Pct10 * TrainingCL * PL_P2KT)
Replace ALL PL_P2TC20 WITH (TrainingCL - Pct20 * TrainingCL * PL_P2KT)
Replace ALL PL_P2TC30 WITH (TrainingCL - Pct30 * TrainingCL * PL_P2KT)
Replace ALL PL_P2TC40 WITH (TrainingCL - Pct40 * TrainingCL * PL_P2KT)
Replace ALL PL_P2TC50 WITH (TrainingCL - Pct50 * TrainingCL * PL_P2KT)
Replace ALL PL_P2TC60 WITH (TrainingCL - Pct60 * TrainingCL * PL_P2KT)
Replace ALL PL_P2TC70 WITH (TrainingCL - Pct70 * TrainingCL * PL_P2KT)
Replace ALL PL_P2TC80 WITH (TrainingCL - Pct80 * TrainingCL * PL_P2KT)
Replace ALL PL_P2TC90 WITH (TrainingCL - Pct90 * TrainingCL * PL_P2KT)
Replace ALL PL_P2TC100 WITH (TrainingCL - Pct100 * TrainingCL * PL_P2KT)


*********************************************************************************************
** PAIR MEDIUM complexity projects and PRIOR PAIR EXPERIENCE (BOTH HAVE EXPERIENCE)
*********************************************************************************************
* Effort based on survey responses
Replace ALL PM_P2Ef WITH mcx_hrs for Prior2_ef = 0
Replace ALL PM_P2Ef WITH mcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
Prior2_ef = 1
Replace ALL PM_P2Ef WITH mcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior2_ef = 2
Replace ALL PM_P2Ef WITH mcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior2_ef = 3
Replace ALL PM_P2Ef WITH mcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior2_ef = 4
Replace ALL PM_P2Ef WITH mcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
Prior2_ef = 5
Replace ALL PM_P2Ef WITH mcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
Prior2_ef = 6
Replace ALL PM_P2Ef WITH mcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Prior2_ef = -1
Replace ALL PM_P2Ef WITH mcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior2_ef = -2
Replace ALL PM_P2Ef WITH mcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior2_ef = -3
Replace ALL PM_P2Ef WITH mcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior2_ef= -4
Replace ALL PM_P2Ef WITH mcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  for
Prior2_ef = -5

Replace ALL PM_P2ECo WITH PM_P2Ef * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
Replace ALL PM_P2Def WITH mcx_Defect FOR Prior2_df = 0
Replace ALL PM_P2Def WITH mcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR Prior2_df = 1
Replace ALL PM_P2Def WITH mcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR Prior2_df = 2
Replace ALL PM_P2Def WITH mcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR Prior2_df = 3
Replace ALL PM_P2Def WITH mcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR Prior2_df = 4
Replace ALL PM_P2Def WITH mcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2))
FOR Prior2_df = 5
Replace ALL PM_P2Def WITH mcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2))
FOR Prior2_df = 6
```

```
Replace ALL PM_P2Def WITH mcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))
FOR Prior2_df = -1
Replace ALL PM_P2Def WITH mcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2))
FOR Prior2_df = -2
Replace ALL PM_P2Def WITH mcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2))
FOR Prior2_df = -3
Replace ALL PM_P2Def WITH mcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2))
FOR Prior2_df = -4
Replace ALL PM_P2Def WITH mcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior2_df = -5

Replace ALL PM_P2DCo WITH (PM_P2Def * m_Size * FixingS) * payRate

* Duration
Replace ALL PM_P2DR WITH PM_P2Ef/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PM_P2Slack WITH PM_P2DR - M_Ideal
Replace ALL pm_p2Sco WITH  -1 * pm_p2slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR
pm_p2Slack <= 0
Replace ALL pm_p2Oco WITH  pm_p2slack * HrsPerday8 *  team * ((payrate + payrate2) / 2) FOR pm_p2Slack >
0

* Knowledge Transfer based on survey responses
Replace ALL PM_P2KT WITH SM_KT FOR Prior2_kt = 0
Replace ALL PM_P2KT WITH SM_KT * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior2_kt = 1
Replace ALL PM_P2KT WITH SM_KT * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior2_kt = 2
Replace ALL PM_P2KT WITH SM_KT * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior2_kt = 3
Replace ALL PM_P2KT WITH SM_KT * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior2_kt = 4
Replace ALL PM_P2KT WITH SM_KT * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior2_kt = 5
Replace ALL PM_P2KT WITH SM_KT * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior2_kt = 6
Replace ALL PM_P2KT WITH SM_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior2_kt = -1
Replace ALL PM_P2KT WITH SM_KT * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior2_kt = -2
Replace ALL PM_P2KT WITH SM_KT * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior2_kt = -3
Replace ALL PM_P2KT WITH SM_KT * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior2_kt = -4
Replace ALL PM_P2KT WITH SM_KT * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior2_kt = -5

* Convert KT into training cost
Replace ALL PM_P2TC10 WITH (TrainingCM - Pct10 * TrainingCM * PM_P2KT)
Replace ALL PM_P2TC20 WITH (TrainingCM - Pct20 * TrainingCM * PM_P2KT)
Replace ALL PM_P2TC30 WITH (TrainingCM - Pct30 * TrainingCM * PM_P2KT)
Replace ALL PM_P2TC40 WITH (TrainingCM - Pct40 * TrainingCM * PM_P2KT)
Replace ALL PM_P2TC50 WITH (TrainingCM - Pct50 * TrainingCM * PM_P2KT)
Replace ALL PM_P2TC60 WITH (TrainingCM - Pct60 * TrainingCM * PM_P2KT)
Replace ALL PM_P2TC70 WITH (TrainingCM - Pct70 * TrainingCM * PM_P2KT)
```

```
Replace ALL PM_P2TC80 WITH (TrainingCM - Pct80 * TrainingCM * PM_P2KT)
Replace ALL PM_P2TC90 WITH (TrainingCM - Pct90 * TrainingCM * PM_P2KT)
Replace ALL PM_P2TC100 WITH (TrainingCM - Pct100 * TrainingCM * PM_P2KT)

********************************************************************************************** PAIR
HIGH complexity projects and PRIOR PAIR EXPERIENCE (BOTH HAVE EXPERIENCE)
**********************************************************************************
* Effort based on survey responses
Replace ALL PH_P2Ef WITH hcx_hrs for Prior2_ef = 0
Replace ALL PH_P2Ef WITH hcx_hrs * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  for
Prior2_ef = 1
Replace ALL PH_P2Ef WITH hcx_hrs * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior2_ef = 2
Replace ALL PH_P2Ef WITH hcx_hrs * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior2_ef = 3
Replace ALL PH_P2Ef WITH hcx_hrs * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior2_ef = 4
Replace ALL PH_P2Ef WITH hcx_hrs * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) for
Prior2_ef = 5
Replace ALL PH_P2Ef WITH hcx_hrs * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) for
Prior2_ef = 6
Replace ALL PH_P2Ef WITH hcx_hrs * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) for
Prior2_ef = -1
Replace ALL PH_P2Ef WITH hcx_hrs * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) for
Prior2_ef = -2
Replace ALL PH_P2Ef WITH hcx_hrs * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) for
Prior2_ef = -3
Replace ALL PH_P2Ef WITH hcx_hrs * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) for
Prior2_ef = -4
Replace ALL PH_P2Ef WITH hcx_hrs * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  for
Prior2_ef= -5

Replace ALL PH_P2ECo WITH PH_P2Ef * ROUND((payRate + payRate2)/2, 2)

* Defect based on survey responses
Replace ALL PH_P2Def WITH hcx_Defect FOR Prior2_df = 0
Replace ALL PH_P2Def WITH hcx_Defect * (1 + ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior2_df = 1
Replace ALL PH_P2Def WITH hcx_Defect * (1 + ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior2_df = 2
Replace ALL PH_P2Def WITH hcx_Defect * (1 + ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior2_df = 3
Replace ALL PH_P2Def WITH hcx_Defect * (1 + ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior2_df = 4
Replace ALL PH_P2Def WITH hcx_Defect * (1 + ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior2_df = 5
Replace ALL PH_P2Def WITH hcx_Defect * (1 + ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior2_df= 6
Replace ALL PH_P2Def WITH hcx_Defect * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior2_df = -1
Replace ALL PH_P2Def WITH hcx_Defect * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior2_df = -2
Replace ALL PH_P2Def WITH hcx_Defect * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior2_df = -3
Replace ALL PH_P2Def WITH hcx_Defect * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior2_df = -4
```

