PREDICTING THE USE OF PAIRED PROGRAMMING: APPLYING THE ATTITUDES OF APPLICATION DEVELOPMENT MANAGERS THROUGH THE TECHNOLOGY ACCEPTANCE MODEL

by

Mark S. Zecca

A Dissertation Presented in Partial Fulfillment

Of the Requirements for the Degree

Doctor of Philosophy

Capella University

July 2010

Abstract

Business managers who look for ways to cut costs face difficult questions about the efficiency and effectiveness of software engineering practices that are used to complete projects on time, on specification, and within budget (Johnson, 1995; Lindstrøm & Jeffries, 2004). Theoretical models such as the Theory of Reasoned Action (TRA) have linked intention and attitude to predictable behavior. The Technology Acceptance Model (TAM) as proposed by Davis (1985, 1989) furthers the theory of predicting behavior to the acceptance of technology practices through the understanding of perceived ease of use and perceived usefulness. These theories are applied to the practices of paired and individual programming. The research in this study surveys the attitudes of software development managers towards the practices of paired and individual programming and applies a generally accepted technology acceptance model to the collected data as a theoretical framework to indicate the acceptance and usage of such a practice in their software engineering environments. The findings do not support the position that the paired programming practice is used more than the individual programming practice. The data also indicates that while a software development managers' type of business does not affect usage of the paired or individual programming practice, the manager's years of experience does. This study suggests follow-on research and possible experimentation in the future use of paired programming as a viable, cost effective practice for software engineering.

Dedication

This work is dedicated to the memory of my father, George Zecca, who always told me that whatever you can think of, you can do. I spend more time trying to disprove his theory, yet his advice and wisdom continue to be proved true. I also dedicate this work to the memory of my mother, Betty Zecca, whose limitless support, humor, and assurance gave me the energy to move through my studies. My faith tells me they are with me and even now, I am just starting to realize the unlimited support they were to me.

My professional life would not be possible without the example and mentorship of Mr. Virgil Slayton, former Vice President of Operations for American Airlines. He taught me honesty, loyalty, and truth in the workplace and was the embodiment of the perfect professional leader. He provided me a model and an example that I still strive to become. I wish also to dedicate this work to Mr. Max Hopper, former CIO at American Airlines-AMR Corp, whose insight and forward vision of technology have always inspired me. I wish to recognize the current President and CEO of Mitchell 1, Dave Ellingen, for consistently supporting me and encouraging my efforts to complete this program.

Finally, I dedicate this work to my lovely wife, Barbara. Her patience and understanding is boundless and her loving support allowed me the time and the access to complete my research studies. She was and continues to be my cheerleader and source of energy. Without her, I would not have seen the end of this endeavor. With her, I know all things are possible!

Acknowledgments

The wisdom and understanding of many people contributed to this work. I would like to acknowledge my mentor and committee chair, Dr. Alan Chmura. I was his first Mentee at Capella University, and together we traversed the jungles of process and policy as well as various avenues of research represented in this study. Doctor C, as I have called him, is a man of infinite patience, a resource of infinite wisdom, and a man with a heart the size of Wyoming. Without him, I would simply not have progressed nor would I have produced this document. He remains for me and will always be a life mentor, tutor, friend, and overall *Wiseman from Wyoming*!

I would also like to acknowledge Dr. Kathleen Hargiss, my committee member from the School of Business and Technology. Dr. Hargiss continues to be an inspiration and sounding board, especially in research methods as well as attention to detail. During my research, I was constantly thinking what would Dr. Hargiss want to see? I will continue to ask this question for any research I do in the future. She embodies the meaning of what it means to be a scholarly practitioner.

I would like to acknowledge Dr. Steven Brozovich, my external committee member and co-worker at Snap-on Corporation. Steven was my grounding for theory and scientific application as well as part-time uncle, long lost brother, and father confessor! His math-prowess is only exceeded by his generosity. I would also like to offer a special thank you to my colleagues, Dr. Gary Shelton and Dr. Virginia Ross. Their advice, editorial assistance, collaborative support, and especially their friendship were invaluable. Virginia, thank you so much for being the *editor extraordinaire*!

Table of Contents

List of Tables vii

List of Figures viii

CHAPTER 1. INTRODUCTION 1

Introduction to the Problem 1

Background of the Study 4

Statement of the Problem 14

Purpose of the Study 15

Rationale 16

Research Questions 18

Significance of the Study 24

Definition of Terms 26

Assumptions and Limitations 31

The Nature and Theoretical Framework of the Study 32

Organization of the Remainder of the Study 33

CHAPTER 2. LITERATURE REVIEW 36

The Theory of Reasoned Action and the Theory of Planned Behavior 38

The Technology Acceptance Model 47

The Paired Programming Practice 64

CHAPTER 3. METHODOLOGY 92

Research Design 94

Sampling 99

Instrumentation and Measures 102

Data Collection 103

Data Analysis 105

Validity and Reliability of the Survey 116

Ethical Considerations 126

CHAPTER 4. RESULTS 128

Survey Response Results 129

Primary Hypotheses Analysis 134

CHAPTER 5. DISCUSSION, IMPLICATIONS, RECOMMENDATIONS 154

Discussion 154

Implications 163

Recommendations 166

REFERENCES 167

APPENDIX; ASSESSING THE ATTITUDES OF SOFTWARE DEVELOPMENT MANAGERS ON THE USE OF PAIRED PROGRAMMING AS COMPARED TO INDIVIDUAL PROGRAMMING 186

List of Tables

Table 1. The Technology Acceptance Model Mathematical Formula 21

Table 2. Comparison of Individual and Paired Programmer Efforts 79

Table 3. U.S. Government Standard Industrial Classification Codes 97

Table 4. Pre-test for Reliability Using the Cronbach Alpha Method 121

Table 5. Construct Items Applied to the Practice of Paired/Individual Programming 135

Table 6. Mean TAM Score Between Paired and Individual Programming for PEU. 136

Table 7. Mean TAM Score Between Paired and Individual Programming for PU 138

Table 8. Correlation of Constructs for Paired Programming 141

Table 9. Correlation of Constructs for Individual Programming 143

Table 10. Correlations Between Perceived Usefulness, Perceived Ease of Use, and Self-Reported Usage 144

Table 11. Regression Analysis on the Effect of Usefulness and Ease of Use Usage 146

Table 12. Correlational & Mean Difference Tests for Business-Type Analysis 147

Table 13. ANOVA Between Paired and Individual Usage and Business Types 148

Table 14. Analysis of Variance Between Paired and Individual Programming Experience 150

Table 15. ANOVA Between Paired and Individual Programming Experience 151

Table 16. Review of Hypotheses Conclusions 152

List of Figures

Figure 1. The Agile Methodology Movement 2

Figure 2. Underlying Concept of the Basic User Acceptance Model 10

Figure 3. Technology Acceptance Model 11

Figure 4. Theoretical Framework for this Study 20

Figure 5. A Time Line of Theoretical Models Used in this Study 37

Figure 6. Studies on Paired Programming 40

Figure 7. Sample Survey Scale 103

Figure 8. Research Model with Correlations Variables 107

Figure 9. A Sample Question from the eSurvey 129

Figure 10. Research Model with Correlation Values – Paired Programming 142

Figure 11. Research Model with Correlation Values – Individual Programming 144

Figure 12. Plotting a Manager's Experience Level to Practice Usage 159

CHAPTER 1. INTRODUCTION

Introduction to the Problem

A software development crisis was first publicly revealed in the Standish Group's 1994 publication, *The Chaos Report*. The crisis was simply that software development projects were failing to be completed or even abandoned prior to completion due to incomplete requirements and poor estimations of software engineering resources and time. This and other studies were important in highlighting a need for better software development methods that were accurate, cost effective, and efficient. The traditional means of software engineering included methodologies such as Software Development Life Cycle (SDLC) and Waterfall methodologies (Mugridge, 2008). These generally represented a process of gathering detailed requirements, writing technical specifications, coding to the specifications, integrating various single coding branches, testing the code, checking the code-based functionality against the specifications, and compiling and implementing the code into a working production program. The majority of work was carried out by programmers working individually with segmented specifications (El Emam & Koru, 2008). Various layers of management and integration were needed to weld programs together from the work of various independent software engineers.

Many new software development methodologies have surfaced in the last 15 years (Mugridge, 2008). One such methodology is the *Agile Programming Movement* (see Figure 1).

The Chain of the Agile Programming Movement, Methodology, & Practice

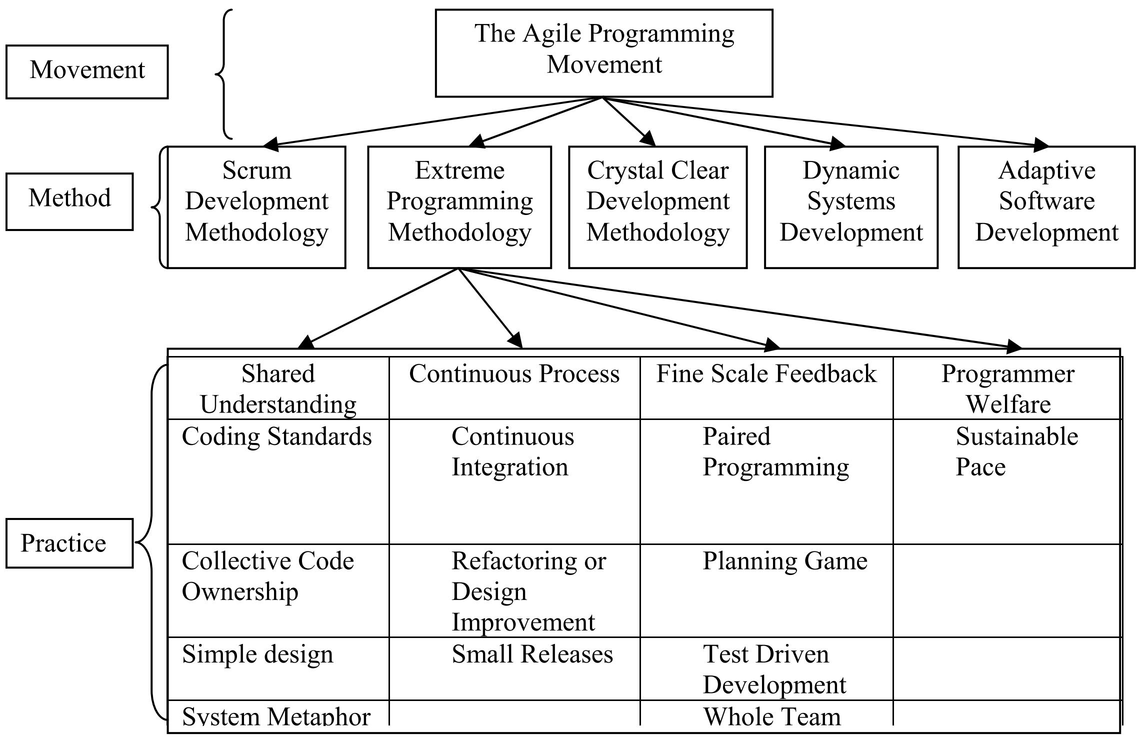


Figure 1. The chain of the Agile programming movement, methodology, and practice. A breakdown of the Agile movement shows the progression of various methodologies and within those methodologies, various practices. Paired Programming is shown as a practice of the Extreme Programming methodology.

The Agile Movement was a key factor in the development of a loose consortium of software development managers and software business leaders. Their purpose was to attempt to pursue alternative and more cost-effective and accurate means of software engineering. Members of this consortium wanted to depart from traditional methodologies and engage in iterative development. This included tight communication between customers and developers in a rhythm of communicated requirements to expertly authored code (Cao & Ramesh, 2008). Within the Agile movement, the *Extreme Programming* development method became popular (Highsmith, 1999). Noted for its 12 practices, it employed radically new forms of software engineering in an attempt to produce accurate code related to customer requirements (Beck, 1999).

Part of the Extreme Programming methodology included the practice of paired programming. In the practice of paired programming, two programmers work in tandem (programming pairs) at the same workstation, in contrast to the traditional practice of programmers working individually. This practice has become popular over the last 10 years. Various studies, as indicated in Chapter 2, propose that this practice is more efficient and productive than traditional, individual programming. It appears that many businesses are questioning the costs and effectiveness of this practice. It also appears that the excitement of early adoption of this practice has given way to questions about its actual cost effectiveness. There are also questions about the accuracy of some of the studies that initially indicated that this practice was more efficient than traditional practices.

A good measure of the paired programming practice's success is its acceptability in software development groups (Williams, 1999). There is a significant volume of peer-reviewed literature with experimental results on various aspects of the paired programming practice. Those results do not necessarily translate to the actual usage and employment of paired programming in today's software development groups. Many of the experiments attempted to demonstrate the relative empirical value of paired programming compared to individual programming and/or other types of programming methodologies (Arisholm & Sjøberg, 2003). Understanding the comparative efficiency or effectiveness of paired programming and other methodologies does not predict its current and future use or acceptance in the field. This study measures the intentions and attitudes of software development managers about their software engineering practices in order to understand the usage and acceptance of the paired programming practice. Specifically, the focus of this study is on the perceived ease-of-use and perceived usefulness of paired programming, using a structured and accepted scientific methodology. The focus of the research concentrates on how software development managers have perceived paired and individual programming practices, and how they have developed and engaged behaviors that have actually employed paired programming as a software engineering practice.

Background of the Study

History of the Agile Methodology Movement

In February of 2001, at a ski resort lodge in the Wasatch Mountains of Snowbird Utah, 17 people met to discuss, debate, and find common ground on a better form of software development that was practical and simple. What emerged was the Agile Methodology Movement of software development. Representatives were present from methodologies such as Extreme Programming (XP), SCRUM, Dynamic Systems Development Method (DSDM), Adaptive Software Development (ASD), Crystal Development Method (CDM), Feature-Driven Development, Pragmatic Programming, and other software development methodologies (Southerland, 2004). The representatives from these methodologies were sympathetic to the need for an alternative to traditional documentation driven software development processes. What emerged from this meeting was the *Manifesto for Agile Software Development*, better known as the Agile Manifesto (Highsmith, 1999). This group, named *The Agile Alliance,* included attendees who did not believe that a collection of independent and creative thinkers could agree on any substantive principles, let alone the 12 Agile principles (Highsmith).

At the root of the Agile Methodology Movement, there are a series of basic principles that helped create the *Manifesto.* These included: (a) Customer satisfaction by rapid, continuous delivery of useful software, (b) Working software that is delivered frequently in weeks rather than months, (c) The most significant and principal measure of progress is actual working software, (d) Changes in requirements, even late ones, are welcomed, (e) Close, daily cooperation between business people and developers is fostered, (f) Face-to-face conversation and co-location is the best form of communication, (g) Projects are built around motivated individuals, who should be trusted, (h) Continuous attention to technical excellence and good design is always maintained, (i) Simplicity in code is elegant, (j) Self-organizing teams are the best to set up, (k) Regular adaptation to changing circumstances is promoted and supported, and (l) At regular intervals the programming team reflects on better ways to do things and then tunes their processes accordingly (Highsmith, 2002, 2004).

Although the Manifesto provides some specific ideas about more efficient and effective software development, a more significant motivation drives many software development managers. Software development managers are motivated by the values of trust and respect for others and the promotion of organizational goals (Highsmith, 2001). These goals promote a model based on people, collaboration, creativity, and the building of organizational communities. Agile practitioners concentrate on delivering good products to customers by operating in an environment that meets the needs of the customer and promotes a value-based culture (Highsmith, 2004).

A series of studies were conducted prior to the formation of the Agile Software Programming Methodologies Movement (Beck, 1999; Nosek, 1998). One of the methodologies studied was Extreme Programming (XP). This methodology actually existed prior to the advent of the Agile Alliance. Extreme Programming is credited with providing the basic motivation of the Agile Methodology Movement in providing a more rapid, less bureaucratic method of software engineering.

A History of Extreme Programming

Extreme Programming was created by Kent Beck in 1996 during his work on the Chrysler Comprehensive Compensation System (C3) payroll project. Beck became the C3 project leader in March of that year and began to refine the development methodology used in the project. Beck's publication, *Extreme Programming Explained* (2000), was the seminal work on the Extreme Programming methodology. Chrysler canceled the C3 project in February 2000, and although some saw that initial failure as a problem with XP, the methodology became popular throughout the software engineering field, especially during the dot.com era. Currently tens of thousands of software-development projects continue to use Extreme Programming as a methodology of choice (Beck, 2000).

Although Extreme Programming itself is relatively new, many of its practices have been in use for a considerable amount of time. It is the purpose of this methodology to take software engineering best practices to an extreme or never-tried-before level. For example, the practice of test-first-development -- planning and writing tests before each story or set of requirements -- was used as early as NASA's Project Mercury in the early 1960s (Larman & Basili, 2003). Other practices now used in the XP methodology such as refactoring, modularity, bottom-up, and incremental designs were described by Brodie (1984).

The focus of Extreme Programming is to reduce the cost of software change. In traditional system development methodologies, such as the Structured System Application Design Methodology (SSADM), the requirements for the system are determined at the beginning of the development project and often fixed or refactored from that point on. This means that the cost of changing the requirements at a later stage (a common feature of traditional software engineering projects) will be high. By implementing the practices of Extreme Programming, a software development manager can reduce the cost of change by introducing basic values, principles, and practices. By applying Extreme Programming practices, a software development project can be more flexible to changes without major disruptions in actual efficiency and effectiveness of the team.

A Review of Paired Programming

Extreme Programming's paired programming practice *pairs up* software development engineers or programmers into two-person teams. One person is called the *driver* or *director*, the other is called the *navigator* or *co-pilot*. The pair uses a single workstation to construct code with the director actually typing the code and the navigator dictating the code line and constructing the code logic. Both members make a constant review of the code for correct syntax and logical form. The pair uses stories, use cases, and pre-arranged acceptance tests to guide them in the engineering of the code. The customer or requirement provider presents stories, which are not tight specifications, but rather loose explanations of what the customer would like to see in the program (Beck & Fowler, 2001).

Unlike the traditional or the *Waterfall* approach to software development, stories within the Extreme Programming methodology provide only a guideline to the software engineering pair. The results of their work are generated from their knowledge and the application of currently employed architecture and software practices. The pair determines the best way to present the user's needs as represented through their interaction with the customer (in person and in stories), and then apply software-enabling technologies to provide the desired results for the customer. Customers join the pair during engineering, contributing clearer explanations of the desired work or results as the pair creates the code. Groups of stories that work together and are executable are called *iterations*. This allows the iterations to be placed into production sooner rather than waiting until the end of the entire programming development cycle to determine if everything works. Each iteration is considered to be *executable* or able to provide the required results as desired by the customer. The chain of iterations or executable segments, tied together, form the working application program (Beck, 1999, 2000).

Today the controversy continues over whether paired programming is as cost effective and/or efficient, as compared to individual programming. Continued experimentation in empirically valid venues can increase the reliability of the data that may point to whether paired programming is or is not perceived as easy to use and as more useful than individual programming. It does not appear that previous experimentation can necessarily predict the actual adoption of the paired programming practice or to what extent its use will be. Development managers who tend to engage software engineering methodologies and practices for their companies constantly make decisions about those practices based on their past experience and attitudes (Davis, 1989).

By understanding the attitudes of software development managers concerning paired programming practices, it may be possible to predict the continued and/or future use of paired programming as a viable software development practice based on perceived ease of use and perceived usefulness.

The Role of a Theoretical Framework to Advance the Study

Davis (1989) indicated that a methodological consideration of metrics for explaining and predicting the use of technology in processes and systems had significant practical value. By measuring intentions and attitudes, he was able to predict behavior and usage through a process called The Technology Acceptance Model (TAM) (see Figure 2). He noted this model's value was evident in the commercial environment by providing better business systems at better cost-points. This process continues to be important for information technology managers who must select and implement processes and products that support the operation and growth of their companies.

Underlying Concept of the Basic User Acceptance Model

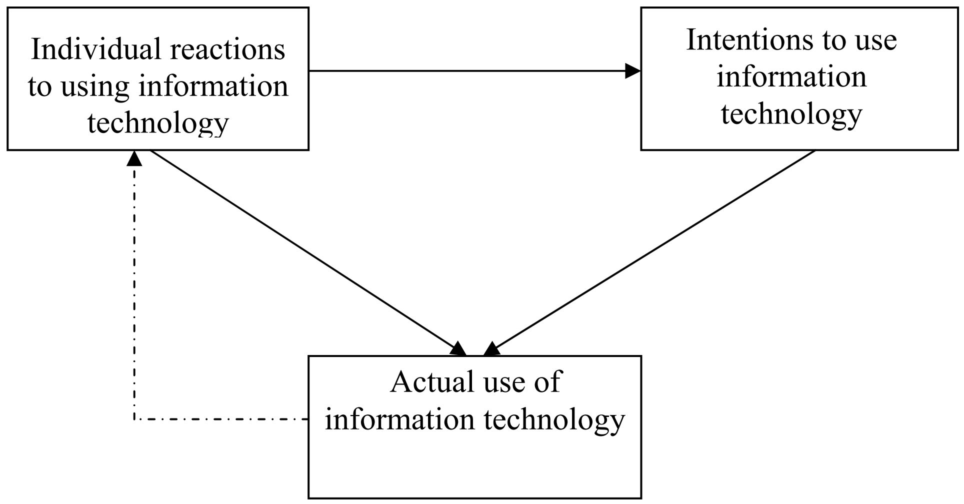

Individual reactions to using information technology
Intentions to use information technology
Actual use of information technology

Figure 2. Underlying concept of the Basic User Acceptance Model. The progression graph shows a representation of Davis' concept of user acceptance, which is a core premise of his Technology Acceptance Model. From *A technology acceptance model for empirically testing new end-user information systems: Theory and results.* (Doctoral dissertation, Massachusetts Institute of Technology-Sloan School of Management, 1985) *MIT Management Library* No. 1721.1/15192, p. 10. Copyright 1985 by the Massachusetts Institute of Technology Press.

The purpose of such metrics lies in a growing concern within the IT industry of controlling development costs and providing solid returns on application development investments. Users of metrics and methodologies have sought, for several decades, to eliminate the many false starts and incomplete software development projects as well as the many failed or abandoned projects experienced by businesses (see Figure 3).

Technology Acceptance Model

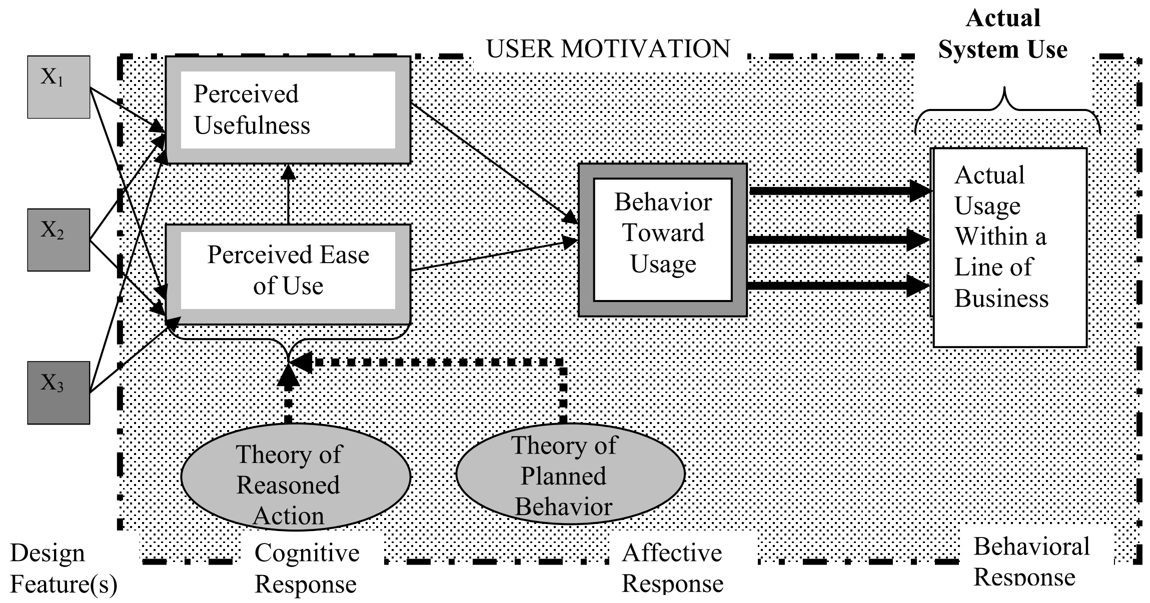


Figure 3. The Technology Acceptance Model. The model as designed by Davis (1985, 1989) showing the applicable psychometric supporting theories.

IT executives today are pressured to control the human capital and financial resources consumed by software development projects especially when there are sometimes questionable or even completely non-valuable results (El Emam & Koru, 2008). The Johnson and Standish Groups found, in a study of over 360 executives, that over half of all software development projects presented significant cost and time overruns (Johnson, 1994, 1995). They found that in 1995, businesses in the United States over-spent their Information Technology (IT) development budgets by $59 billion and threw away another $81 billion on projects that were not completed or were prematurely halted (Montealegre & Keil, 2000; Smith & Keil, 2003; Wallace & Keil, 2004). In a later study, they found that over 70% of information technology (IT) software projects failed or were abandoned (The Standish Group, 2001, 2004). The logical conclusion is that IT software development capability cannot contribute to the corporation's financial performance when software projects are not completed or delivered on time and within planned budgets (Markus & Keil, 1994).

Successful software development projects have discernable elements, which include involvement by the customer or user, controlled and minimized scope with reasonable and manageable milestones, standardized software infrastructure, and the discipline of a formal software development methodology (Standish Group, 2001). In the last decade, the software development industry has uncovered several ways and means to improve the success of software projects. These initiatives include focused efforts to create better and more understandable software development practices, iterative or modular based programming efforts, and the reuse of existing knowledge and software artifacts for future coding projects (Wallace & Keil, 2004). By improving the methods by which software engineering is accomplished, improvements can be made in the successful implementation of software projects and an increase in valuation of software development investments can be realized within businesses.

The last 30 years have demonstrated a very explosive environment in software engineering. Development managers must produce a never-ending backlog of application development projects with continual demands for the most up-to-date features using the latest in development methodologies (Lindstrøm & Jeffries, 2004). Multiple empirical surveys confirm that most software projects fail when measured against even the most minor success metrics. Most projects fail due to unclear requirements, requirements that fail to solve underlying business issues, requirements that consistently change, untested software, and software that is created from incomplete requirements (Lindstrøm & Jeffries, 2004).

These conditions spawned efforts by many developers to drop multiple step methods and detailed design practices. Some wrote papers that proposed lighter, more flexible and open methods. In 1999, Beck published his book, *Extreme Programming Explained: Embrace Change* that solidified the Extreme Programming method of software development (Beck, 1999). It described the Extreme Programming method as a series of practices that was lighter and more open in its approach to software engineering. The trend of developing software in a more open and flexible environment became popular in software development circles (see Figure 1).

Extreme programming is the most popular and pervasive methodology used in the Agile movement. Its primary tenets include simplicity, communication, feedback, and courage. It not only espouses the particular values of the Agile Manifesto, but also employs a flexible yet simple set of programming practices. Paired programming is one of the most popular of its practices (Lindstrøm & Jeffries, 2004). This practice engages a review by an additional programmer, other than the coder, resulting in better design, more complete testing, and overall code results that are more accurate and bug free. Bugs are errors in code that fail to return results or fail to return the correct results (Grenning, 2001).

Critics of paired programming cite that two resources working in the place of one are cost prohibitive and wasteful (Cockburn & Williams, 2001). As indicated above, initial experiments at the University of Utah indicated that interactions between the pair working in a collaborative environment, when measured for overall effectiveness as compared to programmers working alone, contributed to more effective and efficient code production (Williams, 2000). These experiments were held in an academic setting, used students with prior knowledge of the aims of the experiment, and did not use commercially directed business programmers. Questions continue to mount throughout the Information Technology industry as to the viability and practical applicability of paired programming in actual business environments (Cockburn & Williams, 2001).

Statement of the Problem

Negative economic factors are pressuring businesses today to cut costs and operate leaner. The use of the paired programming practice, which places two programmers together coding on one workstation, generates the question of why this programming practice is economically viable in today's business climate. Specifically, it raises the question as to what are software development managers' intentions and attitudes toward using this practice and what is the possible future use of paired programming as compared to individual programming. If the paired programming practice is a solution to the problems of software engineering being on time and within budget, is it successfully being used by software development managers and to what extent?

There is evidence that the perception of many business leaders, especially those not experienced in the software or applications development business, view paired programming as a costly personnel resource practice (The Standish Group International, 2004). Many of those same business leaders apply pressure on their software development managers to maintain an effective and efficient work environment (Hartwick & Barki, 1994; Ives & Olsen, 1984). The pressure to complete software projects that are completed on time and within budget is a common theme in industrial publications as well as academic journals (Mugridge, 2008). Software development managers look for better and easier ways to produce useful code, while driving toward on-time results that are within or under budget (Krishnakumar & Sukumaran, 1997; Putnam, 1978).

Purpose of the Study

The objectives of this research study are focused on providing an empirical basis on which practitioners, namely software development managers and business leaders with engaged software development resources, might decide to adopt or not adopt the practice of paired programming as a software development practice. Specifically, the objectives of this research are to: (a) Survey development managers on their attitudes toward the use of paired programming as opposed to individual programming, (b) Use the results of the data collected on attitudes and intentions and then apply The Technology Acceptance Model to the variables of perceived usefulness and perceived ease of use, (c) Determine whether the standard business type, as defined by the standard industrial classification (SIC) of a participating manager's company, correlates with the use or non-use of paired programming, and (d) Establish the predicted use or non-use of paired programming by sending a validated and reliable sample of software development manager attitudes and intentions through Davis' (1989) Technology Acceptance Model.

A review of the current literature shows no evidence of a study that has applied the TAM to predict the acceptance and use of paired programming. The results of this research contribute to the body of knowledge through the survey of development managers and the engagement of that survey data through the mathematical TAM model (Davis, 1993). The ramifications to the field are widespread. Strong affirmation for the use of paired programming will continue to support the current peer-studies, which find that paired programming is a cost effective and nimble alternative to the practice of individual programming. Conversely, a lack of support for paired programming will provide additional momentum for a tier of software development and business leaders that feel this practice is over-rated, costly, and inefficient (Nawrocki & Wojciechowski, 2001). Moderate support for either paired programming or individual programming practices will indicate that the field is still ambivalent towards paired programming and that more education, experience, and data are needed to form a more delineated industry direction.

It is normal for practitioners to evaluate various practices, processes, tools and even systems to understand their acceptability and/or understand their reasons for a lack of acceptance (Davis, 1989). “A study of how usefulness and ease of use can be influenced by various functional and interoperable factors such as externally engaged software development methods is important” (Alavi, 1984, p. 562). The purpose of this study is not to promote the practice of paired programming but to understand user acceptance and usage in the attempt to thwart what Davis (1989) says is “The possibility of dysfunctional impacts generated by information technology [practices]” (p. 335).

Rationale

The result of this study indicates a gap in the body of knowledge regarding the predicted usage of paired programming by analyzing the responses of software development managers. The methodology used in this study is a widely applied theoretical model called The Technology Acceptance Model, to determine the predictable use of the software technology practice of paired programming. Perceived ease of use and perceived usefulness are the basic variables of The Technology Acceptance Model. The results of the research from this study support or discredit previously and/or currently held beliefs that paired programming is more easy to use and is more useful as a software engineering practice than individual programming practices.

In Chapter 2 of this study, there is substantive research demonstrated in the areas of The Theory of Reasoned Action, The Theory of Planned Behavior, and The Technology Acceptance Model. Sufficient reliability and validity have been established for the use of The Technology Acceptance Model as a viable theoretical framework from which to study the acceptance and use of a technology practices, processes, or tools. Over 146 various technology practices and tools have used The Technology Acceptance Model as a method to demonstrate its acceptance and use. The operation of this study follows the same track, using The Technology Acceptance Model to measure the acceptance and usage of the paired programming practice. There has not been a study that applied the paired programming and individual programming practices through The Technology Acceptance Model to determine possible usage. The results of this study add to the body of knowledge concerning the degree of acceptance and use of paired programming as perceived by software development managers, using a well-tested and accepted theoretical modeling tool such as The Technology Acceptance Model.

The findings resulting from this research could yield some significant results, including: (a) Influencing whether paired programming becomes established or remains established at company types closely related to the US Department of Commerce, Standard Industrial Code, (b) Giving business leaders a reason to support or not support the practice of paired programming in their companies, (c) Determining if the use or acceptability of paired programming is influenced by the type of business being considered, and (d) Determining the general acceptability of paired programming as a software development practice.

In a separate, but related consideration, the results support (or challenges) the research method and applicability of The Technology Acceptance Model as a predictive tool in the determination of the perceived use and perceived usefulness of a technology practice (Bagozzi, 2007). To date, no research study has applied the TAM to the practice of paired programming to determine its perceived ease of use and perceived usefulness, acceptance and usage in a business environment.

Davis (1989), in his seminal work on perceived usefulness, ease of use, and user acceptance of information technology, states the simple rationale for a study such as this as well as other studies that attempt to determine the acceptance of technology by users. He states,

> Although there has been a growing pessimism in the field about the ability to identify measures that are robustly linked to user acceptance, the view taken here is much more optimistic. User reactions to computers are complex and multifaceted. But if the field continues to systematically investigate fundamental mechanisms driving user behavior, cultivating better and better measures and critically examining alternative theoretical models, sustainable progress is within reach. (p. 335)

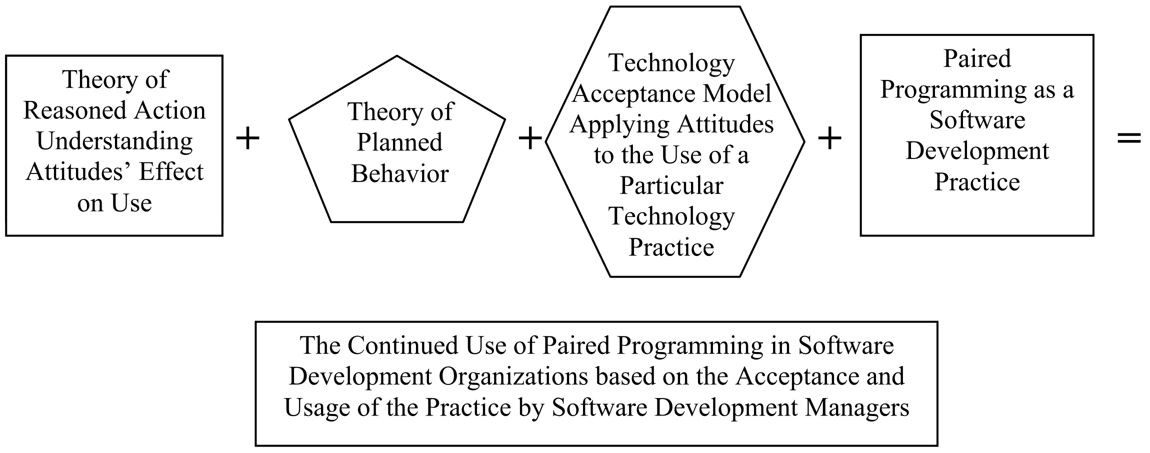
Research Questions

The primary research question of this study is how and to what extent, software development managers, perceive the paired programming practice as useful and easy to use, as compared to individual programming practices. Consequently, to what extent can The Technology Acceptance Model (TAM) predict the acceptance and continued usage of the paired programming practice? Other research questions that are addressed in this study include: (a) R1: Are there observed differences in perceived ease of use of the practice of paired programming versus individual programmers working alone? (b) R2:

Are there observed differences in perceived usefulness of the practice of paired programming versus individual programmers working alone? (c) R3: To what extent is the paired programming practice more acceptable/used than individual programmers working alone? (d) R4: How does the type of business correlate to the acceptance/usage value of paired programming? (e) R5: What is the relationship between a software development manager's development experience and his or her acceptance/use of the paired programming practice?

It is important to consider a number of factors in order to answer this primary question and the various subsequent questions. Some factors are based on pertinent studies from relevant and supporting theory such as the Theory of Reasoned Action (TRA). This theory associates attitudes with reasoned or predicted action. The Technology Acceptance Model (TAM) is another theory that predicts the value (amount) of an action or adoption based on empirical data representing attitudes of possible adopters. The objective in this study is to identify correlations between key elements and factors of software development managers' intentions and attitudes about paired programming. The results of the research project the general probability of adoption, use of the paired programming practice, and provide data for future studies and/or experiments (see Figure 4). There are few scientific studies existing today where paired programming has been systematically researched within actual software engineering environments. There is little solid evidence or empirical data showing the perceived benefits of this practice (Hulkko & Abrahamsson, 2005). By addressing these research questions, future experimentation can focus on parameters that yield greater amounts of relevant data for this field study and ultimate decision-making information for software development managers.

The Theoretical Models Used in this Study.



Theory of Reasoned Action Understanding Attitudes' Effect on Use
+
Theory of Planned Behavior
+
Technology Acceptance Model Applying Attitudes to the Use of a Particular Technology Practice
+
Paired Programming as a Software Development Practice
=
The Continued Use of Paired Programming in Software Development Organizations based on the Acceptance and Usage of the Practice by Software Development Managers

Figure 4. The theoretical models used in this study. The figure shows the various theories and models that provide the basis for the research methodology in the study of the acceptance and usage of the paired programming practice.

Hypotheses

The following hypotheses were explored using Davis' (1989) original research methods for proving the acceptance and usage of a technology practice and/or tool, and Rigopoulos and Askounis' (2007) application of the TAM framework to a technology practice (see Table 1). The hypotheses are best viewed through the five constructs or general variables of the TAM survey. These include perceived ease of use, perceived usefulness, acceptance/usage, effects of programming experience, and effects of a programmer's business type. Additional respondent experience data such as business type and previous use and experience with paired programming was collected and used for statistical analyses that support the confirmation or rejection of the following hypotheses.

Table 1. The Technology Acceptance Model Mathematical Formula

The Technology Acceptance Model is expressed in four linear and progressive formulas:

	Variable		Formula
1)	EOU	=	$\sum_{i=1,n} \beta X_i + \varepsilon$
2)	USEF	=	$\sum_{i=1,n} \beta_i X_i + \beta_{n+1}\,\text{EOU} + \varepsilon$
3)	ATT	=	$\beta_1\,\text{EOU} + \beta_2\,\text{USEF} + \varepsilon$
4)	USE	=	$\beta_1\,\text{ATT} + \varepsilon$

In the case where:

Variable		Definition
X_i	=	Design Feature *i, i*=1, *n*
EOU	=	Perceived Ease of Use
USEF	=	Perceived Usefulness
ATT	=	Attitude Toward Using
USE	=	Actual Use of the System
β_i	=	Standardization Partial Regression Coefficient
ε	=	Random Error Term

Note. From Davis, F. (1985). *Technology Acceptance Model for Empirically Testing New End-User Information Systems: Theory and Results.* Massachusetts Institute of Technology, Sloan School of Management, p. 25. Copyright 1985 by the Massachusetts Institute of Technology Press.

The first hypothesis, which attempts to address research question R1: Are there observed differences in perceived ease of use of the practice of paired programming versus individual programmers working alone, is expressed as:

Ho1: The mean TAM score of the perceived ease of use construct will not be significantly different for paired programmers versus individual programmers working alone.

Ha1: The mean TAM score of the perceived ease of use construct will be significantly different for paired programmers versus individual programmers working alone.

The second hypothesis, which attempts to address research question R2: Are there observed differences in perceived usefulness of the practice of paired programming versus individual programmers working alone, is expressed as:

Ho2: The mean TAM score of the perceived usefulness construct will not be significantly different for paired programmers versus individual programmers working alone.

Ha2: The mean TAM score of the perceived usefulness construct will be significantly different for paired programmers versus individual programmers working alone.

The hypothesis, which attempts to address research question R3: To what extent is the paired programming practice more acceptable/used than the practice of individual programmers working alone, is expressed in the following statements:

Ho3: The paired programming practice will not be significantly perceived to be more used by software development managers than the individual programming practice as determined through a comparison of means of the USE variable in each practice.

Ha3: The paired programming practice will be significantly perceived to be more used by software development managers than the individual programming practice as determined through a comparison of means of the USE variable in each practice.

The hypothesis, which attempts to address research question R4: How does the type of business correlate to the acceptance/usage value of paired programming, is expressed in the following statements:

Ho4: There is no statistically significant mean difference between a software development manager's type of business and the level of acceptance/usage of the paired programming practice.

Ha4: There is a statistically significant mean difference between a software development manager's type of business and the level of acceptance/usage of the paired programming practice.

The hypothesis, which attempts to address research question R5: What is the relationship between a software development manager's development experience and their acceptance/use of the paired programming practice, is expressed in the following statements:

Ho5: There is no statistically significant mean difference between the variables of a software development manager's development experience and the level of their acceptance/usage of the paired programming practice.

Ha5: There is a statistically significant mean difference between the variables of a software development manager's development experience and the level of their acceptance/usage of the paired programming practice.

The statistical tests and methods of analysis are found in the *Data Analysis* section of Chapter 3. The values derived from the tests in Hypothesis 1 through Hypothesis 5 establish a basis upon which future studies, especially those using a longitudinal approach, might be made. This and other reflections of the possibilities for extended use of these tests are made in Chapter 5.

Significance of the Study

In several controlled experiments, it has been shown that paired programming provided improvements, sometimes significant, over individual programming (Arisholm, Gallis, Dybå, & Sjøberg, 2007). These improvements ranged from functional quality or correctness to other measures of quality including a reduced time to market with only minor additional overhead or cost of effort (Müller, 2004; Nosek, 1998). Yet additional studies counter these experimental results indicating that there were little or no functional improvements or improvements on correctness as compared with individual programmer development efforts (Nawrocki & Wojciechowski, 2001). Still other studies point to paired programming as being more a label or a popular movement that focuses on skilled and motivated workers with already existing high morale. The results of these types of workers are going to be naturally productive despite the paired practice or any other systematic method as long as they are not overly constrained by organizational bureaucracy (Hilkka, Tuure, & Matti, 2005).

These studies leave software development managers in a quandary as how best to service their companies with the most cost effective and efficient code producing groups. Business managers and leaders are faced with economic downturn limits and reductions in capital investment capabilities (Ebert & deNeve, 2001). Using two people where at some point in the past there was only one, does not seem to make empirical, logical, or business sense (Lindstrøm & Jeffries, 2004). This can put development managers and business leaders at odds in determining the possible success (measured in terms of ease of use and usefulness) of the paired programming practice. By surveying and analyzing the attitudes of development managers about paired programming, using an acceptable theoretical construct such as the TAM, the research results in an added predictive value to either paired programming's use or rejection within any generic applications development organization. Using the results of this study, software development managers could sway the use or rejection of the paired programming practice and possibly create new directions in software development through changes in their own software engineering practices.

There are over 146 reported studies on the use of The Technology Acceptance Model to determine and predict current and future behavior and acceptance of a technology practice, process, or tool. The common use of TAM provides a rich methodological backdrop for this study and a solid foundation of previous research using its theoretical framework. Baker-Eveleth, Eveleth, O'Neill, and Stone (2006) applied the TAM to the use of laptop exams and security software. Keat and Mohan (2004) applied the TAM to various electronic commerce tools to determine ease of use and usefulness. Pei, Zhenxiang, and Chunping (2007) applied the TAM to measure the acceptance, ease of use, and usefulness of Chinese business-to-business website designs. Perez, Sanchez, Carnicer, and Jimenez (2004) applied the TAM to teleworking processes and practices. Pikkarainen, Pikkarainen, Karjaluoto, and Pahnila (2004) applied the TAM to the acceptance of online banking in third world countries.

These are just examples of the application of the TAM toward various practices, processes, and tools. This study's results will replicate the success of these previous studies with a focus on the paired programming practice of software development engineering. A review of the literature reveals that paired programming has yet to be applied to The Technology Acceptance Model and there are no formal reported results or formal studies associated with this model in peer-reviewed works. The results of this research provide pertinent data to fill the gap in the body of knowledge concerning the acceptance of paired programming by software development managers as calculated through the application of The Technology Acceptance Model.

Definition of Terms

The following definition of terms expose the reader to a greater understanding of the concepts, hypotheses, and findings of this study:

Agile Methodologies – A title that refers to a group of software development methodologies that promote development iterations, open collaboration, and process adaptability throughout the life cycle of a project. The Agile methodology emphasizes working software as the primary measure of progress. Combined with the preference for face-to-face communication, agile methods usually produce less written documentation than other methods.

Director – A position or role in the technique where two programmers work together at one workstation, one programmer (the *driver or director*) is assigned to typing the actual code while the other (the *navigator*) reviews each line of code as it is typed.

Ease of Use –The measure or degree to which a person believes that using a particular system will result in a freedom of effort. A person may believe that the system is easy enough to use and that the performance benefits of usage are outweighed by the effort of using the application.

Effectiveness – The production of a desired, decided, or decisive result and/or the fulfillment of a purposeful or intent, especially as viewed after a related or associated event.

Efficiency – The production of a desired effect without waste. It is the ratio of the useful energy delivered by a dynamic system in relation to the energy supplied to it.

Extreme Programming – A software engineering methodology (and a form of agile software development) prescribing a set of daily stakeholder practices that embody and encourage the specific values of Communication, Simplicity, Feedback, Courage, and Respect. Proponents believe that exercising these values will lead to a development process that is more responsive to customer needs and be more *agile* than traditional methods, while creating software of better quality.

Information Technology (IT) – The general term and acronym used in this study to include the various disciplines in the management information system sciences and information system disciplines. It refers to previous acronyms such as MIS (management information systems) or IS (information systems). For simplicity and convention, the term IT will be a general term meant to encompass these and other acronyms usually associated with Information Technology sciences and disciplines.

Iterative Development – The scheduling strategy within Agile methodologies where time is set aside to revise and improve parts of the system. It does not presuppose incremental development, but works very well with it. The output from an *iteration* is reviewed for modifications and/or for any revisions that could be targets for successive iterations.

Navigator – A position or role that reviews the code that is written in the technique where two programmers work together at one workstation. While reviewing, the observer or navigator also considers the strategic direction of the work, coming up with ideas for improvements and likely future problems to address. This frees the director to focus all of his or her attention on the tactical or syntactical aspects of completing the current tasks, using the navigator as a safety net and guide. The two programmers switch roles frequently.

Paired Programming – is a software engineering practice in which two programmers work together at one workstation. One types in code while the other scans each line of code as it is typed. The person typing is called the driver or director. The person reviewing the code is called the observer or navigator. The two programmers switch roles frequently. While reviewing, the observer also considers the strategic value of the work, creating new ideas for improvements and possible future problems that might occur. This liberates the driver to focus on the operational aspects of completing the planned task, using the observer as both a guide and a safety net. XP programmers write all production code in pairs, two programmers working together at one machine. Paired programming has been shown by many experiments to produce better software at similar or lower cost compared to programmers working by individually. For the purposes of this study, the term pair programming and paired programming will be considered synonymous.

Single Programming – A traditional programming practice where one programmer works alone engineering and/or coding software routines toward a final executable programming on toward a contribution to a team-effort that combines the contributions of many single programmers into an executable running program.

Software Development Life Cycle (SDLC) – The model or methodology that is used to develop systems, most generally computer software systems. It adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation. The oldest model that is originally regarded as *SDLC* is the waterfall model. It is a model that involves a sequence of stages in which the output of each stage becomes the input for the next.

Standard Industrial Classification (SIC) – The Standard Industrial Classification (*SIC*) is a United States government system for classifying industries by a four-digit code. Established in 1937, it was recently updated by using a six-digit code and was renamed the North American Industry Classification System (NAICS), released in 1997. Certain government departments and agencies, such as the U.S. Securities and Exchange Commission (SEC), still use the acronym SIC to represent standard industry (six-digit) codes (see Table 3).

Technology Acceptance Model (TAM) – A model designed to predict a particular technology tool, process, or practice's acceptance and usage. It embraces two variables, that of perceived usefulness and perceived ease of use.

Theory of Planned Behavior (TpB) – The theory, based on the Theory of Reasoned Action, that indicates the possibility of predicting planned and deliberate behavior from attitudes and non-voluntary behavior characteristics (Ajzen & Fishbein, 1980; Ajzen, 1991).

Theory of Reasoned Action (TRA) – The theory and model that is drawn from social psychology, which also represents a fundamental as well as influential force of basic human behavior. It is an individual's positive or negative feelings about performing a target behavior or demonstrating an acceptance or rejection of an action, process, tool, or application.

Usefulness – The measure or degree to which a person believes that using a particular system would enhance his or her job performance or help them do their job better.

User Stories – A user story is a software system requirement formulated as one or two sentences in the everyday language of the user. These are written by the customers of a software project and are their main instrument to communicate their requirements and influence the development of the software and its functionality.

Waterfall Programming Methodology –A sequential software development model for the creation of application software in which development is seen as flowing steadily downwards (like a waterfall) through the phases of requirements analysis, design, implementation, testing (validation), integration, and maintenance. The origin of the term *waterfall* is often credited to Royce (1970). Although Royce did not use the actual term waterfall in this article, he ironically presented the model as an example of a flawed, non-working software engineering method. Today the term is often used as a derogatory label to software development methods that are heavily bureaucratic, wasteful, and over-complex.

Assumptions and Limitations

The goal of the research in this study is to collect the intentions and attitudes of software development managers on paired programming and compare them to their intentions and attitudes on single programmer work. An email-connected web survey was the vehicle used to collect data on intentions and attitudes associated with perceived usefulness and perceived ease of use of the paired programming practice. To achieve a reasonable response and correlation of data the following assumptions and challenges were encountered: (a) Although Extreme Programming is a popular and well-known methodology, not all software development managers responding to the survey are familiar with this methodology (Turk, France, & Rumpe, 2005), (b) Development managers will respond honestly to the survey questions, (c) A random list of 500 software development managers will be a representative sample of software development managers' intentions and attitudes, (d) The preponderance of emails will be from, but not limited to, the United States, (e) Surveys are common in the IT industry and IT executives tend to be cooperative when asked to do surveys that will yield new technology information, (f) In this study, the intentions and attitudes of software development managers were measured; and when applied to the theoretical model of the TAM, they indicate the likelihood of adoption or rejection of the paired programming practice, (g) It is assumed that Davis' Technology Acceptance Model (1985, 1989) is an accurate mathematical representation of an acceptance model for a technology practice, (h) Paired programming has been identified to include variations in skill sets among programmers, which can affect the possible acceptance of this practice by businesses (Arisholm et al., 2007), (i) It was assumed that a small pre-test sample of questionnaires will be sufficient to validate the data, and (j) It was assumed that at least 150 out of 500 surveys would be returned completed.

The Nature and Theoretical Framework of the Study

The Technology Acceptance Model (TAM) as proposed by Davis (1985, 1989) and refined by Davis, Bagozzi, and Warshaw (1989) and Venkatesh, Morris, Davis, and Davis (2003) are employed in this study as the theoretical framework. To provide a basis for The Technology Acceptance Model, the Theory of Reasoned Action (TRA) (Fishbein & Ajzen, 1975; Hale, Householder, & Greene, 2003) was introduced. This forms a psychological grounding for the use of the TAM. The TRA is drawn from social psychology and is a significantly influential and fundamental theory of human behavior. The TRA has been used to anticipate a wide range of behaviors and practices (Venkatesh et al., 2003). Davis (1985) verified that the TRA was consistent in output/results with various studies when compared within the context of other behaviors (Venkatesh et al., 2003).

The TAM is specifically built for IT contexts and environments, and was designed to predict IT acceptance of a practice and its acceptance in the context of the job (Venkatesh et al., 2003). Measurement of perceived usefulness and perceived ease of use (Davis, 1985, 1989) are used in conjunction with the measurement of actual behavior and usage (Arisholm & Sjøberg, 2003) to predict the continued use of paired programming as a software engineering practice. The research in this study exposes the intentions and attitudes of software development managers as data and applies that data through the TAM formula to indicate the continued use or rejection of the paired programming practice in software engineering.

Statistical Package for Social Sciences (SPSS) version 15.0 and Microsoft Excel 2003 Data Analysis Packages were employed to do the electronic calculations and reporting for the analysis. The use of a part-time statistician from Modern Analytics, Inc. was arranged to assist with the statistical analysis calculations and data review. Two statistical tests are core to the TAM, correlations, and linear regression. These tests were applied to the variable constructs of perceived usefulness, perceived ease of use, and self-reported usage in keeping with Davis' (1985, 1989) original model (see Table 1).

Organization of the Remainder of the Study

Chapter 4 – Results

In the fourth chapter of this study, the data collected, through the survey of software development managers, is analyzed through the TAM formula and hrough a series of standard statistical tests. The survey used in this study returned data on the intentions, attitudes, and use of paired programming from a sample of software development managers. There are four basic constructs to the survey, three of which are part of the TAM formula. These constructs are used as independent variables with the fourth construct being used as a dependent variable. Through the process, independent and dependant variables are identified, and through analysis, demonstrate their ability to confirm or reject the listed null hypotheses. Tables and graphs are used to display the data and analysis results. Each hypothesis uses a specific statistical analysis for confirmation or rejection of its hypothesis statement.

Data collected from the administered surveys are presented in output tables. Each section of the six-section survey, found in The Appendix, is highlighted in the first section of Chapter 4. Data table summaries are displayed along with the mean values associated with the sample frame used for this study. Summary calculations are made for each survey section, and in some cases, for specific questions. The second section of this chapter addresses each hypothesis, draws the summary data relevant to that hypothesis, and demonstrates the results of statistical analysis against the relevant data. Tabular representation of the analysis results pertinent to each hypothesis are graphically as well as numerically presented.

Chapter 5 – Discussion, Implications, Recommendations

The fifth chapter synthesizes and evaluates the analysis of the collected data. Correlations of the findings produce conclusions about the acceptability and intention to use the paired programming or individual programming practice by a representative sample of software development managers. The resultant data collected and analyzed on perceived ease of use and perceived usefulness are correlated to the variables of self-reported usage and behavioral use. The results of the study also reveal the possible effects of business types on the use and acceptance of paired or individual programming practices. The study also reveals whether software development managers who have used paired programming in the past will continue or discontinue using it in the future. The affect of business types and the continued use of paired programming, after having used it in the past, are findings that will promote future studies.

The sections in this chapter address the evaluation and statistical analysis of each hypothesis. Based on the analysis results, the hypotheses are confirmed or rejected with an explanation and/or recommendation as to the meaning of the results. As this is the first study to apply the TAM to the use of the paired programming and individual programming practices, the findings indicate a need for further study and possible new experimentation to determine the continued usefulness and viability of paired and individual programming practices.

CHAPTER 2. LITERATURE REVIEW

The literature is rich in the areas of the TRA, TAM, and paired programming. The following review of the literature is segmented into three sections that cover the TRA, TAM, and paired programming respectively. These general areas make up the theoretical model and structured methodology of this study.

A review of the TRA provides the basis on which the human condition navigates between various types of responses that create direction of intention or mode of action (Ajzen, 2005). This theory can be applied to the acceptance and usage of a practice or a process by understanding the intentions and behaviors of an actor. Seminal works by Ajzen and Fishbein (1980) and Ajzen (1985, 1991) reveal that connected psychological processes between belief and attitude consistently produce a directional intention. Ajzen and Fishbein later perfected this theory through the introduction of subjective norms and created a series of theories around planned behavior. By understanding one's beliefs, it is possible to predict its effect on an actor's attitudes, which in turn provides an indicator of that actor's intention toward using a process or a practice. Considerations of these theories yield a basis on which to consider the advanced theories of The Technology Acceptance Model (Davis, 1985, 1989).

In the second part of this review, Davis' Technology Acceptance Model (1985, 1989) provides a logical (and mathematical) continuation of Ajzen's and Fishbein's work. The literature will show the various applications of this model, based on the theory of reasoned action and the theory of planned behavior. It establishes that the intention of an actor can be predicted based on the perceived (by the actor) ease of use and perceived usefulness of the proposed action or process (Davis, 1989). By understanding and/or testing the variable strength of an actor's perception of a process, it is possible to predict the actual engagement of that process by the actor through comparative and empirical values. The Technology Acceptance Models' mathematical formula, when applied to most technology processes, tools, or practices, can produce measurable and predictive results about their acceptance and use (see Figure 5 for a timeline of various models used in this study).

A Time Line of Theoretical Models Used in this Study

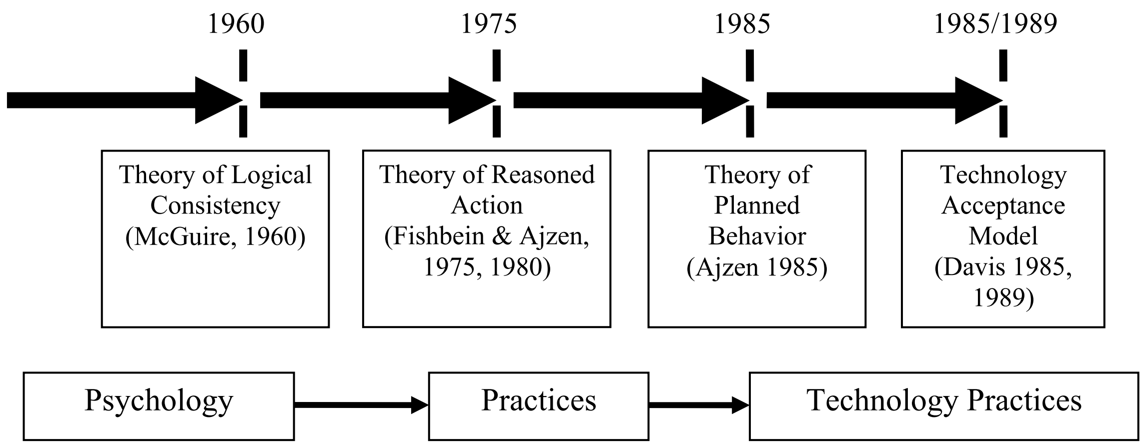

1960
1975
1985
1985/1989
Theory of Logical Consistency (McGuire, 1960)
Theory of Reasoned Action (Fishbein & Ajzen, 1975, 1980)
Theory of Planned Behavior (Ajzen 1985)
Technology Acceptance Model (Davis 1985, 1989)
Psychology
Practices
Technology Practices

Figure 5. A time line of theoretical models used in this study. The graph shows the time frames and progression of the various theoretical models used in the development of The Technology Acceptance Model.

The third part of this review will demonstrate the various contributions on the subject of paired programming. By an understanding of the specific elements of this Extreme Programming practice, a foundation can be constructed upon which the practice of paired programming can be applied to The Technology Acceptance Model. The basis of this study, which is to determine the acceptance of paired programming by software development managers, is achieved by applying the collected intentions, beliefs, and attitudes of those managers regarding their possible use (behavior) of paired programming in their software engineering environments. By an enriched understanding of the current writings on paired programming, it is possible to establish a knowledge base on how software development managers might apply (or not apply) the practice of paired programming based on those managers' attitudes about the practice's perceived usefulness and perceived ease of use. The literature will also support additional inquiries into the relative values of perceived effectiveness and perceived efficiency as software development managers continue to support the demands of their businesses with productive and cost-sensitive code engineering.

The Theory of Reasoned Action and the Theory of Planned Behavior

Literature in human psychological development indicates that people are naturally consistent in their outward responses due to the methods they employ in processing sensory data and information resulting in decision-making (Ajzen, 2005). The most notable works in the translation of human attitudes-to-intention-to-action include those of Ajzen (1985, 1991, 2002) and Fishbein and Ajzen (1975). The first theory, that of reasoned action came from McGuire and Weiss' theories on logical consistency (1976) that drew together formal and statistical probability. Considering two or three related known variables (intention and attitude), it is possible to determine a third unknown (actual behavior). Ajzen and Fishbein matured this into their theory of reasoned action.

They later matured the theory to include the effects of controlled behavior and that of environmental influence, to form the theory of planned behavior (Ajzen, 2005).

Theory of Reasoned Action

Fishbein and Ajzen (1975) first proposed the theory of reasoned action with the concept that humans employ logical synthesis in their thinking and acting. Developing this concept further, Ajzen and Fishbein (1980) proposed the theory of reasoned action. Rather than treating the logical constructs of human information processing as a purely psychological activity, they proposed that [human] responses were demonstrative, observable reflections of the cognitive factors and could be explained through independent constructs of belief, attitude, and intention. What they indicated was much more than a construct. It was a progressive process of believing, then thinking, then acting on a cognitive stimulus. A residual effect of this process results in attitudes. They held that attitudes are logical results from beliefs humans have about an object of that attitude. Intentions are created around that attitude, and the actions taken derive as well from that attitude (Ajzen, 2005). This clearly makes a connection between what is believed and thought, forming intentions upon which humans ultimately act (see Figure 6). By understanding the objects of those beliefs and attitudes, researchers can then project a person's possible intentions. Together, beliefs, attitudes, and intentions can help predict a person's resulting actions (Donald & Cooper, 2001).

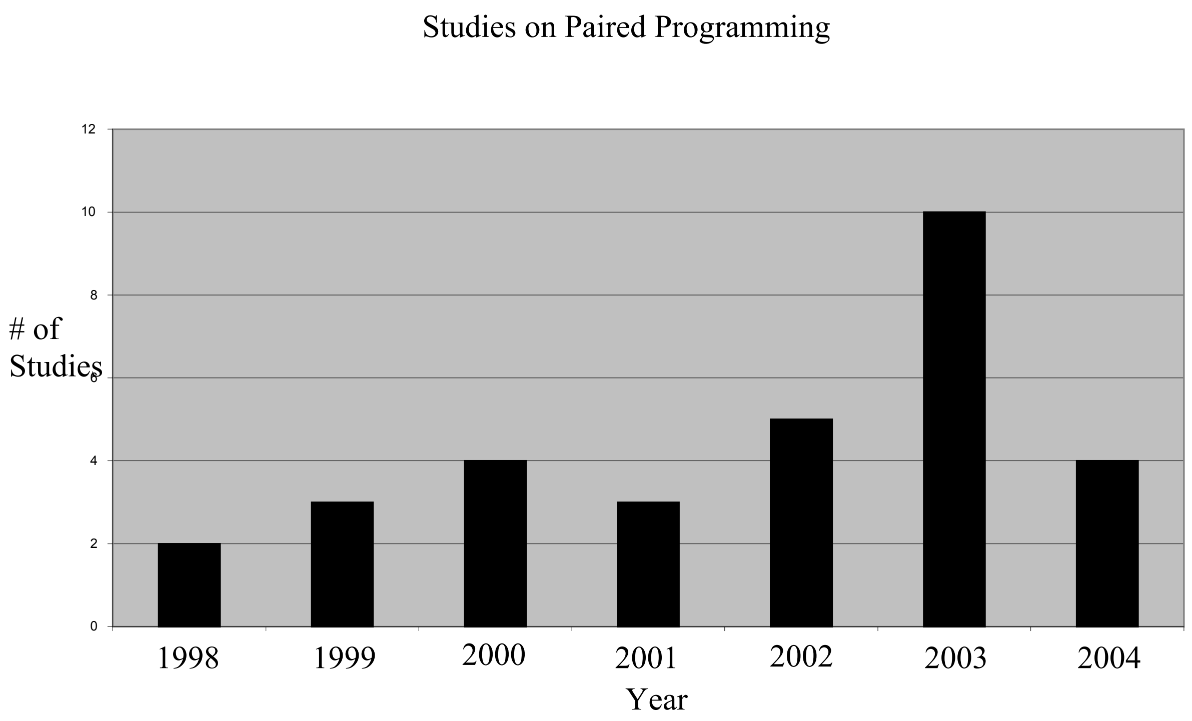


Figure 6. Studies on paired programming. Number of studies on paired programming per year.

Ajzen's and Fishbein's work attempted to place empirical values or estimates between attitude and behavior. Their studies dealt with mostly voluntary behavior, which was counter to their later findings that indicated behavior was not necessarily voluntary nor was it completely controllable (Ajzen, 1991). These findings added another consideration of perceived behavior control. This enriched the concept of reasoned behavior to one of planned behavior. This adds to the TRA a predictability of behavior when it is deliberate and planned. This improved the results of the TRA and created an additional theory, the Theory of Planned Behavior (TpB) (Ajzen, 1991).

Theory of Planned Behavior

The Theory of Planned Behavior proposes that specific attitudes projected toward behavior of a subject will predict the behavior toward that subject, but only that subject (Ajzen, 2002). In addition to measuring an actor's attitudes to predict behavior, it is also important to measure the influence on an actor's attitude by the attitudes of other people surrounding or involved with the actor. In this situation, the actor must have some measure of value or reliance on the environment and/or those people that are a part of it. The ability to predict a person's intentions requires this additional understanding of that person's surrounding influences and how that influence is different from their beliefs. This understanding is termed the influence of subject norms (Terry, Hogg, & White, 1999).

The last influence that helps to create intention is the actor's perception whether he or she has the ability to perform the specific behavior. If people are not able to see themselves in control of this behavior, the intention to act is not sufficiently motivated and less likely to happen (Notani, 1998). The previous two elements of what effects behavior (a favorable attitude toward the specific behavior and environmentally affected subject norms) helps to form the perceived ability to control the one's behavior and strengthens a person's resolve or intention to actually perform that specific behavior (Manstead & Parker, 1995).

TRA and TpB Scope and Application

Ajzen (1991) provided a clear set of instructions to develop studies and questionnaires on the theories of reasoned action and planned behavior. He demonstrated the ability to obtain data on attitudes and intention and predicted specific behavior, through a questionnaire instrument. He validated this instrument with follow-up reviews conducted by direct observation and other self-reporting tools. Specifically, he suggested that the theory of reasoned action could conform to a question-based tool that would support the evaluation of particular studies such as voting behavior, disease prevention behavior, birth control behavior, and behavior leading to consumption (Ajzen, 1991). Examples of the use of a question-based tool for the theory of planned behavior would be whether to wear a seat belt, whether to examine oneself for diseases, or whether to use condoms during sex (Ajzen, 1991).

Issues with the Application of the TRA. Mykytyn and Harrison (1993) based their investigation on the TRA for purposes of competitive advantage in organizations leveraging computing systems and computing products. They applied the TRA to market studies to determine if the relationship of attitude and behavior would result in a measure of intention and a predictor of action. They found (and confirmed) that a positive relationship existed between behavior, intention, attitude, and subject norms (environmental influences on behavior) as indicated by Ajzen (1991). They added that "salient consequences" (p. 4) and the corresponding evaluations of those consequences were the motivating actions toward the actual adoption of a course of action (or the acceptance of a process). Mykytyn and Harrison (1993) confirmed Ajzen and Fishbein's TRA and noted its wide acceptance and capability to stand up to a positivistic structure through an equation-based representation.

Davis, Bagozzi, and Warshaw (1989) had earlier completed a supportive meta-analysis on user acceptance of computing technologies and confirmed the theories of Ajzen & Fishbein's original work. They also based their conclusion on an exhaustive study by Sheppard, Hartwick, and Warshaw (1988) that consolidated over 100 various studies that successfully applied the TRA to various practices, processes, and adoptions.

Considering that adopters' or decision-makers' intentions, attitudes, and subjective norms associated with a specific adoption or decisive behavior, can be predicted, Mykytyn and Harrison (1993) propose easier, more effective, and more efficient ways to change or alter that behavior in order to produce desired results. They conclude that Ajzen and Fishbein not only provided the requisite theoretical basis regarding predictive behavior, but also provided six criteria to form a solid measurement methodology. These criteria included: (a) Define the focused behavior with regards to actions, subject, process, or time, (b) Identify consequences of the focused behavior and any social entities capable of influencing it, (c) Make a choice on the most salient consequences and influential entities within the focus, (d) Measure behavior beliefs, normative, and surrounding beliefs, and the pressures to get others to comply, (e) Measure intention, attitudes, and the subjective norm (based on behavior of interest), and (f) Integrate and measure all of the above into a single questionnaire for exploring the behavioral question (Mykytyn & Harrison, 1993).

The TRA is not without those that question its validity. Budd (1987) indicated that some people might be aware of the theory's assumptions and skew their response toward a predetermined end, altering an honest result. Fazio, Lenn, and Effrein (1984) argued that some people might not have formed an intention or attitude about a particular subject, but due to being interviewed or by taking a survey, they may acquire an attitude formation rather than responding with an assessment. Mykytyn and Harrison (1993) countered by indicating that the subject matter serious enough to be surveyed, such as important decisions in an IT area (e.g. what ERP system should be purchased), do not lend themselves to quick judgments or hasty responses, and therefore the criticisms are only minor considerations.

Sheppard, Hartwick, and Warshaw (1988) noted that there are limitations to the TRA and whereas the TRA has been successfully applied in many cases for a prediction of behavior based on intention and attitude, there are cases where the model fails to provide the necessary framework for correct predictions. Sheppard et al., in the meta-analysis of the TRA, demonstrated that that the model fails when the subject behavior is: (a) Not under the person's control, (b) The situations involves multiple possibilities or choices, and (c) The person's intentions are analyzed when that intention has not been formed completely or confidently.