```
Replace ALL PH_P2Def WITH hcx_Defect * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))
FOR Prior2_df= -5

Replace ALL PH_P2DCo WITH (PH_P2Def * h_Size * FixingS) * payRate

* Duration
Replace ALL PH_P2DR WITH PH_P2Ef/(Team * HrsPerDay7)

*LABOR SLACK & Opportunity Cost
Replace ALL PH_P2Slack WITH PH_P2DR - H_Ideal
Replace ALL ph_p2Sco WITH  -1 * ph_p2slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR ph_p2Slack
<= 0
Replace ALL ph_p2Oco WITH  ph_p2slack * HrsPerDay8 * team * ((payrate + payrate2) / 2) FOR ph_p2Slack > 0

* Knowledge Transfer based on survey responses
Replace ALL PH_P2KT WITH SH_KT FOR Prior2_kt = 0
Replace ALL PH_P2KT WITH SH_KT * (1 +  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2))  FOR
Prior2_kt = 1
Replace ALL PH_P2KT WITH SH_KT * (1 +  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior2_kt  = 2
Replace ALL PH_P2KT WITH SH_KT * (1 +  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior2_kt  = 3
Replace ALL PH_P2KT WITH SH_KT * (1 +  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior2_kt = 4
Replace ALL PH_P2KT WITH SH_KT * (1 +  ROUND((DIF5H - DIF5L + 0.01) * RAND() + DIF5L, 2)) FOR
Prior2_kt  = 5
Replace ALL PH_P2KT WITH SH_KT * (1 +  ROUND((DIF6H - DIF6L + 0.01) * RAND() + DIF6L, 2)) FOR
Prior2_kt  = 6
Replace ALL PH_P2KT WITH SH_KT * (1 -  ROUND((DIF1H - DIF1L + 0.01) * RAND() + DIF1L, 2)) FOR
Prior2_kt  = -1
Replace ALL PH_P2KT WITH SH_KT * (1 -  ROUND((DIF2H - DIF2L + 0.01) * RAND() + DIF2L, 2)) FOR
Prior2_kt  = -2
Replace ALL PH_P2KT WITH SH_KT * (1 -  ROUND((DIF3H - DIF3L + 0.01) * RAND() + DIF3L, 2)) FOR
Prior2_kt  = -3
Replace ALL PH_P2KT WITH SH_KT * (1 -  ROUND((DIF4H - DIF4L + 0.01) * RAND() + DIF4L, 2)) FOR
Prior2_kt  = -4
Replace ALL PH_P2KT WITH SH_KT * (1 -  ROUND((DIF5H2 - DIF5L2 + 0.01) * RAND() + DIF5L2, 2))  FOR
Prior2_kt = -5

* Convert KT into training cost
Replace ALL PH_P2TC10 WITH (TrainingCH - Pct10 * TrainingCH * PH_P2KT)
Replace ALL PH_P2TC20 WITH (TrainingCH - Pct20 * TrainingCH * PH_P2KT)
Replace ALL PH_P2TC30 WITH (TrainingCH - Pct30 * TrainingCH * PH_P2KT)
Replace ALL PH_P2TC40 WITH (TrainingCH - Pct40 * TrainingCH * PH_P2KT)
Replace ALL PH_P2TC50 WITH (TrainingCH - Pct50 * TrainingCH * PH_P2KT)
Replace ALL PH_P2TC60 WITH (TrainingCH - Pct60 * TrainingCH * PH_P2KT)
Replace ALL PH_P2TC70 WITH (TrainingCH - Pct70 * TrainingCH * PH_P2KT)
Replace ALL PH_P2TC80 WITH (TrainingCH - Pct80 * TrainingCH * PH_P2KT)
Replace ALL PH_P2TC90 WITH (TrainingCH - Pct90 * TrainingCH * PH_P2KT)
Replace ALL PH_P2TC100 WITH (TrainingCH - Pct100 * TrainingCH * PH_P2KT)

*********************************************************************************
* Create datasets for bootstrap and analysis
*********************************************************************************
* Data for hypothesis testing
```

```
SELECT Respondent, PairYes, Lcx_Hrs, Mcx_Hrs, Hcx_Hrs, Lcx_Defect, Mcx_Defect, Hcx_Defect, kt1, kt2, kt3,
Lcx_ef, Mcx_ef, Hcx_ef, Jr2_ef, Jrsr_ef, Sr2_ef, Prior0_ef, Prior1_ef, Prior2_ef,
Lcx_df, Mcx_df, Hcx_df, Jr2_df, Jrsr_df, Sr2_df, Prior0_df, Prior1_df, Prior2_df,
Lcx_kt, Mcx_kt, Hcx_kt, Jr2_kt, Jrsr_kt, Sr2_kt, Prior0_kt, Prior1_kt, Prior2_kt
FROM c1
into CURSOR cCursor1

COPY TO hypoData.dbf TYPE foxplus

* Data for Cost information

SELECT c1.respondent, PairYes, c1.team, c1.fixingS, c1.lcx_hrs, c1.mcx_hrs, c1.hcx_hrs,  c1.lcx_defect,
c1.mcx_defect, c1.hcx_defect, c1.kt1, c1.kt2, c1.kt3, ;
000000000.00 as SL_Cost, 000000000.00 as SM_Cost, 000000000.00 as SH_Cost, ;
000000000.00 as PL_Cost, 000000000.00 as PM_Cost, 000000000.00 as PH_Cost, ;
000000000.00 as PL_JJCost, 000000000.00 as PM_JJCost, 000000000.00 as PH_JJCost, ;
000000000.00 as PL_JSCost, 000000000.00 as PM_JSCost, 000000000.00 as PH_JSCost, ;
000000000.00 as PL_SSCost, 000000000.00 as PM_SSCost, 000000000.00 as PH_SSCost, ;
000000000.00 as PL_P0Cost, 000000000.00 as PM_P0Cost, 000000000.00 as PH_P0Cost, ;
000000000.00 as PL_P1Cost, 000000000.00 as PM_P1Cost, 000000000.00 as PH_P1Cost, ;
000000000.00 as PL_P2Cost, 000000000.00 as PM_P2Cost, 000000000.00 as PH_P2Cost, ;
000000000.00 as SL_Cost2, 000000000.00 as SM_Cost2, 000000000.00 as SH_Cost2, ;
000000000.00 as PL_Cost2, 000000000.00 as PM_Cost2, 000000000.00 as PH_Cost2, ;
000000000.00 as PL_JJCost2, 000000000.00 as PM_JJCost2, 000000000.00 as PH_JJCost2, ;
000000000.00 as PL_JSCost2, 000000000.00 as PM_JSCost2, 000000000.00 as PH_JSCost2, ;
000000000.00 as PL_SSCost2, 000000000.00 as PM_SSCost2, 000000000.00 as PH_SSCost2, ;
000000000.00 as PL_P0Cost2, 000000000.00 as PM_P0Cost2, 000000000.00 as PH_P0Cost2, ;
000000000.00 as PL_P1Cost2, 000000000.00 as PM_P1Cost2, 000000000.00 as PH_P1Cost2, ;
000000000.00 as PL_P2Cost2, 000000000.00 as PM_P2Cost2, 000000000.00 as PH_P2Cost2, ;
000000000.00 as SL_EDCost, 000000000.00 as SM_EDCost, 000000000.00 as SH_EDCost, ;
000000000.00 as PL_EDCost, 000000000.00 as PM_EDCost, 000000000.00 as PH_EDCost, ;
000000000.00 as PL_JJEDCo, 000000000.00 as PM_JJEDCo, 000000000.00 as PH_JJEDCo, ;
000000000.00 as PL_JSEDCo, 000000000.00 as PM_JSEDCo, 000000000.00 as PH_JSEDCo, ;
000000000.00 as PL_SSEDCo, 000000000.00 as PM_SSEDCo, 000000000.00 as PH_SSEDCo, ;
000000000.00 as PL_P0EDCo, 000000000.00 as PM_P0EDCo, 000000000.00 as PH_P0EDCo, ;
000000000.00 as PL_P1EDCo, 000000000.00 as PM_P1EDCo, 000000000.00 as PH_P1EDCo, ;
000000000.00 as PL_P2EDCo, 000000000.00 as PM_P2EDCo, 000000000.00 as PH_P2EDCo, ;
SL_ECost, SL_DCost, SL_DR, SL_OCost, SL_SCost, SL_TC40 as SL_TC,  ;
SM_ECost, SM_DCost, SM_DR, SM_OCost, SM_SCost, SM_TC40 as SM_TC,    ;
SH_ECost, SH_DCost, SH_DR, SH_OCost, SH_SCost, SH_TC40 as SH_TC,  ;
PL_ECost, PL_DCost, PL_DR, PL_OCost, PL_SCost, PL_TC40 as PL_TC,   ;
PM_ECost, PM_DCost, PM_DR, PM_OCost, PM_SCost, PM_TC40 as PM_TC,  ;
PH_ECost, PH_DCost, PH_DR, PH_OCost, PH_SCost, PH_TC40 as PH_TC,  ;
PL_JJECo, PL_JJDCo, PL_JJDR, PL_JJOCo, PL_JJSCo, PL_JJTC40 as PL_JJTC, ;
PM_JJECo, PM_JJDCo, PM_JJDR, PM_JJOCo, PM_JJSCo, PM_JJTC40 as PM_JJTC,  ;
PH_JJECo, PH_JJDCo, PH_JJDR, PH_JJOCo, PH_JJSCo, PH_JJTC40 as PH_JJTC,  ;
PL_JSECo, PL_JSDCo, PL_JSDR, PL_JSOCo, PL_JSSCo, PL_JSTC40 as PL_JSTC,  ;
PM_JSECo, PM_JSDCo, PM_JSDR, PM_JSOCo, PM_JSSCo, PM_JSTC40 as PM_JSTC, ;
PH_JSECo, PH_JSDCo, PH_JSDR, PH_JSOCo, PH_JSSCo, PH_JSTC40 as PH_JSTC, ;
PL_SSECo, PL_SSDCo, PL_SSDR, PL_SSOCo, PL_SSSCo, PL_SSTC40 as PL_SSTC, ;
PM_SSECo, PM_SSDCo, PM_SSDR, PM_SSOCo, PM_SSSCo, PM_SSTC40 as PM_SSTC, ;
PH_SSECo, PH_SSDCo, PH_SSDR, PH_SSOCo, PH_SSSCo, PH_SSTC40 as PH_SSTC, ;
PL_P0ECo, PL_P0DCo, PL_P0DR, PL_P0OCo, PL_P0SCo, PL_P0TC40 as PL_P0TC, ;
PM_P0ECo, PM_P0DCo, PM_P0DR, PM_P0OCo, PM_P0SCo, PM_P0TC40 as PM_P0TC, ;
PH_P0ECo, PH_P0DCo, PH_P0DR, PH_P0OCo, PH_P0SCo, PH_P0TC40 as PH_P0TC, ;
PL_P1ECo, PL_P1DCo, PL_P1DR, PL_P1OCo, PL_P1SCo, PL_P1TC40 as PL_P1TC, ;
```

```
PM_P1ECo, PM_P1DCo, PM_P1DR, PM_P1OCo, PM_P1SCo, PM_P1TC40 as PM_P1TC, ;
PH_P1ECo, PH_P1DCo, PH_P1DR, PH_P1OCo, PH_P1SCo, PH_P1TC40 as PH_P1TC, ;
PL_P2ECo, PL_P2DCo, PL_P2DR, PL_P2OCo, PL_P2SCo, PL_P2TC40 as PL_P2TC, ;
PM_P2ECo, PM_P2DCo, PM_P2DR, PM_P2OCo, PM_P2SCo, PM_P2TC40 as PM_P2TC, ;
PH_P2ECo, PH_P2DCo, PH_P2DR, PH_P2OCo, PH_P2SCo, PH_P2TC40 as PH_P2TC;
FROM C1, C2, C3;
WHERE c1.respondent = c2.respondent  AND c1.respondent = c3.respondent ;
INTO table costData

SELECT costData

* Labor Cost
Replace  ALL SL_EDCost WITH  SL_ECost + SL_DCost
Replace  ALL SM_EDCost WITH  SM_ECost + SM_DCost
Replace  ALL SH_EDCost WITH  SH_ECost + SH_DCost

Replace  ALL PL_EDCost WITH  PL_ECost + PL_DCost
Replace  ALL PM_EDCost WITH  PM_ECost + PM_DCost
Replace  ALL PH_EDCost WITH  PH_ECost + PH_DCost

Replace  ALL PL_JJEDCo WITH  PL_JJECo + PL_JJDCo
Replace  ALL PM_JJEDCo WITH  PM_JJECo + PM_JJDCo
Replace  ALL PH_JJEDCo WITH  PH_JJECo + PH_JJDCo

Replace  ALL PL_JSEDCo WITH  PL_JSECo + PL_JSDCo
Replace  ALL PM_JSEDCo WITH  PM_JSECo + PM_JSDCo
Replace  ALL PH_JSEDCo WITH  PH_JSECo + PH_JSDCo

Replace  ALL PL_SSEDCo WITH  PL_SSECo + PL_SSDCo
Replace  ALL PM_SSEDCo WITH  PM_SSECo + PM_SSDCo
Replace  ALL PH_SSEDCo WITH  PH_SSECo + PH_SSDCo

Replace  ALL PL_P0EDCo WITH  PL_P0ECo + PL_P0DCo
Replace  ALL PM_P0EDCo WITH  PM_P0ECo + PM_P0DCo
Replace  ALL PH_P0EDCo WITH  PH_P0ECo + PH_P0DCo

Replace  ALL PL_P1EDCo WITH  PL_P1ECo + PL_P1DCo
Replace  ALL PM_P1EDCo WITH  PM_P1ECo + PM_P1DCo
Replace  ALL PH_P1EDCo WITH  PH_P1ECo + PH_P1DCo

Replace  ALL PL_P2EDCo WITH  PL_P2ECo + PL_P2DCo
Replace  ALL PM_P2EDCo WITH  PM_P2ECo + PM_P2DCo
Replace  ALL PH_P2EDCo WITH  PH_P2ECo + PH_P2DCo

* All Cost
Replace  ALL SL_Cost WITH  SL_ECost + SL_DCost + SL_TC + SL_OCost  - SL_SCost
Replace  ALL SM_Cost WITH  SM_ECost + SM_DCost + SM_TC + SM_OCost - SM_SCost
Replace  ALL SH_Cost WITH  SH_ECost + SH_DCost + SH_TC + SH_OCost - SH_SCost

Replace  ALL PL_Cost WITH  PL_ECost + PL_DCost + PL_TC + PL_OCost - PL_SCost
Replace  ALL PM_Cost WITH  PM_ECost + PM_DCost + PM_TC + PM_OCost - PM_SCost
Replace  ALL PH_Cost WITH  PH_ECost + PH_DCost + PH_TC + PH_OCost - PH_SCost

Replace  ALL PL_JJCost WITH  PL_JJECo + PL_JJDCo + PL_JJTC + PL_JJOCo - PL_JJSCo
Replace  ALL PM_JJCost WITH  PM_JJECo + PM_JJDCo + PM_JJTC + PM_JJOCo - PM_JJSCo
Replace  ALL PH_JJCost WITH  PH_JJECo + PH_JJDCo + PH_JJTC + PH_JJOCo - PH_JJSCo
```

```
Replace  ALL PL_JSCost WITH  PL_JSECo + PL_JSDCo + PL_JSTC + PL_JSOCo - PL_JSSCo
Replace  ALL PM_JSCost WITH  PM_JSECo + PM_JSDCo + PM_JSTC + PM_JSOCo - PM_JSSCo
Replace  ALL PH_JSCost WITH  PH_JSECo + PH_JSDCo + PH_JSTC + PH_JSOCo - PH_JSSCo