Ajzen and Fishbein (1980) specifically noted the model's limitations, particularly when goal intentions were used instead of behavioral intentions. The TRA was developed to address behavioral intentions such as taking a pill or applying for a loan. It was not meant to address established outcomes or results such as being relieved of a headache or having a debt. Essentially Sheppard et al. agree with Ajzen and Fishbein on the basic limitations of the model, that is, actions that are partially or totally out of the person's voluntary control fall outside the parameters of the model. Sheppard et al. (1988) points out that Fishbein and Ajzen initially indicated that few conditions of behavior were outside a person's voluntary control (1975). Ajzen (1985) later modified this position while noting a person's ability to project him or herself into the role or behavior in question. This extended the model to include the theory of planned behavior. It is significant to understand that, in the use of the model, responses to intentions are different from responses to estimations. Conclusively there are situations where one intends to do something versus situations where one actually expects to do something.

Although a subtle difference, Sheppard et al. (1988) found that even a slight shift in intention could alter the predictability of the outcome and the validity of the model.

Practitioner Experience with the TRA. The warnings of Sheppard et al. (1988) are enough to warrant a review of various applications, in vitro, of the TRA and to what extent possible validity issues or failures in predictability were experienced. Singh, Leong, Tan, and Wong (1995) employed the TRA to measure voting behavior and model an empirical test for predicting that behavior. The authors used belief and importance components to ascertain the intention of the respondents to vote along a party line, choose affiliation to a party, predict what candidate a respondent might vote for, and motivation to be swayed by media for or against a candidate. The co-relational and regression analysis used in their study indicated that the predictive power of the TRA model was verified, although they suggested continued testing in certain empirical measurements (Singh et al., 1995). From a review of the literature search, it was evident the authors were aware of the considerations of Sheppard, Hartwick and Warshaw, and even referred to them in various areas. The researchers did not recognize the brittle nature between goal behavior and intentional behavior or the pending results that might have been produced by the behavior. Their methodology did not appear to be affected by the discrete differences in goal behavior and intention-oriented results (Singh et al., 1995).

Randall (1989) demonstrated another successful application of the TRA model in a research study that applied the model in an attempt to explain and predict a situation of unethical conduct. Randall's conclusion indicated that the TRA model was able to provide a framework that could illuminate the underlying structures of unethical behavior, and could provide a predictive understanding of its various occurrences.

Randall agreed with Singh et al. (1995), in questioning the empirical nature of the model. At the time of their writing, there were few empirical tests that supported the use of the model as a predictive tool (Randall, 1989).

Randall's timeframe of research was somewhat early in terms of the model's acceptance and the amount of a body of knowledge building with results that were both qualitative and quantitative. It was found that several studies were known to Randall at the time of her research that lent positive support for the model's validity and possible empirical value. Sejwacz, Ajzen, and Fishbein (1980) applied the model empirically to predict the success of choosing a weight loss program. Jaccard and Davidson (1972) applied the model to the use of birth control in a study attempting to predict the possible use of various methods of family planning. Glassman and Fitzhenry (1976) used the model to develop one of the first marketing and brand choice applications of the TRA model. Once again, they confirmed the predictive value of the model but with less empirical measures than causation and logical result measures.

Thomas, Bull, and Clark (1978) used the model successfully to predict the use of public transportation. Hom, Katerberg, and Hulin (1979) used the model successfully to predict re-enlistments in various military services. Pomazal and Jaccard (1976) used the model successfully to predict the participation levels and pre-disposition for blood donations. These and other researchers found that the TRA model was successful as a predictive tool when applied to a particular intentional behavior. Although most were quantitative in their approach, the point that Singh et al. (1995) indicates concerning a lack of empirical evidence capability within the model can be observed. This may be a perspective of time as Singh et al. published 15 years after the initial body of studies that lent credence to the TRA model.

Extending the TRA Model to Applications in the TpB Model. The Theory of Reasoned Action was developed from models involved in the study of psychological expectancy values (Ajzen, 1987). The TRA was an attempt to estimate the differences between behavior and attitude with behavior considered voluntary. It was found that behaviors were sometimes involuntary and, in some cases, not under the control of the subject. The Theory of Planned Behavior introduced the addition of how subjects perceived their ability to control their environment and/or the outcome of the mixture of intention and attitude. The ability to control behavior became an added element to the TRA, creating the Theory of Planned Behavior and the ability to predict behavior, especially when it is deliberate and consciously planned (Manstead & Parker, 1995). Predicting behavior is fundamental to the theoretical framework of this study. As will be seen in the next section, The Technology Acceptance Model is a framework built upon the ability to predict use or adoption of a process or action by surveying the attitude, intention, and finally the applied behavior of the subject.

The Technology Acceptance Model

Sharda, Barr, and McDonnell (1988), noted that information technology had the capability of enabling business performance, creating growth, and substantially improving overall production. As Bowen (1986) points out, these types of improvements in business capability are retarded due to end users' reluctance to accept new technologies or use new systems.

Information systems research has struggled with measuring user acceptance of technology. Many individual studies have focused on certain variables and traits, but have failed to produce an overall set of metrics that can accurately correlate to system, practice, or process use (DeSanctis, 1983). This resulted in the need for an industrially recognized and theoretical measurement for the adoption of technology practices (Davis, 1989).

Beyond the theoretical aspects of an improved metric for technology adoption and use, a more accurate metric for predicting the use of a process, practice, or particular technology has significant practical value (Davis, 1989). Information technology professionals as well as manufacturers and vendors stand to gain from empirical assessments of specific technologies. In the past, these measures were simply not available or were provided from user opinion surveys that were highly subjective (Klein & Beck, 1987). With the use of subject measurements, questions of accuracy and repeatable value could constantly be raised (Shneiderman, 1987). Technology managers who base decisions upon these types of un-validated and non-empirical metrics are at risk of making misinformed decisions concerning the usefulness of a new system or process (Davis, 1985, 1989).

A Basic Understanding of The Technology Acceptance Model

The goal of The Technology Acceptance Model is to produce better measures for predicting the use of technology systems and processes (Davis, 1985, 1989). Davis first proposed the use of two theoretical constructs to predict technology adoption in his dissertation on how to test new end user systems. His study focused on what caused people to adopt or ignore various technology processes or systems. He found that two variables were important to this metric. People will accept or adopt a process or system if they believe it will assist them in their job or will improve their chances of achieving a goal. Davis referred to this as *perceived usefulness* (1989). Even if users believe that a process will be beneficial, they must also believe that it will be free from difficulty or effort and easy to use. Davis referred to this as *perceived ease of use* (1989).

Davis constructed a survey that collected data in three clusters for each of the two variables. For perceived usefulness, the criteria of job effectiveness, time savings/productivity, and relative importance to the job were measured in a series of questions. For perceived ease of use, the criteria of physical effort, mental effort, and the ease to which the system or process could be learned was measured in another series of questions (Davis, 1989). Davis found that the scales resulting from his study produced excellent psychometric results and validity was heavily supported by the multi-trait method. Perceived ease of use and perceived usefulness were highly correlated to indications of strong system acceptance and predictive use (Davis, 1989). With Davis' initial results in 1985 and refined results in 1989, there was a strong call for additional research and measures within various areas of information technology (Anderson & Olsen, 1985; Gould & Lewis, 1985; Johansen & Baker, 1984).

Additional Literature and Applications of The Technology Acceptance Model

Like the TRA and the TpB, the TAM was constructed to measure and predict behavior in the form of acceptance and use of particular processes and/or system (Rawstorne, Jayasuriya, & Caputi, 2000). These models not only describe intention and behavior but also identify the motivators behind intention and behavior especially when the purpose is for prediction (Sutton, 1998). It is important to consider issues that might arise in general measurements of behavior and intention.

Understanding Problems Associated with General Measures of Behavior and Intention

Rawstorne et al. (2000) indicate that there are two significant problems when attempting to measure intention and behavior at the same time. The first issue explains that when subjects complete surveys, which measure intention and behavior together, there are forces that drive a strong correlation between these two elements. People attempt to avoid psychological discomfort that comes when there is a discrepancy between what a person intends to do versus what he or she actually does (Bem, 1967).

The second problem involved with the measurement of intention and behavior together is that it may not be an accurate test of the model's ability to predict future behavior. Instead, the mix of measured intention and behavior will tend only to highlight present behavior (Rawstorne et al., 2000). Ajzen and Fishbein (1980) suggest that the time interval between measures of intention and measures of behavior are somewhat measures of greater accuracy of future behavior and provide predictions that are more astute. This may be true as unplanned factors or events may interrupt the intention-behavior relationship during that interval, causing added inaccuracies and error in the measurements. Karahanna (1993) suggests that there are too many dependent variables to determine what valid relationships exist and to what extent error enters the measurement between intention and behavior. Additional research, particularly longitudinal research, appears to be necessary in considering general measures of intention and behavior in models of prediction of future action.

Comparing the Theory of Planned Behavior and The Technology Acceptance Model

Rawstorne et al. (2000) set out to compare the theory of planned behavior and the technology acceptance model through a series of three different behaviors. Their study was designed to determine if behaviors could be predicted within each model, as well as what amount of accuracy could be measured when both models were compared. The study found that the TpB was only able to predict accurately one behavior of the three. A coefficient of planned behavior dominated the relationship between intention and behavior. Perceived usefulness was not mediated in the relationship between perceived ease of use and intention and there were no direct effects on the variables for behavior by the mediating effects of intention (Rawstorne et al.

This study found that the TAM was accurately able to predict two of the three behaviors with perceived usefulness partially mediating the interchange between intention and perceived ease of use. The failure to predict one of the three behaviors may have been caused by the formation of intention. Davis (1985) indicated that in some occasions, the intention to use a process or technology may require an amount of time in which to form an intention, since some or even many systems require time to learn them and to form an intention to use (or not use) once a workable understanding is achieved. If this is applied to the Rawstorne et al. (2000) study, there is significant evidence that the TAM surpassed the TpB in the prediction of technology processes or systems use.

A Review of The Technology Acceptance Model's Acceptance

Burton-Jones and Hubona (2005) indicates that The Technology Acceptance Model has emerged as one of the more popular and useful theories in predicting the usage of technology processes and systems. In this 2005 study, the authors found that individual differences were crucial in considering further predictability and accuracy of the TAM.

Zmud (1979) agreed with this assertion and demonstrated that the TAM's belief framework only partially mediates the effects of individual differences and heightens the results of usage when a consideration of external variables is made. In a review of contemporary TAM usage, Venkatesh and Davis (2000) found that the TAM was 40% more accurate in predicting usage intentions and about 30% more accurate in predicting usage behavior when external variables were considered (Meister & Compeau, 2002). Theses authors noted that this is higher than other current models, but admit that a richer experience is still needed. Critics of these percentages indicate that accuracy of the model is still marginal and more research is required before the model has substantive empirical value (Legris, Ingham, & Collerette, 2003; Plouffe, Hulland, & Vandenbosch, 2001).

To facilitate a greater understanding of what additional characteristics would increase the accuracy of the TAM, Burton-Jones and Hubona (2005), selected three individual criteria: (a) Educational level, (b) Seniority, and (c) Age. Agarwal and Prasad (1999) predicted that these same criteria would directly affect the perceived usefulness and perceived ease of use. Burton-Jones and Hubona (2005) expected only a partial mediating effect on predictive usage. Their results indicated that the above additional criteria did directly affect usage in addition to the affects of perceived usefulness and perceived ease of use.

Burton-Jones and Hubona's (2005) final conclusions indicated that the TAM, albeit a widely used and respected tool for predicting technology adoption and usage, was lacking in accuracy due to missing criteria and specific external variables that directly affect usage measures. The authors proposed that researchers and practitioners use more robust acceptance tools to improve usage prediction. They also proposed that users' individual differences must be taken into account in whatever model that is applied; and that for more accurate results organizational, social and belief-factors must be included.

Over 200 studies have involved the use of The Technology Acceptance Model as the theoretical framework for the study (Yousafzai, Foxall, & Pallister, 2007). What many of these studies have in common are the various positions and adaptations their authors have found in the use of the TAM. In some cases, the TAM is a basis on which to highlight their focused study or subject of investigation (Yousafzai et al., 2007). In other cases, they use the TAM as a starting point in which to expand or extend its theoretical elements with new, more relevant elements (Goodhue, 2007). Yet in others, the TAM is used as a fulcrum of theoretical debate as to the applicability, validity, or elemental reliability of its empirical and psychological components altogether (Benbasat & Barki, 2007).

Examples of The Technology Acceptance Model Applied

McCloskey (2004) found that the elemental aspects of the TAM, perceived ease of use and perceived usefulness, were integral components of electronic commerce. The two components provided a conduit for determining predictable usage. Usefulness was demonstrated through the predictable engagement of electronic commerce media. Ease of use was demonstrated through the measurement of whether a consumer would engage in online commerce because it was easy to use. McCloskey extended her study in 2006 to include the value of trust by consumers when making electronic commerce decisions. Trust was found to extend the impact of ease of use and usefulness in electronic commerce.

Almutairi (2007) applied the TAM to attempt to predict the acceptance of information services within the Kingdom of Kuwait's governmental organizations. The study found that a relationship existed between ease of use and usefulness. It found there were too many external affects of culture, Arabic organizational structure, and types of information system usage that did not support the use of the TAM. A closer review of this study indicates that the measures of data applied to the TAM were not intention oriented but rather goal oriented. Davis (1985, 1989) and Venkatesh & Davis (2000) noted that the applicability of the TAM was questionable in circumstances where respondents had already begun to implement a process or practice. Their intentions had given way to goal orientation, those goals being to see the practice or process implemented. The TAM does not apply well in these circumstances because the perception and the motivating attitude are not constructed to predict, but rather to illuminate an already existing condition (Venkatesh & Davis, 2000).

Lee, Fiore, and Kim (2006) were able to use the TAM to explain the effects of image interactive technologies on attitude and behavioral intention toward online retailers. Like McCloskey, they added or extended the TAM to include perceived enjoyment. Klopping and McKinney (2004) also applied the TAM to online commerce. What they found was that ease of use was directly linked to the intention to shop online and make actual purchases. There appeared to be no direct link to usefulness. Their research indicated that another model, the Technology Task Fit (TTF) was a valuable addition to the TAM when analyzing and predicting online shopping actions.

Experience was shown to provide an increased valuation in perceived usefulness when applied to predictability and the intentions regarding application development outsourcing (Benamati & Rajkumar, 2002). In the process of the research already conducted there is evidence that indicates that certain relationships, as well as the external environment, were additions to perceived usefulness and perceived ease of use in determining the use of outsourcing. The external environment including cultural affectations were found to influence the applicability of the TAM in Veiga, Floyd, and Dechant's 2001 study of culture on IT acceptance and engagement. Like Benamati and Rajkumar, Veiga et al. (2002) noted that not only external influences were relevant and additional to the concept of perceived ease of use and usefulness, but that specific cultural and cross-cultural aspects must be taken into account when considering acceptance and predictability of use. There is a strong relationship between the external influences of culture on perceived ease of use and usefulness (possibly more relevance issues). Scope is a natural coefficient in any discussion that includes a study area meeting cross-cultural parameters.

Savitskie, Royne, Persinger, Grunhagen, and Witte (2007) found similarly that international settings and cultural differences caused a lack of validation and perceived usefulness when the TAM was applied. Their study is the first to note the phenomenon of technology use and/or the engagement of computing practices or processes using the TAM as a possible theoretical construct (Stylianou & Jackson, 2007). Other studies, including Veiga et al. (2001), have solidly documented the effect of culture and cross-cultural conditions on the TAM's ability to predict usage. Both of the studies mentioned above do not treat intention clearly nor do they completely define how international pressures could affect perceived ease of use and perceived usefulness. This is the confusion between intention and goal orientation, or what can be predicted versus an explanation of what has already happened. In the case of the latter, it is a given that the TAM is clearly inaccurate in determining predictability for something that already exists.

Criticisms of the TAM appear to gravitate toward an observed misuse of the TAM variables and the incorrect outcomes. There are numerous demonstrations of TAM applicability to various technologies as well as its ability to work with other theories of predictive analysis. Gong, Xu, & Yu (2004) noted that the TAM clearly provided a solid framework for measuring resistance to IT in an educational setting. Their study also engaged the help of the Social Cognitive Theory (SCT) that added additional considerations of how people develop knowledge and retain it. This, together with the perceived ease of use and usefulness of IT computing capability allowed the study to determine the influences of teacher's technology acceptance and follow-on behavior. The TAM was successfully applied to various medical and health-care situations. Hu, Chau, Sheng, & Tam (1999), applied the TAM to a study of physicians' acceptance of technology that empowered telemedicine in health-care. Perceived ease of use was not necessarily found to be a strong indicator of attitude or intention. Perceived usefulness was a strong indicator of both attitude and intention.

The TAM's success in predicting behaviors from attitudes and intentions is well documented in business and technology applications. Analysis of intranet use within organizations was successfully aided by the TAM (Horton, Buck, Waterson, & Clegg, 2001). Analysis of the use of specific financial services within a retail environment was successfully supported by the TAM (McKechnie, Winklhofer, & Ennew, 2006). The acceptability and use of computer-based training was successfully analyzed through the TAM framework (Schneberger, Amoroso, & Durfee, 2007). The TAM was also found to be a useful product management tool in the overall research of mobile commerce (Snowden, Spafford, Michaelides, & Hopkins, 2006), and the ability to trust mobile or electronic commerce from a users usage perspective (Dasgupta, Granger, & McGarry, 2002). The use of the TAM for business and technology acceptance challenges has a strong record of successful application. Many researchers feel that the TAM requires modifications to its basic theoretical constructs. Even though the TAM has been paired with various other theories for increased and more robust meta-analysis, some researchers feel that the TAM must be altered to make way for new revelations in predictive analysis.

Modifying The Technology Acceptance Model

Chau (1996) indicated that perceived usefulness was a more complex component than originally applied by Davis. He believed that usefulness had near-term and long-term aspects that affected the predictive results substantially and called for further research into a modified model of the TAM. Davis revised the TAM in 1989, with it later being labeled TAM 2 (Venkatesh & Davis, 2000). Szajna (1996) reviewed the changes and felt that even modifications to the TAM by the TAM creator may not have been needed. Szajna did concede that the introduction of experience into the TAM model did enhance its prediction capability. As indicated above, the notion of acceptance from intentions and behaviors predicted from attitudes is psychologically based. Schwarz and Chin (2007) concluded that when considering alternative IT technologies, it was important to have other psychological methods to determine alternative IT acceptance. Schwarz and Chin were able to identify six additional psychological methods of acceptance. Although they do not attempt or call for a modification of the TAM, they do suggest that other methods either singularly or in concert with the TAM are more effective in determining psychological acceptance.

Rigopoulos and Askounis (2007) developed a modified TAM to apply to IT based banking services. Their findings indicated that when the original TAM is specifically modified to the study, its relevance and accuracy increase. Yu, Liu, and Yao (2003) modified the TAM extensively by adding additional variables to include perceived targets differences, perceived complexity, and perceived social influences. They noted that the acceptance of the technology in question was more accurately defined using additional variable traits. Similarly, McCoy, Galletta, and King (2007) found that the original TAM was relevant for U.S. application, but did not pass rigorous testing in a multi-cultural surrounding. They suggested that aspects such as low uncertainty avoidance, high masculinity, and high collectivism counter the effects of perceived ease of use and usefulness. They did not attempt a modified or re-engineered TAM, but rather suggested that when applying the TAM in studies across more than 20 countries other than the U.S., caution should be applied (McCoy et al., 2007).

One of the most significant meta-analysis of the TAM is the two-part study by Yousafzai, Foxall, and Pallister (2007). Their attempt at TAM modification was through a qualitative gap analysis of over 145 studies on the TAM. Their aim was not only a gap analysis but also a TAM unification proposal. In part two of their study, they applied a quantitative analysis of 569 study results from over 95 TAM research projects. The empirical investigation focused on self-reporting usage and its effect on behavior. Their study raised several questions about the possible exclusion of attitude from the TAM when technology usage behavior is actually mandatory instead of voluntary.

Additionally, they asked if self-reporting measurements on intentions was valuable instead of employing actual usage measures (Yousafzai et al., 2007). The outcome for them is essential to a unified, and modified, perception of the TAM.

Venkatesh and Davis (2000) using data from longitudinal research addressed the issues of mandatory usage versus voluntary usage through a model that was modified to include social influences such as subjective norm, voluntariness, job relevance, and output quality. This study found that when these influences were added to perceived usefulness and perceived ease of use and measured during periodic milestones of activity, there was improved adoption behavior over the TAM model without the added influences. It is clear that previous additions and modifications to the TAM supported or at least motivated that particular research direction. The use of longitudinal measures was the first to indicate that there was empirical evidence that supported the use of actually modifying the TAM to acquire more accurate behavior/adoption predictions. This conclusion, and the subsequent conclusions of other researchers support as well as lead to the conclusions of Venkatesh and Davis (2000). They support the addition of quality-metric influences of efficiency and effectiveness on perceived ease of use and usefulness (see Figure 3).

Criticisms of The Technology Acceptance Model

A literature review of the TAM must include contributions of the critics, comparisons to those that find relevance in the model, and those that find modifications to improve the model's accuracy, application, and relevance. Benbasat and Barki (2007) presented a qualitative critique on the TAM's basic tenets as well as studies and researchers that have attempted to modify the TAM to adapt it to changing IT environments. They indicated that the TAM was an illusion of progress and diverted attention away from important research topics. They found the work by other researchers to change or modify the TAM to be a cause of confusion as to the actual version of the TAM that should be employed (Benbasat & Barki, 2007). Goodhue (2007) echoes their concerns about the TAM's obfuscation role in adoption research. His review indicated that the TAM assumes that more usage will result in a more accurate behavioral prediction, and that increased usage will result ultimately in higher performance and higher adoption (Goodhue, 2007).

There is compelling logic to Benbasat and Barki's (2007) as well as Goodhue's (2007) arguments, but there is little empirical evidence that there is chaos or widespread obfuscation, as they want one to believe. Benbasat and Barki failed to unhinge the foundations of psychological interaction embedded in the Theory of Reasoned Action that is basic to the TAM. They also failed to counter the findings of progressive behavioral predictions contributed by the Theory of Planned Behavior. Essentially, they employed a qualitative argument to uphold an empirical critical position. They also assumed that the variations on the TAM are chaotic and/or are at the center of confusion within the environment. This is not supported by other researchers and is not even moderately mentioned in the literature (Zhang, Prybutok, & Koh, 2006). Their assumption that variation is somehow chaotic challenges the dynamism of information technology and the constant movements and changes general information technology sciences have and do undergo. The scope of any research could vary with the subject and it could be said that a requirement of an essential aspect of a valid theory is that it has the growth capability to meet the dynamic movement of the environment it is studying.

Goodhue's (2007) critique of the TAM indicates it assumes more usage is better. He states that the higher the usage the higher the performance of the system, and the higher the systems' general acceptability. This causation is not empirically fed through the psychological process-bound movement of intention to attitude to behavior, but rather an unintentional outcome of two independent variables coming together in a random result. Actually usage is not necessarily a dependant variable. Predictive use (usefulness) may have influences outside of perceived ease of use depending on the environment (Yousafzai et al., 2007). The previous highlighted studies demonstrated this. Applying influences that further focus perceived ease of use and perceived usefulness do not attempt to engage quantity usage, but rather create deliberate shifts in focus that are consummate with its surroundings (known as the subjective norm).

Other criticisms of the TAM come from the post-positivistic movement. Silva (2007) takes this perspective and reviews the criteria of the TAM using an empirical approach. Using several (positivistic) theoretical models, Silva attempts to test the connection between intention and action and to apply the criteria of normal science to the TAM. Lastly, Silva notes that the TAM, as a science, is advancing. A science must advance as a part of its validation criteria and not decline. Similar to Benbasat and Barki, Silva points out that the apparent growth is a combination of confusion and knowledge accumulation, which may not be scientific at all. This position seems to state the criteria of science then proceeds to argue why fulfilling the criteria is not scientific. It is apparent from the literature that the TAM's ability to draw variations and specific subject influences causes immediate questions for quantitative researchers. Ma and Liu (2004) take a similar position. They employ statistical significance, direction, and magnitude as critical criteria. Their study, unlike Silva, measured the relationships between usefulness and acceptance as well as usefulness and ease of use. They did find that there is only a weak connection between ease of use and acceptance (Ma & Liu, 2004). Silva also questioned this connection (2007) but found it an emotional (non-qualitative) connection and not an empirically sound one.

A Unified Technology Acceptance Model

The review of the literature so far yielded several aspects of the TAM, one of which indicates that a unification of variables is necessary for some form of application or practitioner consistency. Although most authors are unclear as to why consistency is important (other than for itself), it nevertheless is a criterion in evaluating the various styles of TAM employment. The call for unification is found specifically in many of the modified TAM proposals. One of the most exhaustive works in unifying user acceptance of technology is a model developed by Venkatesh, Morris, Davis, & Davis (2003). This model combined eight acceptance determinant theories and applied four criteria across them to create a unified set of theories. These criteria include: (a) A review of current literature and identification of support for the theory's relevance and general acceptance in the industry, (b) An empirical comparison of each theory along with supporting material, (c) The formulation of a model that incorporates the successful elements of each of the various models, and (d) Validation of the unified model through empirical means.

The combined result of this work was labeled the Unified Theory of Acceptance and Use of Technology (UTAUT). The purpose of the UTAUT is to give IT managers that need to evaluate the implementation of new technologies an understanding of the drivers of acceptance in order to prepare environments and user groups for the adoption of the new technologies (Venkatesh et al., 2003). This theory is based on the concept that in order for technologies to increase productivity they must be used and accepted by a community of users within the organization (Hu et al., 1999). Researchers are faced with many models that meet various criteria, but which lack other requirements. The UTAUT offers the researcher the ability to meet a majority if not all of the generally accepted relevant criteria in one unified model of measuring and predicting acceptance of a technology or process (see Figure 2).

The UTAUT was the result of longitudinal studies conducted at four locations using individuals introduced to new technologies at their work site. Great care was taken to create a homogeneous environment between the locations, the organizations, the technologies employed and the background of the each individual involved (Venkatesh et al., 2003). A series of surveys were conducted in both voluntary and mandatory settings at various points of the technology implementation. Aspects of the eight various acceptance models were represented in the survey questions. The result indicated that the UTAUT provided a filtered view of what determines intention and behavior and how these evolve over a period. Moderating influences seen in many of the individual models entered into the values of the unified model. Aspects such as age and gender appeared to present unexamined interactions and results (Venkatesh & Morris, 2000). Other determinant factors only show that a complex series of moderating influences continue to operate around an increasingly dynamic picture of what individual perceptions of intentions and behaviors toward acceptance of technology might have. Many of the models, both individually and within this unified construct, predict intention and usage. Specific confirmation of usage or even motivations of individual acceptance remain weak or unsubstantiated. Technology creators may continue to struggle under an incomplete picture of what is acceptable and to what level of acceptance certain technologies may be judged (Venkatesh & Davis, 2000).

In comparison to other, individual models, the UTAUT has not received a significant amount of acceptance or use in the field or within the existing literature. Despite the call for a more unified technology acceptance model, the result of a hyper-consolidated model was highly complex, and for most practitioners, unusable. With over 64 cross-validation variables, the UTAUT is less likely to be used by a technology manager at an organization attempting to integrate new technology processes, tools, or practices. It is a powerful research tool with strategic value in plotting expectancy, usage, and behavior through predictive analysis on a variety of internal and external influences. Venkatesh et al. (2003), admits that with the UTAUT the practical limit to explaining individual technology acceptance and usage decisions in organizations is at its limit. If actual employment of the model is an indication of industry acceptance, the lack of substantial literature showing the use or employment of the UTAUT speaks for itself.

The Paired Programming Practice

Paired programming is one of the most significant practices in the Extreme Programming methodology because of its proposed ability to increase product creativity, increase customer satisfaction through improved communication, and increase technical software coding quality (Beck, 1999). It has also become one of the most popular topics of research within the Agile family of methodologies (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003). Nosek (1998) reported the first scientific study of paired programming. There are indications from various literature sources that studies prior to this were conducted and their results popularized in several publications, but without empirical evidence or peer-review. Studies in paired programming escalated in 2000 and remained actively studied producing contributions up to the present (see Figure 6).

Understanding Paired Programming

Hulkko & Abrahamsson (2005) organized studies in paired programming into intuitive categories. These include case studies, experiments, surveys and experience reports. Another area of recent interest is paired programming in (computer) education. For the purpose of this study, the above categories of Hulkko and Abrahamsson will frame the following literature review. The research problem of this study highlights the issues of paired programming as compared to single programming results. Supporters have claimed benefits of less scheduling time and reduced communication logistics along with higher productivity and shorter release times (Williams, 2000). Yet there are counter reports on paired programming with less than optimum revelations. The following review will reveal the various camps that have appeared and the evidence associated with each of their positions.

Case studies on Paired Programming

Case studies in paired programming introduce an informative aspect of the practice that discerns its performance in actual software project settings, as well as demonstrates the impact it has on software quality (Muller, 2003). Case study literature suggests that the major subject of paired programming studies focus around maintainability and reliability (Muller, 2003; Williams, Kessler, Cunningham, and Jeffries, 2000; Wood & Kleb, 2003). Gallis, Arisholm, & Dybå (2003) indicated that beyond the research of how paired programming is beneficial, it is important to evaluate how paired programming is easy to use and useful. This is accomplished in case studies where paired programming has been observed in use and where results are readily apparent. Williams and Kessler (2003) note that the benefit of paired programming is best observed when applied to particular tasks or activities. Additionally, Hulkko and Abrahamsson (2005) state that various case studies have observed the suitability of paired programming for only specific tasks under certain conditions. Hulkko and Abrahamsson point out that whereas these observations indicate a positive direction, the general nature, and relative importance to the field continues to indicate and produce questionable metrics.

Testing and Paired Programming Case Studies

The Extreme Programming methodology and other Agile-type methodologies promote testing-first practices. Talby, Hazzan, Dubinsky, and Keren (2006), shifted their study from paired programming being a programmer-oriented focus, to an architectural element that makes up the internal construction of a large project. In a case study of a large scale, critical software project for the Israeli Air Force, Talby et al. (2006), found paired programming to be a modular component to the planning and execution of large-scale projects. Turk, France, and Rumpe (2005) agree, but from the position that Agile based practices such as paired programming are not just for anecdotal applications or short-term experiences. Paired programming supports methodical test-driven programming and design. They raise the subject that although applicable, paired programming and other Agile methodologies that support testing-first practices must be applied differently depending on the scope and critical nature of the project. They also indicate that other Extreme Programming practices should be given more credence in large as well as small project planning.

Crispin (2006) states that test-driven programming or design is essentially not a technique for testing but rather an engineering design technique. The practice is integrated into the pair activities so that for each function point the pair creates, they first must write a function or unit test to complement it. Writing unit tests prior to coding a function requires the pair to understand the components of that function as well as its outcome as it relates to the overall program. The unit test is combined with other unit tests to provide a test-story prior to declaring that iteration's code executable (Mugridge, 2008).

Case Studies Considering the Role Components of a Programming Pair

In a pair of case studies to review the components of a programming pairs, Chong and Hurlbutt (2007) found that terminology was not created prior to the practice, but after it. The terms *driver* and *navigator* were not used in Beck's seminal work (2000) on paired programming practices, but the processes proposed by him relied on the programming pairs working as a team within a larger team. Williams and Kessler (2003) first use the terms to identify the driver and the navigator. The driver works at the keyboard writing the code or design. The navigator thinks strategically, observing the driver's code and thinking of the next several lines to encode.

Few studies have really delved into the inner workings between the navigator and the driver. To some extent, this would seem counter to Beck's original concept of what the pair was doing; opting for more creative freedom than structural role definitions. Chaparro, Yuksel, Romero, and Bryant (2005), indicated that the role of navigator and driver were difficult to define, at least in studies where student programmers were observed in paired programming settings. They did theorize that as programmers matured in their skills they would gravitate more toward structured roles. Bryant (2004) researched the working patterns of programmer pairs and found that student programmers tended to work outside of most defined roles such as navigator and driver. The pairs tended to switch somewhat chaotically between roles. Yet, when Bryant studied more mature or professional pairs, what she found was almost similar behavior with more order in their activities. This led her to conclude that programmers in general seem to be role independent, even though as a programmer matures it is possible for the roles to become seamless through experience and expertise (Bryant, 2005).

Williams and Kessler (2003) observed that variation in pair make-up between novice and experienced programmers yielded more effective and efficient results. Novices that were paired with novices became frustrated early and were ineffective. Experienced programmers paired with other experienced programmers became bored or worked independently despite pair assignment. Balancing experience within a pair seemed to yield the greater productivity and quality. Sfetsos, Stamelos, Angelis, and Deligiannis (2006), found that the productivity of a programming pair was related to the amount of communication between the pair, based on their personalities. Padberg and Müller (2004) observed a connection between a programmers comfort level and the resulting pair's productivity. They failed to actually define what comfort level actually meant or substantiate a metrics for what constitutes comfort within a pair.

Chong and Hurlbutt (2007) sum up the results of these various case studies on pair roles through the results on their own observations. They indicate that in practice the roles within a pair are less structured than the literature may indicate or wish. The natural pattern of interaction takes over and both involve themselves in the actual division of labor (Ho, Raha, Gehringer, & Williams, 2005). The pairs appeared to be the most effective when they took on both the navigator and driver roles, switching the actual control (keyboard) point as they progressed through the code. This indicates that these roles are less of a structured assignment or position, but more of a general explanation of the pair's role at any one point in time; and are naturally fluid (Chong & Hurlbutt, 2007).

Clegg, Waterson, and Axtell (1996) noted that programmer groups are actually intensive work-cells and themselves living organizations. They found the theory of the current period to be unprepared to address this type of practice. What was necessary was a detailed analysis that yielded cognitive explanations and organizational understanding. Their studies on knowledge-intensive work organizations lay a socio-psychological basis for collaborative software engineering (Clegg, 1994). Although there is no indication that Beck (1999) was aware of their research in his seminal work on Extreme Programming and specifically on the practice of pair programming, Clegg et al.'s (1996) initial theories seem to set the stage for the practice of collaborative programming. This is evident in Clegg et al.'s (1996) qualitative action research study with three specific cases. Each case involved a large software project and a team of programmers. Several different programming methodologies were used along with various structured tools. What they found was that most software development managers and software teams were eager to collaborate. In each case study, a socio-technical proposition was independently determined, that is, the adoption of more coordinated efforts, explicit team processes, and the use of "software development cells," an idea borrowed from manufacturing (Clegg et al., 1996, p. 247).