Replace  ALL PL_SSCost WITH  PL_SSECo + PL_SSDCo + PL_SSTC + PL_SSOCo - PL_SSSCo
Replace  ALL PM_SSCost WITH  PM_SSECo + PM_SSDCo + PM_SSTC + PM_SSOCo - PM_SSSCo
Replace  ALL PH_SSCost WITH  PH_SSECo + PH_SSDCo + PH_SSTC + PH_SSOCo - PH_SSSCo

Replace  ALL PL_P0Cost WITH  PL_P0ECo + PL_P0DCo + PL_P0TC + PL_P0OCo - PL_P0SCo
Replace  ALL PM_P0Cost WITH  PM_P0ECo + PM_P0DCo + PM_P0TC + PM_P0OCo - PM_P0SCo
Replace  ALL PH_P0Cost WITH  PH_P0ECo + PH_P0DCo + PH_P0TC + PH_P0OCo - PH_P0SCo

Replace  ALL PL_P1Cost WITH  PL_P1ECo + PL_P1DCo + PL_P1TC + PL_P1OCo - PL_P1SCo
Replace  ALL PM_P1Cost WITH  PM_P1ECo + PM_P1DCo + PM_P1TC + PM_P1OCo - PM_P1SCo
Replace  ALL PH_P1Cost WITH  PH_P1ECo + PH_P1DCo + PH_P1TC + PH_P1OCo - PH_P1SCo

Replace  ALL PL_P2Cost WITH  PL_P2ECo + PL_P2DCo + PL_P2TC + PL_P2OCo - PL_P2SCo
Replace  ALL PM_P2Cost WITH  PM_P2ECo + PM_P2DCo + PM_P2TC + PM_P2OCo - PM_P2SCo
Replace  ALL PH_P2Cost WITH  PH_P2ECo + PH_P2DCo + PH_P2TC + PH_P2OCo - PH_P2SCo

* Data for training part of the cost given the scenarios of 10% impact to 100% impact
SELECT c1.respondent, PairYes,  c1.TrainingCL, c1.TrainingCM, c1.TrainingCH, ;
SL_TC10, SL_TC20, SL_TC30, SL_TC40, SL_TC50,  SL_TC60, SL_TC70, SL_TC80, SL_TC90, SL_TC100,
SM_TC10, SM_TC20, SM_TC30, SM_TC40, SM_TC50,  SM_TC60, SM_TC70, SM_TC80, SM_TC90,
SM_TC100,  SH_TC10, SH_TC20, SH_TC30, SH_TC40, SH_TC50,  SH_TC60, SH_TC70, SH_TC80, SH_TC90,
SH_TC100,  PL_TC10, PL_TC20, PL_TC30, PL_TC40, PL_TC50,  PL_TC60, PL_TC70, PL_TC80, PL_TC90,
PL_TC100,  PM_TC10, PM_TC20, PM_TC30, PM_TC40, PM_TC50,  PM_TC60, PM_TC70, PM_TC80,
PM_TC90, PM_TC100,  PH_TC10, PH_TC20, PH_TC30, PH_TC40, PH_TC50,  PH_TC60, PH_TC70, PH_TC80,
PH_TC90, PH_TC100,  PL_JJTC10, PL_JJTC20, PL_JJTC30, PL_JJTC40, PL_JJTC50,  PL_JJTC60, PL_JJTC70,
PL_JJTC80, PL_JJTC90, PL_JJTC100,
PM_JJTC10, PM_JJTC20, PM_JJTC30, PM_JJTC40, PM_JJTC50,  PM_JJTC60, PM_JJTC70, PM_JJTC80,
PM_JJTC90, PM_JJTC100,  PH_JJTC10, PH_JJTC20, PH_JJTC30, PH_JJTC40, PH_JJTC50,  PH_JJTC60,
PH_JJTC70, PH_JJTC80, PH_JJTC90, PH_JJTC100,  PL_JSTC10, PL_JSTC20, PL_JSTC30, PL_JSTC40,
PL_JSTC50,  PL_JSTC60, PL_JSTC70, PL_JSTC80, PL_JSTC90, PL_JSTC100,  PM_JSTC10, PM_JSTC20,
PM_JSTC30, PM_JSTC40, PM_JSTC50,  PM_JSTC60, PM_JSTC70, PM_JSTC80, PM_JSTC90, PM_JSTC100,
PH_JSTC10, PH_JSTC20, PH_JSTC30, PH_JSTC40, PH_JSTC50,  PH_JSTC60, PH_JSTC70, PH_JSTC80,
PH_JSTC90, PH_JSTC100,
PL_SSTC10, PL_SSTC20, PL_SSTC30, PL_SSTC40, PL_SSTC50,  PL_SSTC60, PL_SSTC70, PL_SSTC80,
PL_SSTC90, PL_SSTC100,  PM_SSTC10, PM_SSTC20, PM_SSTC30, PM_SSTC40, PM_SSTC50,
PM_SSTC60, PM_SSTC70, PM_SSTC80, PM_SSTC90, PM_SSTC100,
PH_SSTC10, PH_SSTC20, PH_SSTC30, PH_SSTC40, PH_SSTC50,  PH_SSTC60, PH_SSTC70, PH_SSTC80,
PH_SSTC90, PH_SSTC100,  PL_P0TC10, PL_P0TC20, PL_P0TC30, PL_P0TC40, PL_P0TC50,  PL_P0TC60,
PL_P0TC70, PL_P0TC80, PL_P0TC90, PL_P0TC100,  PM_P0TC10, PM_P0TC20, PM_P0TC30, PM_P0TC40,
PM_P0TC50,  PM_P0TC60, PM_P0TC70, PM_P0TC80, PM_P0TC90, PM_P0TC100,  PH_P0TC10, PH_P0TC20,
PH_P0TC30, PH_P0TC40, PH_P0TC50,  PH_P0TC60, PH_P0TC70, PH_P0TC80, PH_P0TC90, PH_P0TC100,
PL_P1TC10, PL_P1TC20, PL_P1TC30, PL_P1TC40, PL_P1TC50,  PL_P1TC60, PL_P1TC70, PL_P1TC80,
PL_P1TC90, PL_P1TC100,  PM_P1TC10, PM_P1TC20, PM_P1TC30, PM_P1TC40, PM_P1TC50,  PM_P1TC60,
PM_P1TC70, PM_P1TC80, PM_P1TC90, PM_P1TC100,  PH_P1TC10, PH_P1TC20, PH_P1TC30, PH_P1TC40,
PH_P1TC50,  PH_P1TC60, PH_P1TC70, PH_P1TC80, PH_P1TC90, PH_P1TC100,
PL_P2TC10, PL_P2TC20, PL_P2TC30, PL_P2TC40, PL_P2TC50,  PL_P2TC60, PL_P2TC70, PL_P2TC80,
PL_P2TC90, PL_P2TC100,  PM_P2TC10, PM_P2TC20, PM_P2TC30, PM_P2TC40, PM_P2TC50,  PM_P2TC60,
PM_P2TC70, PM_P2TC80, PM_P2TC90, PM_P2TC100,
PH_P2TC10, PH_P2TC20, PH_P2TC30, PH_P2TC40, PH_P2TC50,  PH_P2TC60, PH_P2TC70, PH_P2TC80,
PH_P2TC90, PH_P2TC100
FROM C1, C2, C3
```

```
WHERE c1.respondent = c2.respondent  AND c1.respondent = c3.respondent
INTO table costdataT
```

D. Training Cost

Training cost assuming 10% to 100% impact of knowledge transfer.

10%	Z-Score			$ Amount from the Mean		
	Low	Medium	High	Low	Medium	High
Solo	.0493	.1883	.0994	7	123	126
Pair	.0128	-.0195	-.0080	2	(13)	(10)
JJ	.0015	.0441	.0194	0	29	25
JS	-.0288	-.0926	-.0439	(4)	(61)	(56)
SS	-.0107	-.0556	-.0351	(1)	(36)	(44)
None	.0134	.0482	.0169	2	32	21
One	-.0146	-.0385	-.0399	(2)	(25)	(50)
Both	-.0230	-.0744	-.0088	(3)	(49)	(11)
Mean	3,320	22,679	64,763			
SD	134	654	1,266			

20%	Z-Score			$ Amount from the Mean		
	Low	Medium	High	Low	Medium	High
Solo	.202	.775	.445	26	493	504
Pair	.052	-.080	-.036	7	(51)	(41)
JJ	.006	.181	.087	1	115	98
JS	-.118	-.381	-.196	(15)	(243)	(222)
SS	-.044	-.229	-.157	(6)	(145)	(178)
None	-.060	.198	.075	(8)	126	85
One	-.060	-.159	-.178	(8)	(101)	(202)
Both	-.094	-.306	-.039	(12)	(195)	(44)
Mean	3,216	20,513	55,912			
SD	131	636	1,132			

30%	Z-Score			$ Amount from the Mean		
	Low	Medium	High	Low	Medium	High
Solo	.311	1.324	.746	40	739	755
Pair	.081	-.137	-.060	10	(77)	(61)
JJ	.010	.310	.145	1	173	147
JS	-.181	-.651	-.330	(23)	(364)	(334)
SS	-.067	-.391	-.263	(9)	(218)	(266)
None	.084	.339	.126	11	189	128
One	-.092	-.271	-.299	(12)	(151)	(303)
Both	-.145	-.523	-.066	(19)	(292)	(67)
Mean	3,147	19,069	50,011			
SD	128	558	1,012			

40%	Z-Score - 40% Impact			$ Amount from the Mean		
	Low	Medium	High	Low	Medium	High

Solo	.422	1.875	1.081	53	986	1,007
Pair	.109	-.194	-.087	14	(102)	(81)
JJ	.013	.439	.211	2	231	196
JS	-.246	-.923	-.477	(31)	(485)	(445)
SS	-.091	-.553	-.381	(11)	(291)	(355)
None	.115	.480	.183	14	252	171
One	-.125	-.384	-.433	(16)	(202)	(404)
Both	-.197	-.740	-.495	(25)	(389)	(462)
Mean	3,078	17,625	44,110			
SD	126	526	932			

50%	Z-Score			$ Amount from the Mean		
	Low	Medium	High	Low	Medium	High
Solo	.537	2.480	1.450	66	1,232	1,259
Pair	.139	-.257	-.117	17	(128)	(102)
JJ	.017	.581	.282	2	289	245
JS	-.313	-1.220	-.640	(39)	(606)	(556)
SS	-.116	-.732	-.511	(14)	(364)	(444)
None	.146	.635	.246	18	316	213
One	-.158	-.508	-.581	(20)	(252)	(505)
Both	-.250	-.980	-.128	(31)	(487)	(111)
Mean	3,009	16,182	38,210			
SD	123	497	869			

60%	Z-Score			$ Amount from the Mean		
	Low	Medium	High	Low	Medium	High
Solo	.655	3.132	1.829	79	1,478	1,511
Pair	.170	-.324	-.148	21	(153)	(122)
JJ	.020	.734	.356	2	346	294
JS	-.382	-1.541	-.808	(46)	(728)	(667)
SS	-.142	-.924	-.645	(17)	(436)	(533)
None	.178	.802	.310	22	379	256
One	-.193	-.641	-.733	(23)	(303)	(606)
Both	-.306	-1.237	-.161	(37)	(584)	(133)
Mean	2,940	14,738	32,309			
SD	121	472	826			

70%	Z-Score			$ Amount from the Mean		
	Low	Medium	High	Low	Medium	High
Solo	.776	3.815	2.182	351	3,081	1,763
Pair	.201	-.395	-.176	91	(319)	(143)
JJ	.024	.894	.425	11	722	343
JS	-.453	-1.877	-.964	(205)	(1,516)	(779)
SS	-.168	-1.126	-.769	(76)	(909)	(621)
None	.211	.977	.370	95	789	299
One	-.229	-.781	-.875	(104)	(630)	(707)
Both	-.362	-1.507	-.193	(164)	(1,217)	(156)

Mean	2,871	13,294	26,408			
SD	120	452	808			

80%	Z-Score			$ Amount from the Mean		
	Low	Medium	High	Low	Medium	High
Solo	.900	4.501	2.471	106	1,971	2,014
Pair	.233	-.466	-.200	27	(204)	(163)
JJ	.028	1.054	.482	3	462	393
JS	-.525	-2.215	-1.092	(62)	(970)	(890)
SS	-.195	-1.328	-.871	(23)	(582)	(710)
None	.244	1.153	.419	29	505	341
One	-.266	-.921	-.991	(31)	(403)	(807)
Both	-.420	-1.778	-.218	(49)	(779)	(178)
Mean	2,802	11,850	20,508			
SD	118	438	815			

90%	Z-Score			$ Amount from the Mean		
	Low	Medium	High	Low	Medium	High
Solo	1.025	5.157	2.673	119	2,218	2,266
Pair	.266	-.534	-.216	31	(230)	(183)
JJ	.032	1.208	.521	4	520	442
JS	-.598	-2.538	-1.181	(70)	(1,091)	(1,001)
SS	-.222	-1.522	-.943	(26)	(654)	(799)
None	.278	1.321	.453	32	568	384
One	-.303	-1.055	-1.072	(35)	(454)	(908)
Both	-.478	-2.037	-.236	(56)	(876)	(200)
Mean	2,733	10,406	14,607			
SD	116	430	848			

100%	Z-Score			$ Amount from the Mean		
	Low	Medium	High	Low	Medium	High
Solo	1.152	5.750	2.789	132	2,464	2,518
Pair	.299	-.595	-.226	34	(255)	(204)
JJ	.035	1.347	.544	4	577	491
JS	-.672	-2.829	-1.232	(77)	(1,213)	(1,112)
SS	-.249	-1.697	-.984	(29)	(727)	(888)
None	.313	1.472	.473	36	631	427
One	-.340	-1.177	-1.118	(39)	(504)	(1,009)
Both	-.538	-2.271	-1.246	(62)	(973)	(1,125)
Mean	2,664	8,963	8,706			
SD	115	429	903			

REFERENCES

Abdel-Hamid, T. K., Sengupta, K., & Swett, C. (1999). The Impact of Goals on Software Project

Management: An Experimental Investigation. *MIS Quarterly, 23*(4), 531-555.
Adèr, H.J., Mellenbergh, G.J., and Hand, D.J. (2008). *Advising on research methods: A consultant's companion:* Johannes van Kessel Publishing, The Netherlands.
Agile Manifesto. (2011). Principles behind the Agile Manifesto. Retrieved August 23, 2011, from http://agilemanifesto.org/principles.html
Aiken, J. (2004). Technical and Human Perspectives on Pair Programming. *ACM SIGSOFT Software Engineering Notes, 29*(5), 1-14.
Al-Kilidar, H., Parkin, P., Aurum, A., & Jeffery, R. (2005). *Evaluation of Effects of Pair Work on Quality of Designs*. Paper presented at the 2005 Australian conference on Software Engineering.
Ally, M., Darroch, F., & Toleman, M. (2005). *A Framework for Understanding the Factors Influencing Pair Programming Success.* Paper presented at the XP 2005 Conference.
Ambler, S. W. (2007). Dr. Dobb's Agile Newsletter 12/07. Retrieved October 1, 2009, from http://www.ddj.com/architect/204804676
Ambler, S. W. (2008). Dr. Dobb's Agile Newsletter 04/08. Retrieved October 1, 2009, from http://www.ddj.com/architect/207401013
Anthes, G. (2004). Sabre Takes Extreme Measures. Retrieved October 1, 2009, from http://www.computerworld.com/s/article/print/91646/Sabre_takes_extreme_measures?taxonomyName=Development&taxonomyId=11
Arisholm, E., Gallis, H., Dybå, T., & Sjøberg, D. I. K. (2007). Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Transactions on Software Engineering, 33*(2), 65-86.
Bagozzi, R. P. (1980). Causal Methods in Marketing. *New York: John Wiley and Sons*.
Baheti, P., Gehringer, E., & Stotts, D. (2002). Exploring the Efficacy of Distributed Pair Programming. *XP Universe, 2418*, 208-220.
Balijepally, V., Mahapatra, R., Nerur, S., & Price, K. H. (2009). Are Two Heads Better than One for Software Development? The Productivity Paradox of Pair Programming. *MIS Quarterly, 33*(1), 91-118.
Banker, R. D., Davis, G. B., & Slaughter, S. A. (1998). Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study. *Management Science, 44*(4), 433-450.
BBC News. (1998). Education GCSE Error Means Upgrade for Pupils. Retrieved September 27, 2009, from http://news.bbc.co.uk/2/hi/uk_news/education/167755.stm
BBC News. (2009). Software Error Affects Tax Forms. Retrieved September 27, 2009, from http://news.bbc.co.uk/2/hi/europe/isle_of_man/7819150.stm
Beck, K. (2000). *Extreme Programming Explained: Embrace Change*: Adison-Wesley.
Belshee, A. (2005). *Promiscuous Pairing and Beginner's Mind: Embrace Inexperience.* Paper presented at the Agile Development Conference (ADC'05).
Bevan, J., Werner, L., & McDowell, C. (2002). Guidelines for the use of pair programming in a freshmanprogramming class. *Software Engineering Education and Training*, 100-107.
Boudreau, M., Gefen, D., & Straub, D. (2001). Validation in IS research: A state-of-the-art assessment. *MIS Quart, 25*(1), 1-16.
Brooks, F. P. (1995). *The Mythical Man Month: Essays on Software Engineering*: Addison-Wesley.