The result of Clegg et al.'s (1996) study was a fear that the software development world would move toward several negative trends if socio-technical elements were not considered. These trends include the growing use of the factory example as a process example for software engineering, the use of highly structured computer aided software engineering (CASE) tools for standardized work, and the fragmenting of productivity processes and knowledge bases because of outsourcing (Clegg et al., 1996). The first fear reduced the software engineering team to mechanic-like properties, disregarding the scientific aspects of the work (problem solving versus mechanical iterative actions). The second fear place a significant amount of trust and expectation on the tool rather than the user of the tool. This fragmented knowledge, expertise, and understanding for the use of expensive tools and highly structured processes. The third fear recognized that the knowledge and intellectual property were lost when outsourcing was utilized in sort of a plug-and-play method (Clegg et al., 1996). The case study of Clegg et al. (1996) provides a fertile basis on which to ground the need for collaborative software engineering in the form of paired programming. Whether the founders of the pair programming practice considered it, is not as important as the groundwork, which has provided an understanding of the need for such a practice in software engineering maturity.

Experiments in Paired Programming

Since 2000, there have been many pronouncements in favor of paired programming versus individual programming. Various experiments within the last 10 years provide empirical evidence that this is the case. There are also studies that seem opposed to these findings or at least moderate the benefits significantly. This section discusses the experimentation and empirical results related to the value of paired programming and the ongoing literary and research debate over the evidence.

The first formal studies of paired programming can be found in Flor and Hutchins' (1991) study on distributed cognition among software development teams. This is where they first introduced the concept of paired programming as a way to distribute task knowledge and divide task productivity. This work was qualitative in nature and it was not until 1998 that Flor followed up on their initial study with a research project to measure and promote side-by-side software development or paired programming (Flor, 1998). Flor returned to the subject of paired programming to emphasize the empirical evidence for the use of pairs in software development. He recognized that businesses are now under severe economic pressure to employ distributed software groups in various locations and/or even around the globe. Returning to his previous studies, he restates the elements of successful pairing and applies the limitations of distance collaboration to the situation (Flor, 2006). The results of his review reveal the need for visual, manual, and audio links between distributed locations, but essentially the tenets of pair programming continue to remain the same, with the same expectation of success (Carmel & Agarwal, 2001).

Despite Flor and Hutchin's (1991) initial analysis on pair programming, many theorists cite Nosek's (1998) initial empirical study of collaborative programming as the first true empirical work specifically on the practice. In his study, he hypothesized that programmers working in pairs would provide more and better code than programmers working individually, that pairs will take less time on a problem than an individual working alone, and that pairs will express higher confidence levels in their work (Nosek, 1998). Although Nosek's study was small and somewhat limited, it provided the first in a series of experiments and empirical studies that demonstrated pair programming benefits. Nosek also provided questions with his findings. These included questions that asked if two average programmers teamed up, would this provide enough productivity that would not be possible if attempted separately. He also asked if companies could use collaborative programming to shorten development and product delivery time. Finally, he asked if collaborative programming provided a competitive edge for companies that use it (Nosek, 1998). These and other similar questions are proposed throughout the literature. Many have attempted to answer these questions through experiments, surveys, and experience reports.

Visaggio (2005) attempted an empirical assessment to help address Nosek's questions. The results, not unlike previous findings, were clearly found in three areas. First, it appears that paired programming improved developer productivity as compared to individual programming. Second, paired programming supported significant knowledge transfer between the members of the pair, especially when both were of equal educational backgrounds. Lastly, attempting to distribute the pair over geographical distances seriously deteriorated productive results if collaboration and communication tools were not solidly in place (Canfora, Cimitile, DiLucca, & Visaggio, 2006: Visaggio, 2005).

The first experimental work on paired programming was conducted in an academic setting, the results of which were not published in peer-reviewed forums. The experience was communicated through various works. Almost immediately, researchers set out to replicate and renew the research to further ends. Williams (1999), who took part in some of the original experiments as a graduate student at the University of Utah, continued to propose more extensive testing on the use, practice, and effects of paired programming. She followed up the 1999 experiment experience with a series of experiments between the years of 2001 and 2002 at North Carolina State University. Empirical data was collected in several regular and longitudinal studies on the effects of paired programming on productivity and output (Williams, Wiebe, Yang, Ferzli, & Miller, 2002).

A review and summary of experiments to date was made by Mendes, Al-Fakhri, and Luxton-Reilly (2006). Although these experiments were clearly more structured and more populated, (over 100 students participated) than the original study, they still kept within the academic arena and produced questionable results for applications in business and commercial enterprises. Additionally, their findings were a mix of empirical metrics and qualitative measures based on opinion surveys of the students participating in the experiment. The results matched other similar studies in that students that employed paired programming successfully completed the course at a much higher rate than other students that followed individual programming methods (Braught, Eby, & Wahls, 2008). What remained to be accomplished from this and other studies was a follow-on investigation on how the successful students performed in subsequent programming courses when either working in pairs or working individually (Mendes et al., 2006).

Measuring the Benefits of Paired Programming. The focus of the research in this study is on understanding the attitudes and intentions of software development managers to measure the perceived ease of use and perceived usefulness of paired programming in order to predict the practice in the future. The reason for this study is the contradiction in measures for the use of paired programming over the last nine years. There is also a grass roots concern in many businesses that they are spending too much for software development when observing two people work at one workstation (Flor, 2006).

One of the proposed benefits of paired programming is a shorter time to completion or increased velocity. This is accomplished through improved problem solving activities of a pair of programmers rather than an individual programmer (Williams, 2000). There is also less administrative and communication overhead because the pair is already stitched together in a formal way. Williams, Kessler, Cunningham, and Jeffries, (2000) found that in a 2000 experiment on pair programmer productivity, pairs finished 40%-50% faster than did individual developers. Lui and Chan (2003) indicate that there is only a 5% timesaving gained by pair programming. Müller (2003) found that pairs could cut quality assurance and unit testing in half. Yet Nawrocki and Wojciechowski (2001) reported that there are no added gains by employing paired programming or the Extreme Programming methodology over individual programming and a waterfall software engineering process. They found the paired programming practice to be less productive than the practice of individual programming when employing other forms of Agile or Extreme Programming methods. Over time, they found that paired programming became more efficient than individual programming and to some small extent, the practice proved more productive or effective over time (Nawrocki and Wojciechowski, 2001). Humphrey, (1995) found that individual programming, when applied in a group setting, was linear, that is, increased incrementally and linearly as additional individuals were added. Nawrocki and

Wojciechowski (2001) found that paired programming efforts were not linear in relation to the increased population of programmers working on the project.

Other contradictions about greater or lesser coding efficiencies and effectiveness in paired programming exist throughout the literature. Gallis et al. (2003) indicated that one of the reasons for this issue is that there are no consistent and/or agreed upon frameworks for research for paired programming. Their research based on previous paired programming metrics, current research on collaborative programming, and various group research theories resulted in a framework that suggests a hierarchy foundation for analyzing paired programming research. Review of the literature does not indicate significant support for this foundation, but there is evidence to indicate that other contributing authors desire a more disciplined approach.

Costs in relation to greater efforts and the ensuing expenditures are other methods of measuring the benefits of paired programming. Williams and Kessler (2003) found that whereas certain paired programming efforts might increase overall project costs, it is worth the added expense in part due to the resulting software's improved quality. Measurable points of quality include the constant review of code lines by the navigator as the director types in the code. It also includes continued defect review and prevention with four eyes versus two scanning the code lines, and the pressure of the pair which imposes a natural conformance to standard work and the courage to refactor (rebuild or reconstruct code) when required (Beck, 1999). Wood and Kleb (2003) support Beck's position and indicate that through the paired interaction, coding standards are actually tried and reconfirmed in the course of use. Gallis et al. (2003) indicates that stricter use of coding standards will strengthen the software, provide for better readability by others, and increase the accuracy and completeness of knowledge transfer between developers.

Actual output and productivity of the pairs as well as the work effort is central within much of the literature and concern about paired programming by businesses (Hulkko & Abrahamsson, 2005). It is also substantive to the research question of this study and behind the assumption by many business managers that when using two programmers for the same task, the effort spent and costs expended are doubled. Williams (2001) notes the pairs expended only 15% more effort when doing a task than their individual programmer counterparts. Ciolkowski & Schlemmer (2002) found a 10% increase and Lui & Chan (2003) found a 21% increase. Additionally, Williams, Shukla, and Antón (2004), noted the productivity impact of new members as pairs to a delayed project and found that with paired programming, assimilation of the pair and increases in productivity were greater with the pairs than if the team members worked individually.

Jensen (2003) reported a 127% increase in productivity by using paired programming over individual programmers, measured over large and time-extensive software projects. Yet as already noted above with Nawrocki and Wojciechowski (2001), productivity was greater using individual programmers under the Extreme Programming method. They did find that after a certain period, the productivity of the pairs increased beyond the individual programmers. Williams et al., (2000) called the initial ramp-up of expended work necessary for a pair to be significantly productive (beyond the value of an individual programmer), *pair jelling*. This may also explain the phenomenon observed by Nawrocki and Wojciechowski (2001).

Additional evidence and metrics can be found on the quality results of paired programming through the findings of empirical studies. Shorter or reduced code length is a manner of measure that indicates the code is more maintainable and easier for others to read and manage (Cockburn & Williams, 2000; Wood & Kleb, 2003). Shorter or reduced code length is also a general indicator of superior design and architecture. Sometimes considered more efficient or *tighter*, short code length indicates a more quality engineering effort borne of expertise and experience (Ciolkowski & Schlemmer, 2002; Cockburn & Williams, 2001).

Associated with a quality consideration, but more specifically measured are defect rates in code lines along with higher rates of initial acceptance test or test case segments passed as a result of paired programming (Jensen, 2003; Tomayko, 2002; Williams, 2001). Overall quality ratings that combine a mixture of these metrics along with some subjective elements such as readability, ease of change, executable speed, and intuitive logic are combined to form a measure of quality that has a positive impact on the final code product as a result of applying paired programming (Nilsson, 2003; Williams, McDowell, Nagappan, Fernald, & Werner, 2003). All of these measures point to a positive direction for the use of paired programming. Yet there remains no specific, quantitative measure that fully indicates an unquestionable position. Part of the reason lies in the measures themselves. Some are general; some are undefined or too subjective. Others lack a standard methodology on which to rely (Chaffey, 1998). In some cases, the cost benefits have been used to indicate quality or acceptability or as a justification for further use of the practice (Hulkko & Abrahamsson, 2005). What can be concluded from the literature to this point is that paired programming appears to have benefits that at least are equal to or greater than the disadvantages. To what extent this is the case remains to be proven through more quantitative measures and more substantive theoretical frameworks in which to study ongoing results.

Paired Programming Studies

The Agile Methodology Movement proposes to create faster code that is more desirable, through an iterative process that produces working results through the constant interaction between the customer and the programming pair. Many times paired programming is termed *collaborative programming* or *iterative programming*, depicting its logical segmentation, yet independent executable capability. Nosek's (1998) study provided the first empirical evidence that paired programming was possibly more efficient and more effective than traditional single programming methods. An experiment at the University of Utah in 1999 further solidified the use of paired programming as a viable practice that contributed to the effectiveness and efficiency of software coding (Williams, 2000).

In 2003, Arisholm and Sjøberg replicated Nosek's research and the experiment of the original paired programming study. A total of 295 junior, intermediate, and senior professional Java consultants (99 individuals and 98 pairs) from 29 international consultancy companies in Norway, Sweden, and the UK were hired for one day to participate in a controlled experiment on paired programming. The subjects used professional Java tools to perform several change tasks on two alternative Java systems with different degrees of complexity. The results of this experiment did not support the hypothesis that paired programming in general reduces the time required to solve the tasks correctly or increases the proportion of correct solutions (efficiency). On the other hand, there was a significant 84 percent increase in effort measured to perform the tasks correctly (effectiveness). On the more complex system, the paired programmers had a 48 percent increase in the proportion of correct solutions but no significant differences in the time taken to solve the tasks correctly (effectiveness). For the simpler system, there was a 20 percent decrease in time taken (efficiency) but no significant differences in correctness (see Table 2) (Arisholm & Sjøberg, 2003).

Table 2. Comparison of Individual and Paired Programmer Effort Expended in a Generic Coding Problem

Programmer Category	Individual	Pair	Difference
All (average)	73	135	84 %
Juniors Only	82	172	111 %
Intermediate Only	89	128	43 %
Senior Only	62	114	83 %

Note: Adapted from "A controlled experiment with professionals to evaluate the effect of a delegated versus centralized control style on the maintainability of object-oriented software," by Arisholm, R. & Sjøberg, D. I. K. (2003), *Technical Report 2003-6, Simula Research Laboratory.* Copyright 2007 by the IEEE Computer Society.

It appears that the moderating effect of system complexity depends on the programmers' expertise in the subject areas. The observed benefit of paired programming for complex systems appears to be more useful when junior programmers are involved. The observed benefit of paired programming for simple systems appears to be more easy to use when performed by intermediate and senior programmers. Arisholm, Gallis, Dybå, and Sjøberg, (2007) experiment proposed that the future benefits of paired programming would exceed the results obtained in their experiment for larger, more complex tasks. It was also noted that benefits would increase if paired programmers were given the chance to work together over longer durations (Arisholm et al., 2007).

Perceived Ease of Use and Usefulness of Paired Programming

The experiment by Arisholm et al. (2007) indicated that there were variances between efficiency of paired programming and the effectiveness of its application. Added to these findings, there appears to be growing concerns by businesses about the cost of software programming and the general inability of software development groups to complete projects on time and within budget (The Standish Group, 2004). It is only natural to think that two programmers sitting at one workstation is inefficient despite the initial findings of Nosek's (1998) and the University of Utah studies that indicated paired programming was more efficient than single programming. The effectiveness of paired programming continues to come under scrutiny as cost-conscious business leaders and lean-driven product managers find it hard to believe that a pair is a more efficient arrangement than a series of individual programmers working in a focused manner (Nosek, 1998; Mendes, Al-Fakhri, & Luxton-Reilly, 2006; Williams, 2000).

Paired Programming Surveys

This segment of the literature review focuses on relevant survey studies in paired programming and the Extreme Programming development processes. Some surveys have been developed to uncover group perceptions of paired programming in use. Others have been executed to determine if paired programming is considered useful and might be used in the future (similar to the focus of this study). A preponderance of paired programming survey studies focused on the educational benefits of preparing new software development engineers. Some have targeted supporting functions to paired programming and their importance in the overall process of software development. Only a small body of knowledge was found within this realm of the study. Despite that, this is an important segment of study if a complete review of paired programming literature is desired.

Hanks (2006) surveyed students about the use of paired programming in their training and study as well as in their assignments. The survey indicated strong agreement on every sample taken supporting various aspects of paired programming. One of the aspects not anticipated in the survey was that women were shown to have a more positive attitude about paired programming as opposed to men. The statistical significance was relatively small, but the trend was clear. This trend reflected similar findings by Werner, Hanks and McDowell (2005) and Margolis and Fisher (2002) in a series of studies dedicated to researching the affect of paired programming on female users/programmers. Additionally it was found that students who were the most confident also found paired programming more beneficial overall. This finding also matched other studies (of a non-student nature) where programmers who were more confident in their software engineering abilities also were more positive about the use of paired programming (Thomas, Ratcliffe, & Robertson, 2003).

Experience Reports on Paired Programming

Experience reports are usually compilations of experiments, case studies, and surveys brought together to propose or promote a particular topic or process. Various experience reports exist in the body of knowledge for paired programming. The general indication of the literature indicates that since 1999, there has been an increasing empirical and structured approach to the study of Extreme Programming, particularly the practice of paired programming. The growing efforts have matured the science of the study in the last nine years. This segment represents samples of such research within the area of paired programming (Cockburn & Williams, 2000).

Erdogmus and Williams (2003) produced a research work that brought together various types of past research and surveys into their own study on the various economic aspects of collaborative programming (including paired programming) and personal software processes (individual programmer efforts). Their study yielded notable differences between the performance of paired programming and individual programming. The results of their experiments, past research, and surveys were used to determine the economic feasibility of paired programming (Williams et al., 2000).

Cockburn and Williams (2001) noted that many reject paired programming because they believe that software engineering costs will double with two programmers working on the same task. If the practice cannot demonstrate economic feasibility then most managers will not use it. They stated that businesses determine whether to adopt certain practices based on bottom-line outcomes. These outcomes are modeled using net present value (NPV) calculations and breakeven analysis (Levy, 1987). Erdogmus and Williams' (2003) findings demonstrated the economic value of paired programming as an alternative to individual programming. In both value models, they noted that paired programming produced superior economic results from other previous studies and engineering literature. In particular, they found that the value of paired programming was enhanced when development tasks were incrementally structured through frequent releases, a long-standing engineering economic measure (Boehm, 1981). They also found that paired programming value increased as defect rates dropped from 60% (at an individual programming par) to 20% indicating a significant connection between paired programming value and improvement in code quality (Erdogmus & Williams, 2003).

In an explanatory work on Extreme Programming and the use of paired programming practices, Lindstrøm and Jeffries (2004), take a values based approach to the comparison of present software development models. This report simply outlines the various values, principles, and practices of five popular models: Capability Maturity Model Integration (CMMI), Structured Analysis & Structured Design (SASD), Rational Unified Process (RUP), Agile, and Extreme Programming (XP). Their findings show by comparison that only Extreme Programming contains value based engineering structures, principle based coding measures, and practices based task work. Further analyzed, the paired programming practice is shown to provide better code and tests (practices), ability to communicate and spread knowledge (values), and improve skills and professional knowledge (principles) (Lindstrøm & Jeffries, 2004). Waguespack and Schiano (2004) reflected this same thinking in their experienced based work on component architecture. This is a different perception of software engineering, with a preference for component building and integration rather than a linear process of engineering development. They explore various component frameworks such as Active X, Corba, Microsoft DCOM, Microsoft .NET, and enterprise Java and J2EE. With these components, they apply three usability characteristics such as utility, capacity, and versatility. The results of their comparison was that the iterative aspects of paired programming produced the best results in all the architecture categories and provided the most output with the greatest rapidity for the three usability characteristics above (Waguespack & Schiano, 2004).

Another important experience report in paired programming comes from a detailed ethnographic study of the driver and navigator. Bryant, Romero, & duBoulay (2005), focused on how tools are used in the paired programming cycle and when combined with engineering dialogue, produces a simple but powerful management and communication tool. This observational study took place in several business workplaces with programmers working on real tasks and problems. What they found was a series of cost-benefits due to the evolved roles. Whereas the roles did not always align specifically to the paired programming *rules*, the collaborative setting and role orientation was adequately followed with positive results (Heilberg, Puus, Salumaa, & Seeb, 2003).

The motivation to reduce the cost of development and increase positive benefits was at the basis of Hayes' experience article on expensive bug tracking (2002). Hayes estimated that over $59.5 billion was spent on repairs and corrections to software (bugs). Of that amount, Hayes considered over $22.2 billion or 37% could have been saved through process improvements and better methodological practices (2002). The reason this number has grown over the years is the lack of more precise software engineering methods and tools. Hayes suggested that businesses are at an impasse and must decide on better methods of code development that are faster, simple, yet robust enough to tackle the increasing complexity of growing code bases. He suggested practices such as programming in pairs, radical increase in automation testing, early unit tests, all of which have lightweight best practices and require a commitment on the part of the company and the programmer (Hayes, 2002).

Many more experience reports exist extolling the benefits of paired programming and countering the observational concerns of two people doing what appears to be the job of one. Increased quality, speed of production, tighter code bases, and better unit testing are but a part of the benefits. Yet programmers have been taught to work alone, work in straight line or linear patterns, keeping to tight regimens and rules. There is room for not only new processes but also new educational experiences to create a new breed of programmers ready to collaborate and generate high-quality software in an open and shared environment (Williams & Kessler, 2000). The last section of this literature review will explore the educational challenges and opportunities that lay ahead.

Paired Programming and Education

Research and general literature reviews indicate that there are many educational benefits to the use of paired programming in instructional settings (Williams, 2007). There is a strong indication that paired programming creates an atmosphere of advanced learning and active collaboration. This was found to reduce student frustration, increase confidence, and build additional interest in Information Technology (Berenson, Slaten, Williams, & Ho, 2004). This does not mean that paired programming is the only method that should be taught. Combinations of collaborative and individual programming course work are central to a well-rounded software development education (Berenson, Williams, & Slaten, 2005). The introduction and focus on programming collaboratively is a new addition to the technology educational arsenal.

Collaborative programming, especially working in pairs, has positive and negative results for both the student and the instructor. Williams, McDowell, Nagappan, Fernald, and Werner (2003) found that programming in pairs appears to increase knowledge retention especially among female students. It tends to reduce certain negative conditions of traditional individual programming, particularly in the beginning years of training. Oblinger (2003) found that contemporary students prefer to work collaboratively and that an educational process that mimics the industrial world's demand for better teamwork and communication produces more successful graduates and programmers. The negative aspects are small. Some students will always want to work alone. This could be due to being more intelligent and not wanting to drop to another student's level. Another is the actual management of time to collaborate. Many times, especially with younger students, time-management is not an acquired skill; not to mention the regular distractions of college life (Williams, 2007).

McDowell et al., (2003) take a different approach to paired programming education in their observation study of collaborative programming in the classroom. They noted that most important projects use teams or groups of people to accomplish the work. It is normal for many trained programmers to look over the shoulders of others or to discuss a particular direction or method while in the coding phase of the project. With the advent of paired programming, more collaborative methods have been used to accomplish tasks. The experiences in colleges and universities have not kept pace with the professional world. McDowell et al. (2003) propose that as student's progress, their ability to work collaboratively should be increased. McDowell, Werner, Bullock, and Fernald (2006) suggest that requiring students to do projects alone should be stopped in lieu of working collaboratively. Experimentation and ethnographic studies have indicated that benefits such as more students passing computer science courses, higher quality instruction programs, reduced project completion times, increased student satisfaction, and increased numbers of students pursuing an Information Technology career provide motivation to move forward with collaborative programming instruction (Williams et al., 2002).

Additional variables can play a part in a successful educational strategy to teach collaborative programming. The classroom can be arranged for a highly guided and monitored environment or there can be little to no instructional supervision leaving the programmers to themselves to solve problems and ask questions of one another (Bevan, Werner, & McDowell, 2002). McDowell et al. (2003) found very little difference in time spent by pairs versus time spent by individuals in the completion of their assignments. Williams et al. (2000) indicates that the total time to complete a task decreases as the pair works together over time.

Preston (2005) contributes to the study of the educational aspects of collaborative programming in his observational study of paired programming practices in introductory software engineering courses. His research supports paired programming as an effective instructional method for teaching programming and an efficient method consistent with computer sciences and its constantly changing disciplines (Cliburn, 2003; DeClue, 2003; McDowell et al., 2003). The benefits of paired programming being taught at all levels, from the observations of the studies include higher level program quality, decreased time to complete programs, improved understanding of the programming process, increased course completion rates, and improved exam performance. Five critical attributes that were common to a paired programming instructional approach include common task suitable for a collaborative process, small group learning, cooperative behavior, interdependence, and individual accountability (Davidson, 1994).

Jacobson and Schaefer (2008) presented a study that established the ability to try paired programming in their undergraduate basic computer science curriculum. The objections to using paired programming were reflective of objections in the industry. One basic concern presented included whether problems could occur with paired programming that would reduce the effectiveness of imparting computer science knowledge to students at the basic levels. Another concern presented indicated that paired programming would remove the ability to assess each student’s programming capability individually and might hurt the ability for the student to work independently in other courses and/or in industry. Information collected from student surveys, teacher assistant observations, and object student exams indicated that computer science information was disseminated properly. The second concern was addressed by the successful use of individual programming exams that demonstrated comparative similar scores to non-paired programming courses (Jacobson, 2000). The scores had actually improved with the paired programming practices used in the course. Course expectations were laid out clearly with distinct outlines on how collaborative behavior was to be engaged. The study found that there were no indications of a loss of individual programming capability or rampant pair breakups (Jacobson & Schaefer, 2008).

Zin, Idris, and Subramaniam (2005) found that paired programming was the most applicable method for learning programming through e-learning or distance education. They found that normally e-learners were disconnected from other students as well as the instructor. Paired programming created a means and a motivation to connect with a peer and improve the e-learning experience while practicing a programming methodology (Herbskeb & Grinter, 1999). They termed this virtual pair programming (VPP) (Baehti, Gehringer, & Stotts, 2002; Hanks, 2004; Kiercher, Jain, Corsaro, & Levine, 2001). Instrumental to the success of e-learning was the availability of an open forum portal for collaboration, information, and knowledge sharing. At the end of the courses, students were given a questionnaire to collect their perceptions on the effect of paired programming in the e-learning environment. The results indicated that students perceived he or she had acquired the confidence and capability to go to the next level of programming training through their collaboration with their peers. They experienced less anxiety and greater support than if he or she were alone in their e-learning experience. Logistical improvements were suggested by the students to include the use of an online compiler and the capability of instant messaging to assist in communication (Zin, Idris, & Subramaniam, 2005).

Collaborative programming in computer education programs appears to be a substantive part of improved program success rates and better experiences for students and instructors alike. McDowell et al., (2006) engaged a study to confirm these perceptions and to review the reasons for a low population of women in the field. They found that paired programming when used in a software engineering learning environment increased the number of women and men continuing in their previously indicated degree pursuits, especially in computer science (Williams et al., 2003). Additionally, paired programming, when used in software engineering instruction, created increased satisfaction with the problem-solving process and a greater confidence in the resulting solutions. The study found that some instructors continued to rely on solo programming in an academic setting for fear that one of the pair might not learn as much as the other and/or that assignments would not be completed adequately (McDowell et al., 2006). Although there was no empirical answer to this concern, McDowell et al. (2006) felt that this was not paired programming and that some instructional monitoring could catch problems earlier in the learning process to avoid these problems.

Another aspect of this study was whether using collaborative pairing as a learning aid would influence course completion results and computer science, learning behaviors as indicated by pass rates and continuation of study in the information technology field. McDowell et al. (2006) found students who were involved with pairing were significantly more likely to stay the course through the final exam than students who were not paired. An increase in pass rates was noted for students who were paired but the increase was not statistically significant. Pass rates for men and women who were paired were at similar levels. *Pair Pressure* (Williams & Kessler, 2000) is the influence on the work of a closely working peer. This may be a part of the reason for the higher participation rates in the study. Students who were part of a collaborating pair showed a significant increase of those who declared a computer science major as compared with those who programmed individually. The continued low representation of women in the computer field highlights the need for practices that foster women's interest and result in their educational and professional success (Werner, Hanks, & McDowell, 2005). Paired programming is a practice that offers that possibility with a wide range of benefits for both men and women (McDowell et al., 2006).

Summary

The literature is highly complimentary of the paired programming practice. Benefits abound in studies, experiments, surveys, and experience reports indicating that the general practice of collaboration in software engineering has financial as well as technical values. A review of educational writings indicates that paired programming practices improve education, participation, knowledge retention, and future pursuits of computer related studies. Yet there appears to be a reticence to employ paired programming in some business areas. The literature reveals that biases continue as two programmers working at the same terminal cause fears of waste and costly technology employment. The research methods and activities outlined in the following chapters will describe an empirical survey of software development managers' attitudes and intentions in an effort to understand whether and to what extent the practice of paired programming might be used in the future.

CHAPTER 3. METHODOLOGY

Today businesses are faced with growing costs, higher demands for speed of software delivery, and better efficiency in software engineering (Hayes, 2002). When first introduced, Extreme Programming Methodology and the paired programming practice offered to provide quicker and more reliable software delivery results to businesses. Today many software development professionals are questioning those first findings and challenging the cost effective nature of the paired programming practice (Levy, 1987; Williams et al., 2000). This is primarily due to the use of two programmers at a single workstation instead of individual programmers working independently (Hayes, 2002). The purpose of this study is to demonstrate, through the sampling of the intentions and attitudes of software development managers, the perceived ease of use and perceived usefulness of the paired programming practice compared to individual programming practices.

The literature reflects a wide and variant set of methods in programming today, even in the application of paired programming practices. Many authors have called for continued research to produce additional empirical data on the practice of paired programming. Part of that general call for more empirical data requires an understanding of what the perceptions of software managers are today relative to paired programming. An understanding of these perceptions will contribute to predictions of how and to what extent the paired programming practice will be used in the future. A quantitative method of analyzing the collected data using The Technology Acceptance Model's mathematical formula, a validated and reputable survey instrument, will be employed. The survey provided in The Appendix, establishes the independent variables of the study and the boundaries of the research. The results of this study produce some of the output called for by Davis (1989) in his statements for future research. He stated, “More research is needed to understand how measures such as those introduced here [acceptance of technology practices, processes, tools, and systems] perform in applied design and evaluation settings” (p. 335). By applying the practices of paired programming and individual programming practices to The Technology Acceptance Model, a technology practice is evaluated through an accepted and peer-reviewed measurement framework. The results will also confirm the value of the modeling practice for not only theoretical but practical and field-use as well (Shneiderman, 1987).

Understanding attitudes and intentions, relative to paired programming, adds to the body of knowledge and gives credence to its acceptability and possible future practice in business and industry. Simply understanding intentions and/or attitudes is not enough to provide a scientific accounting of paired programming’s value as a practice or its possible future acceptance and adoption by business and industry. As seen in Chapter 2, various experimental studies have found paired programming to be an efficient practice in software engineering. Yet other studies have found flaws in those findings. Those studies have offered explanations and/or suggested changes in the practice to achieve efficiencies that are more productive.

There is room for continued experimentation in the search of empirical data proving or disproving the effectiveness and efficiency of paired programming. There is also a need to determine what the current level of intention and attitude might be among software development managers who are responsible for setting the methodological directions of software engineering in their respective companies. Through an understanding of their intentions and attitudes it is possible to predict their present and future behavior with respect to the use (or non-use) of paired programming as a practice. This can be accomplished using The Technology Acceptance Model, a tool that has been proven to produce values that establish the behavior and/or acceptance of a technology process, practice, or tool. This understanding will result in either a call for more detailed experimentation in the use of paired programming, or a revelation that its use may be less pervasive or less important than originally thought. The results of this research will contribute to a foundation for building future empirical research in the determination of paired programming's ultimate contribution to businesses.

Research Design

Methodological Approach Background

A methodological approach is necessary in this study to provide a structured and scientific framework to apply the surveyed opinions of development managers on the use of paired programming. The use of Davis' Technology Acceptance Model (1989) is a solid and often used methodological framework for technology practices, processes, and tools. A clear foundation is necessary on how The Technology Acceptance Model provides the scientific basis of predicting acceptance and ultimate usage.

In the early 1980's it was determined through psychological research that intentions and attitudes could be measured and projected into current and future behavior. What people thought about a subject and how they felt about it could be analyzed to the point of predicting their behavior toward that subject (Ajzen & Fishbein, 1980). This is the basis of the Theory of Reasoned Action. Variations on this theory built up throughout the years as studies indicated that the more a person knew and/or believed in a subject or process, the more he or she would behave toward acting on those subjects or using those processes (Ajzen, 1985, 1987, 1991). There was an essential bridge at this point in the research to produce not just a psychological profile, but a sociological reality that predicted the behavior of people based on their attitudes and intentions. This reality was determined by substantive measures that produced coordinated results from attitude and intention surveys and comparative analysis with actual demonstrative actions of people to subject areas (Ajzen, & Fishbein, 1980; Ajzen, 1987, 2005). These measures laid the groundwork for a more formulative and mathematical application of this theory. This provided Davis (1985) the opportunity to focus these theories toward the acceptance of various processes, practices, and tools within the area of technology.

Davis, in his 1985 dissertation from MIT's Sloan School of Business, devised a formulative mathematical application using the Theory of Reasoned Action and the Theory of Learned Behavior called The Technology Acceptance Model (TAM) (1989). This model provided for the measurement of perceived ease of use and perceived usefulness of a process or subject based on the surveyed attitudes and intentions of test groups. It was also based upon the results of a mathematical and statistical formulation indicating acceptance (or lack thereof) of a subject or process (see Table 1 and Figures 2 & 3). The literature is robust with scientific documentation on the application of the TAM in various technology areas. Over 400 studies reflect the use of the TAM to demonstrate acceptance of subject or process ranging from medical procedures and tools to the acceptance of personal computers and cell phones. Substantive discussion exists on the accuracy of the TAM in many of these applications. There is a preponderance of evidence that the TAM has scientific value in predicting the present and continued use and usefulness of processes or practices based on the collected attitudes and intentions of respondents familiar with the process or practice (Ma & Liu, 2004). One practice area that has not yet been subjected to the analysis of the TAM is the use of the paired programming practice by software development managers. This is the focus and subject area of this study.

General Methodological Approach

Consistent with previous scientific and positivistic studies, The Technology Acceptance Model and its mathematical and statistical formulas were used to apply the responses of software development managers on the subject of the perceived usefulness and perceived ease of use of the paired programming practice in software engineering. This was accomplished by a random survey of 500 software development managers from around the United States about their attitudes and intentions regarding the use and ease of use of the paired programming practice. The response data was entered into the TAM formulae and through various statistical tests, helped to reject or not reject the null hypotheses presented in Chapter 1.

Over 146 documented studies and over 1000 peer-reviewed articles have used or referenced the TAM methodological approach for various technology subject areas. In addition to the two variables basic to the TAM, behavior and usage results are added in the analysis of the data from software development managers. Precedent for such additions in TAM research exists extensively in literature and appears to be significant in the progression of the TAM's influence in technology acceptance research (Venkatesh &

Davis, 2000). The results of this study using actual attitudes and intentions of software development managers will indicate whether and/or to what extent the paired programming practice might be used in software engineering groups. Demographic type data from the survey instrument will also allow the data to be broken down into various business type categories, consistent with the U.S. Government Standard Industrial Classification (SIC) system's 10 major divisions (see Table 3). This will support the hypotheses associated with the effect of business type on use of paired programming—Ho4: Observations of paired programming compared to individual programmers working alone will not be affected by the reporting software development managers' type of business.

Table 3. U.S. Government Standard Industrial Classification Codes by Major Division

Division	Description
Division A:	Agriculture, Forestry, and Fishing
Division B:	Mining
Division C:	Construction
Division D:	Manufacturing
Division E:	Transportation, Communications, Electric, Gas, and Sanitary Services
Division F:	Wholesale Trade
Division G:	Retail Trade
Division H:	Finance, Insurance, and Real Estate
Division I:	Services
Division J:	Public Administration

Note: Adapted from http://www.osha.gov/pls/imis/sic_manual.html.