Bryant, S. (2004). *Double Trouble: Mixing Qualitative and Quantitative Methods in the Study of eXtreme Programmers.* Paper presented at the IEEE Symposium on Visual Languages

and Human Centric Computing.
Bureau of Labor Statistics (2011). Occupational Employment Statistics. Retrieved Feb 27, 2011, from http://www.bls.gov/oes/current/oes151131.htm
Canfora, G., Cimitile, A., Garcia, F., Piattini, M., & Visaggio, C. A. (2007). Evaluating performances of pair designing in industry. *The Journal of Systems and Software, 80*(8), 1317-1327.
Canfora, G., Cimitile, A., Lucca, G. A. D., & Visaggio, C. A. (2006). How Distribution Affects the Success of Pair Programming. *International Journal of Software Engineering and Knowledge Engineering, 16*(2), 293-313.
Canfora, G., Cimitile, A., & Visaggio, C. A. (2005). *Empirical Study on the Productivity of the Pair Programming.* Paper presented at the 6th International Conference on Extreme Programming And Agile Processes in Software Engineering.
Cao, L., & Xu, P. (2005). *Activity Patterns of Pair Programming.* Paper presented at the 38th Annual Hawaii International Conference on System Sciences
Chong, J. (2005). *Social Behaviors on XP and non-XP Teams: a Comparative Study.* Paper presented at the 2005 Agile Conference.
Chong, J., & Hurlbutt, T. (2007). *The Social Dynamics of Pair Programming.* Paper presented at the 29th International Conference on Software Engineering.
Ciolkowski, M., & Schlemmer, M. (2002). *Experiences with a Case Study on Pair Programming.* Paper presented at the First International Workshop on Empirical Studies in Software Engineering.
Cliburn, D. C. (2003). Experiences with Pair Programming at a Small College. *Journal of Computing Sciences in Colleges, 19*(1), 20-29.
Cockburn, A., & Williams, L. (2001). The Costs and Benefits of Pair Programming. In *Addison-Wesley* (pp. 223-248).
Cohen, J. (1988). Statistical Power Analysis for the Behavioral Sciences, Hillsdale, NJ: L: Erlbaum Associates.
Computing Cases. (2009). Therac-25 Case Materials. Retrieved September 27, 2009, from http://computingcases.org/case_materials/case_materials.html
Constantine, L. L. (1995). *Constantine on Peopleware*: Yourdon Press.
Copeland, L. (2001a). Programming Gets Extreme, Development Method Takes Off, but Not in U.S. Retrieved October 1, 2009, from http://www.computerworld.com/s/article/59388/Programming_gets_extreme
Copeland, L. (2001b). Extreme Programming. Retrieved October 1, 2009, from http://www.computerworld.com/s/article/66192/Extreme_Programming?taxonomyId=63&pageNumber=1
Cronbach, L. J. (1971). *Test validation*. Washington D.C.: American Council on Education.
Cusumano, M., MacCormack, A., Kemerer, C. F., & Crandall, B. (2003). Software Development Worldwide: The State of the Practice. *IEEE Software, 20*(6), 28-34.
Data Monitor. (2008a). Industry Profile: Software in the United States. Retrieved July 1, 2008, from http://www.datamonitor.com
Data Monitor. (2008b). Industry Profile: Global Software. Retrieved July 1, 2008, from http://www.datamonitor.com
Dawande, M., Johar, M., Kumar, S., & Mookerjee, V. S. (2008). A Comparison of Pair versus Solo Programming under Different Objectives: An Analytical Approach. *Information Systems Research, 19*(1), 71-92.

DeClue, T. H. (2003). Pair Programming and Pair Trading: Effects on Learning and Motivation in a CS2 course. *Journal of Computing Sciences in Colleges, 18*(5), 49-56.

Dennis, A., Wixom, B. H., & Tegarden, D. (2005). *Systems analysis and design with UML version 2.0*: Wiley.

Dick, A. J., & Zarnett, B. (2002). *Paired Programming & Personality Traits.* Paper presented at the 3rd International Conference on Extreme Programming and Flexible Processes in Software Engineering.

Dybå, T., Arisholm, E., Sjoerg, D. I. K., Hannay, J. E., & Shull, F. (2007). Are Two Heads Better than One? On the Effectiveness of Pair Programming. *IEEE Software, 24*(6), 12-15.

Efron, B. (1979). Bootstrap methods: another look at the jackknife. *The annals of Statistics,* 7(1), 1-26.

Efron, B. (1982). The jackknife, the bootstrap and other resampling plans. *SIAM Monograph,* 38.

Efron, B., and Tibshirani, R.(1993). *An introduction to the bootstrap:* Chapman & Hall, New York.

Erdogmus, H., & Williams, L. (2003). The Economics of Software Development by Pair Programmers. *Engineering Economist, 48*(4), 283-319

Erickson, J., Lyytinen, K., & Siau, K. (2005). Agile Modeling, Agile Software Development, and Extreme Programming :

The State of Research. *Journal of Database Management, 16*(4), 88-100. **??????**

Espinosa, J. A., Slaughter, S. A., Kraut, R. E., & Herbsleb, J. D. (2007). Familiarity, Complexity, and Team Performance in Geographically Distributed Software Development. *Organization Science, 18*(4), 613-630.

Fitzgerald, B., & Harnett, G. (2005). A Study of the Use of Agile Methods within Intel. *Business Agility and Information Technology Diffusion,* 187-202.

Flor, N. V. (2006). Globally Distributed Software Development and Pair Programming. *Communications of the ACM, 49*(10), 57-58.

Flor, N. V., & Hutchins, E. (1991). *Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance.* Paper presented at the Empirical Studies of Programmers: Fourth Workshop.

Freedman, D.A., and Peters, S.C. (1984). Bootstrapping a regression equation: Some empirical results. *Journal of the American Statistical Association* , 79(385), 97-106.

Gallis, H., Arisholm, E., & Dyba, T. (2003). *An Initial Framework for Research on Pair Programming.* Paper presented at the International Symposium on Empirical Software Engineering.

Gehringer, E. F. (2003). *A Pair-Programming Experiment in a Non-Programming Course*. Paper presented at the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.

Geoghegan, T. (2005). Progress... pah! Retrieved September 27, 2009, from http://news.bbc.co.uk/2/hi/uk_news/magazine/4606263.stm

Glass, R. L. (2001). Extreme Programming: The Good, the Bad, and the Bottom Line. *IEEE Software, 18*(6), 112-111.

Greene, B. (2004). *Agile Methods Applied to Embedded Firmware Development.* Paper presented at the Agile Development Conference.

Hanks, B., McDowell, C., Draper, D., & Krnjajic, M. (2004). Program Quality with Pair Programming in CS1. *ACM SIGCSE Bulletin 36*(3), 176-180.

Harel, E. C., & McLean, E. R. (1985). The Effects of Using a Nonprocedural Computer Language on Programmer Productivity. *MIS Quarterly, 9*(2), 109-120.

Haungs, J. (2001). Pair Programming on the C3 Project. *Computer, 34*(2), 118-119.

Heiberg, S., Puus, U., Salumaa, P., & Seeba, A. (2003). Pair-Programming Effect on Developers Productivity. *Lecture Notes in Computer Science, 2675*, 215-224.

Hulkko, H., & Abrahamsson, P. (2005). *A Multiple Case Study on the Impact of Pair Programming on Product Quality.* Paper presented at the 27th International Conference on Software Engineering.

Humphrey, W. (1995). *A Discipline for Software Engineering*: Addison Wesley.

Humphrey, W. S. (1989). *Managing the Software Process*: Addison-Wesley.

Hurt, H. (2009). Turning Whimsy Into a Video Classic. Maybe. . Retrieved September 25, 2009, from http://www.nytimes.com/2009/02/07/business/07pursuits.html?_r=1&sq="lines%20of%20code"&st=cse&adxnnl=1&scp=5&adxnnlx=1253930633-DNWMTY2OELEn7n2lXHqQhQ

Janes, A., Russo, B., Zuliani, P., & Succi, G. (2003). *An Empirical Analysis on the Discontinuous Use of Pair Programming.* Paper presented at the Extreme Programming 2003, Genova, Italy.

Jensen, R. W. (2003). A Pair Programming Experience. *CrossTalk, 16*(3), 22-24.