At the conclusion of the random survey of software development managers, the survey data were analyzed, compiled, and the results applied to The Technology Acceptance Model. The results of the TAM calculations indicated the possible usage of the paired programming practice as compared to the individual programming practice. The findings also indicated how the type of business affected software development managers' behavior and acceptance of the paired programming practice compared to the individual programming practice. The methodology employed in this study was designed to rely on a proven framework used in previous technology studies as well as a solid progressive scientific framework of prediction from recorded behaviors, captured intentions, and attitudes of respondents. Through the findings of this study, software development practitioners will be aided in making informed judgments about the use of the paired programming practice in their respective software engineering areas. These results will also serve to guide technology and software business leaders in their understanding of the usefulness and ease of use of paired programming for their enterprises. The results from this study will help to guide future research, especially experimentation, in determining the effectiveness and efficiency of paired programming as compared to traditional individual programming.

Through this research, a solid contribution to the building of the body of knowledge is being made on the perceived use and perceived usefulness of paired programming as an efficient and effective practice for software engineering. This knowledge base continues to progress and evolve. It becomes richer with the results of studies such as this one and others that assimilate observation and future research into paired programming's contribution to business.

Sampling

The research design for this study was a systematic random sample survey of 500 software development managers from around the United States. The survey designed for this study was taken, with already established permission, from the survey used by Davis (1989) that determined the predicted ease of use and predicted usefulness of several technology subjects. The data collected from the survey used in this study was processed through The Technology Acceptance Model's mathematical formulation (see Table 1). The results indicate a significant relationship between perceived ease of use and perceived usefulness of the paired programming practice as well as a measured significance between these variables within the individual programming practice construct. In this study, a statistical comparison of the predicted ease of use and predicted usefulness of both practices along with a comparison of self-reported usage were made. The findings reject or fail to reject the hypotheses proposed in Chapter 1.

Target Population

The specific target population was software development managers within the United States. The term: *software development manager*, can mean any person responsible for software development or code engineering of software programs for use in computing environments. It should be noted that this could mean software intended for use internal to the business (such as software developed for an Enterprise Resource Planning system) or as commercial off the shelf products (such as shrink-wrapped software applications for commercial or semi-commercial sale). The level or title of a software development manager is not considered significant in this study. Assumptions made with this title include the general responsibility of the software development leader to determine the methodology and practices employed by his or her software development group. Chief Technology Officers and Chief Information Officers are considered software development managers for the purpose of this study because of their responsibilities associated with the groups that produce software code. This responsibility for software engineering is also relevant for application software leaders who hold the title of Director, Vice-President of Software Engineering, or other such titles that indicate a basic responsibility for software development practices within their software engineering groups. Email listings are publicly available for those responsible for leading software development groups. These listings are valuable to the survey's basic capability to export an electronic questionnaire as part of the data gathering and research process.

Sample Frame Random Stratified Selection

A random sample of 500 software development manager emails was acquired from the target population by a sequenced ratio selection of the total target population. To achieve a random sample, the total value of the population (Z) was divided by 500 and resulted in a sequence number to be applied to the total population. For example, with a target population (Z) of 1500 emails, divided by 500 (X), would result in every third email selected from the sample frame. The sampling ratio or probability of selection (P) is equal to the sample size (X) divided by the population size (Z) (P = X/Z). For this example, the probability of selection is 0.2% (0.002). Records in the target population presented to the sorting engine in alphabetical order of the email's address line provided a reasonable sample without location or business-type influences or biases (Deming, 1960).

Sampling Methods

The method of sending surveys was through electronic mail. The mail message had an embedded hyperlink that took the respondent to a professional survey site. The respondent had the choice to click on the hyperlink to go to the survey or exit the email and delete it. The email had no agents or cookies deposited on the respondent's workstation. Once the respondent clicked the hyperlink to go to the survey site, a further explanation of the survey was given. The respondent at this point had the opportunity to accept (go onto the survey) or escape (leave the survey site). At all times and on all pages of the survey, the respondent had the ability to stop the survey and escape (log out of) the survey site. Several ethical and procedural conditions are met in operating the survey in this manner. First, the two-staged process of first seeing an introductory email explaining the study with the ability to escape/delete or move forward, establishes voluntary and informed consent. The second, after landing on the survey site where a consent form is presented; the respondent has the choice of moving forward or escaping the site. Throughout the survey, the respondent could escape any time and abandon the survey. This confirms voluntary and informed consent and establishes the conscious decision of the respondent to complete the survey.

The actual survey instrument was produced from a commercial survey generation site called *SurveyMonkey*. Using the pre-established sample frame, an introduction email was send to the proposed respondents. The respondent had the chance to take the survey, delete the email and not take the survey, or wait and take the survey at another time. The initial survey was posted for three weeks, after which the survey site URL was expired. *SurveyMonkey* automatically collected the survey responses and provided the raw data as well as a series of general summation reports from which data analysis and statistical tests were applied.

Instrumentation and Measures

The survey instrument used in this study is patterned after the TAM survey template developed by Davis (1989), (see The Appendix). Permission to use this survey template was obtained in electronic form from Davis on July 16, 2008. There are six sections to the survey. Each of these sections in the survey instrument address a particular variable: (a) Usefulness for paired programming practices, (b) Usefulness for individual programming practices, (c) Ease of Use for paired programming practices, (d) Ease of Use for individual programming practices, (e) Self-reported Usage, and (f) The collection of demographic and respondent experiential information. There is nothing in the survey instrument that identifies the respondent personally or through a position within a specific company. No company names or specific titles are used in the survey. No specific or descriptive titles are given in these sections in order to eliminate possible pre-conclusions by the respondent. Instead, sequential Roman numeral designators are used.

A seven point Likert scale was used to measure responses in five of the six subject sections of the instrument. On the survey, explanatory descriptors were used to aid in the placement of the respondent's selection (see Figure 7 for a scale sample). An effort was made to equalize all measurement scales in order to provide a more accurate comparative analysis. Demographic data was provided to sort records easily and support hypotheses related to the effects of gender, business type, experience with paired programming, and experience with software development as well as actual titled position, and number of years of experience in paired programming environments and/or individual programming environments.

Sample Survey Scale

Using Paired Programming in my Software Development Group would enable me to accomplish tasks more quickly.

Likely	extremely	quite	slightly	neither	slightly	quite	extremely	Unlikely
	↓	↓	↓	↓	↓	↓	↓	
Likert Values	7	6	5	4	3	2	1	

Figure 7. Sample – survey scale. This sample shows the Likert scale used in the research survey (see The Appendix)

Data Collection

Using an eSurvey Engine

Data collection for this study was accomplished through *SurveyMonkey*, a web-based commercial survey-software engine provider. Their services used in this study included the collection of data and the application of several statistical analysis tools. The data was provided through a series of Microsoft Excel spreadsheets and was easily portable to the SPSS version 15.0 data analysis application program and to Microsoft Office Professional 2007 Excel-Data Analysis. It is not anticipated that later versions of the SPSS application will affect the outcome, validity, or reliability of analyses in this study. These programs were used to produce the necessary statistical analysis that has been reported in Chapter 4 of this study.

The data research process within this study used email addresses to connect with possible respondents. This method kept anonymity high for the respondents and provided for a more random and unbiased sampling process. The email listing was obtained through *BizExUSA.com,* a marketing contact web site and IEEE, a technology academic and professional association. The first generation pass for *Software Development Managers* yielded 1312 hits. Names, addresses, and business types were excluded from the records report with primary emails being the key identifier for the record. The email listing was randomly sorted to provide a listing of the primary 500 records. With these records extracted, another random sort was made to yield a secondary 500 record listing.

It was anticipated that there would be between a 15% to 25% response rate to this survey. An option was planned should the initial survey fail to yield these results. This included the execution of a secondary survey drawn from the original target sample. A method was available to extract the original 500 records from the secondary sample frame prior to reapplication to the target population. This is how an additional set of possible respondent emails was processed. As described in Chapter 4, this process was repeated three times before sufficient data was collected to obtain a successful sampling rate of approximately 10%.

Consent

An electronic mail listing was engaged that when opened delivered an email message with a web-link to the survey site. This method was used to contact possible respondents from the randomly selected target group derived from a sample frame of software development managers. The email explained the reason for the contact and included a web hyperlink provided for the respondent to select and then go forward to the survey or to select the *Exit* button that returned the respondent back to the email inbox. If the respondent selected the web hyperlink, a survey instruction page appeared along with an informed consent form. The respondent had the opportunity to select either the *Continue* button or the *Exit* button as the manner of consent or non-consent respectively to participate in the survey. Selecting the exit button stopped all forward progress toward the survey and returned the respondent to the original email. Selecting the continue button advanced the respondent to the survey introduction page and the actual survey (see The Appendix). The respondent was able to exit the survey at anytime by selecting the *Exit* Survey button.

At all times, the respondent had the choice of exiting the email and deleting the email message, exiting the email and saving it for attention at a later time, or clicking the hyperlink to send the survey to the Survey Monkey location where the survey was administered. Once in the survey site, the respondent was able to see a further explanation of the survey. At that point, the respondent could exit the survey engine and delete the engaging email, exit the survey engine and come back later, or select *Continue* to move forward to the survey. These movements and choices along with the opportunity for the respondent to exit the survey in two places, at any time prior to taking the survey and during the survey, formed the basis of the respondent's informed consent for the survey.

Data Analysis

Meeting Statistical Guidelines

There are four general areas in the analysis associated with hypotheses one through three: (a) Perceived usefulness of paired and individual programming, (b) Perceived ease of use of paired and individual programming, (c) Self-reported usage, and acceptance, and (d) Calculated usage data (Davis, 1989). For analysis purposes, a

Cronbach Alpha reliability coefficient calculation for each construct was conducted for the survey used in this research. The original calculations that were made by Davis (1989) achieved the desired result for each construct, which was above the minimum standard of 0.70 desired for social science or management research. Data collected followed a normal distribution as indicated by Davis' actual results and validity measures (1989). A normality test for distribution was made and reported to assure that the values followed planned normal distribution.

Statistical Analysis Tools

The statistical analysis tools used in this study are correlation and regression analysis consistent with the TAM formula. Correlation analyses are applied to the four constructs. The Technology Acceptance Model is the theoretical framework used in this study and is built on the relationships between intention and attitude that, when measured through correlation and confirmed through regression analysis, can predict the behavior of usage for a technology process, tool, or practice (Davis, 1985, 1989). The outcome of these relationships provides a basis for prediction of behavior toward the subject as well as an acute estimation of usage (Mason & Bramble, 1989). Linear regression and regression analysis is used to compare the correlational variable content that will be applied in the TAM formula (see Table 1). For the constructs, the results and associated *p*-values of significance are presented between perceived usefulness and perceived ease of use, between perceived usefulness and perceived ease of use and behavioral intention to use, and between behavioral intention to use and actual usage. Data is then loaded into Microsoft Office Professional 2007-Excel and SPSS version 15.0 application for statistical presentation and analysis, which is further explained in Chapter 4 of this study.

In order to perform the required analysis used within this study, the TAM methodology was followed using both Davis' 1989 study and an application of the TAM as outlined by Money and Turner (2004). Their research study provided a peer-reviewed and accepted model for applying the TAM to a technology process, practice, or tool. Relationships and correlations between the key factors of TAM and the collected responses from software development managers on the practice of paired programming and individual programming, form the research and analysis framework for this study (see Figure 8).

The TAM Research Model with Correlations Variables

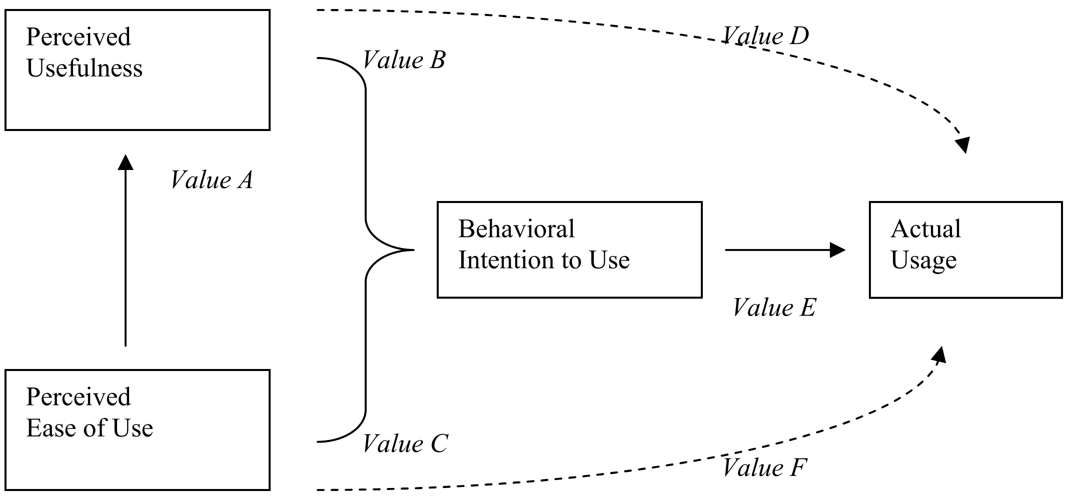


Figure 8. The TAM research model with correlations. This graph shows how the various values applied to the TAM are combined to form a measure of actual usage. Adapted with permission from *A TAM framework to evaluate users' perception towards online electronic payments* (p. 5), by G. Rigopoulos, and D. Askounis, 2007. *Journal of Internet Banking and Commerce, 12*(3). Copyright 2007 by the Journal of Banking and Internet Commerce.

Statistical Tests Employed

In The Technology Acceptance Model, there are two standard statistical tests, *Correlation Analysis*, and *Linear Regression*, as well as mean (arithmetic) values from the combinations of question-values from the groups of the various constructs of the survey. The correlational analysis is bivariate, meaning there are two continuous variables measured using an interval scale. The resulting outcome does not recognize an independent and dependent variable, as both variables are symmetrical. These primary variables or constructs are perceived ease of use (EOU) and perceived usefulness (USEF). Each is treated symmetrically as the coefficient $r_{(eou)(usef)}$ has the same interpretation as $r_{(usef)(eou)}$ (Cooper & Schindler, 2006). The variable, *Attitude and Behavior* (ATT) is derived from the self-reporting usage data. The final variable, *Actual Use* (USE), results from the primary variables delivering data into the TAM formula and processing it through the regression and correlation operations (see Table 1) (Davis, 1989).

The second statistical test in the TAM is a simple linear regression or regression analysis. This provided the predictive values necessary to confirm the correlational determinates of usage and acceptance of the subject practice within the TAM. Regression explains relationships. Within the TAM, the relationship was explored between perceived ease of use and perceived usefulness once the coefficient of correlation is determined (see Table 1, step 3). Bivariate linear regression was then applied to the resultant correlational values of the constructs EOU and USEF. This resulted in a value that represented the attitude toward using the subject practice (paired programming) represented by the variable ATT (see Table 1, step 3). Predictive use was then confirmed by a calculation applying a simple regression against the attitude variable to form the resultant output of usage (USE) (see Table 1, step 4). It should be noted that for all steps in the TAM formula, ε symbolizes the deviation of the ith observation from the mean as a form of error checking (see Table 1).

The process of applying the statistical tools of correlation and regression found in the TAM is simple. The values from the surveys were calculated to $\sum$ sigma summation for perceived ease of use (EOU) and correlated. A $\sum$ summation for perceived usefulness (USEF) data was also made, correlated, and applied to the perceived ease of use variable (Robson, 2002). Both EOU and USEF were then correlated to the $\sum$ summation of self-reported behavior to use or intention for usage (ATT). EOU and USEF were also correlated to usage USE to determine the final value for predicted future use. Both results for paired and individual programming were independently regressed and compared (see Table 1) (Davis, 1985, 1989). Both variable constructs for the paired programming practice and the individual programming practice were compared for actual values to determine the one most and/or least used. This pointed to a rejection or non-rejection of the hypothesis Ho2.

Correlation tables are constructed and displayed in Chapter 4 to show associated values and results of perceived usefulness, perceived ease of use, and self-reported usage for both paired programming practice and individual programming practice. The variable subjects of the paired programming practice and the individual programming practice are combined and presented in a correlational display between the variables shown in various tables. Other tables found in Chapter 4 show the mathematical representation of the TAM method in formula form (see Table 1), and represent the statistical testing model for rejection or non-rejection of the null Hypothesis (Ho3). Prior to that, the mean values of perceived ease of use, perceived usefulness were correlated separately (between paired and individual programming) to show independent effects on each of the constructs.

A regression analysis table is displayed in Chapter 4 to show the effect of perceived usefulness and perceived ease of use based on self-reported usage as a confirmation of the correlational tests (Davis, 1989). Although a regression value was not specifically a part of the TAM, it was used by Davis to demonstrate the validity of the correlational values (particularly perceived ease of use and perceived usefulness) when compared to self-reported usage. The regression chart shows the pooled (combined) value and displays regression values against usefulness, ease of use, and the resulting R^2. Associated *p* values depicting significance for each calculation are noted. This is not a calculation for the confirmation of the hypotheses, but rather a calculation that Davis (1989) used in confirming the relevance of the TAM formulae. It should be noted that Davis did not segment behavioral intent to use and self-reported usage, but pooled these values. In this study, these values are segmented into behavioral intent to use and self-reported usage, following the methodology of Rigopoulos and Askounis (2007).

Statistical Tests Linked to Hypotheses

The first hypothesis (H1) tests the correlation of perceived ease of use (EOU) of paired programming and individual programming. The summations of the survey values were collected and the mean values compared and correlated. These values were then entered into the TAM formula (see Table 1, step 1). Davis indicated that the TAM values had normal distribution in his studies (1989). In this study, a test was conducted to ensure the data was normally distributed. The data collected was found to be normally distributed and the sample size large (more than 30 samples were collected with values that are ordinal) as well as with independent variables. This resulted in a Z test being used to reject or not reject the null hypothesis. (Had the data collected been a normally distributed set with a small sample for the independent variables, [less than 30] a *t* test would have been run to reject or not reject the null hypothesis.) The results produce a 95% confidence factor that will accept the null hypothesis as correct if the calculation is less than two. If the calculation produces a value greater than 1.96, (more than 1.96 standard deviations of separation) then the null hypothesis is rejected. The value was then entered into the EOU variable of the TAM.

The second hypothesis (H2) tests the correlation of perceived usefulness (USEF) of paired programming and individual programming. The summation of the survey values was collected and the mean values compared and correlated. The mean values were then entered into the TAM formula (see Table 1, step 1). A Gaussian (normal) distribution was completed to represent the combined data. Davis indicated that the TAM values had normal distribution in his studies (1989). In this study, a test was run to insure the data was normally distributed. The data collected was normally distributed and the sample size large (more than 30 samples were collected with the values appearing to be ordinal) as well as there were independent variables. This resulted in a Z test being used to reject or not reject the null hypothesis. (Had the data collected been a normally distributed set with a small sample for the independent variables, [less than 30] a *t* test would have been run to reject or not reject the null hypothesis). The results produce a 95% confidence factor that will accept the null hypothesis as correct if the calculation is less than two. If the calculation produces a value greater than 1.96, (more than 1.96 standard deviations of separation) then the null hypothesis is rejected. The value was entered into the USEF variable of the TAM.

To answer the primary research question and third hypothesis (H3) of this study, it was necessary to determine the value of actual usage for both the paired programming practice and the individual programming practice. The value of usage (USE) for each of these constructs was measured by a progression of separate correlational tests on the mean values of summation of the questionnaire constructs and *t* tests (normalized data, with a low sample value [<30]) to determine significance. The tests reject the null hypothesis based on difference between correlated means of usage compared between the two constructs (paired and individual). A confidence factor of 95% was chosen, which is close to two standard deviations of separation between the means. The values in the *t* table represent the area under the normal curve for the number of standard deviations selected. 95% confidence factor indicates there is only a 5% chance that the means will be greater than two standard deviations by chance. This results, with a 95% confidence factor, will confirm the null hypothesis as correct if the calculation is less than 1.96. If the calculation produces a value greater than 1.96 (more than 1.96 standard deviations of separation), then the null hypothesis is rejected. There is only a 5/100th chance this will happen by chance. If there is a result of five, this would mean the number of times, out of 100, this could happen by chance. This would actually be significantly less and would reject the null hypothesis with higher confidence.

The *t* tables used for this test are set for a range of +1.96 to -1.96 standard deviation. These are the 95% confidence limits for confirming the null hypothesis with 95% confidence. If the calculated value is greater than 1.96 then the null hypothesis is rejected. At this point 2.5% of the normal curve's area is greater than 1.96 and 2.5% is less than -1.96. At that point, 5% of the normal area is outside these limits, which means that if the calculated number is greater than 1.96 there is only a 5% chance this could have happened by chance. Using this test, the null hypothesis values typically ignore the sign and thus take absolute values (Dorofeev & Grant, 2006).

The TAM formula calculates the intentions and attitudes of software development managers through the correlation and regression of the mean summation values from the data of perceived usefulness (USEF), perceived ease of use (EOU) and self-reported attitude toward usage (ATT-which is a combination or pooled value of behavioral intention to use and actual self-reported usage). This process is replicated for the constructs of the paired programming practice and individual programming practice each having a usage (USE) result. The usage results were correlated and compared to determine the final values against the criteria for rejection or non-rejection of the null hypothesis. The value of usage (USE) also provided a variable for other tests in this study. The hypothesis assumed that the resultant relationships demonstrated a positive acceptability and usage of paired programming over individual programming.

Additional Hypotheses Tests

The TAM offers the opportunity for determining the value of acceptance/usage of a practice. That value can have significant relationships that heretofore have not been discovered. The following variables relate specific conditions to the resultant value of TAM-usage.

The first of these variables (related to Ho4) provides for an analysis of the mean relationship between the business type and the resultant usage value of paired and individual programming. There are 10 general business types (see Table 3) or divisions designated by the U.S. Government Standard Classification Codes (2008). Survey responses were sorted by these various types and correlated to determine if there is a variation of significance between any of the types. This correlation indicates the possibility that a business type might influence the usage or acceptance of the paired programming practice in a general type of business. The research within this study offers the opportunity to understand the relationships of acceptance and usage (of paired programming) with the types of businesses that might employ this practice. An analysis of variance (ANOVA) comparison of the TAM results based on the USE variable and the variable of business type found in section VI of the survey, provided an appropriate and highly informational data point on what business types might employ the paired programming practice as compared to other business types. In this study, the relationship of business type to behavioral acceptance of the paired programming practice has significant practitioner benefits. This analysis departs from Davis' (1985, 1989) research, but is consistent with his call to extend the research to practical field applications and relationships. Follow-on research will also benefit from the collection of this data and the analysis of the findings; especially when considering the business type and whether to employ or not employ the paired or individual programming practice for a particular business type or industry.

The next variable (related to Ho5) provides for an analysis of the relationship between the software development manager's experience (in software development) and the level of acceptance/usage of the paired programming practice. In the literature review of Chapter 2, there is evidence that paired programming is a practice being used to teach new and junior programmers. It is possible that the use of the paired programming practice is a result of current software development pedagogy and not related to a software manager's experience. There may be evidence that the more experienced a software development manager, the less likely that manager may be aware of/or positive toward the usage of paired programming. A non-parametric test of the variable of software development managers' experiences to the result of acceptance/usage is appropriate in this situation. The use of an ANOVA can test the differences between the software development manager's experiences and the result of acceptance/usage (USE) to reject or non-reject the null hypothesis (Ho5). The data is ordinal in nature, leading to some minor errors. The abundant sample size helped to support the significance of differences in the two variables (Cooper & Schindler, 2006). This result has future value for follow-on studies related to types of practices employed by software development managers. This relationship highlights the possible connection of the respondents' experience and their acceptance/usage of the paired programming practice, compared to the use of the individual programming practice. This simple mean measure indicates the effect, if any of the previous development managers' experience on their acceptance and/or use of paired programming. There is some indication, as was indicated in Chapter 2 above, that previous application development experience may be inversely proportional to the level of paired programming acceptance.

Displaying Results

As with Davis (1989) and Rigopoulos & Askounis (2007), tabular displays of statistical tests (for Ho1 through Ho3) are provided in Chapter 4. The first set of tables display the correlations between perceived usefulness, perceived ease of use, and self-reported practice usage. Value entries are coded for their respective *p* values to indicate significance. Columns include the two subject variables of paired programming, individual programming, the correlation of usefulness and usage, the correlation of ease of use and usage, and the correlation of ease of use and usefulness.

Validity and Reliability of the Survey

Survey Construction and Validity Measures

Original Survey Validity Measures. The initial survey developed by Davis (1989) attained a Cronbach Alpha (1951) reliability value of 0.97 for the first variable of usefulness and .91 for the second variable for ease of use. Discriminant and convergent validity were attained through the multi-trait/multi-method matrix (MTMM) (Campbell & Fiske, 1959). It appears that the methods used in Davis' original study were a combination of initial surveys, individual, face-to-face interviews, and post-questionnaires. Two methods were used: same traits-different method and different method-same trait. These were applied to the constructs of perceived usefulness and perceived ease of use. Because of the MTMM correlations, Davis (1989) altered some of his scaling to improve the survey for perceived ease of use. The final survey instrument forms the foundation of the one used in this study.

The MTMM matrix covers the inter-co-relational methods applied to the two variable traits (usefulness and ease of use). Convergent validity applies to the scale if it acts as if it is measuring a common construct and elements that measure the same variables should display a high rate of correlation with one another (Campbell & Fiske, 1959). Davis' (1989) results produce high correlations between the two variables of usefulness (95%) and ease of use (95.6%).

The other form of validity, discriminant validity, should display a measure that differentiates between the various objects or variables measured. If there is little or no differentiation, there may be a common thread of variance that is part of the survey instrument or hidden variations in how respondents answer the same questions (Campbell & Fiske, 1959). The test for discriminant validity indicates that an object should have a high correlation with other objects that are measuring the same trait as opposed to a comparison of objects with different traits. The Davis scales indicated less than 3% exceptions for comparisons made, resulting in a high variance of objects, which are not influenced by questions or the internal scales (Davis, 1989).

There is an importance to these elements. The basis of the survey instrument used in this study is the original survey from the Davis 1989 study. It has already demonstrated a high validity rate and should perform equally as well in this study as long as consistency to the scale and the methodological artifact is maintained. The same question patterns are used to achieve as close an alignment to the original survey artifact as possible. Specific definitions in the survey explanation for perceived ease of use and perceived usefulness are provided to reduce possible discriminant validity issues.

In the initial study, Davis (1989) was concerned with factorial validity as well. This is a possible crossover of the scales between the variables and the possible extent to how much crossover occurs between those scales. Data for both the variables indicated that there was little to no crossover between the variables (Davis, 1989). Since this study does not change the essential questions used by Davis (1989), no particular problems with the survey were anticipated. Additionally, behavior and usage were measured in order to meet the variable requirement of the TAM formula (see Table 1 and The

Appendix, Section V). Except for subject name changes, this section is replicated directly from Davis' (1989) survey.

Requirements for Validating the Current Survey

The survey in this study is taken by permission from the original Davis study (Davis, 1989; personal communication, July 16, 2008). The survey used in this study makes nominal changes to the terminology used in Davis' original survey. For example, in the original study, the TAM was applied to several automated charting tools and processes. A survey on the perceived ease of use and perceived usefulness was conducted for each charting tool to be studied. The focus of research in this study's survey used paired programming as the subject of interest. Individual programming was considered the antithesis of the paired programming practice. These were then compared and contrasted. Measures on the Likert scale indicate to what extent the respondent favored the paired programming practice. Next measures on a Likert scale indicated to what extent the respondent favored the individual programming practice. It was then possible for the respondent to express a like (or distain) on a measured scale for either or both or none of the practices. The elements and wording of the questions in the survey remain substantially the same as in the original Davis survey (C. Butler & V. Coxon, personal communication, January 16, 2009).

Since the subject names were changed in this study's survey instrument, this introduced a small foreign factor to an exact replication of Davis' original survey. As a result, a short validation pre-test was conducted using a Cronbach Alpha measure for internal reliability and a face-to-face follow-up interview for validity (Humboldt State University Survey Website, 2008). This was accomplished during a pre-test of at least 12 respondents. The survey was administered in hard copy and not through an email hyperlink-to-survey engine. It used the same survey explanation directives as the formal, emailed survey version and had the same capabilities for the respondent to withdraw from the survey at anytime.

Pre-Test Survey Plan for Validity and Reliability

During the pre-test of the survey instrument, a group of 12 experienced senior programmers was asked to assume the role of software development managers from one of their former companies. They were asked to complete the survey in printed form and then note any comments about the survey. 24 hours after the survey was taken, a sub-group (approximately 40%) of the programmers was individually interviewed in a face-to-face meeting. A standard list of questions was asked each respondent to determine if their answers on the survey were consistent with the question’s intended meaning. Any comments or questions about the survey were gathered at the time of the interview. The results of the interview and the post-survey questionnaire were correlated to the values from the pre-test survey to obtain a validation result.

The pre-test was conducted to find ways to increase actual participant interest, ensure participants would complete the survey, ensure word and content level were adequate, determine if questions required more or less understanding, and to determine if there were quality issues with the survey (Cooper & Schindler, 2006). The results of the pre-test indicated only minor changes were needed. No additional pre-tests were indicated at that time. The pre-test results were analyzed using the Cronbach Alpha statistical test for reliability and a post-survey questionnaire and interview were used to test for validity (see Table 4). The pre-test was engaged prior to the application and approval for research to the Institutional Review Board (IRB) for the execution of any pilot survey and the main survey.

The pre-test surveys were each given a random alphanumeric code. A card with the same alphanumeric code was given to the respondent. This code was also used on the post-test questionnaire and interview comment form to support validation calculations. Rigopoulos and Askounis (2007) used the term *constructs* in their TAM survey model to represent groups of questions. This term was also used in the same manner for this study.

Results of the Pre-test Survey for Validation and Reliability

The data analyzed in the pre-test resulted in a Cronbach Alpha reliability coefficient (see Table 4) for the constructs of Perceived Usefulness of 0.81, Perceived Ease of Use of 0.62, and Behavioral Intention to Use of 0.93. These values indicate a strong reliability measurement construct for Perceived Usefulness and Behavioral Intention to Use but a weak reliability measure for Perceived Ease of Use. During the follow-up interview, it was learned that Q11, Q21, Q23, Q24, and Q26 proved somewhat confusing as respondents found it hard to respond to implementing a practice from a manager's role. They indicated that since they had not done this in the past, it was hard to respond to the question. By inserting an average value of the standard deviation for these five questions, the reliability coefficient for Perceived Ease of Use was raised to 0.80. By eliminating this record from the post-test collection, the reliability coefficient for Perceived Ease of use was raised to 0.81. Using either method meets the minimum reliability score of 0.70 held for most social sciences (Compeau & Higgins, 1995).

Table 4. Pre-test for Reliability Using Cronbach Alpha Method

Item Analysis of Q1 through Q7

Variable	Count	Mean	StDev
Q1	11	4.273	1.421
Q2	11	3.545	1.508
Q3	11	3.545	1.508
Q4	11	3.182	1.250
Q5	11	3.364	1.027
Q6	11	3.727	1.348
Q7	11	3.545	1.368
Total	11	25.182	6.462

Cronbach Alpha = 0.8071

Item Analysis of Q8 through Q13

Variable	Count	Mean	StDev
Q8	11	4.455	1.128
Q9	11	3.364	1.362
Q10	11	3.364	1.433
Q11	11	3.818	0.982
Q12	11	4.364	1.286
Q13	11	4.273	1.348
Total	11	23.636	4.456

Cronbach Alpha = 0.6187

Item Analysis of Q14 through Q20

Variable	Count	Mean	StDev
Q14	11	3.364	1.433
Q15	11	3.909	1.300
Q16	11	4.182	1.328
Q17	11	4.636	1.433
Q18	11	3.909	1.300
Q19	11	3.000	1.265
Q20	11	4.182	1.471
Total	11	7.182	6.539

Cronbach Alpha = 0.8115

Item Analysis of Q21 through Q26

Variable	Count	Mean	StDev
Q21	11	2.000	0.632
Q22	11	2.727	1.009
Q23	11	2.091	0.539
Q24	11	2.364	0.924
Q25	11	2.909	1.300
Q26	11	2.364	0.809
Total	11	14.455	3.205

Cronbach Alpha = 0.6265

Item Analysis of Q27 through Q31

Variable	Count	Mean	StDev
Q27	11	3.727	1.794
Q28	11	3.273	1.421
Q29	11	3.182	1.328
Q30	11	3.364	1.433
Q31	11	3.636	1.748
Total	11	17.182	6.824

Cronbach Alpha = 0.9249

Consolidated Results

Construct	Cronbach Alpha
Perceived Usefulness	0.8093
Perceived Ease of Use	0.6226
Perceived Ease of Use (adjusted)*	0.7993
Behavioral Intention & Usage	0.9249

Note: *Adjustment due to respondent anomaly, average question standard deviation used.

The response rate of 11 completed surveys out of 12 surveys distributed provided a good test backdrop. For validation purposes, six alphanumeric codes were picked from the 12 distributed. One of the codes represented the survey that was not returned. This resulted in five post-test questionnaires and five interviews. Each respondent was asked to complete a questionnaire of six questions identified only with the alphanumeric code from the card given to him or her during the pre-test. These summarized the actual field-test survey questions as well as asked the respondent about his or her ability to think and respond as a development manager instead of a programmer. A match of the alphanumeric codes between the pre-test and the questionnaire were made. The results from four out of five respondents indicated that their initial pre-test answers were consistent with the post-test questionnaire for Perceived Usefulness. The results from three out of five respondents indicated their initial pre-test answers were consistent with the post-test questionnaire for Perceived Ease of Use. Four out of five respondents indicated their initial pre-test answers were consistent with the post-test questionnaire for Behavioral Intention of Use. Three out of five respondents indicated that they were able successfully to maintain the role of a software development manager to answer the field-test survey questions.

A review of the questionnaires indicated that the same person was inconsistent for three of the constructs and was not able successfully to answer the questions in the role of a software development manager. Another respondent also was inconsistent in answering questions between the pre-test survey and the post-test questionnaire for the construct of perceived ease of use. The coding indicated they were also not able successfully to maintain the role of a software development manager. The post-test questionnaire was consistent with the anomaly indicated in the Cronbach Alpha results for the construct of perceived ease of use. It appeared that attempting to maintain a role as a software development manager for the purpose of the pre-test was successful for some and not for others. A review of the questions with low standard deviations for the construct of perceived ease of use did require some dedication to the role of software development manager. The anomaly of a lower reliability score, and the inconsistency of answers between the pre-test survey and the post-test questionnaire, was understandable.

A follow-up face-to-face interview was held with five of the respondents after the questionnaires were analyzed. During the interviews, each respondent was asked if he or she had any comments or reactions to the pre-test survey and/or to the post-test questionnaire. Four respondents indicated they did not have any comments, but indicated that the survey was either interesting or that they were interested in the outcome. One respondent indicated that it was hard for him to put himself in the role of the software development manager. A review of his coded alphanumeric card indicated this respondent was the one who had been inconsistent between the pre-test survey and the post-test questionnaire. All respondents indicated that the survey questions would be relatively easy for software development managers to answer since all the questions were reasonably in the realm of a software manager's experience.

It appears from the Cronbach Alpha reliability scores and the follow-up questionnaire and interviews, that the survey has a solid basis of validity for respondents that are software development managers (see Table 4). It appears that the Davis (1989) survey instrument, as altered for obtaining perceived usefulness, perceived ease of use, and behavioral intention and usage values used in this study, is valid and reliable. The TAM and the associated survey instrument have been well tested in the field. Over 146 separate research studies have used the TAM successfully to determine behavior and usage toward a process, product, or tool. There appears to be no reason to subject the TAM to a pilot study for the purpose of this research endeavor.

Execution of the Main Research Survey

The sample frame was calculated through a selection program built through Microsoft Excel 2007. The selected email listings were connected with the survey invitation letter and sent through the Internet. The contact data consisted of the email addresses of randomly selected software development managers from all over the United States. The process of presenting the survey electronically was accomplished in two stages. The first stage consisted of the introduction email. This introduced the proposed respondent to the survey. The proposed respondent had the option of: (a) Clicking on the URL that would take him or her to the survey home page, (b) Bookmark the URL for later attention, or (c) Delete the email completely. Proposed respondents were not tracked and were not contacted for reconsidered or follow-up participation.

The second stage involved the first two pages of the survey that provided the instructions and definitions of the survey. Each page throughout the survey allowed the respondent to either click to continue or exit the survey. These controls were provided to allow the respondent to complete or depart the survey without the issue of being coerced into completing the survey. In this regard, informed consent was provided, allowing the respondent complete control of the experience. Respondents were informed in the introduction email and on the survey instructions that the survey would take 8 to 12 minutes to complete. This time was established through a pre-test of 12 senior-level, experienced programmers completing the survey in an average of 10 minutes.

The first four sections of the survey used in this study copied Davis' "Final Measurement Scales for Perceived Usefulness and Perceived Ease of Use" (1989, p. 340). This scaled survey was validated in Davis' original 1985 Doctoral Dissertation and again in his 1989 research study. In the 1989 study, Davis used several charting applications as the subject of the acceptance model study. For this study, the paired and individual programming practices will be the primary constructs. To maintain validation of the original study, the charting application names in the Davis (1989) research were replaced by the term paired or individual programming practice. The fifth section introduced the measures for behavioral intention (to use) and actual usage (Rigopoulos & Askounis, 2007). This provided the necessary data for the last two formulas in the TAM calculation model (see Table 1).

The sixth section collected demographic and segmentation data. A test was provided to make sure the survey was being taken by a software development management professional. The first three response possibilities in Question # 31 provided cause for the survey not to be counted in the primary data gathering. These responses indicated that the respondent does not act as a software development manager, thus polluting any consolidated data. The responses in question # 30 provided a segmentation opportunity by type of business. These categories are consistent with the first 10, high-level divisions of the Standard Industrial Code (SIC) classification and provide the capability to co-relate data by business type to detect possible influences of a business type on the perceived ease of use and perceived usefulness of paired programming. The other questions in this section provide quality co-relative attributes to orient the data for further analysis.

Once the survey was completed, the respondent was asked if he or she would like a copy of the results. If the respondent wished a copy of the results, he or she was asked to provide an email address where an electronic copy could be shipped. The email captured in this email box was stored in a separate file from the survey data and was provided by the survey engine vendor in a distribution listing. This distribution listing was used to send a copy of the analysis, findings, and conclusions derived from this study. Upon the completion of this study and the final reporting and distribution of findings, this email file will be destroyed. At no time will the distribution listing be shared or sold to any entity.

Ethical Considerations

This study focused on the attitudes toward usage of software development managers toward the technology practice of paired programming, an element of the Extreme Programming methodology. The central point of the research in this study was to gather the attitudes and intentions of software development managers about paired and individual programming practices. The goal of the study was to predict the current and future use of paired and individual programming using The Technology Acceptance Model (TAM) as a theoretical framework. The data that was gathered focused on development manager's attitudes and intentions. This data was aggregated for both analysis and implementation into the TAM mathematical model. At no time during the study was there an identification process of the respondents and/or their respective businesses or any personal identification information collected or processed. The closest data field element for segmentation is a high-level business type segmentation category (10 categories at the division level) based on the U.S. Government Standard Industrial Code specifications. The study results yielded a scientific finding based on non-personal, aggregated data that placed no person or business at jeopardy or at risk. At all times the Code of Ethics (2008) from the Academy of Management and the Capella University Institutional Review Board (IRB), was used to guide the actions taken in this research and formed the basis of the ethical measures used to monitor those actions.

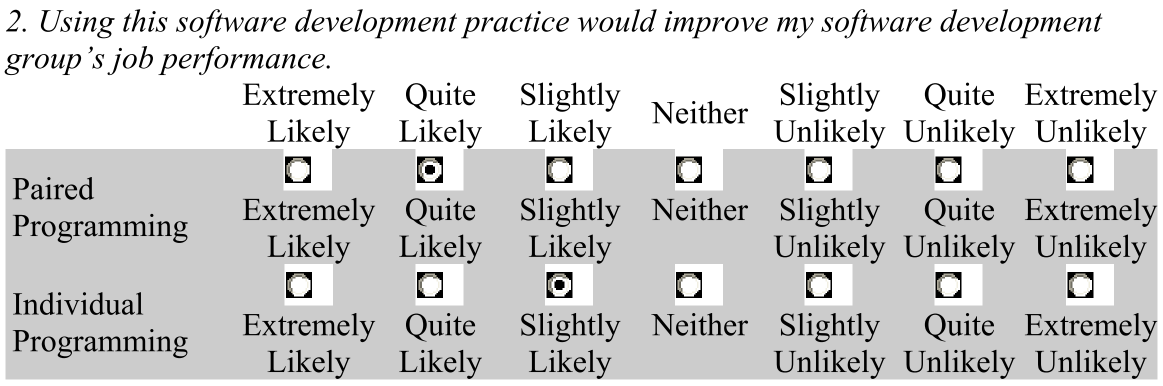
The results of this study might point to changing company policies, directions, or decisions in programming methods. The results may also suggest changes in philosophical as well as operational methods for software engineering practices. One of the reasons for this study was that business leaders and Information Technology Managers continue to question the value of paired programming, especially after its very enthusiastic introduction to the industry in 1999. Some businesses could view the results of this study as a reason to change their software development practices. In other cases, business and technology leaders might leverage this study to make critical organizational and financial decisions as well as techno-process oriented decisions. An underlying premise within this study is the recognition and encouragement of change and forward movement within the software development community, especially concerning programming practices that create more efficient and effective results for businesses. Through this study, scientific information and statistical analysis from industry-based field data can provide a result that may be used by software development managers for any business looking to improve its software development practices. This study takes all reasonable precautions to provide data and information in a random, non-biased manner. All elements of data gathering, retention, and analysis are open and demonstrative for easy and accurate replication and review.

CHAPTER 4. RESULTS

There are a number of factors that affect the use and adoption of any practice within software engineering. This is no different for the practice of paired programming. Software engineering practitioners continue to search for faster, more reliable coding methods. A significant amount of studies as demonstrated in Chapter 2, herein, reveal the industry's concern and preoccupation with improving these methods. Practices within these methods continue to reflect the need for more effective and efficient ways to make coding easier and more useful for developers. The scope of this study has been limited to a focus on paired programming and individual programming. Using The Technology Acceptance Model as a framework for executing a preliminary research study, this study set out to identify the correlations between key factors and provide data that might initiate and support a broader study into the effect of paired programming and its ability to support faster, flawless code engineering (Rigopoulos & Askounis, 2007).

The study was conducted via an Internet Survey engine as described in the section, *Using an eSurvey Engine,* in Chapter 3 above. Emails were sent inviting possible respondents to take the survey, engaged through a web-link in the invitation email. The actual survey questions and their format, found in The Appendix, were used exactly as they appear. However, in the eSurvey Engine, it was possible to stack the construction of the questions for paired programming and individual programming in a single question packet. This does not violate basic survey technique of asking multiple questions within a single question, but rather, provides an efficient method for collecting similar ideas applied to two variable constructs (see Figure 9). Each construct also had its own measurement collection line (scale), creating two single questions with two single data collection points. The result of this form of economizing the survey's real estate, helped to provide a cleaner look and feel as well as an instrument with only 14 question packets.

A Sample Question from the eSurvey



2. Using this software development practice would improve my software development group's job performance.

	Extremely Likely	Quite Likely	Slightly Likely	Neither	Slightly Unlikely	Quite Unlikely	Extremely Unlikely
Paired Programming	○ Extremely Likely	[✓ Quite Likely]	○ Slightly Likely	○ Neither	○ Slightly Unlikely	○ Quite Unlikely	○ Extremely Unlikely
Individual Programming	○ Extremely Likely	○ Quite Likely	[✓ Slightly Likely]	○ Neither	○ Slightly Unlikely	○ Quite Unlikely	○ Extremely Unlikely

Figure 9. This sample shows how a question from the eSurvey used in this study was displayed with two complete questions and measurement lines within one question packet.

Survey Response Results

Email listings were obtained from *BizExUSA.Com* in block amounts of 500 non-opt-in email addresses. These emails were sorted for the general value of software development lead, supervisor, manager, director, vice-president, CTO, CIO, and President of a software development or applications development organization. The related possible respondent pool based on this sort was 47,351. A test of 500 emails revealed that approximately 62.3% of these emails were filtered through a Network Security Scan (NSS) as possible spam and 22.6% were no longer valid emails as reported by the user's SMTP server. 4,500 emails were acquired (the original 500 used for testing email passage were not included). Of the 4,500 emails, 1,131 appeared to pass through filtering and be validly constructed and representative emails. Of the emails sent to 1,131 valid email addresses, 212 responses were received during the three-week survey period. The initial response rate was 18.7%. 49 surveys were not satisfactorily completed (more than three questions from each of the sections were not completed), seven surveys were returned blank, and 53 were completed by non-software development management respondents (see question 31 in The Appendix). 103 surveys met the research criteria and were satisfactorily completed. This resulted in a final survey response rate of 9.12%.

Analyzing the Response Rate

The response rate based on the original e-listings was hampered by the randomness of the survey design and non opt-in of the list population. An opt-in is the consent by possible respondents to have their email address used in marketing or survey listings, usually for the purpose of a particular subject inquiry. This means that the respondents would need to have known about the subject and made a decision at some point to participate in general surveys associated with questions or subjects tuned to software development management. To provide a sample that was random and not already tainted toward one of the constructs in the survey, it was necessary to use a marketing listing that filtered only title. There were no listings found that provided an opt-in for possible e-survey respondents open to completing questions on software development practices. Of the 103 respondents to the survey, 49 requested follow-up information on the results of the survey and 16 responded to the delivery email address (sender email) with specific questions or request for additional information on the subject of paired programming. This indicated that those responding to the survey possessed a genuine interest in the subject, lending credibility to their input through the survey.

Validating the Sample

It is important to question the validity of the sample as a primary step in the analysis of the research data. This was done by first considering the confidence level. This is usually 95% for similar research projects (Creative Research Systems, 2010). Next, the confidence interval was computed based on the sample size and population with a percentage margin of error. For the research project described herein, the margin of error was set conservatively at 50%. This yielded a confidence interval of 9.21%. When calculated with a population of 1130 complete responses, this yielded a valid sample rate, considering the criteria above, to be a minimum of 103 responses. The formula used to calculate the sample size and confidence interval is represented by $SS = (Z^2 \text{ X } p \text{ X } (1 - p)) / C^2$. Where Z is the value of 1.96 to achieve a confidence level of 95%, p is the percentage of picking a choice, and C is the confidence interval of 9.21%, which is a range that is + or – the value of that number. (Creative Research Systems, 2010). The actual received samples were above 103, thereby confirming the validity of the sample size used in this research study.

Additional Considerations beyond The Technology Acceptance Model Survey Data

Prior to analyzing the data through the Davis' Technology Acceptance Model, it is important to understand the relationship of perceived usefulness and perceived ease of use on actual usage as influenced by individual differences. This is basic to the survey that collected the data in this research study and the foundation of the TAM formulae that will provide a comparative view of usage for the practices of paired and individual programming. When measuring responses based on individual attitudes and projecting them into behavior in the present and predicted for the future, there are various psychological and sociological issues involved in perceived usefulness and perceived ease of use that must be understood in the interpretation of the data. Three possible influencing elements should be considered when analyzing individual responses to the TAM survey.

Three Influencing Considerations on Individual Response Differences

According to Ajzen (1985, 1991), when a person's behavior is not completely voluntary, the perceived behavioral controlling element directly and strongly affects the person's intentions to behave (and use), over their attitudes and other subjective influences. Venkatesh and Davis (2000) indicated that perceived ease of use tends to cancel this out or at least mediate it. Other studies involving Information Technology practices tend to support the direct effect on individual differences of perceived behavioral control on the outcome of actual usage and/or predicted use (Igbaria, Guimaraes, & Davis, 1995; Mathieson, Peacock, & Chin, 2001; Taylor & Todd, 1995).

Another consideration as to why perceived usefulness and perceived ease of use may only partially affect individual differences with regards to actual usage is that behavior can be linked to self-identity and this can be a strong driver of behavior. A person's perception or self-identity of themselves as being in a particular camp or within a particular named group directly affects their intentions, even beyond attitudes, subjective norms, or various types of behavioral control (Sparks & Shepherd, 1992). Multiple studies confirm this condition (Ajzen, 1991; Armitage & Conner, 1999; Sparks & Guthrie, 1998). An important study pertinent to technology found that a person's self-identity was directly linked to his or her usage of particular technologies and practices (Speier & Venkatesh, 2002). This study did not use the TAM as a theoretical model, diluting a more direct comparison and conclusion. Nevertheless, there is sufficient evidence that indicates above one's intentions, subjective norms, and behavior, the forces of self-identity may play a substantive role in usage with only superficial effects from perceived usefulness and perceived ease of use (Burton-Jones & Hubona, 2005).

This concept is strongly relevant to this study as practitioners of paired programming may, as indicated in near-term history, follow the *New Age* of programming practices linked to popular/current methodologies such as Extreme Programming. The momentum of those popular methods, giving way to popular practices may have influenced software development managers to identify with being practitioners of new methods and practices. The effect of perceived usefulness and perceived ease of use may have a lesser influence with regards to usage. The survey helps to mitigate this influence as the social and professional subject norms are reduced through the anonymous execution of the instrument and its data collection.

Assumptions in the TAM methodology are that behavior is energized through perceived usefulness and perceived ease of use based on intentions and attitudes about a thing or practice. Some researchers hold that behavior is more a condition of habitual occurrence than intentional occurrence, where habits directly affect behavior and usage beyond the influences of attitudes, intentions, behavioral control, and subjective norms (Oullette & Wood, 1998; Ronis, Yates, & Kirscht, 1989). Venkatesh and Davis (2000) found this to be the case for technology. In their study, it appeared that after three months of using a tool or a practice, the best predictor of usage was an individual's prior use. As this study was to look for other influencing elements other than perceived usefulness and perceived ease of use, these constructs were not a part of that study and were not compared to other elements such as attitude, subjective norms, and behavior control. It is still important to note the effect of habit on individual differences as measures toward current and future usage.

Critics of the TAM noted that when considering human intentions, attitudes, and influencing external norms, there can be no exact science to predict behavioral acceptance and usage. The above three elements seem to support that the methodology has limits and that there are constraints that must be considered in a full analysis of the data. The ability to predict a finite conclusion of acceptance or lack of acceptance is not attempted here, nor is it in the many cases of the application of the TAM as a framework to determine usage. Behavioral acceptance and usage is a continuum of scale and not a logic gate of decision. This must be considered in the presentation and analysis of the data presented.

Primary Hypotheses Analysis

An internet eSurvey study was conducted (see The Appendix) in order to test the research model and confirm or reject the null hypotheses. A web-based questionnaire was accessed by randomly invited software development managers. That questionnaire was based on the constructs depicted in Table 5 and used to collect various points of data in accordance with Davis' (1989) Technology Acceptance Model. The questionnaire used a Likert scale ranging from 7 = extremely likely to 1 = extremely unlikely. The two major constructs were that of paired programming and individual programming. Using the Davis (1989) survey model for the TAM, the variables engaged in both constructs included Perceived Usefulness, Perceived Ease of Use, and self-reported usage. A shortened version of the survey questions are provided in Table 5.

Statistical Analysis

Table 5. Construct Items Applied to the Practice of Paired and Individual Programming.

The Construct Items
<u>I. Perceived Usefulness</u>
1. Using this practice would enable quicker software development
2. Using this practice would improve developer's job performance
3. Using this practice would increase developer productivity
4. Using this practice would enhance effectiveness of producing code
5. Using this practice would make coding easier for programmers
6. Developers would find this practice useful in software development
<u>II. Perceived Ease of Use</u>
1. Using this practice would enhance the efficiency of producing code
2. Learning and implementing this practice would be easy for my programmers
3. Programmers would find this practice easy in producing desired code
4. Implementing this practice would be clear and understandable to programmers
5. Programmers would find this practice to be flexible and easy to use
6. Programmers would find it easy to become skillful in this practice
<u>III. Behavioral Intention to use</u>
1. Programmers would find this type of practice easy to use
<u>IV. Usage</u>
1. I intend to use this practice for my development group

Hypotheses Analysis for Perceived Ease of Use

The first hypothesis to be reviewed proceeds from the research question that asks if there are observed differences in perceived ease of use (PEU) between the practice of paired programming and the practice of individual programming (R1)? Restated, the hypothesis (Ho1) indicates that the mean score for the perceived ease of use construct will not be significantly different for paired programmers versus individual programmers working alone. To test this hypothesis, the results of the mean value produces a 95% confidence factor that will accept the null hypothesis as correct if the Z calculation is less than two. If the Z calculation produces a value greater than 1.96, (more than 1.96 standard deviations of separation) then the null hypothesis is rejected.

Table 6. Mean TAM Score Between Paired and Individual Programming for PEU.

Paired Programming	Mean
1. Learning/implementing is easy	4.18
2. Easy in producing desired code	4.62
3. Clear and understandable	4.54
4. Flexible and easy to use	4.17
5. Easy to become skillful	4.70
6. Practice that is easy to use	4.33
Average Mean	4.42***
Individual Programming	Mean
1. Learning/implementing is easy	5.46
2. Easy in producing desired code	5.15
3. Clear and understandable	5.70
4. Flexible and easy to use	5.53
5. Easy to become skillful	5.55
6. Practice that is easy to use	5.54
Average Mean	5.49***

***p<.001 **p<.01 *p<.05

The mean average for Paired Programming is 4.42 compared to the Individual Programming mean average of 5.49, a 19.41% difference between the mean variables. The Pearson correlation coefficient of -1.59 indicates that the relationship between the variables is somewhat inversely related (when one gets larger, the other gets smaller). This is consistent with those that form perceived ease of use judgments for Individual Programming over Paired Programming. Although the possibility of the measure could be equal, (the respondent could like both), the data reflects otherwise as those selecting individual programming clearly relegate the selection of paired programming to a lesser status. Variations in the questions for each construct prove to be tight or highly consistent amongst themselves with a standard deviation of 0.18 for the individual practice and 0.23

for the paired practice. Individual practice scores are leptokurtic or pile up to a peak value, whereas the paired practice scores are platykuric or more evenly distributed. This indicates that the questions are correctly related in tapping perceived ease of use, but the evidence shows there is a preference toward the individual practice area more so than in the paired practice area. Skewness also shows a marked disparity between individual programming and paired programming data.

A Z parametric test was run to determine the statistical significance between the distributed means. The p value showed significance between the variables with a value of 0.0326. The analysis indicated that there was a 19.41% variation between the variables mean values with a Z value of 1.845. At the same time, the p value confirmed the significance of the variables' relationship to each other. Since the criteria to confirm the null hypotheses was that there was no significant difference between the variables, and the calculated Z value was less than 1.96. This fails to reject hypothesis Ho1, in other words, there is no significant difference in software development managers' perceived ease of use for the individual programming practice over the paired programming practice.

Hypotheses Analysis for Perceived Usefulness

The second hypotheses attempts to address the second research question (R2), which asks if there are observed differences in perceived usefulness (PU) between the practice of paired programming and individual programmers working alone. The hypothesis Ho2 indicates that the mean score of the perceived usefulness construct will not be significantly different for paired programmers versus individual programmers working alone. To test this hypothesis, the results of the mean value produces a 95%

confidence factor that will fail to reject the null hypothesis if the Z calculation is less than 1.96. If the Z calculation produces a value greater than 1.96, (more than 1.96 standard deviations of separation) then the hypothesis Ho2 is rejected.

Table 7. Mean TAM Score Between Paired and Individual Programming for PU.

Paired Programming	Mean
1. Efficiency producing code	4.60
2. Enable quicker software development	5.04
3. Improve developer's job performance	4.55
4. Increase developer productivity	5.04
5. Enhance effectiveness producing code	4.75
6. Make coding easier for programmers	4.75
Average Mean	4.76***
Individual Programming	**Mean**
1. Efficiency producing code	4.74
2. Enable quicker software development	4.35
3. Improve developer's job performance	4.54
4. Increase developer productivity	4.25
5. Enhance effectiveness producing code	4.55
6. Make coding easier for programmers	4.97
Average Mean	4.56***

***p<.001 **p<.01 *p<.05

The mean average for Paired Programming is 4.76 compared to the mean average of Individual Programming of 4.56. The Pearson Correlation Coefficient of -0.59 indicates that the relationship between the variables is somewhat inversely related (when one gets larger the other gets smaller), but at a moderate to slow rate. In this case, the movement is toward the Paired area. The variance between the constructs is small: 0.02, indicating a low degree of significant variance. This indicates that both variables would be considered useful.

Variations in the questions for each construct prove to be tight or highly consistent amongst themselves with a standard deviation of 0.002 between individual programming and paired programming. The individual scores are leptokurtic or pile up to a peak value, whereas the paired scores are platykuric or more evenly distributed as described above. This indicates that the questions are correctly related in tapping perceived usefulness, but the evidence shows there is a more evenly distributed value or preference for specific selections in the Paired area more so than in the individual area, but only slightly. Skewness also shows a minimal disparity between Individual and paired practice variables. There is a 4.26% variance between the two constructs with the paired practice area showing a trend toward a more asymmetrical distribution (see *Statistical Tests Linked to Hypotheses* section in Chapter 3 above).

A Z parametric test was conducted to determine the statistical significance between the distributed means. The p value shows a low but acceptable significance relationship between the variables with a value of 0.0433. The analysis indicated a Z value of 1.7141. At the same time, the p value of .0432 confirmed the moderate significance of the variables' relationship to each other. Both results for perceived ease of use and perceived usefulness were independently regressed. The criteria to fail to reject the null hypotheses were that there was no significant difference between the variables, and the calculated Z value was less than 1.96. This fails to reject the null hypothesis (Ho2); in other words, there is no significant difference in software development managers' perceived usefulness for the individual programming practice over the paired programming practice.

The literature indicates that in past studies there was a bias for usefulness among more seasoned professionals. Their tendency was to see usefulness as a tool in itself, and not necessarily in comparison with something else. This would be particularly applicable to software managers, seasoned in the field of applications development, judging the usefulness of paired versus individual programming. In this case, the paired and individual programming practices appear to be considered useful. Although the possibility of the measure could always be equal, the result indicates that in this construct it is very close.

Hypotheses Analysis for the Correlation of Perceived Usefulness and Ease of Use

The third hypothesis attempts to address research question (R3), which indicates to what extent is the paired programming practice more acceptable/used than the practice of individual programmers working alone. The hypothesis (Ho3) indicates that the paired programming practice will not be significantly perceived to be more used by software development managers than the individual programming practice. This is determined through a correlation (of constructs) between the mean scores of the constructs (see Tables 8 through 9) of perceived usefulness, perceived ease of use, behavioral intention to use, and self-reported usage; as well as a regression analysis of the effect of perceived usefulness and perceived ease of use on self-reported usage (see Tables 10 and 11).

The TAM formula as shown in Table 1 is a simple manipulation of the correlation coefficients between perceived ease of use, perceived usefulness, behavioral intention to use, and self-reported usage (Davis, 1985, 1989; Rigopoulos & Askounis, 2007). Davis, Bagozzi, and Warshaw (1989) and Sheppard et al (1988), noted that the addition of behavioral intention or the self-prediction of use was a significant predictor of future behavior. Studies by Burton-Jones and Hubona (2005), and Rigopoulos and Askounis (2007), among others, made a distinction between the psychometrics of behavioral intention and self-predicted usage. These variables were represented in their survey forms as separate questions. The behavioral use values and those of self-predicted usage bond to form a strong predictor of future use. The statistical models below reflect that bond, while demonstrating the correlational constructs (see Table 10). The paired programming practice had a non-significant correlation between usefulness and ease of use (0.0136, *p*<.01). For paired programming, usefulness was moderately correlated to intent to use (0.489, *p*<.03) but was not significantly correlated to usage (0.013, *p*<.06). Ease of use was significantly correlated between intent to use (0.637, p<.0001) but was not significantly correlated to usage (0.050, *p*<.01). Actual usage showed a statistically significant correlation (0.647, *p*<.02).

Table 8. Correlation of Constructs for Paired Programming

Constructs	(1)	(2)	(3)	(4)
(1) Perceived Usefulness	1	.014	.489	.413
(2) Perceived Ease of Use	.014	1	.638	.050
(3) Behavioral Intention to Use	.489	.638	1	.647
(4) Self-Reported Usage	.014	.050	.647	1

The TAM Research Model Applied for Paired Programming (see Figure 10) demonstrates the correlational values (with *p* values exposed) for the construct of paired programming. The internal relational values explained above support the resultant value of a significant correlation between actual usage and the variables of perceived ease of use, perceived usefulness, behavioral intention to use, and self-reported usage. The data suggests that there are strong correlational ties between perceived usefulness and usage, with less of a tie between ease of use and usage. The model indicates that there is a moderately strong correlation for actual usage (0.647, *p*<.02) of the paired programming practice by the data from software development managers.

The TAM Research Model with Correlation Values for Paired Programming

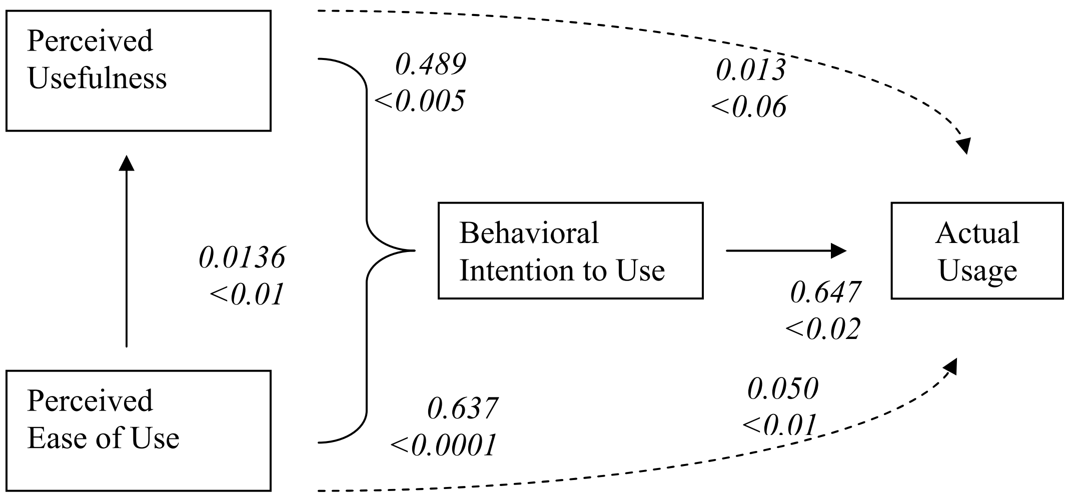


Figure 10. The TAM research model with paired programming correlation values. This graph shows how the various values derived from the survey data are applied to the TAM to form a measure of actual usage. Adapted with permission from *A TAM framework to evaluate users' perception towards online electronic payments* (p. 5), by G. Rigopoulos, and D. Askounis, 2007. *Journal of Internet Banking and Commerce, 12*(3). Copyright 2007 by the Journal of Banking and Internet Commerce.

The TAM Research Model applied for Individual Programming (see Figure 11) demonstrates the correlational values (with *p* values exposed) for the construct of the individual programming practice. The internal relational values explained above support the resultant value of a significant correlation between actual usage and the variables of perceived ease of use, perceived usefulness, behavioral intention to use, and self-reported usage. The data suggests somewhat different results to that of the paired programming practice construct.

Usefulness and ease of use showed a moderately significant correlation (0.289, *p*<.0001). There is a stronger correlation between usefulness and intent to use (0.262, *p*<.03) and a significant correlation between usefulness and usage (0.542, *p*<.005). There is a significant correlation between ease of use and intent to use (0.845, *p*< 0001) and a non-significant correlation between usefulness and usage (0.037, *p*<.005). The model indicates that there is a significantly strong correlation between intent to use and actual usage (0.954, *p*<.0001)

Table 9. Correlation of Constructs for Individual Programming

Constructs	(1)	(2)	(3)	(4)
(1) Perceived Usefulness	1	.289	.262	.542
(2) Perceived Ease of Use	.289	1	.845	.037
(3) Behavioral Intention to Use	.262	.845	1	.954
(4) Self-Reported Usage	.542	.037	.954	1

The comparison of the paired and individual programming practice models indicates that the individual programming practice was predicted to be more used than the paired programming practice. Although the correlational variance is only moderate (.307) in favor of individual programming, it is nonetheless apparent that there is more confidence in the future use of the individual programming practice than the paired programming practice.

Research Model with Correlation Values for Individual Programming

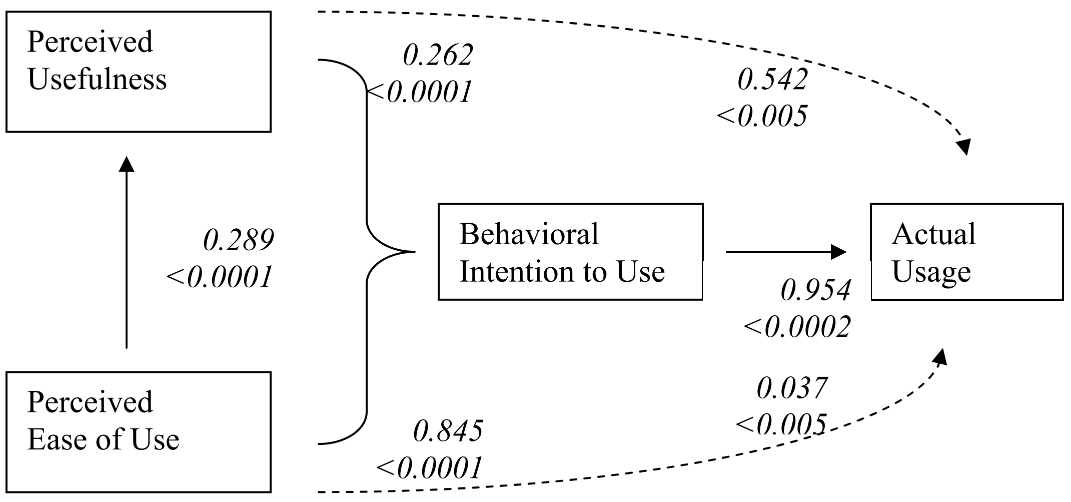


Figure 11. The TAM research model with individual programming correlation values. The graph shows actual usage survey data applied to the TAM. Adapted with permission from *A TAM framework to evaluate users' perception towards online electronic payments* (p. 5), by G. Rigopoulos, and D. Askounis, 2007. *Journal of Internet Banking and Commerce, 12*(3). Copyright 2007 by the Journal of Banking and Internet Commerce.

The criteria for rejecting or failing to reject the hypothesis is reflected in Chapter 3, *Statistical Tests Linked to Hypotheses*. The values found in Figure 10 and 11 are summarized in Table 10. The results, with a 95% confidence factor, would fail to reject the null hypothesis if the usage calculation was less than 1.96. If the calculation produces a value greater than 1.96 (i.e. significantly perceived to be more used – a value of more than two standard deviations of separation), then the null hypothesis would be rejected.

Table 10. Correlations Between Perceived Usefulness, Perceived Ease of Use, and Self-Reported Usage

	Correlation			
	Usefulness & Usage	Ease of Use & Usage	Ease of Use Usefulness	Pooled Usage
Paired Practice	.013	.050**	.014*	.647*
Individual Practice	.542**	.037**	.289***	.954***

***p<.001 **p<.01 *p<.05

The null hypothesis, Ho3, indicated that the paired programming practice would not be significantly perceived to be more used by software development managers than the individual programming practice. The TAM model resulted in a value for the paired programming practice of 0.647 ($p<.05$) and a value for the individual programming practice of 0.954 ($p<.001$). Paired programming was not perceived to be more used, as the data indicated that it was actually less used than the individual programming practices by a moderate amount. Although there is a slight difference between the usage values, favoring the individual practice, the criterion to fail to reject the null hypothesis is upheld. Conceptually this is consistent with the opportunity of choice by the respondents for both practices. The data supports a stronger perceived usefulness for the individual programming practice, but only by a non-significant margin.

Regression Test to Confirm the Effects of Usefulness and Ease of Use on Usage. In Davis' study (1989) applied the TAM to charting software use and then used regression analyses to confirm the effect of perceived usefulness on usage, when ease of use is controlled or limited. Davis' significance values were stronger in his study compared to this study (see Table 11). Here the paired programming practice for usefulness shows a value of 0.50 and an ease of use value of 0.69 with no significance. For individual programming, usefulness shows a value of 0.80 and an ease of use value of 0.88. Both are moderately significant. The regression coefficients calculated for the paired practice and the individual practice were not significantly different (0.25 and 0.15 respectively). This confirms Davis' findings, which found that usefulness tends to mediate the effects of ease of use when applied to usage. Considered in another way, usefulness contains the behavioral intentions of ease of use.

Table 11. Regression Analysis on the Effect of Usefulness and Ease of Use Usage

	Independent Variables		
	Usefulness	Ease of Use	R^2
Paired Programming	.50	.69	.24
Individual Programming	.80**	.88*	.12

***p<.001 **p<.01 *p<.05

Davis pooled the values of behavioral intent to use and self-reported usage. When this was done to the data in this study, the values were only slightly moderated by less than 2%. In this study, behavioral intent to use and self-reported usage was separated to discern better-detailed values following the study methods of Rigopoulos and Askounis (2007). The values for regression do not alter the analysis for the confirmation or rejection of the hypothesis, but rather help to confirm the value of the TAM model for determining usage from behavior intent (Davis, 1989).