Ji, Y., Mookerjee, V. S., & Sethi, S. P. (2005). Optimal software development: a control theoretic approach. *Information Systems Research, 16*(3), 292.

Juran, J. M. (1974). *Quality Control Handbook* (3rd ed.): McGraw-Hill New York.

Kampenes, V. B., Dybl, T., Hannay, J. E., & Sjrberg, D. I. K. (2007). A Systematic Review of Effect Size in Software Engineering Experiments. *Information and Software Technology, 49*(11-12), 1073-1086.

Kandt, R. K. (2009). Experiences in Improving Flight Software Development Processes. *IEEE Software, 26*(3), 58-64.

Katira, N., Williams, L., Wiebe, E., Miller, C., Balik, S., & Gehringer, E. (2004). On Understanding Compatibility of Student Pair Programmers. *ACM SIGCSE Bulletin, 36*(1), 7-11.

Kemerer, C. F. (1995). Software complexity and software maintenance: A survey of empirical research. *Annals of Software Engineering, 1*(1), 1-22.

Kirby, E. J. (2007). Cancer Patients' Battle for Justice. Retrieved September 27, 2009, from http://news.bbc.co.uk/2/hi/programmes/from_our_own_correspondent/7049319.stm

Kirschner, F., Paas, F., & Kirschner, P. (2008). Individual versus Group Learning as a Function of Task Complexity: An Exploration into the Measurement of Group Cognitive Load. In *Beyond Knowledge: The Legacy of Competence* (pp. 21-28): Springer Netherlands.

Kline, R. B. (2005). *Principles and Practice of Structural Equation Modeling*: The Guilford Press.

Ko, D. G., & Kirsch, L. (2005). Antecedents of Knowledge Transfer from Consultants to Clients in Enterprise System Implementations. *Management Information Systems Quarterly, 29*(1), 59-85.

Langr, J. (2005). Pair Programming Observations. Retrieved October 1, 2009, from http://langrsoft.com/articles/pairing.shtml

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.
Luck, G., & ThoughtWorks, I. (2004). *Subclassing XP: Breaking its Rules the Right Way.* Paper presented at the 2004 Agile Development Conference.
Lui, K., & Chan, K. C. C. (2003). When Does a Pair Outperform Two Individuals? *Lecture Notes in Computer Science, 2675*, 225-233.
Lui, K. M., & Chan, K. C. C. (2006). Pair Programming Productivity: Novice-Novice vs. Expert-Expert. *International Journal of Human-Computer Studies, 64*(9), 915-925.
Madeyski, L. (2006). The Impact of Pair Programming and Test-Driven Development on Package Dependencies in Object-Oriented Design — An Experiment *Lecture Notes in Computer Science, 4034*, 278-289.
Mahalo. (2009). Answered Question. Retrieved September 28, 2009, from http://www.mahalo.com/answers/programming/how-many-software-developers-are-there-in-the-world-and-how-many-in-the-us-india-and-china-respectively
March, J., & Simon, H. (1958). *Organizations*. New York: Wiley.
Marchenko, A. (2008a). XP Practice: Pair Programming. Retrieved October 1, 2009, from http://agilesoftwaredevelopment.com/xp/practices/pair-programming
Marchenko, A. (2008b). Five Risks of Solo Programming. Retrieved October 1, 2009, from http://agilesoftwaredevelopment.com/blog/artem/five-risks-of-solo-programming
Marchenko, A. (2008c). Pair Programming. What Researches Say on the Costs and Benefits of the Practice. Retrieved October 1, 2009, from http://agilesoftwaredevelopment.com/blog/artem/pair-programming-what-researches-say
Markoff, J. (1998). Sun Microsystems Introducing A New Version of Unix Today. Retrieved September 25, 2009, 2009, from http://www.nytimes.com/1998/10/27/business/sun-microsystems-introducing-a-new-version-of-unix-today.html?scp=18&sq=%22software%20errors%22&st=cse
McDowell, C., Hanks, B., & Werner, L. (2003). Experimenting with Pair Programming in the Classroom. *ACM SIGCSE Bulletin, 35*(3), 60-64.
McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). *The Effects of Pair-Programming on Performance in an Introductory Programming Course.* Paper presented at the 33rd SIGCSE Technical Symposium on Computer Science Education.
McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2003). *The Impact of Pair Programming on Student Performance, Perception and Persistence.* Paper presented at the 25th International Conference on Software Engineering, Portland, Oregon.
McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair Programming Improves Student Retention, Confidence, and Program Quality. *Communications of the ACM, 49*(8), 90-95.
Mendes, E., Al-Fakhri, L. B., & Luxton-Reilly, A. (2005). *Investigating Pair-Programming in a 2nd-year Software Development and Design Computer Science Course.* Paper presented at the 10th annual SIGCSE Conference on Innovation and Technology in Computer Science Education.
Microsoft. (2009). Every Project Plan is a Triangle. Retrieved Septebmer 25, 2009, from http://office.microsoft.com/en-us/project/HA010211801033.aspx#1
Morales, A. W. (2002). Going to Extremes. Retrieved October 1, 2009, from http://www.informationweek.com/news/software/development/showArticle.jhtml?articleID=6500648
Müller, M. M. (2004). Are Reviews an Alternative to Pair Programming? *Empirical Software*

Engineering, 9(4), 335-351.
Müller, M. M. (2005). Two Controlled Experiments Concerning the Comparison of Pair Programming to Peer Review. *Journal of Systems and Software, 78*(2), 166-179.
Müller, M. M. (2006). A Preliminary Study on the Impact of a Pair Design Phase on Pair Programming and Solo Programming. *Information and Software Technology, 48*(5), 335-344.
Müller, M. M., & Padberg, F. (2002). *Extreme Programming from an Engineering Economics Viewpoint.* Paper presented at the Fourth International Workshop Economics-Driven Software Engineering Research.
Müller, M. M., & Padberg, F. (2003). On the Economic Evaluation of XP Projects. *SIGSOFT Software Engineering Notes, 28*(5), 168-177.
Müller, M. M., & Padberg, F. (2004). *An Empirical Study about the Feelgood Factor in Pair Programming.* Paper presented at the 10th International Symposium on Software Metrics.
Müller, M. M., & Tichy, W. F. (2001). *Case Study: Extreme Programming in a University Environment.* Paper presented at the 23rd International Conference on Software Engineering.
Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., et al. (2003). Improving the CS1 Experience with Pair Programming. *ACM SIGCSE Bulletin, 35*(1), 359-362.
Nagappan, N., Williams, L., Wiebe, E., Miller, C., Balik, S., Ferzli, M., et al. (2003). Pair Learning: With an Eye Toward Future Success. *Lecture Notes in Computer Science, 2753*, 185-198.
Natsu, H., Favela, J., Moran, A. L., Decouchant, D., Martinez-Enriquez, A. M., de la Computacion, C., et al. (2003). *Distributed Pair Programming on the Web.* Paper presented at the Fourth Mexican International Conference on Computer Science.
Nawrocki, J., & Wojciechowski, A. (2001). Experimental Evaluation of Pair Programming. *European Software Control and Metrics*.
Nedland, M. (2005). Two Heads, One Focus. Retrieved October 1, 2009, from http://www.ddj.com/architect/184415309
Nilsson, K. (2003). *A Summary from a Pair Programming Survey: Subpart from an Undergraduate Master Thesis Paper.* Blekinge Institute of Technology.
Nosek, J. T. (1998). The Case for Collaborative Programming. *Communications of the ACM, 41*(3), 105-108.
Olsen, P. R. (2009). For Writing Software, a Buddy System. Retrieved September 29, 2009, from http://www.nytimes.com/2009/09/20/jobs/20pre.html?_r=1&emc=eta1
Padberg, F., & Müller, M. M. (2003). *Analyzing the Cost and Benefit of Pair Programming.* Paper presented at the 9th International Symposium on Software Metrics