Hypotheses Analysis for the Effect of Business Type on Usage

The fourth hypothesis addressed the research question (R4) that asked how the type of business relates to the acceptance and usage of the paired programming and individual programming practices. The hypothesis (Ho4) indicated that there would be no statistically significant mean difference between a software development managers' (respondent's) type of business and the level of acceptance and usage for both the paired programming and individual programming practices. The test criterion for this hypothesis was the measure of variation in the calculated standard deviation values between the variables and the mean standard deviation between each of the constructs. The criteria or condition that fails to reject the null hypothesis was if the usage standard deviation calculation between the variables (business types) was less than 1.96 and/or if the mean usage standard deviation calculation between the two constructs (paired and individual) was less than 1.96. If the calculations produced a value that was greater than 1.96, (i.e. two standard deviations of separation with significant variation between the variables and/or constructs) that condition would reject the null hypothesis.

Table 12. Correlational & Mean Difference Tests for Business-Type Analysis

Usage for:	Paired Mean	Individual Mean	Var	Standard Deviation
Wholesale Trade	6.00	4.62	0.885	0.941
Retail Trade	5.71	4.29	1.008	1.004
Finance & Insurance	5.71	4.29	1.008	1.004
Manufacturing	4.44	6.33	1.786	1.336
Service Industry	4.10	5.69	1.264	1.124
Transportation & Comms	5.50	5.00	0.125	0.353
Public Admin & Government	5.50	5.00	0.125	0.353

There was no significant difference in usage for most of the business types in either paired programming or individual programming (see Table 12). The Wholesale Trade type showed a slight usage variable difference in favor of paired programming (variance = 0.885). Manufacturing showed a measurable usage variable difference in favor of individual programming (variance = 1.786) as well as the Service Industry showed a measureable usage variable difference (variance = 1.264). Yet the mean standard deviation did not vary significantly between paired and individual practice for all business types in each construct (0.717 and 0.747 respectively). The analysis indicated there were no other significant departures or variations from the constructs and the relationship with the respondents' business type.

An ANOVA test and a correlational test were conducted for each of the business types. The values of change and variation were consistent among the business types, with the exception of manufacturing and the service industries. These business types showed a higher source of variation between the paired programming and the individual programming practices, with values favoring the individual programming practice. Correlational analysis indicated that between the business types for paired programming there was a standard deviation of 0.717 with (a low) significance (p value) of 0.997, between paired and individual practices. For individual programming, the standard deviation among the business types was 0.7474 with (a low) significance (p value) of 0.6673, between the business types for both paired and individual practices.

Table 13. ANOVA Between Paired and Individual Usage and Business Types

ANOVA Summary	Count	Sum	Average	Variance
Wholesale Trade	2	10.67	5.335	0.885
Retail Trade	2	10.0	5.0	1.001
Finance & Insurance	2	10.0	5.0	1.001
Manufacturing	2	10.77	5.385	1.786
Service Industry	2	9.79	4.895	1.264
Transport & Comms	2	10.5	5.25	0.125
Public Admin/Gov	2	10.5	5.25	0.125
Paired Usage	7	36.96	5.28	0.514
Individual Usage	7	35.27	5.039	0.559

Source of Variation	SS	df	MS	F	P-value	F crit
Rows	0.438	6	0.073	0.073	0.997	4.284
Columns	0.204	1	0.204	0.204	0.667	5.987
Error	5.997	6	0.999			
Total	6.639	13				

Given the non-significant data of only a 0.031 variation in standard deviation between the constructs and no value greater than two standard deviation points between any of the business type variables, it appeared there was no significant difference in the means. This was despite some indication that at least two business types (and possibly more) have some minor differences in favor of the individual programming practice. An ANOVA test (see Table 13) indicated that the critical value (*F*-value) was less than the critical test value (4.283 < 5.996). This fails to reject the null hypothesis and concludes there are no statistically significant differences between the pair of means. Additionally, the *p* value (P<.667) was more than the significance level (α = .05), providing another confirmation of the (Ho4) null hypothesis (Snedecor & Cochran, 1967).

Hypotheses Analysis on the Effect of Software Development Manager Experience on Usage

The fifth hypothesis addressed research question (R5), that asked if there is a relationship between a software development manager's development experience (in years by range) and their acceptance and use of the paired programming and individual programming practices. The null hypothesis (Ho5) indicated that there was no statistically significant mean difference or variation between a software development manager's development experience (in years) and the level of their acceptance and usage of the paired programming and individual programming practice.

The literature provides an understanding of paired programming as a practice that is representative of a new arena of software development. As a part of the Agile programming movement, the Extreme Programming methodology challenges legacy forms of standard or individual programming, providing various alternatives to producing cleaner code at a more rapid and accurate rate. The paired programming practice is just one of many new alternative tools within the application engineering area. Whether these new practices are used may have a direct relationship with the experience level of the manager setting the style and method of software engineering. The question remains as to whether a manager's programming experience influences the usage of the paired programming practice as compared to the individual programming practice.

Table 14. Variance & Mean for Usage and Respondent Experience Level

Respondent Experience	Paired Mean Usage	Individual Mean Usage	Mean Difference	Variance
<1 to 2 Years	5.00	6.25	1.25	0.78
3 to 6 Years	5.70	5.50	0.02	0.02
7 to 10 Years	5.05	5.38	0.33	0.05
11 to 14 Years	4.71	5.50	0.79	0.31
15 to 18 Years	4.00	5.78	1.78	1.58
> 18 Years	4.05	5.59	1.54	1.19

The statistics in Table 14 account for behavioral use and actual self-reported usage for both paired programming and individual programming. The view is two-dimensional: a) Mean difference between the layers of experience in years (left column), and b) The mean difference between paired and individual programming practices. The criterion for the test of the hypothesis was whether a statistically significant variation occurs between either one or both strata. Through an ANOVA test, the calculated value (*F* value) was compared to the critical test value (significance level [p<.05] and degrees of freedom). If the calculated value is greater than the critical value the null hypothesis is rejected and a conclusion that there is a significant statistical difference between the pair of means (Snedecor & Cochran, 1967).

It appears for paired and individual comparisons that although there is some variation in the first years of programming experience, there is only a slight mean variance (0.59) in the median year brackets. Between the total program experience values, the mean difference is no greater than 1.58 and the variation is no greater than 1.56. The mean difference between the experience levels shows an increasing variance departure in the experience year brackets of 15 to 18 years and >18 years. The variance skews toward acceptance of individual practice over paired practice in these brackets by over 70%. This can simply be explained as those with deeper backgrounds in individual programming felt more comfortable (perceived ease of use) and have had the opportunity to have successful engagements through that practice (perceived usefulness). They have formed behaviors that return to that practice as a way of actually producing software (behavioral usage).

Table 15. ANOVA Between Paired and Individual Programming Experience

ANOVA Summary						
Groups	Count	Sum	Average	Variance		
Paired Experience	6	28.51	4.751667	0.421817		
Indiv. Experience	6	34	5.666667	0.099347		
ANOVA Source of Variation						
	SS	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	2.511675	1	2.511675	9.638725	0.011162	4.964603
Within Groups	2.605817	10	0.260582			
Total	5.117492	11				

Since the calculated value (see Table 15) is greater than the critical value (9.6387 > 4.9646) and the *p* value equals .0112, which is less than the significance level threshold (*p*<.05); the null hypothesis (Ho5) is rejected and the conclusion is that there is a statistically significant difference between the pair of mean usage for paired and individual programming.

A Review of Data Analysis and Hypotheses Support

A review of the support for or lack of support for the null hypotheses, in this study, is found in Table 16. The rationale of the research was generally to determine if paired programming was or could be predicted to be used *more* as a software programming practice than the individual programming practice. To understand the term *more,* a statistically significant difference or variance between the responses for paired programming and individual programming was sought. The data applied to statistical tests against the five hypotheses indicated some difference (see Table 13 and 14), but in many cases, not statistically significant.

Table 16. Review of Hypotheses Conclusions

Tag	Hypothesis	Result
Ho1	The mean TAM score of the perceived ease of use construct will not be significantly different for paired programmers versus individual programmers working alone.	Failed to reject
Ho2	The mean TAM score of the perceived usefulness construct will not be significantly different for paired programmers versus individual programmers working alone.	Failed to reject
Ho3	The paired programming practice will not be significantly perceived to be more used by software development managers than the individual programming practice.	Failed to reject
Ho4	There will be no statistically significant mean difference between a software development managers' (respondent's) type of business and the level of acceptance and usage for both the paired programming and individual programming practice.	Failed to reject
Ho5	There is no statistically significant mean difference or variation between a software development manager's development experience (in years) and the level of their acceptance and usage of the paired programming and individual programming practice.	Rejected

The literature review in Chapter 2 outlines a significant positioning of paired programming as a major tool in software improvement practices. The data does not support the excitement expressed in the literature. The data also indicates that paired programming is a practice that is used, even though not necessarily, as much in some cases as individual programming. The significance of this finding is that paired programming has already taken a place in the list of practices software development managers use to engineer software. The data suggests that paired programming has not taken a lead role over individual programming yet, contrary to much of the literature currently available and despite studies indicating that using the paired programming practices yields more efficient and effective results compared to individual or waterfall type programming practices. A better understanding of this finding is described in Chapter 5.

CHAPTER 5. DISCUSSION, IMPLICATIONS, RECOMMENDATIONS

The objectives of this research study were to focus on providing an empirical basis on which practitioners, namely software development managers and business leaders with engaged software development resources, might decide to adopt or not adopt the practice of paired programming as a software development practice. Sheil (1981) stated the foundation of this study's nature clearly, "Most innovations in programming languages and methodology are motivated by a belief that they will improve the performance of the programmers who use them" (p. 101). The research within this study attempts to answer the question as to whether the belief that paired programming as a practice within the Extreme Programming methodology, is perceived to improve the performance of programmers through its usage by software development managers as a practice of choice in their development practice environments.

Discussion

A review of the literature indicates that there has not been even a moderate amount of study on the Agile software methodologies, specifically Extreme Programming and its practices. Further, those studies in paired programming and the associated literature provided little in structured methodology to ascertain the predicted behavior and/or usage by those contemplating its employment in their organizations. Bahli and Zeid, (2005), indicated The Technology Acceptance Model (TAM) created by Davis' (1989) provided a structured methodology in which to study Extreme

Programming. Their study reflected the move and experiences of a Canadian company as it moved from the Waterfall or Individual programmer methodology to the Extreme Programming methodology. They raised the question and called for further research in this area of programming practices. The activities in this study have focused on the next iteration on that line of research by applying the TAM to the practice of paired programming, in an attempt to understand the possible usage values between this practice and that of the practice of individual (waterfall) programming.

Davis' study conducted two experiments, one in an internal lab setting using respondents familiar with the practices being measured. The other was external using random surveys and responses. The data was compared, correlated, and even pooled to discern the relative strength of the TAM as a tool. Davis determined that the external measures were stronger for external data collection. This coincides with questions concerning related subject research in Extreme Programming, specifically in studies related to paired programming. Many of the studies conducted were done in laboratory settings with graduate students familiar with the practice. Random practitioner metrics were not always used, lending a somewhat questionable result, especially to critics of the practice. A random field study of actual practitioners was needed to take the next steps as a contribution to the body of knowledge on Extreme Programming practices and in determining if paired programming is a behavior that is or might be used by software development managers (practitioners), especially when compared to the possible use of individual programming practices.

Review of the Findings

The first hypothesis considered the effect of The Technology Acceptance Model construct, perceived ease of use, between the paired programming and individual programming practices. The data suggested that the perceived ease of use construct was relevant to both practices. The data also suggested that there was no statistically significant difference between the practices, although there were slight variations favoring the individual programming practice. This is counter to the literature that indicated the paired programming practice was easier and possibly more popular to use than traditional waterfall or individual programming practices. It does not appear this is true for managers that control the practices used in their software engineering environments. Even though there is a slight difference in the correlation between paired and individual programming practices, both values show no significant difference. This strongly suggests that software programming managers feel that they can use both practices, but favor the individual programming practice.

The second hypothesis considered the effect of The Technology Acceptance Model construct, perceived usefulness, between the paired programming and individual programming practices. The data suggested that the perceived usefulness construct was relevant to both practices. The data also suggested that there were no statistically significant differences between correlations of the practices. Davis indicated that, "One of the most significant findings is the relative strength of the usefulness-usage relationship compared to the ease of use-usage relationship" (1989, p. 333). The data confirms this, as perceived usefulness is strong for both constructs with very little variance between them. Yet in a choice of paired versus individual programming practices, the data indicated that programming managers found both practices useful with a slight tendency to favor the individual programming practice over the paired programming practice. Despite the strong sense of programming capability for the paired programming practice reflected in the literature, there appears to be no over-riding emphasis by programming managers toward that practice over the individual programming practice.

The third hypothesis considered the effect of The Technology Acceptance Model framework and structured methodology between the paired programming and individual programming practices. This framework correlated the values of the constructs of perceived ease of use and perceived usefulness for both practices as well as self-reported usage. The data suggested that this framework was relevant to both practices. The data also suggested that there was a positive usage relationship between both perceived ease of use and perceived usefulness, especially with the individual programming practice. There was a slight to moderate favor toward the individual programming practice by respondents, than toward paired programming. This was despite the promotion of paired programming as a more efficient and effective practice indicated by the literature.

The fourth hypothesis considered the correlational effect of The Technology Acceptance Model output of usage for both paired and individual programming practices and their relationship to the respondent's business type. The output of the TAM when correlated with the values of usage and the business type of the respondents indicated that there was no statistically significant difference between the usage values and the business type. There was also no statistical difference between the mean values of business types and the two practices. Upon further investigation, there appeared to be some slight variances between the business type of Manufacturing (toward individual programming) and Wholesale Trade (toward paired programming). The data suggested that there is a slight effect of the business type to usage. The effect is not significant and the variations fall within recognized business types that coincide with one particular programming style historically. For example, when the forms of programming used for manufacturing are made up of complex, structured applications usually engaged through waterfall-type and/or highly structured programming processes. Conversely, Wholesale Trade is made up of rapidly changing applications, especially web based applications that range from point of sale processing to electronic payment applications. These forms of applications are more related to the iterative style of the paired programming practice. Overall, the data suggest there is no appreciable link between the respondent's type of business and their usage of paired or individual programming practices.

The fifth hypothesis was based on a correlation of the relationship of the development managers' programming background with their response to usage for both paired and individual programming practices. It is assumed that software development managers carry their experience forward from the beginning of their programming experience to the point where their managerial duties influence designating a software practice in their areas of influence. The data suggested that experience did play some slight role in the influence of using individual programming for those respondents with 15 years of experience and more. There was no statistically significant difference for that period or for the other periods as well. This seems to be consistent with the logical connection that Agile methods, especially Extreme Programming and the various practices that are employed, did not exist 15 years ago. Those with shorter experience tenure would have experienced Agile-programming methods in their internships and basic training. Those with 15 or more years would not have had this experience in their basic or formative training. They would also have had several years of experience in traditional programming methodologies prior to the advent of Agile. There was a significant statistical difference in the data between a respondent's experience level and his or her usage of paired or individual programming (see Figure 12). This certainly does not preclude those with programming tenure greater than 15 years the inability to engage Agile or Extreme Programming methods, but it does explain the tendency for more experienced respondents to recognize and rely on traditional programming practices more than contemporary practices.

Plotting a Manager's Experience Level to Practice Usage

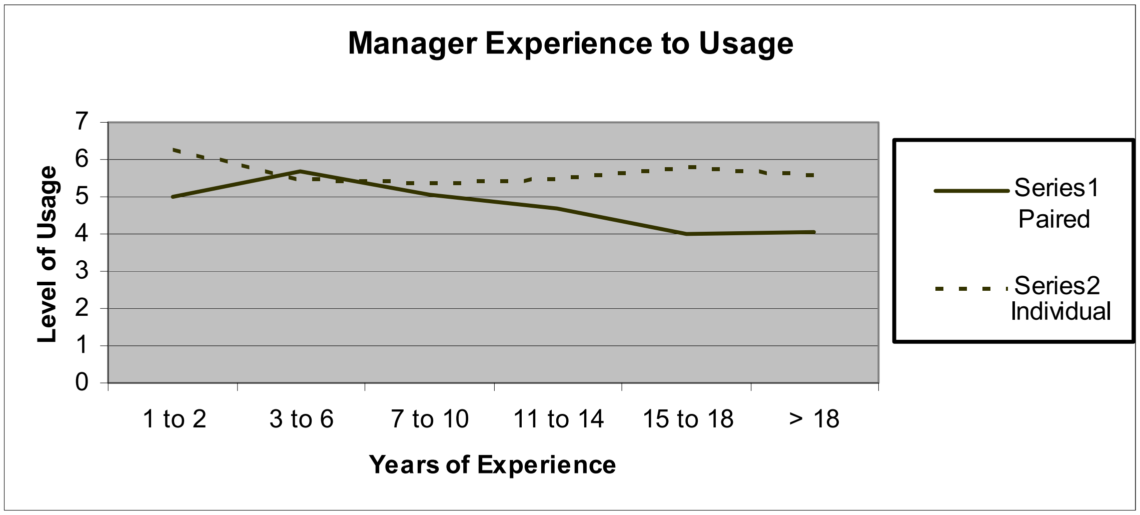


Figure 12. This graph shows the divergence in usage levels of either paired or individual programming practices and the level of experience of software manager respondents. The level of usage represents the ∑ mean values assigned to the survey question associated with self-reported usage (see Figure 7 above).

Limitations of the Study

The data within the study has been subjected to psychometric testing and analysis. There are numerous critics that state the results of such data are general at best and not representative of scientific or empirical testing (Benbasat & Barki, 2007). At the same time, the use of the TAM as a means to predict usage based on psychometric responses has been well documented. The results of the TAM are not meant to establish a single, unarguable value, but rather a relative descriptor of predicted usage, especially when compared to an antecedent of some similar analytic quantity.

Davis favored a research study using external validity over an internal (lab) setting, as a more realistic and empirical manner of observing psychometric patterns and empirical associations (1989). Although, in this study, a single external user population was used, that population was questioned about two antecedent variables (paired programming and individual programming). Not totally unlike Davis' study of two to four variables (three various chart applications and the pooling of the three to make up a fourth variable), this is still a limited sample of empirical measure. A more focused choice on whether to use or not use paired programming could have been designed to extend the behavior intention of the respondent to a clearer selection. Although the comparison of paired programming and individual programming are antecedent, they are not mutually exclusive. The choice of *either-or* is supplemented with a *both*, which could confuse or even obfuscate the results. The survey tool could have forced a choice between paired and individual programming on the same scale line. Alternatively, a question could have been asked: "To what extent is one practice used over the other?" Where this may create an empirical conundrum, is the practitioner's reality where both practices are actually present. Both are capable of being used, they are not mutually exclusive, and both have properties of success (perceived usefulness and perceived ease of use).

Another limitation includes the sampling of typical software development managers and their attitudes, behaviors, and predicted usage of one or both of the practices. It is possible that the assumption that the typical software managers' ability to dictate the method of programming and the practices employed in his or her company might be too liberal. Some organizations may already have decided this within a broader, more corporate strategy. In other cases, the style and methods used may be dictated by business rules or corporate agreements. In this case, the measure of behavioral intentions and usage are not linked and therefore invalid as a respondent measure. To solve this possible problem, a follow up questionnaire may be used to validate the respondent's environment and conditional membership in the sample group.

An additional limitation should be considered in the working definition of paired versus individual programming practices. Paired programming in its current and past state has been well defined. Its activities and tasks, albeit somewhat general in nature, are well accepted and tightly understood by most practitioners. Individual programming practices are varied and widely defined. Software engineering in assembly-based languages through C# language could all be applied to the individual programming practice if the code was implemented from a cascade or waterfall approach. Individual programming can be construed to mean a systematic process versus an iterative approach as is the case in paired programming. Although many would accept this distinction, the exactness of the respondents' consideration of this distinction cannot be verified by the processes used in this study.

The research represented here does not overcome the issues presented by Arisholm et al (2007). These issues represented: (a) Differences in the sample population (the types and experiences of software development managers), (b) Possible variation in types of business environments represented (not to be confused with the business type or general business line categorized in hypothesis 4), (c) Different methods of addressing the variable constructs (paired programming is specific whereas individual programming or waterfall programming encompasses a wide range of practice specifics), and (d) The variation in the complexity of the tasks that framed the respondent's opinion and behavioral intentions for usage. As Arisholm et al (2007) stated, "In light of existing research in software engineering and social psychology we expected that system complexity and programmer expertise would have a significant impact on when and how [Paired Programming] is beneficial compared with individual programming" (p. 65).

It should be noted that the original TAM model, or the TAM I model, was used in this study. There are more contemporary and more complex versions of the TAM including an 18-sectioned model in a unified version as well as a TAM III model. With these newer models, there is also a significant complexity of both output and design, rendering them severely inappropriate for this type of study or for any study that seeks to be practitioner oriented. There are noted (in the literature) limitations to the original TAM, such as the analysis of only two main constructs and only a couple of independent variables. There is however, a direct proportion of the number of variables (and coverage) to the amount of complexity and ability for usefulness on the part of the business/practitioner. A tool for evaluating predictable usage is not much good if its complexity and difficulty make calculating and integrating its results near impossible.

Implications

The state of the literature to date shows an overwhelming acceptance of paired programming as a positive, adjunctive practice that adds effectiveness and quality to the engineering of software code (Williams, 2000). Various studies have contributed to this popularity in the vacuum of needed practices that will help mature the software development industry to engineer better and more reliable code, faster and with greater accuracy and less cost (Ghezzi, Jazayeri, & Mandrioli, 1991). Studies have attempted to measure paired programming against various other programming practices to show positive results when two programmers are paired together at one terminal versus when programmers work alone. Various percentage increases in output and quality are at the root of the measurement to show that the investment in two people is better than the output of two people working by themselves independently (Williams, 2000).

The research questions in this study reflect a challenge to measure empirically the use of the paired programming and individual programming practice by software development managers. These measures were processed through a generally acceptable theoretical model (the TAM). The results do not completely coincide with the literature trends. Whereas paired programming was recognized in the literature as a popular practice of some noteworthy accomplishments (and measure), the study results demonstrated that it was not predicted to be used more than individual programming by software development managers. There were indications that the business type of the software development manager did slightly, but not significantly, influence their acceptance of the paired programming practice. There was evidence that the more experience the programming manager had, the less likely he or she was prone to implement paired programming.

Critics of paired programming reported on studies that showed little in the way of efficiencies and effectiveness gains by the paired programming practice (Nawrocki & Wojciechowski, 2001). The data from this study would tend to support some of those findings. It appears there is still much to be studied and a need for tighter empirical testing standards and with less questionable results.

The implications of the data produced in this study indicate a less popular and less used practice than the waterfall or individual programmer practice. This does not say that it is false to state that paired programming produces better quality code or that over various periods, the investment of two programmers working together at one terminal will be more productive than two programmers working separately. It does say that software managers, who make decisions about the type of practices that are employed in their development environments, tend to continue to favor individual programming practices over paired programming practices based on their perceived usefulness to a large extent and perceived ease of use to a moderate extent. The data collected and analyzed here also indicates that the experience level of the manager plays some role in the decision to use one practice over another. Very little is indicated in the literature that considers the manager’s experience level as a controlling variable in the success or adoption of paired programming over other practices.

There is significant attention given in the literature to a programmer’s experience as a key to the success and productivity output for paired programming versus individual programming practices. The literature suggests that early training in paired practices yields a more productive pairing partner. It also indicates that productivity and effectiveness increase as the experience level of the programmer increases. The data in this study confirms that the more experienced the development manager, the more the tendency is to select individual programming over paired programming, especially after 15 years or greater of programming experience. The data demonstrates that the more experience a development manager has, the greater the likelihood he or she will use individual programming practices. This may change over time. Follow-up studies are highly recommended to monitor the possible shift of usage should that occur over the next several years.

The challenge of the data in this study compared to the direction of the literature is evident. Well-published and popular contributors to software development maturity have supported the more progressive moves to radically different practices in order to thwart the growing tide of disappointment over software engineering's lack of productivity and success. Yet it appears that despite the charged, popular environment, software development managers are not readily accepting this popular movement with abandon.

The data and analysis drawn from the research in this study also points to the successful use of The Technology Acceptance Model as both a template of analysis and a method by which a technology practice can be evaluated for predictable usage. Despite the other forms of the TAM and other technology acceptance models available, the Davis 1989 model still appears to be the most basic and seminal model available.

Recommendations

Future research is needed to extend the sampling of usage for paired and individual programming practices and determine the usage and acceptance at various levels of software development management. Additional research is needed to bridge an empirical understanding between the usage prediction of the software development manager and the acceptability/use of paired programming, individual programming, or other form of a software coding practice. Additional research is also needed in comparing the paired programming practice with other practices (there are 12 practices just within the Extreme Programming methodology). These results would yield a measure that might also serve as a controlling factor in understanding how other practices are considered or used to a greater or lesser extent than paired programming. Ghezzi, Jazayeri, & Mandrioli (1991), stated:

> Successful software engineering requires the application of engineering principles guided by informed management. The principles must themselves be rooted in sound theory. While it is tempting to search for miracles and panaceas, it is unlikely that they will appear. The best course of action is to stick to age-old engineering principles. There simply are no "silver bullets." (p. 33)

The purpose and goal of this research, as well as similar studies, is to seek an aid for software development managers to find better practices that will support their effort to produce code that is more reliable and can be produce in a cost efficient and timely manner. The future challenges remain significant, and it is only through continued investigation and research will greater progress be made toward this goal.

REFERENCES

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New directions on Agile methods: A comparative analysis. *Proceedings of the International Conference on Software Engineering,* Copenhagen, Denmark.

Academy of Management. *Code of Ethics.* Retrieved November 12, 2008 from http://www.aomonline.org/aom.asp?id=268

Agarwal, R. & Prasad, J. (1999). Are individual differences germane to the acceptance of new information technologies? *Decision Sciences, 30*(2), 361-391.

Ajzen, I. & Fishbein, M. (1980). *Understanding attitudes and predicting social behavior*. Englewood Cliffs, NJ: Prentice-Hall.

Ajzen, I. (1985). From intentions to actions: A theory of planned behavior. In J. Kuhl & J. Beckman (Eds.), *Action-control: From cognition to behavior*. Heidelberg, Germany: Springer.

Ajzen, I. (1987). Attitudes, traits, and actions: Dispositional prediction of behavior in personality and social psychology. In L. Berkowitz (Ed.), *Advances in experimental social psychology* (Vol. 20, pp. 1-63). New York: Academic Press.

Ajzen, I. (1991). The theory of planned behavior. *Organization Behavior and Human Decision Processes, 50*, 179-211.

Ajzen, I. (2002). Perceived behavioral control, self-efficacy, locus of control, and the theory of planned behavior. *Journal of Applied Social Psychology, 32,* 665-683.

Ajzen, I. (2005). *Attitudes, personality and behaviour*, 2nd ed. Berkshire, Great Britain: McGraw-Hill Education.

Alavi, M. (1984, June). An analysis of the prototyping approach to information systems development. *Communications of the ACM, 27*(6), 556-563.

Almutairi, H. (2007). Is the 'Technology Acceptance Model' universally applicable?: The case of the Kuwaiti ministries. *Journal of Global Information Technology Management, 10*(2), 57-80. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1400763801).

Anderson, N. S. & Olsen, J. R. (1985). Methods for designing software to fit human needs and capabilities: *Proceedings of the Workshop on Software Human Factors*. Washington D.C.: National Academy Press.

Arisholm, R., 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.

Arisholm, R. & Sjøberg, D. I. K. (2003). A controlled experiment with professionals to evaluate the effect of a delegated versus centralized control style on the maintainability of object-oriented software. *Technical Report 2003-6, Simula Research Laboratory.* Retrieved August 10, 2008 from http://www.simula.co/erika.

Armitage, C. J. & Conner, M. (1999). The theory of planned behavior: Assessment of predictive validity and perceived control. *British Journal of Social Psychology, 38*, 35-54.

Baehti, P., Gehringer, E., & Stotts, D. (2002). Exploring the efficacy of distributed pair programming. *Proceedings of Extreme Programming and Agile Methods – XP/Agile Universe 2002, Second XP Universe and First Agile Universe Conference*, Chicago, IL, 208-220.

Bagozzi, R. P. (2007). The legacy of The Technology Acceptance Model and a proposal for a paradigm shift. *Journal of the Association for Information Systems, 8*(4), 243-254. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1316739731).

Bahli, B. & Zeid, E.S.A. (2005) Information and Communications Technology 2005: Enabling Technologies for the New Knowledge Society. *Proceedings of the ITI 3rd International Conference,* Cairo, Egypt, 75-87.

Baker-Eveleth, L., Eveleth, D. M., O'Neill, M., & Stone, R. W. (2006). Enabling laptop exams using secure software: Applying The Technology Acceptance Model. *Journal of Information Systems Education, 17*(4), 413-420. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1211107121).

Beck, K. (1999). *Extreme programming explained: Embrace change*. San Francisco: Addison-Wesley.

Beck, K. (2000). *Extreme programming explained*. New York: Addison Wesley Longman.

Beck, K. & Fowler, M. (2001). *Planning extreme programming*. New York: Addison-Wesley.

Bem, D. J. (1967). Self-Perception: An alternative interpretation of cognitive dissonance phenomena. *Psychological Review, 74*, 183-200.

Benamati, J. & Rajkumar, T. M. (2002). The application development outsourcing decision: An application of the technology acceptance model. *The Journal of Computer Information Systems, 42*(4), 35-43. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 147588681).

Benbasat, I. & Barki, H. (2007). Quo vadis, TAM? *Journal of the Association for Information Systems, 8*(4), 211-218. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1316739691).

Berenson, S. B., Slaten, K. M., Williams, L., & Ho, C. (2004, March). Voices of women in a software engineering course: Reflections on collaboration. *ACM Journal on Educational Resources in Computing, 4*(1).

Berenson, S. B., Williams, L., Slaten, K. M. (2005). Using pair programming and Agile development methods in a university software engineering course to develop a model of social interactions. *Crossing Cultures, Changing Lives Conference*, Oxford, United Kingdom.

Bevan, J., Werner, L., & McDowell, C. (2002). Guidelines for the use of pair programming in a freshman programming class. *In the 15th Conference on Software Engineering Education and Training.* (pp. 100-107), Covington, KY.

Boehm, B. W. (1981). *Software engineering economics*. Englewood Cliffs, NJ: Prentice-Hall.

Bowen, W. (1986, May). The puny payoff from office computers. *Fortune*, 20-24.

Braught, G., Eby, L. M., & Wahls, T. (2008, March). The effects of pair programming on individual programming skill. *Proceedings of the SIGCSE 2008*, 200-204, Portland, Oregon.

Brodie, Leo (1984) *Thinking Forth*. New York: Prentice-Hall.

Bryant, S. (2004). Double trouble: Mixing qualitative and quantitative methods in the study of eXtreme programmers. *Proceedings of the VL/HCC*, Rome, Italy, 55-61.

Bryant, S. (2005). Rating expertise in collaborative software development. *Proceedings of the Paired Programming Information Group,* Brighton, UK, 19-29.

Bryant, S., Romero, P., & duBoulay, B. (2005, November 6-9). Pair programming and the re-appropriation of individual tools for collaborative programming.

Proceedings of ACM Group '05 Convention, (332-333). Sanibel Island, Florida, USA,.

Budd, R. I. (1987). Response bias and the theory of reasoned action. *Social Cognition, 5*(2), 95-107.

Burton-Jones, A. & Hubona, G. S. (2005). Individual differences and usage behavior: Revisiting a technology acceptance model assumption. *The Data Base for Advances in Information Systems, 36*(2), 58-77.

Campbell, D. T. & Fiske, D. W. (1959, March). Convergent and discriminate validation by the Multitrait-multimethod matrix. *Psychological Bulletin, 56*(9), 81-105.

Canfora, G., Cimitile, A., DiLucca, G. A., & Visaggio, C. A. (2006). How distribution affects the success of pair programming. *International Journal of Software Engineering and Knowledge Engineering, 6*(2), 293-313.

Cao, L. & Ramesh, B. (2008). Agile requirements engineering practices: An empirical study. *Journal of IEEE Software, 25*(1), 60

Carmel, E. & Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. *IEEE Software, 18*(2), 22-29.

Chaffey, D. (1998). *Groupware, workflow and intranets*. Boston, MA: Digital Press.

Chaparro, E. A., Yuksel, A., Romero, P., & Bryant, S. (2005). Factors affecting the perceived effectiveness of pair programming in higher education. *Proceedings of the Pair Programming Information Group*, Brighton, UK, 5-18.

Chau, P. Y. K. (1996). An empirical assessment of a modified technology acceptance model. *Journal of Management Information Systems, 13*(2), 185-204. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 10578717).

Chong, J. & Hurlbutt, T. (2007). The Social dynamics of pair programming. *Proceedings of the 29th International Conference on Software Engineering (ICSE'07)* (Document ID # 0-7695-2828-7/07).

Ciolkowski, M. & Schlemmer, M. (2002). Experiences with a case study on pair programming. *First International Workshop on Empirical Studies in Software Engineering,* Finland.

Clegg, C. W. (1994). Psychology and information technology: The study of cognitions in organizations. *British Journal of Psychology, 85*, 449-477.

Clegg, C. W., Waterson, P. E., & Axtell, C. M. (1996). Software development: Knowledge-intensive work organizations. *Behavior & Information Technology, 15*(4), 237-249.

Cliburn, D. (2003). Experiences with pair programming at a small college. *The Journal of Computing in Small Colleges, 19*(10), 20-29.

Cockburn, A. & Williams, L. (2000). The costs and benefits of pair programming, in Extreme Programming and Flexible Processes. In *Software Engineering (XP 2000)*, Cagliari, Sardinia, Italy, New York: Addison Wesley.

Cockburn, A. & Williams, L. (2001). The costs and benefits of pair programming. In G. Succi & M. Marchesi (Eds.), *Extreme Programming Examined*, Boston: Addison Wesley.

Compeau, D. & Higgins, C. (1995). Computer self-efficacy: Development of a measure and initial test. *MIS Quarterly, 19*(2), 189-211.

Cooper, D. & Schindler, P. S. (2006). *Business research methods*. New York: McGraw-Hill/Irwin.

Creative Research Systems (2010). Sample size calculator. *The Survey System.* Retrieved January 16, 2010 from http://www.surveysystem.com/sscalc.htm.

Crispin, L. (2006). Driving software quality: How test-driven development impacts software quality. *The Journal of IEEE Software, 23*(6), 70-71.

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika, 16*(3), 297-334.

Dasgupta, S., Granger, M., & McGarry, N. (2002). User acceptance of e-collaboration technology: An extension of the technology acceptance model. *Group Decision and Negotiation, 11*(2), 87-100. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 403883861).

Davidson, N. (1994). Cooperative and collaborative learning: An integrative perspective. In J. Thousand, R. Villa, & A. Nevin (Eds.), *Creativity and Collaborative Learning: A Practical Guide to Empowering Students and Teachers*. (pp. 13-10), Baltimore, MD: Paul H. Brookes.

Davis, F. D. (1985). A technology acceptance model for empirically testing new end-user information systems: Theory and results (Doctoral dissertation, Massachusetts Institute of Technology-Sloan School of Management, 1985) *MIT Management Library* (No. 1721.1/15192).

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly, 13*(3), 319-340.

Davis, F. D. (1993). User acceptance of information technology: System characteristics user perceptions and behavior impacts. *International Journal of Man Machine Studies, 38*, 475-487.

Davis, F. D., Bagozzi, R. P., & Warshaw, P. R. (1989). User acceptance of computer technology: A comparison of two theoretical models. *Management Science, 35*(8), 982-1003.

DeClue, T. (2003). Pair programming and pair trading: Effects on learning and motivation in a CS2 course. *The Journal of Computing in Small Colleges, 18*(5), 49-56.

Deming, W. E. (1960). *Sample design in business research*. New York: Wiley.

DeSanctis, G. (1983). Expectancy theory as an explanation of voluntary use of a decision support system. *Psychological Reports, 52*, 247-260.

Donald, I. & Cooper, S. R. (2001). A facet approach to extending the normative component of the theory of reasoned action. *British Journal of Social Psychology, 40*, 599-621.

Dorofeev, S. & Grant, P. (2006). Statistics for real-life sample surveys. Melbourne, Australia: Cambridge University Press.

Ebert, C. & deNeve, P. (2001). Surviving global software development. *IEEE Software, 18*(2), 62-69.

El Emam, K. & Koru, A. G. (2008, September/October). A replicated survey of IT software project failures. *IEEE Software, 34*(4), 84-90.

Erdogmus, H. & Williams, L. (2003). The economics of software development by pair programmers. *The Engineering Economist, 48*(4), 283-319.

Fazio, R. H., Lenn, T. M., & Effrein, E. A. (1984). Spontaneous attitude formation. *Social Cognition, 2*(3), 217-234.

Fishbein, M., & Ajzen, I. (1975). *Belief, attitude, intention, and behavior: An introduction to theory and research*. Reading, MA: Addison-Wesley.

Flor, N. V. (1998). Side-by-side collaboration. *International Journal of Human-Computer Studies, 49*, 201-222.

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. J. Koenemann-Belliveau, T. Moher, & S. Robertson, Eds. *Empirical Studies of Programmers: Fourth Workshop*. Norwood, NJ: Ablex Press.

Gallis, H., Arisholm, E., & Dybå, T. (2003). An initial framework for research on pair programming. *Proceedings of the International Software Engineering Syposium and Exposition*, Italy.

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

Glassman, M. & Fitzhenry, N. (1976). Fishbein's subjective norm: Theoretical considerations and empirical evidence. In B. B. Anderson, (Ed.), *Advances in Consumer Research, Vol. 3*. Ann Arbor, MI: Association for consumer Research, 477-483.

Gong, M., Xu, Y., & Yu, Y. (2004). An enhanced Technology Acceptance Model for web-based learning. *Journal of Information Systems Education, 15*(4), 365-374. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 793505851).

Goodhue, D. L. (2007). Comment on Benbasat and Barki's "Quo Vadis TAM" article. *Journal of the Association for Information Systems, 8*(4), 219-222. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1316739701).

Gould, J. D. & Lewis, C. (1985, March). Designing for usability: Key principles and what designers think. *Communications of the ACM, 28*(3), 300-311.

Grenning, J. (2001). Launching extreme programming at a process-intensive company. *IEEE Software, 18*(6), 27-33.

Hale, J. L., Householder, B.J., & Greene, K.L. (2003). The theory of reasoned action. In J.P. Dillard & M. Pfau (Eds.), *The persuasion handbook: Developments in theory and practice* (pp. 259 – 286). Thousand Oaks, CA: Sage.

Hanks, B. F. (2004). Distributed pair programming: An empirical study. *Proceedings of Extreme Programming and Agile Methods – XP/Agile Universe 2004: 4th Conference on Extreme Programming and Agile Methods*, (pp. 81-91). Calgary, Canada.

Hanks, B. F. (2006). Student attitudes toward pair programming. *Proceedings of the ITiCSE 2006*, Bologna, Italy.

Hartwick, J. & Barki, H. (1994). Explaining the role of user participation in information systems use. *Management Sciences, 40*, 440-465.

Hayes, F. (2002). Don't shrug off bugs. *Computerworld, 36*(27), 50-51.

Heilberg, S., Puus, U., Salumaa, P., & Seeb, A. (2003). Pair-programming effect on developers productivity. *Fourth International Conference on Extreme Programming and Agile Processes in Software Engineering (XP2003)* (pp. 215-224). New York: Springer-Verlag.

Herbsleb, J. D. & Grinter, R. E. (1999). Architecture, coordination and distance: Conway's law and beyond. *IEEE Software, 16*(5), 63-70.

Highsmith, J. (1999). *Adaptive software development: A Collaborative approach to managing complex systems*. Boston, MA: Dorset House.

Highsmith, J. (2001). *The agile manifesto for software development*. Retrieved July 16, 2008 from www.agilemanifestor.org.

Highsmith, J. (2002). *Agile software development ecosystems.* New York: Addison-Wesley Pearson Education.

Highsmith, J. (2004). *Agile project management: Creating innovative products.* New York: Addison-Wesley Professional.

Hilkka, M., Tuure, T., & Matti, R. (2005). Journal of Database Management. *Hershey Journal, 16*(4), 21-41.

Ho, C. W., Raha, S., Gehringer, E., & Williams, L. (2005). Sangam: A distributed pair programming plug-in for eclipse. *The Proceedings of the OOPSLA Workshop on Eclipse Technology eXchange*, Vancouver, BC, Canada, 73-77.

Hom, P. W., Katerberg, W. R., & Hulin, C. L. (1979). Comparative examination of three approaches to the prediction of turnover. *Journal of Personality and Social Psychology, 11*, 123-128.

Horton, R. P., Buck, T., Waterson, P. E., & Clegg, C. W. (2001). Explaining intranet use with The Technology Acceptance Model. *Journal of Information Technology, 16*(4), 237-249. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 667615981).

Hu, P. J., Chau, P. Y. K., Sheng, O. R. L., & Tam, K. Y. (1999). Examining The Technology Acceptance Model using physician acceptance of telemedicine technology. *Journal of Management Information Systems, 16*(2), 91-112.

Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 47523432).

Hulkko, H. & Abrahamsson, P. (2005, May). A multiple case study on the impact of pair programming on product quality. *Journal of the ACM*, 495-504.

Humboldt State University Survey Website. (n.d.). *Validating a Survey*. Retrieved December 29, 2008 from http://www.humboldt.edu/~storage/surveysite/survey_validation.html.

Humphrey, W. S. (1995). *A discipline for software engineering*. New York: Addison-Wesley.

Igbaria, M., Guimaraes, T., & Davis, G. B. (1995). A path analytic study of individual characteristics, computer anxiety, and attitudes towards microcomputers. *Journal of Management, 15*(3), 373-388.

Ives, B. & Olsen, M. H. (1984). User involvement and MIS success: A review of research. *Management Sciences, 30*, 586-603.

Jaccard, J. J. & Davidson, A. R. (1972). Toward an understanding of family planning behaviors: An initial investigation. *Journal of Applied Social Psychology, 2*, 228-235.

Jacobson, N. (2000). Using on-computer exams to ensure beginning students' programming competency. *SIGCSE Bulletin, 32*(4), 53-56.

Jacobson, N. & Schaefer, S. K. (2008). Pair programming in Computer Science 1: Overcoming objections to its adoption. *Inroads-SIGCSE, 40*(2), 93-96.

Jensen, R. W. (2003). A pair programming experience. *CrossTalk, The Journal of Defense Software Engineering, 16*, 22-24.

Johansen, R. & Baker, E. (1984). User needs workshops: A new approach to anticipating user needs for advanced office systems. *Office Technology and People, 2*, 103-119.

Johnson, J. (1994). *My life is a failure: 100 things you should know to be a project leader*. Boston, MA: The Standish Group International Press.

Johnson, J. (1995). Chaos: The dollar drain of IT project failures. *Application Development Trends, 2*(1), 41-47.

Karahanna, E. (1993). *Evaluative criteria and user Acceptance of end-user information technology: A study of end-user cognitive and normative pre-adoption beliefs.* Unpublished doctoral dissertation, University of Minnesota.

Keat, T. K. & Mohan, A. (2004). Integration of TAM based electronic commerce models for trust. *Journal of American Academy of Business, Cambridge, 5*(1/2), 404-410. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 653886231).

Kiercher, M., Jain, P., Corsaro, A., & Levine, D. (2001). Distributed eXtreme programming. *Proceedings of XP2001*, Kuala Lumpur, Malaysia.

Klein, G. & Beck, P. O. (1987). A decision aid for selecting among information systems alternatives, *MIS Quarterly, 11*(2), 177-186.

Klopping, I. M. & McKinney, E. (2004). Extending the technology acceptance model and the task-technology fit model to consumer e-commerce. *Information Technology, Learning, and Performance Journal, 22*(1), 35-48. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 719370511).

Krishnakumar, P. & Sukumaran, N. V. (1997). A model for software development effort and cost estimation. *IEEE Transactions on Software Engineering, 23*(8), 485-497.

Larman, C. & Basili, V. (2003). Iterative and incremental development: A brief history. *Computer IEEE Computer Society, 36*(6), 47-56.

Lee, H. H., Fiore, A. M., & Kim, J. (2006). The role of the technology acceptance model in explaining effects of image interactivity technology on consumer responses. *International Journal of Retail & Distribution Management, 34*(8), 621-644. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1073444641).

Legris, P., Ingham, J., & Collerette, P. (2003). Why do people use information technology? A critical review of the technology acceptance model. *Information & Management, 30*(2), 1-21.

Levy, L. S. (1987). Taming the tiger: Software engineering and software economics. In H. Ledgard (Ed.), *Springer Books on Professional Computing*. New York: Springer-Verlag.

Lindstrøm, L. & Jeffries, R. (2004, Summer). Extreme programming and agile software development methodologies. *Information Systems Management, 21*(3), 41-52.

Lui, K. M. & Chan, K. C. C. (2003). When does a pair outperform two individuals? *Proceedings from XP2003*, Rome, Italy.

Ma, Q. & Liu, L. (2004). The technology acceptance model: A meta-analysis of empirical findings. *Journal of Organizational and End User Computing, 16*(1), 59-72. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 533172441).

Manstead, A. S. R. & Parker, D. (1995). Evaluating and extending the theory of planned behavior. In W. Stroebe & M. Hewstone (Eds.), *European Review of Social Psychology,* 6, 69-96.

Margolis, J. & Fisher, A. (2002). *Unlocking the clubhouse: Women in computing*. Cambridge, MA: MIT Press.

Markus, M. L. & Keil, M. (1994). If we build it, they will come: Designing information systems that users want to use. *Sloan Management Review, 35*(4), 11-25.

Martin, K. & Hoffman, B. (2007). An open source approach to developing software in a small organization. *Journal of IEEE Software, 24*(1), 46-53.

Mason, E. J. & Bramble, W. J. (1989). *Understanding and conducting research*. New York: McGraw- Hill Publishing.

Mathieson, K., Peacock, E., & Chin, W. W. (2001). Extending the technology acceptance model: The influence of perceived user resources. *The Data Base for Advances in Information Systems, 32*(3), 86-112.

McCloskey, D. W. (2004). Evaluating electronic commerce acceptance with the technology acceptance model. *The Journal of Computer Information Systems, 44*(2), 49-57. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 538723971).

McCloskey, D. W. (2006). The Importance of ease of use, usefulness, and trust to online consumers: An examination of the technology acceptance model with older consumers. *Journal of Organizational and End User Computing, 18*(3), 47-65. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1060110261).

McCoy, S., Galletta, D. F., & King, W. R. (2007). Applying TAM across cultures: The need for caution. *European Journal of Information Systems, 16*(1), 81-90. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1244874171).

McDowell, C., Hanks, B., & Werner, L. (2003). Experimenting with pair programming in the classroom. *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education, ACM SIGCSE Bulletin, 35*(3), 60-64.

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.

McGuire, T. W. & Weiss, D. L. (1976). Logically consistent market share models II. *Journal of Marketing Research, 13*(3), 296-303.

McKechnie, S., Winklhofer, H., & Ennew, C. (2006). Applying the technology acceptance model to the online retailing of financial services. *International Journal of Retail & Distribution Management, 34*(4/5), 388-410. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1044969001).

Meister, D. B. & Compeau, D. R. (2002). *Infusion of innovation adoption: An individual perspective.* Winnipeg, Manitoba: Proceedings of the ASAC

Mendes, E., Al-Fakhri, B., & Luxton-Reilly, A. (2006). Investigating pair programming in a 2nd year software development and design computer science course. *Proceedings of ITiCSE Bulletin, 34*(1), 296-300.

Mendes, E., Al-Fakhri, B., & Luxton-Reilly, A. (2006, June). A replicated experiment of pair-programming in a 2nd-year software development and design computer science course. *Proceedings of the ITiCSE,* 108-112, Bologna, Italy.

Money, W. & Turner, A. (2004). Application of The Technology Acceptance Model to a knowledge management system. *Proceedings of the 37th Hawaii International Conference on System Sciences*, Track 8, Honolulu, HI

Montealegre, R. & Keil, M. (2000). De-escalating information technology projects: lessons from the Denver International Airport. *MIS Quarterly* 24(3), 417-447.

Mugridge, R. (2008). Managing Agile project requirements with story, test-driven development. *Journal of IEEE Software, 25*(1), 68.

Müller, M. M. (2003). Are reviews an alternative to pair programming? *7th International Conference on Empirical Assessment in Software Engineering*, London, United Kingdom.

Müller, M. M. (2004). Two controlled experiments concerning the comparisons of pair programming to peer review. *Journal of Systems and Software, 78*(2), 166-179.

Mykytyn, P. P. & Harrison, D. A. (1993, Spring). The application of the theory of reasoned action to senior management and strategic information systems. *Information Resources Management Journal*, 6(2), 15-27.

Nawrocki, J. & Wojciechowski, A. (2001). Experimental evaluation of pair programming. *Proceedings of European Software Control and Metrics Conference* (ESCOM), London, United Kingdom.

Nilsson, K. (2003). A summary from a pair programming Survey: Increasing quality with pair programming. *Journal of computing in Small Colleges, 18,* 57-65.

Nosek, J. T. (1998, March). The case for collaborative programming. *Communications of the ACM, 41*(3), 105-108.

Notani, A. S. (1998). Moderators of perceived behavioral control's predictiveness in the theory of planned behavior: A meta-analysis. *Journal of Consumer Psychology, 7*, 247-271.

Oblinger, D. (2003, July/August). Boomers, gen-xers, and millennials: Understanding the new students. *Educause Review, 38*(4), 37-47.

Oullette, J. A. & Wood, W. (1998). Habit and intention in everyday life: The multiple processes by which past behavior predicts future behavior. *Psychological Bulletin, 124*, 54-74.

Padberg, F. & Müller, M. (2004). An empirical study about the feel-good factor in pair programming. *Metrics, 10*, 151-158.

Pei, Z., Zhenxiang, Z., & Chunping, H. (2007). An extended TAM model for Chinese B2C websites design. *Journal of Global Information Technology Management, 10*(1), 51-66. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1262575201).

Perez, M. P., Sanchez, A. M., Carnicer, P. L., & Jimenez, M. J. V. (2004). A technology acceptance model of innovation adoption: the case of teleworking. *European Journal of Innovation Management, 7*(4), 280-291. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 748515681).

Pikkarainen, T., Pikkarainen, K., Karjaluoto, H., & Pahnila, S. (2004). Consumer acceptance of online banking: an extension of the technology acceptance model. *Internet Research, 14*(3), 224-235. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 672962091).

Plouffe, C. R., Hulland, J. S., & Vandenbosch, M. (2001). Research report: richness versus parsimony in modeling technology adoption decisions-understanding merchant adoption of a smart card-based payment system. *Information Systems Research, 12*(2), 208-222.

Pomazal, R. J. & Jaccard, J. J. (1976). An informational approach to altruistic behavior. *Journal of Personality and Social Psychology, 33*, 317-326.

Preston, D. (2005). Pair programming as a model of collaborative learning: A review of the research. *Consortium for Computing Sciences in Colleges: Central Plains Conference,* 39-45.

Putnam, L. H. (1978). A general empirical solution to the macro software sixing and estimating problem. *IEEE Transactions on Software Engineering, SE4*(4), 345-357.

Randall, D. M. (1989). Taking stock: Can the theory of reasoned action explain unethical conduct? *Journal of Business Ethics, 8*(11), 873-882.

Rawstorne, P., Jayasuriya, R., & Caputi, P. (2000). Issues in predicting and explaining usage behaviors with the technology acceptance model and the theory of planned behavior when usage is mandatory. *Proceedings from the Twenty-first International Conference on Information Systems*, (pp. 35-44).

Rigopoulos, G. & Askounis, D. (2007). A TAM framework to evaluate users' perception towards online electronic payments. *Journal of Internet Banking and Commerce, 12*(3), 1-6. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1454507231).

Robson, C. (2002). *Real world research* (3rd ed.). Malden, MA: Blackwell.

Ronis, D. L., Yates, J. F., & Kirscht, J. P. (1989). Attitudes, decisions, and habits as determinants of repeated behavior. In A. R. Pratkanis, S. J. Breckler, & A. G. Greenwald (Eds.), *Attitude Structure and Function*, (pp. 213-239). NJ: Lawrence Erlbaum Associates.

Royce, W. W. (1970). Managing the Development of large software systems: Concepts and techniques. In: *Technical Papers of Western Electronic Show and Convention (WesCon)* August 25-28, Los Angeles, USA.

Savitskie, K., Royne, M. B., Persinger, E. S., Grunhagen, M., & Witte, C. L. (2007). Norwegian internet shopping sites: An application & extension of the technology acceptance model. *Journal of Global Information Technology Management, 10*(4), 54-73. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1382170461).

Schneberger, S., Amoroso, D. L., & Durfee, A. (2007). Factors that influence the performance of computer-based assessments: An extension of the technology acceptance model. *The Journal of Computer Information Systems, 48*(2), 74-90.

Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1427015491).

Schwarz, A. & Chin, W. (2007). Looking forward: Toward an understanding of the nature and definition of IT acceptance. *Journal of the Association for Information Systems, 8*(4), 230,232-243. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1316739721).

Sejwacz, D. I., Ajzen, I., & Fishbein, M. (1980). Predicting and understanding weight loss: Intentions, behaviors and outcomes'. In I. Ajzen and M. Fishbein (Eds.), *Understanding Attitudes and Predicting Social Behaviors*. Englewood Cliffs, NJ: Prentice-Hall, 101-112.

Sfetsos, P., Stamelos, I., Angelis, L., & Deligiannis, I. S. (2006). Investigating the impact of personality types on communication and collaboration-viability in pair programming. *XP/Agile, 7*, 43-52.

Sharda, R., Barr, S. H., & McDonnell, J. C. (1988). Decision support system effectiveness: A review and empirical test. *Management Science, 34*(2), 139-159.

Sheil, B. A. (1981). The Psychological Study of Programming. *ACM Computing Surveys, 13*(1), 101-120.

Sheppard, B., Hartwick, J., & Warshaw, P. (1988). The theory of reasoned action: A meta-analysis of past research with recommendations of modifications and future research. *Journal for Consumer Research, 15*(3), 325-343.

Shneiderman, B. (1987). *Designing the user interface*. Reading, MA: Addison-Wesley.

Silva, L. (2007). Post-positivist Review of Technology Acceptance Model. *Journal of the Association for Information Systems, 8*(4), 255-266. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1316739741).

Singh, K., Leong, S. M., Tan, C. T., & Wong, K. C. (1995, January). A theory of reasoned action perspective of voting behavior: Model and empirical test. *Psychology & Marketing, 12*(1), 37-51.

Smith, H. J., & Keil, M. (2003). The reluctance to report bad news on troubled software projects: A theoretical model. *Information Systems Journal* 13(1), 69-95.

Snedecor, G. W., & Cochran, W. G. (1967). *Statistical methods* (6th ed.). Ames, Iowa: Iowa State University Press.

Snowden, S., Spafford, J., Michaelides, R., & Hopkins, J. (2006). Technology acceptance and m-commerce in an operational environment. *Journal of Enterprise*

Information Management, 19(5), 525-539. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1143455361).

Southerland, J. (2004, October). Agile development: Lessons learned from the first SCRUM. *Cutter Agile Project Management Advisory Service, Executive Update, 5*(20), 1-6.

Sparks, P. & Guthrie, C. A. (1998). Self-identity and the theory of planned behavior: A useful addition or an unhelpful artifice. *Journal of Applied Social Psychology, 28*, 1393-1410.

Sparks, P. & Shepherd, R. (1992). Self-identity and the theory of planned behavior: Assessing the role of identification with “Green Consumerism.” *Social Psychological Quarterly, 55*(4), 388-399.

Speier, C. & Venkatesh, V. (2002). The hidden minefields in the adoption of sales force automation technologies. *Journal of Marketing, 66*(3), 98-111.

Standish Group International. (2001). *The 2001 Chaos Report*. The Standish Group International, 27-33.

Standish Group International. (2004). *The 2004 Chaos Report*. The Standish Group International, 49-53.

Stylianou, A. C. & Jackson, P. J. (2007). A comparative examination of individual differences and beliefs on technology usage: Gauging the role of IT. *The Journal of Computer Information Systems, 47*(4), 11-18. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1313492751).

Sutton, S. (1998). Predicting and explaining intentions and behavior: How well are we doing? *Journal of Applied Social Psychology, 28*(15), 1317-1338.

Szajna, B. (1996). Empirical evaluation of the revised technology acceptance model. *Management Science, 42*(1), 85. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 9474485).

Talby, D., Hazzan, O., Dubinsky, Y., & Keren, A. (2006). Agile software testing in a large-scale project. *Journal of IEEE Software, 23*(4), 30-37.

Taylor, S. & Todd, P. A. (1995). Understanding information technology usage: A test of competing models. *Information Systems Research, 6*(2), 144-176.

Terry, D. J., Hogg, M. A., & White, K. M. (1999). The theory of planned behavior: Self-identity, social identity, and group norms. *British Journal of Social Psychology, 38*(3), 225-244.

Thomas, K. H., Bull, C., & Clark, J. (1978). Attitude measurement in the forecasting of off-peak travel behavior. In P. W. Bonsall, Q. Dalvi, & P. J. Hills (Eds.), *Urban transportation planning: Current themes and future prospects.* Tunbridge Wells, UK: Abacus Press.

Thomas, L., Ratcliffe, M., & Robertson, A. (2003). Code warriors and code-a-phobes: A study in attitude and pair programming. *In the Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, 363-367.

Tomayko, J. E. (2002). A comparison of pair programming to inspections of software defect reduction. *Journal of Computer Science Education, 12*, 213-222.

Turk, D., France, R., & Rumpe, B. (2005). Assumptions underlying Agile software development processes. *Journal of Database Management, 16*(4), 62-87.

Veiga, J. F., Floyd, S., & Dechant, K. (2001). Towards modeling the effects of national culture on IT implementation and acceptance. *Journal of Information Technology, 16*(3), 145-158. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 667615621).

Venkatesh, V. & Davis, F. D. (2000). A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science, 46*(2), 186-204. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 52438017).

Venkatesh, V. & Morris, M. G. (2000). Why don't men ever stop to ask for directions? Gender, social influence, and their role in technology acceptance and usage behavior. *MIS Quarterly, 24*(1), 115-139.

Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User acceptance of information technology: Toward a unified view. *MIS Quarterly*, (27:3), 425-478.

Visaggio, C. A. (2005, May). Empirical validation of pair programming. *Proceedings of the ICSE 2005*, St. Louis, Missouri.

Waguespack, L. & Schiano, W. T. (2004, Summer). Component-based IS architecture. *Information Systems Management, 8*(3), 53-60.

Wallace, L., & Keil, M. (2004). Software project risks and their effect on outcomes. *Communications of the ACM,* 47(4), 68-73.

Werner, L., Hanks, B., & McDowell, C. (2005, March). Pair programming helps female computer science students persist. *ACM Journal of Educational Resources in Computing, 4*(1), 127-132.

Williams, L. (1999, November). But isn't that cheating? Collaborative programming. *Proceedings of the 29th Annual Frontiers in Education Conference, 2*, 12B9/26-12B9-27.

Williams, L. (2000). The collaborative software process. (Doctoral dissertation, Department of Computer Science, University of Utah, 2000). (UMI No. 9968742).

Williams, L. (2001). Integrating pair programming into a software development process. *Proceedings from the 14th Conference on Software Engineering Education and Training, USA*.

Williams, L. (2007). Lessons learned form seven years of pair programming at North Carolina State University. *Inroads – SIGCSE Bulletin, 39*(4), 79-83.

Williams, L. & Kessler, R. (2000, May). All I really wanted to know about pair programming I learned in kindergarten. *Communications of the ACM, 43*(5), 108-114.

Williams, L. & Kessler, R. (2003). *Pair programming illuminated*. Boston, MA: Addison-Wesley.

Williams, L. & Kessler, R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software, 17*, 19-25.

Williams, L., McDowell, C., Nagappan, N., Fernald, J., & Werner, L. (2003). Building pair programming knowledge through a family of experiments. *In the International Symposium on Empirical Software Engineering (ISESE) 2003*. (pp. 143-152), Rome, Italy.

Williams, L., Shukla, A., & Antón, A. I. (2004). An initial exploration of the relationship between pair programming and brooks law. *Proceedings of the Agile Development Conference*. San Francisco, CA.

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). Support of pair programming in the introduction computer science course, *Computer Science Education, 12*(3), 197-212.

Wood, W. A. & Kleb, W. L. (2003). Exploring XP for scientific research. *IEEE Software 20*(1), 30-36.

Yousafzai, S. Y., Foxall, G. R., & Pallister, J. G. (2007). Technology acceptance: A meta-analysis of the TAM: Part 1. *Journal of Modeling in Management, 2*(3),

251. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1377079531).

Yousafzai, S. Y., Foxall, G. R., & Pallister, J. G. (2007). Technology acceptance: A meta-analysis of the TAM: Part 2. *Journal of Modeling in Management, 2*(3), 281. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1377079481).

Yu, J. C., Liu, C., & Yao, J. E. (2003). Technology acceptance model for wireless Internet. *Internet Research, 13*(3), 206. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 358432131).

Zhang, X., Prybutok, V. R., & Koh, C. E. (2006). The role of impulsiveness in a TAM-based online purchasing behavior model. *Information Resources Management Journal, 19*(2), 54-68. Retrieved May 6, 2008, from ABI/INFORM Global database. (Document ID: 1010746551).

Zin, A. M., Idris, S., & Subramaniam, N. K. (2005). Implementing virtual pair programming in E-learning environments. *Journal of Information Systems Education, 17*(2), 113-117.

Zmud, R. W. (1979). Individual differences and MIS success: A review of the empirical literature. *Management Science, 25*(10), 966-979.

APPENDIX; ASSESSING THE ATTITUDES OF SOFTWARE DEVELOPMENT MANAGERS ON THE USE OF PAIRED PROGRAMMING AS COMPARED TO INDIVIDUAL PROGRAMMING

Note. From "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology" by Fred D. Davis, PhD, 1989*, MIS Quarterly, 13*(3), p. 340. Copyright 1989 by *MIS Quarterly*. Adapted and reprinted with permission.

Perspectives on Paired Programming as Compared to Individual Programming

Measuring Attitudes on Paired and Individual Software Development Programming
The following questions are designed to measure your experiences and attitudes about paired programming practices as compared to Individual Programming. The term ***paired programming*** is used to denote a specific programming practice used in Extreme Programming. It uses two developers, working at a single workstation to produce code from general requirements or *stories*. The term ***Individual Programming*** is used to represent the traditional, single programmer development practice. This practice of software development generates code from technical specifications and specific requirements, which are then segmented and distributed to individual programmers. The word ***efficient*** is used in this survey to represent an easy, cost-effective means of programming with a low incident of bugs or programming errors. The word ***effective*** is used in this survey to represent the production of a desired result with a reduced effort and a more accurate way of producing useable/executable code. You may exit from this survey at any time by selecting the Exit Survey button.

This survey should only take 10 minutes to complete. Thank you for your candid responses.

Section I

1. Using the Paired Programming practice would enable my software development group to develop software more quickly.

Likely	extremely	quite	slightly	neither	slightly	quite	extremely	Unlikely

2. Using Paired Programming would improve my software development group's job performance.

Likely	extremely	quite	slightly	neither	slightly	quite	extremely	Unlikely

3. Using Paired Programming in my software development group would increase productivity.

Likely ____\ Unlikely
extremely quite slightly neither slightly quite extremely

4. Using Paired Programming in my software development group would enhance the effectiveness of producing code.
Likely ____\ Unlikely
extremely quite slightly neither slightly quite extremely

5. Using Paired Programming would make it easier for my software development group to produce software code.
Likely ____\ Unlikely
extremely quite slightly neither slightly quite extremely

6. My software development group would find Paired Programming useful in carrying out their software development responsibilities.
Likely ____\ Unlikely
extremely quite slightly neither slightly quite extremely

7. Using Paired Programming in my software development group would enhance the efficiency of producing code.
Likely ____\ Unlikely
extremely quite slightly neither slightly quite extremely

Section II

8. Using the Individual Programming practice would enable my software development group to develop software more quickly.
Likely ____\ Unlikely
extremely quite slightly neither slightly quite extremely

9. Using Individual Programming would improve my software development group's job performance.
Likely ____\ Unlikely
extremely quite slightly neither slightly quite extremely

10. Using Individual Programming in my software development group would increase productivity.
Likely ____\ Unlikely
extremely quite slightly neither slightly quite extremely

11. Using Individual Programming in my software development group would enhance the effectiveness of producing code.
Likely ____\ Unlikely
extremely quite slightly neither slightly quite extremely

12. Using Individual Programming would make it easier for my software development group to produce software code.

Likely \\_______\\_______\_______\_______\_______\_______\_______\\ Unlikely
extremely quite slightly neither slightly quite extremely

13. My software development group would find Individual Programming useful in carrying out their software development responsibilities.
Likely \\_______\\_______\_______\_______\_______\_______\_______\\ Unlikely
extremely quite slightly neither slightly quite extremely

---

14. Using Individual Programming in my software development group would enhance the efficiency of producing code.
Likely \\_______\\_______\_______\_______\_______\_______\_______\\ Unlikely
extremely quite slightly neither slightly quite extremely

Section III

15. Learning to employ and implement Paired Programming would be easy for my software development group.
Likely \\_______\\_______\_______\_______\_______\_______\_______\\ Unlikely
extremely quite slightly neither slightly quite extremely

16. My software development group would find it easy to get Paired Programming to produce the type of software code they want.
Likely \\_______\\_______\_______\_______\_______\_______\_______\\ Unlikely
extremely quite slightly neither slightly quite extremely

17. The implementation of Paired Programming would be clear and understandable to my programming group.
Likely \\_______\\_______\_______\_______\_______\_______\_______\\ Unlikely
extremely quite slightly neither slightly quite extremely

18. My software development group would find Paired Programming to be a flexible and easy to engage practice.
Likely \\_______\\_______\_______\_______\_______\_______\_______\\ Unlikely
extremely quite slightly neither slightly quite extremely

19. It would be easy for my software development group to become skillful at using Paired Programming
Likely \\_______\\_______\_______\_______\_______\_______\_______\\ Unlikely
extremely quite slightly neither slightly quite extremely

20. My software development group would find Paired Programming easy to use.
Likely \\_______\\_______\_______\_______\_______\_______\_______\\ Unlikely
extremely quite slightly neither slightly quite extremely

Section IV

21. Learning to employ and implement Individual Programming would be easy for my software development group.

Likely								Unlikely
	extremely	quite	slightly	neither	slightly	quite	extremely	

22. My software development group would find it easy to get Individual Programming to produce the type of software code they want.

Likely								Unlikely
	extremely	quite	slightly	neither	slightly	quite	extremely	

23. The implementation of Individual Programming would be clear and understandable to my programming group.

Likely								Unlikely
	extremely	quite	slightly	neither	slightly	quite	extremely	

24. My software development group would find Individual Programming to be a flexible and easy to engage practice.

Likely								Unlikely
	extremely	quite	slightly	neither	slightly	quite	extremely	

25. It would be easy for my software development group to become skillful at using Individual Programming

Likely								Unlikely
	extremely	quite	slightly	neither	slightly	quite	extremely	

26. My software development group would find Individual Programming easy to use.

Likely								Unlikely
	extremely	quite	slightly	neither	slightly	quite	extremely	

Section V

27. I intend to use the Paired Programming practice for my software development group in the future.

Likely								Unlikely
	extremely	quite	slightly	neither	slightly	quite	extremely	

28. I intend to use the Individual Programming practice for my software development group in the future.

Likely								Unlikely
	extremely	quite	slightly	neither	slightly	quite	extremely	

Section VI

29. Indicate number of years of programming experience you have in any programming language? *(Select only one)*

☐ None ☐ <1 – 2 ☐ 3 - 6 ☐ 7 - 10 ☐ 11 - 14 ☐ 15 – 18 ☐ >18

30. Please indicate within what general type of business are you currently employed or engaged? *(Select only one)*

☐ Agriculture, Forestry, and Fishing	☐ Wholesale Trade
☐ Mining	☐ Retail Trade
☐ Construction	☐ Finance, Insurance, and Real Estate
☐ Manufacturing	☐ Services
☐ Transportation, Communications, Electric, Gas, and Sanitary Services	☐ Public Administration

31. What is your title/present position? *(Select only one)*

☐ Programmer/Analyst	☐ Software Development Director
☐ Senior Programmer/Analyst	☐ Software Development Manager
☐ Senior Software Development Engineer	☐ Software Development VP
☐ Software Development Group Lead	☐ CIO/CTO

If you would like a copy of the study results please type in your email address here. Your email address will not be shared or distributed to any company or organization and will be used only to send you a copy of the study results.

[Submit Survey] [Exit Survey]