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.
Parrish, A., Smith, R., Hale, D., & Hale, J. (2004). A Field Study of Developer Pairs: Productivity Impacts and Implications. *IEEE Software, 21*(5), 76-79.
Pendharkar, P. C., & Rodger, J. A. (2009). The Relationship between Software Development Team Size and Software Development Cost. *Communications of the ACM, 52*(1), 141-144.
Petter, S., Straub, D., & Rai, A. (2007). Specifying formative constructs in information systems research. *MIS Quarterly, 31*(4), 623-656.
Pfleeger, S. L., & Atlee, J. M. (2008). *Software Engineering: Theory and Practice*: Prentice Hall

Phongpaibul, M., & Boehm, B. (2006). *An Empirical Comparison between Pair Development and Software Inspection in Thailand*. Paper presented at the 2006 ACM/IEEE international symposium on Empirical software engineering.

Poole, C., & Huisman, J. W. (2001). Using Extreme Programming in a Maintenance Environment. *IEEE Software 18*(6), 42-50.

Prashant, B., Edward, F. G., & Stotts, P. D. (2002). *Exploring the Efficacy of Distributed Pair Programming*. Paper presented at the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods.

Pressman, R. S. (2005). *Software Engineering: a Practitioner's Approach*: McGraw-Hill New York.

Preston, D. (2005). Pair Programming as a Model of Collaborative Learning: a Review of the Research. *Journal of Computing Sciences in Colleges, 20*(4), 39-45.

Price, D. A. (2009). Do as I Say And as I Do. Retrieved October 1, 2009, from http://online.wsj.com/article/SB10001424052970204731804574385132694479464.html

Radding, A. (2002). Extremely Agile Programming. Retrieved October 1, 2009, from http://www.computerworld.com/s/article/print/67950/Extremely_Agile_Programming

Robbins, S. P. (2000). *Organizational Behavior*: Prentice Hall.

Rogers, T. B. (1995). *The Psychological Testing Enterprise: An Introduction*: Brooks/Cole Pub. Co.

Rostaher, M., & Hericko, M. (2002). *Tracking Test First Pair Programming - An Experiment*. Paper presented at the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods.

Rumpe, B., & Schroer, A. (2002). *Quantitative Survey on Extreme Programming Projects.* Paper presented at the 3rd International Conference on Extreme Programming and Flexible Processes in Software Engineering, Sardinia, Italy.

Sanders, D. (2001). *Student Perceptions of the Suitability of Extreme and Pair Programming.* Paper presented at the XP Universe.

Schach, S. R. (2006). *Object-Oriented and Classical Software Engineering*: McGraw-Hill

Siobhan. (2007). Paired Programming - Benefits for Individual and Team Development. Retrieved October 1, 2009, from http://www.siobhan-optimise.blogspot.com/

Sommerville, I. (1996). *Software Engineering*: Addison-Wesley.

Sommerville, I. (2007). *Software Engineering*: Addison-Wesley.

Sommerville, I. (2009). Case Studies and Examples. Retrieved September 27, 2009, from http://www.cs.st-andrews.ac.uk/~ifs/Resources/CaseStudies/index.html

Srikanth, H., Williams, L., Wiebe, E., Miller, C., & Balik, S. (2004). *On Pair Rotation in the Computer Science Course.* Paper presented at the 17th Conference on Software Engineering Education and Training.

Stephens, M., & Rosenberg, D. (2003). *Extreme Programming Refactored: The Case Against XP*: Apress New York.

Straub, D. W. (1989). Validating instruments in MIS research. *MIS Quarterly, 13*(2), 147-169.

Succi, G., Pedrycz, W., Marchesi, M., & Williams, L. (2002). *Preliminary Analysis of the Effects of Pair Programming on Job Satisfaction.* Paper presented at the Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering

Tassey, G. (2002). The Economic Impacts of Inadequate Infrastructure for Software Testing. *National Institute of Standards and Technology RTI Project*.

Tessem, B. (2003). *Experiences in Learning XP Practices: A Qualitative Study.* Paper presented

at the Extreme Programming Genova, Italy.
Thomas, L., Ratcliffe, M., & Robertson, A. (2003). *Code Warriors and Code-a-Phobes: a Study in Attitude and Pair Programming*. Paper presented at the 34th SIGCSE Technical Symposium on Computer Science Education.
Tomayko, J. E. (2002). A Comparison of Pair Programming to Inspections for Software Defect Reduction. *Computer Science Education, 12*(3), 213-222.
VanDeGrift, T. (2004). Coupling Pair Programming and Writing: Learning about Students' Perceptions and Processes. *ACM SIGCSE Bulletin, 36*(1), 2-6.
Vanhanen, J., & Korpi, H. (2007). *Experiences of Using Pair Programming in an Agile Project.* Paper presented at the 40th Annual Hawaii International Conference on System Sciences.
Vanhanen, J., & Lassenius, C. (2005). *Effects of Pair Programming at the Development Team level: an Experiment.* Paper presented at the 2005 International Symposium on Empirical Software Engineering.
Vanhanen, J., & Lassenius, C. (2007). *Perceived Effects of Pair Programming in an Industrial Context.* Paper presented at the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications.
Vanhanen, J., Lassenius, C., & Mantyla, M. V. (2007). *Issues and Tactics when Adopting Pair Programming: A Longitudinal Case Study.* Paper presented at the International Conference on Software Engineering Advances.
Washington Technology. (1999). Selected Security Events in the 1990s. Retrieved September 27, 2009, from http://washingtontechnology.com/Articles/1999/01/07/Selected-Security-Events-in-the-1990s.aspx?sc_lang=en&Page=1
Weinberg, G. M. (1971). *The Psychology of Computer Programming*: Van Nostrand Reinhold New York.
Williams, L. (1999). *But, Isn't that Cheating?* . Paper presented at the Frontiers in Education Conference.
Williams, L., & Kessler, R. (2003). *Pair Programming Illuminated*: Addison-Wesley
Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the Case for Pair Programming. *IEEE Software, 17*(4), 19-25.
Williams, L., McDowell, C., Nagappan, N., Fernald, J., & Werner, L. (2003). *Building Pair Programming Knowledge through a Family of Experiments.* Paper presented at the International Symposium on Empirical Software Engineering.
Williams, L., Shukla, A., & Anton, A. I. (2004). *An Initial Exploration of the Relationship between Pair Programming and Brooks' Law.* Paper presented at the 2004 Agile Development Conference.
Williams, L., & Upchurch, R. L. (2001). *In Support of Student Pair-Programming.* Paper presented at the 32nd SIGCSE Technical Symposium on Computer Science Education.
Williams, L. A., & Kessler, R. R. (2000). *The Effects of" Pair-Pressure" and "Pair-Learning" on Software Engineering Education.* Paper presented at the 13th Conference on Software Engineering Education and Training.
Williams, L. A., & Kessler, R. R. (2001). Experiments with Industry's" Pair-Programming" Model in the Computer Science Classroom. *Computer Science Education, 11*(1), 7-20.
Woyke, E. (2008). The Next Killer Mobile App. Retrieved September 28, 2009, from www.forbes.com/2008/04/03/ctia-mobile-developer-tech-wire-cx_ew_0403ctia.html
Wray, S. (2010). How Pair Programming Really Works. *IEEE Software*(January/February), 50-55.

Xu, S., & Rajlich, V. (2006). *Empirical Validation of Test-Driven Pair Programming in Game Development.* Paper presented at the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering,Software Architecture and Reuse (ICIS-COMSAR'06).

Yourdon, E. (2004). *Death March: The Complete Software Developer's Guide to Surviving Mission Impossible Projects*: Prentice-Hall: Upper Saddle River, NJ.